Elette Boyle
Mohammad Mahmoody (Eds.)

LNCS 15367

Theory of Cryptography

22nd International Conference, TCC 2024
Milan, Italy, December 2–6, 2024
Proceedings, Part IV

Springer

Lecture Notes in Computer Science **15367**

The series Lecture Notes in Computer Science (LNCS), including its subseries Lecture Notes in Artificial Intelligence (LNAI) and Lecture Notes in Bioinformatics (LNBI), has established itself as a medium for the publication of new developments in computer science and information technology research, teaching, and education.

LNCS enjoys close cooperation with the computer science R & D community, the series counts many renowned academics among its volume editors and paper authors, and collaborates with prestigious societies. Its mission is to serve this international community by providing an invaluable service, mainly focused on the publication of conference and workshop proceedings and postproceedings. LNCS commenced publication in 1973.

Elette Boyle · Mohammad Mahmoody
Editors

Theory of Cryptography

22nd International Conference, TCC 2024
Milan, Italy, December 2–6, 2024
Proceedings, Part IV

Springer

Editors
Elette Boyle
NTT Research
Sunnyvale, CA, USA

Reichman University
Herzliya, Israel

Mohammad Mahmoody
University of Virginia
Charlottesville, VA, USA

ISSN 0302-9743 ISSN 1611-3349 (electronic)
Lecture Notes in Computer Science
ISBN 978-3-031-78022-6 ISBN 978-3-031-78023-3 (eBook)
https://doi.org/10.1007/978-3-031-78023-3

This Springer imprint is published by the registered company Springer Nature Switzerland AG
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

Preface

The 22nd Theory of Cryptography Conference (TCC 2024) was held during December 2–6, 2024, at Bocconi University in Milano, Italy. It was sponsored by the International Association for Cryptologic Research (IACR). The general chair of the conference was Emmanuela Orsini.

The conference received 172 submissions, of which the Program Committee (PC) selected 68 for presentation, giving an acceptance rate of 39.5%. Each submission was reviewed by at least three PC members in a single-blind process. The 50 PC members (including PC chairs), all top researchers in our field, were helped by 185 external reviewers, who were consulted when appropriate. These proceedings consist of the revised versions of the 68 accepted papers. The revisions were not reviewed, and the authors bear full responsibility for the content of their papers.

We are extremely grateful to Kevin McCurley for providing fast and reliable technical support for the HotCRP review software. We also thank Kay McKelly for her help with the conference website.

This was the tenth year that TCC presented the Test of Time Award to an outstanding paper that was published at TCC at least eight years ago, making a significant contribution to the theory of cryptography, preferably with influence also in other areas of cryptography, theory, and beyond. This year, the Test of Time Award Committee selected the following paper, published at TCC 2004: "Notions of Reducibility between Cryptographic Primitives," by Omer Reingold, Luca Trevisan, and Salil P. Vadhan. The award committee recognized this paper "for providing a rigorous and systematic taxonomy of reductions in cryptography, and in particular coining fully black-box reductions and motivating their use in barrier results."

We are greatly indebted to the many people who were involved in making TCC 2024 a success. Thank you to all the authors who submitted papers to the conference and to the PC members for their hard work, dedication, and diligence in reviewing and selecting the papers. We are also thankful to the external reviewers for their volunteered hard work and investment in reviewing papers and answering questions. Finally, thank you to the general chair Emmanuela Orsini and her team at Bocconi University, as well as to the TCC Steering Committee.

October 2024

Elette Boyle
Mohammad Mahmoody

Organization

General Chair

Emmanuela Orsini	Bocconi University, Italy

Program Committee Chairs

Elette Boyle	Reichman University, Israel & NTT Research, USA
Mohammad Mahmoody	University of Virginia, USA

Steering Committee

Yuval Ishai	Technion, Israel
Huijia (Rachel) Lin	University of Washington, USA
Tal Malkin	Columbia University, USA
Jesper Buus Nielsen	Aarhus University, Denmark
Krzysztof Pietrzak	Institute of Science and Technology Austria, Austria
Manoj M. Prabhakaran	IIT Bombay, India
Salil Vadhan	Harvard University, USA

Program Committee

Prabhanjan Ananth	UC Santa Barbara, USA
Benny Applebaum	Tel Aviv University, Israel
Amos Beimel	Ben-Gurion University of the Negev, Israel
Chris Brzuska	Aalto University, Finland
Yilei Chen	Tsinghua University, China
Ran Cohen	Reichman University, Israel
Geoffroy Couteau	CNRS, IRIF, Université Paris Cité, France
Itai Dinur	Ben-Gurion University of the Negev, Israel
Yevgeniy Dodis	New York University, USA
Stefan Dziembowski	University of Warsaw & IDEAS NCBR, Poland
Nils Fleischhacker	Ruhr University Bochum, Germany

Chaya Ganesh	Indian Institute of Science, Bangalore, India
Aarushi Goel	NTT Research, USA
Siyao Guo	NYU Shanghai, China
Mohammad Hajiabadi	University of Waterloo, Canada
Carmit Hazay	Bar-Ilan University, Israel
Justin Holmgren	NTT Research, USA
Aayush Jain	Carnegie Mellon University, USA
Zhengzhong Jin	Northeastern University, USA
Dakshita Khurana	University of Illinois at Urbana-Champaign, USA
Susumu Kiyoshima	NTT Social Informatics Laboratories, Japan
Lisa Kohl	CWI Amsterdam, Netherlands
Ilan Komargodski	Hebrew University of Jerusalem, Israel & NTT Research, USA
Eyal Kushilevitz	Technion, Israel
Huijia (Rachel) Lin	University of Washington, USA
Alex Lombardi	Princeton University, USA
Fermi Ma	Simons Institute and UC Berkeley, USA
Hemanta K. Maji	Purdue University, USA
Giulio Malavolta	Bocconi University, Italy & Max Planck Institute, Germany
Noam Mazor	Tel Aviv University, Israel
Pierre Meyer	Aarhus University, Denmark
Ryo Nishimaki	NTT Social Informatics Laboratories, Japan
Omer Paneth	Tel Aviv University, Israel
Krzysztof Pietrzak	Institute of Science and Technology Austria, Austria
Manoj Prabhakaran	IIT Bombay, India
Willy Quach	Weizmann Institute of Science, Israel
Divya Ravi	University of Amsterdam, Netherlands
Alon Rosen	Bocconi University, Italy and Reichman University, Israel
Lior Rotem	Stanford University, USA
Peter Scholl	Aarhus University, Denmark
Sruthi Sekar	IIT Bombay, India
Luisa Siniscalchi	Technical University of Denmark, Denmark
Eliad Tsfadia	Georgetown University, USA
Prashant Nalini Vasudevan	National University of Singapore, Singapore
Muthu Venkitasubramaniam	Georgetown University, USA
Mingyuan Wang	UC Berkeley, USA
Daniel Wichs	Northeastern University & NTT Research, USA
Takashi Yamakawa	NTT Social Informatics Laboratories, Japan

Additional Reviewers

Behzad Abdolmaleki
Anasuya Acharya
Amit Agarwal
Divesh Aggarwal
Andris Ambainis
Gilad Asharov
Thomas Attema
David Balbás
Laasya Bangalore
James Bartusek
Tyler Besselman
Rishabh Bhadauria
Kaartik Bhushan
Alexander Bienstock
Aniruddha Biswas
Alexander Block
Jeremiah Blocki
Katharina Boudgoust
Nicholas Brandt
Rares Buhai
Alper Cakan
Matteo Campanelli
Ran Canetti
Rutchathon Chairattana-Apirom
Benjamin Chan
Anirudh Chandramouli
Rohit Chatterjee
Megan Chen
Jessica Chen
Binyi Chen
Arka Rai Choudhuri
Sandro Coretti-Drayton
Quand Dao
Pratish Datta
Giovanni Deligios
Marian Dietz
Fangqi Dong
Nico Döttling
Ehsan Ebrahimi
Christoph Egger
Saroja Erabelli
Grzegorz Fabiański
Pooya Farshim
Giacomo Fenzi
Ben Fisch
Pouyan Forghani
Cody Freitag
Phillip Gajland
Karthik Gajulapalli
Rachit Garg
Sanjam Garg
Riddhi Ghosal
Satrajit Ghosh
Suparno Ghoshal
Niv Gilboa
Eli Goldin
Tian Gong
Junqing Gong
Jiaxin Guan
Aditya Gulati
Taiga Hiroka
Iftach Haitner
David Heath
Aditya Hegde
Hans Heum
Minki Hhan
Yao-ching Hsieh
Zihan Hu
Jihun Hwang
Yuval Ishai
Abhishek Jain
Daniel Jost
Eliran Kachlon
Fatih Kaleoglu
Chethan Kamath
Simon Kamp
Julia Kastner
Shuichi Katsumata
Hannah Keller
Hamidreza Amini Khorasgani
Taechan Kim
Elena Kirshanova
Ohad Klein
Karen Klein
Dimitris Kolonelos
Chelsea Komlo

Manu Kondapaneni
Venkata Koppula
Alexis Korb
Nishat Koti
Roman Langrehr
Seunghoon Lee
Keewoo Lee
Zeyong Li
Yunqi Li
Hanjun Li
Xiao Liang
Fuchun Lin
Chuanwei Lin
Haoxing Lin
Yao-Ting Lin
Tianren Liu
Jiahui Liu
Chen-Da Liu-Zhang
Zhenjian Lu
Donghang Lu
Vadim Lyubashevsky
Ulysse Léchine
Nir Magrafta
Bernardo Magri
Nathan Manohar
Xinyu Mao
Marcin Mielniczuk
Ethan Mook
Tomoyuki Morimae
Changrui Mu
Saachi Mutreja
Anne Müller
Varun Narayanan
Barak Nehoran
Ky Nguyen
Hai Hoang Nguyen
Guilhem Niot
Oded Nir
Aysan Nishaburi
Mahak Pancholi
Aditi Partap
Anat Paskin-Cherniavsky
Rutvik Patel
Shravani Patil
Sikhar Patranabis
Alice Pellet-Mary
Paola de Perthuis
Naty Peter
Spencer Peters
Bertram Poettering
Guru Vamsi Policharla
Alexander Poremba
Luowen Qian
Rajeev Raghunath
Debasish Ray Chawdhuri
Hanlin Ren
Doreen Riepel
Ron D. Rothblum
Adeline Roux-Langlois
Lawrence Roy
Elahe Sadeghi
Pratik Sarkar
Rahul Satish
Benjamin Schlosser
Akash Shah
Jad Silbak
Mark Simkin
Fabrizio Sisinni
Tomer Solomon
Fang Song
Katerina Sotiraki
Noah Stephens-Davidowitz
Gilad Stern
Björn Tackmann
Kel Zin Tan
Er-cheng Tang
Athina Terzoglou
Jean-Pierre Tillich
Pratyush Ranjan Tiwari
Daniel Tschudi
Prashant Vasudevan
Ivan Visconti
Benedikt Wagner
William Wang
Benjamin Wesolowski
Jiawei Wu
David Wu
Yu Xia
Zhiye Xie
Jeff Xu

Anshu Yadav
Sophia Yakoubov
Chao Yan
Yibin Yang
Xiuyu Ye
Eylon Yogev
Albert Yu
Ilias Zadik
Runzhi Zeng

Contents – Part IV

Obfuscation and Homomorphism

Indistinguishability Obfuscation from Bilinear Maps and LPN Variants

Seyoon Ragavan, Neekon Vafa(✉), and Vinod Vaikuntanathan

MIT CSAIL, Cambridge, USA
{sragavan,nvafa,vinodv}@mit.edu

Abstract. We construct an indistinguishability obfuscation (IO) scheme from the sub-exponential hardness of the decisional linear problem on bilinear groups together with two variants of the learning parity with noise (LPN) problem, namely large-field LPN and (binary-field) sparse LPN. This removes the need to assume the existence pseudorandom generators (PRGs) in NC^0 with polynomial stretch from the state-of-the-art construction of IO (Jain, Lin, and Sahai, EUROCRYPT 2022). As an intermediate step in our construction, we abstract away a notion of structured-seed polynomial-stretch PRGs in NC^0 which suffices for IO and is implied by both sparse LPN and the existence of polynomial-stretch PRGs in NC^0.

As immediate applications, from the sub-exponential hardness of the decisional linear assumption on bilinear groups, large-field LPN, and sparse LPN, we get alternative constructions of (a) fully homomorphic encryption (FHE) without lattices or circular security assumptions (Canetti, Lin, Tessaro, and Vaikuntanathan, TCC 2015), and (b) perfect zero-knowledge adaptively-sound succinct non-interactive arguments for NP (Waters and Wu, STOC 2024).

1 Introduction

Indistinguishability obfuscation ($i\mathcal{O}$) [BGI+12] is a probabilistic polynomial-time algorithm $\mathcal{O}$ that takes as input a circuit C and outputs an (obfuscated) circuit $\hat{C} \leftarrow \mathcal{O}(C; r)$ satisfying three properties: (a) functionality: C and $\hat{C}$ compute the same function; (b) efficiency: $\mathcal{O}$ runs in polynomial time; in particular, the size of $\mathcal{O}(C)$ is polynomially related to that of C; and (c) security: for any two circuits C_1 and C_2 that compute the same function (and have the same size), the distributions $\mathcal{O}(C_1)$ and $\mathcal{O}(C_2)$ are computationally indistinguishable. While the utility of the $i\mathcal{O}$ definition was not clear for a while, a large body of results building on the breakthrough work of Sahai and Waters [SW14] changed all of that and showed that $i\mathcal{O}$ is indeed a "central hub" of cryptography, implying the existence of a vast swathe of cryptographic primitives, both old and new, as well as new insights in complexity theory.

The first $i\mathcal{O}$ candidate (without a security reduction) was constructed by Garg, Gentry, Halevi, Raykova, Sahai, and Waters in [GGH+13]. Nearly a decade of work later, Jain, Lin, and Sahai [JLS21] showed how to construct IO assuming

E. Boyle and M. Mahmoody (Eds.): TCC 2024, LNCS 15367, pp. 3–36, 2025.
https://doi.org/10.1007/978-3-031-78023-3_1

the sub-exponential hardness of four rather different, but reasonable and well-founded, computational problems:

1. the decisional linear problem on symmetric bilinear groups of prime order p;[1]
2. the learning parity with noise ($\mathsf{FieldLPN}$) problem over exponentially large fields $\mathbb{Z}_p$ where the noise rate is $1/n^\delta$, n being the LPN dimension and $\delta > 0$ being any constant;
3. the existence of a Boolean pseudorandom generator in NC^0 with *polynomial stretch*, namely stretching n bits to $n^{1+\epsilon}$ bits for some constant $\epsilon > 0$; and
4. the learning with errors (LWE) problem with a sub-exponential modulus-to-noise ratio.

The subsequent work by the same authors [JLS22] eliminated assumption (4), namely the LWE assumption.

Given how central $i\mathcal{O}$ is to theoretical computer science as a whole, it is important to understand the minimal assumptions required to construct it.

This Work: $i\mathcal{O}$ from Bilinear Maps and LPN Variants. We make progress in constructing $i\mathcal{O}$ from weaker assumptions by getting rid of assumption (3) above, namely the existence of a pseudorandom generator in NC^0 with polynomial stretch. Instead, our construction relies on assumption (1), together with two variants of the learning parity with noise problem: the first being assumption (2) as used in [JLS21, JLS22], and the second is the hardness of the sparse learning parity with noise ($\mathsf{SparseLPN}$) problem over $\mathbb{Z}_2$.

> The sub-exponential sparse learning parity with noise ($\mathsf{SparseLPN}$) assumption says that there exist constants $\epsilon, \delta \in \mathbb{R}^+, t \in \mathbb{N}$ and, letting $m = n^{1+\epsilon}$ and $\eta = 1/n^\delta$, a distribution[2] $\mathcal{D}$ over matrices $\mathbf{A} \in \mathbb{Z}_2^{m \times n}$ with t-sparse rows such that for any p.p.t. adversary $\mathcal{A}$,
>
> $$|\Pr[\mathcal{A}(\mathbf{A}, \mathbf{As} \oplus \mathbf{e}) = 1] - \Pr[\mathcal{A}(\mathbf{A}, \mathbf{u}) = 1]| \leq \exp(-n^{\Omega(1)}).$$
>
> where the probability is over $A \leftarrow \mathcal{D}, \mathbf{s} \leftarrow \mathbb{Z}_2^n, \mathbf{e} \leftarrow \mathsf{Bern}(\eta)^m$ and $\mathbf{u} \leftarrow \mathbb{Z}_2^m$.

A few words about the $\mathsf{SparseLPN}$ assumption are in order. First of all, variants of the $\mathsf{SparseLPN}$ problem have been studied in many works in the cryptography, constraint satisfaction, as well as the average-case complexity literature [Gol00, CM01, Fei02, Ale11, MST06, AOW15, AL18, KMOW17, ADI+17]. In

[1] [JLS21] relies on the symmetric external Diffie-Hellman (SXDH) assumption on asymmetric bilinear prime-order groups, but [JLS22] as well as our work rely on the DLIN assumption on symmetric bilinear prime-order groups.

[2] For technical reasons, in the sub-exponential regime, we do not require $\mathcal{D}$ to be efficiently sampleable, but we do require that there be an efficient sampler $\mathcal{D}'$ that is in some sense $\Omega(1)$-close to $\mathcal{D}$. In the negligible but not sub-exponential regime, we can (plausibly) sample good $\mathbf{A}$ efficiently by using a sampler of Applebaum and Kachlon [AK23]. For more details, see Sects. 2.3 and 3.3 and Appendix A of the full version [RVV24].

fact, it has been used to build several cryptographic objects, including public-key encryption, cryptography with constant computational overhead, multi-party homomorphic secret sharing, and pseudorandom correlation generators [Ale11, AIK08, IKOS08, ABW10, BCGI18, DIJL23, DJ24]. Secondly, we note that the parameter regime we require for our SparseLPN assumption seems quantitatively weaker than the one required for public-key encryption.

In particular, Applebaum, Barak, and Wigderson [ABW10] show how to build public-key encryption from a variant of SparseLPN where the sparsity, noise rate, and number of samples are all related.[3] In comparison, we do not require any such relation between these parameters.

With that said, we now state our main theorem.

Theorem 1 (Informal). *Under the sub-exponential hardness of assumptions (1) and (2) and the* SparseLPN *assumption, there exists an $i\mathcal{O}$ scheme.*

Similarly to SparseLPN, assumption (2), namely FieldLPN for any inverse polynomial noise rate, is also weaker than public-key encryption to the best of our knowledge, as the (natural finite-field analog of the) public-key encryption in [Ale11] requires $O(1/\sqrt{n})$ noise rate. Therefore, the only one of these assumptions that implies public-key encryption is assumption (1), the decisional linear problem on symmetric bilinear groups of prime order.[4]

Isn't This Easy? At first sight, it might appear that the SparseLPN assumption directly gives us a polynomial-stretch pseudorandom generator computable in NC^0 (assumption 3) which, together with assumptions 1 and 2, is sufficient for the [JLS22] construction. In fact, Applebaum, Ishai, and Kushilevitz [AIK08] show how to build a *linear-stretch* PRG in NC^0 from this assumption. For [JLS22], we need a polynomial-stretch PRG in NC^0, so it would be natural to try to extend the result of [AIK08] to handle polynomial stretch. Unfortunately, as [AIK08] mention, their techniques do not yield a PRG in NC^0 with superlinear stretch, let alone polynomial stretch.

Let us see what goes wrong with the direct construction. To be more precise, consider the function (family) $g_{\mathbf{A}} : \mathbb{F}_2^n \times \mathbb{F}_2^\ell \to \mathbb{F}_2^m$ for a t-sparse matrix $\mathbf{A}$, that

[3] More precisely, their construction can be adapted to work for constant sparsity t, noise rate $o(1/n^\delta)$, and $m = n^{1+(t/2-1)(1-\delta)}$ samples. In Appendix A of the full version [RVV24], we provide a summary of the existing cryptanalysis on SparseLPN, and explain why this separation between parameter regimes may be inherent.

[4] We remark that it may be possible to also instantiate the $i\mathcal{O}$ construction assuming *sparse* LPN over $\mathbb{Z}_p$ instead of standard FieldLPN, so that our construction would rely on just DLIN as well as the SparseLPN assumption over both $\mathbb{Z}_2$ and large fields of prime order. For this variant of SparseLPN over $\mathbb{Z}_p$, we need sparsity $t = \omega(1)$ to allow for an arbitrary polynomial number of samples. Since the sub-exponential SparseLPN assumption over $\mathbb{Z}_p$ does not hold with all but *sub-exponential* probability over the randomness of uniform sparse $\mathbf{A}$ (for $t = n^{o(1)}$), this would require checking that the PPE construction by [JLS22] can be made compatible with our use of FE combiners. For simplicity, we do not pursue this generalization in the current version.

is, where each row of $\mathbf{A}$ has exactly t non-zero entries:

$$g_{\mathbf{A}}(\mathbf{s}, \mathbf{r}) = \mathbf{A}\mathbf{s} + \mathsf{BinSamp}_p(\mathbf{r}) \bmod 2,$$

where $\mathsf{BinSamp}_p(\mathbf{r})$ outputs a vector $\mathbf{e} \in \mathbb{F}_2^m$ such that if $\mathbf{r}$ is a uniformly random vector, then each entry of $\mathbf{e}$ is an independent Bernoulli random variable with parameter p. If $\mathsf{BinSamp}_p(\mathbf{r})$ can be implemented as an NC^0 function, we will have a polynomial-stretch PRG computable in NC^0 from the $\mathsf{SparseLPN}$ assumption, and we would be done.

This turns out to be impossible. By standard results in the analysis of Boolean functions, any function $f : \{0,1\}^n \to \{0,1\}$ that outputs a sample from $\mathsf{Bern}(\eta)$ given a uniformly random input $\mathbf{r} \leftarrow \{0,1\}^n$ must have locality $\Omega(\log 1/\eta)$; in particular, if $\eta = o(1)$, as is needed for polynomial stretch, the locality is $\omega(1)$, which is insufficient in the construction of [JLS22].

Our Idea. In the outline above, there is no need to choose $\mathbf{r}$ from the uniform distribution; indeed, the distribution of $\mathbf{r}$ can be arbitrary, it turns out, subject to three constraints:

(a) $\mathbf{r}$ shouldn't be too long;
(b) expanding $\mathbf{r}$ into the Bernoulli $\mathbf{e}$ should be doable with a degree-$O(1)$ polynomial over $\mathbb{Z}$; and
(c) $\mathbf{r}$ can be sampled by a circuit whose size is sublinear in the PRG output length.

We note in passing that if it were for conditions (a) and (b) alone, one way to come up with such a distribution of $\mathbf{r}$ is to start with a Bernoulli $\mathbf{e}$ and to compress it using low-rank matrices, much the same way as [JLS21, JLS22]. Multiplying out the low-rank matrices ensures that expanding $\mathbf{r}$ to $\mathbf{e}$ can be done in degree 2. Assuming sub-exponential LWE in addition, we could recover the $i\mathcal{O}$ result [GKP+13, LPST16].

To avoid the need for LWE and the bootstrapping results of [GKP+13, LPST16], we instead *implicitly* sample $\mathbf{e}$ by sampling the list of entries where it is nonzero (which will be sublinear in the length of $\mathbf{e}$), and then directly construct the compressed representation using low-rank matrices as in [JLS21, JLS22]. Doing this with a sublinear-time RAM program is relatively straightforward, but some care is needed to show that this can be implemented as a sublinear-size circuit.

More generally, we abstract away our needs from sparse LPN into two separate objects that are both weaker than the existence of a polynomial-stretch PRG in NC^0.

Theorem 2 (Informal, following [JLS22]). *There exists an $i\mathcal{O}$ scheme under assumptions (1) and (2) as well as the existence of*

(a) any "structured-seed" polynomial-stretch Boolean PRG computable by degree-$O(1)$ polynomials over $\mathbb{Z}$ with an arbitrarily small polynomial locality (see Sect. 2 and Definition 7 for a formal definition); and

(b) any linear stretch Boolean PRG *in* NC^0.

Both the SparseLPN assumption as well as the existence of Boolean PRGs in NC^0 imply the existence of both objects (a) and (b) above. Our theorem is thus a common generalization of both [JLS22] and our result, a fact that we hope will be useful in further constructions of $i\mathcal{O}$ from fewer and/or simpler assumptions.

Generalizing SparseLPN *and poly-stretch PRGs in* NC^0. In fact, objects (a) and (b) can be constructed from a much more general assumption, which we refer to as "local functions with noise" or LFN.[5] Informally, the LFN assumption states that we can efficiently sample a Boolean function $f : \mathbb{Z}_2^n \to \mathbb{Z}_2^m$ with locality $t = O(1)$ such that $f(\mathbf{s}) \oplus \mathbf{e}$ looks pseudorandom, where $\mathbf{s} \leftarrow \mathbb{Z}_2^n$, $\mathbf{e} \leftarrow \mathsf{Bern}(n^{-\delta})^m$, and $m = n^{1+\epsilon}$. Alternatively, one can view this assumption as a strengthening of the hardness of $n^{-\delta}$-refutations for balanced random constraint satisfaction problems (CSPs). We refer the reader to Sect. 3.4 for details.

Note that the LFN assumption is implied by either PRGs in NC^0 or SparseLPN: when we do not have any noise, it is the former, and when the function f is linear, it is the latter. Another possible instantiation of LFN is to use Goldreich's PRG [Gol00] with additional noise of rate $n^{-\delta}$.

1.1 Consequences

A rich line of work initiated by [SW14] has shown a plethora of applications of $i\mathcal{O}$ to other problems in cryptography. As a result, our construction implies instantiations of these cryptographic primitives from sets of assumptions that were not previously known, in particular the *sub-exponential* hardness of assumptions (1) and (2) and the SparseLPN (or LFN) assumption. We list some of these below, following [JLS22].

- Fully homomorphic encryption (FHE), noting that bilinear DLIN implies a perfectly rerandomizable encryption scheme [CLTV15].
- Adaptively sound perfectly zero-knowledge succinct non-interactive argument (SNARG) system for any NP language in the CRS model, with CRS length $\mathsf{poly}(\lambda + |C|)$. Here, we define C to be the NP verification circuit [SW14, WW24a, WW24b].
- Public-key functional encryption for Turing machines that is fully succinct and adaptively secure against unbounded collusions [AS16]. Full succinctness means that the runtime of encrypting an input $\mathbf{x} \in \{0,1\}^*$ is simply $\mathsf{poly}(\lambda, |\mathbf{x}|)$, independent of the size and the runtime of the Turing machine being evaluated on $\mathbf{x}$.
- Witness encryption for any NP language, as a special case of $i\mathcal{O}$ for P/poly.
- Secret sharing for any monotone function in NP [KNY17].
- Multiparty non-interactive key exchange in the plain model (without trusted setup) [BZ14, KRS15].

[5] We thank Aayush Jain, Rachel Lin, and an anonymous TCC reviewer for suggesting this generalization.

- Sender deniable encryption [SW14], and fully deniable interactive encryption [CPP20].
- Constant-round concurrent zero-knowledge proofs for all of NP [CLP15].

In addition, assuming the *polynomial* hardness of assumptions (1) and (2) and the SparseLPN assumption (over the explicit distribution of sparse $\mathbf{A}$ matrices given by [AK23]; see Sect. 3.3), we obtain the following (by polynomial hardness, we mean that p.p.t. adversaries in any of the assumptions achieve negligible advantage):

- Public-key functional encryption for polynomial-size circuits that is *adaptively* secure against unbounded collusions and fully succinct, i.e., the runtime of encrypting an input $\mathbf{x}$ is independent of the size of the circuit being evaluated on $\mathbf{x}$ [GS16, LM16, ABSV15, KNTY19]. This follows from our construction in Sect. 7.2 of the full version [RVV24] of PKFE for polynomial-size circuits that is selectively secure with a single key and weakly succinct.
- As a special case of the above, attribute-based encryption for polynomial-size circuits.
- Hardness of PPAD [AKV04, BPR15, GPS16, HY20, KS20].

Organization of the Paper. In Sect. 2, we provide a technical outline of our construction. In Sect. 3, we set up some notation and formally introduce our LPN, SparseLPN, and LFN assumptions. In Sect. 4, we give the construction of structured-seed PRGs from LFN; this is one of the main modifications we make to the construction of [JLS22]. We refer the reader to Sect. 5 onwards in the full version [RVV24] for all remaining details; we first closely follow the constructions and analysis by [JLS22] of ARE, PRE, PKFE, and then make a white-box modification to the SKFE combiner by [JMS20] to bootstrap this to sub-exponentially secure SKFE, which can then be bootstrapped to $i\mathcal{O}$ [KNT22].

2 Technical Overview

Our starting point is the construction of $i\mathcal{O}$ from [JLS22]. Most of our construction follows exactly the same template as in [JLS22], with the main exception being that we replace the polynomial-stretch PRG in NC^0 with weaker forms of pseudorandomness. Our main idea can be split into two:

1. The requirement of a polynomial-stretch PRG in NC^0, as needed in framework of [JLS22], can be replaced with two weaker objects: (a) a *linear*-stretch PRG in NC^0, and (b) a polynomial-stretch *structured-seed PRG* (SPRG), which we define shortly.
2. We can build both such PRGs from the assumption that LFN holds for some fixed constant locality, some fixed polynomial number of samples, and some fixed inverse polynomial error rate. A linear-stretch PRG in NC^0 from LFN comes from a straightforward modification of [AIK08]. We sketch how to construct a polynomial-stretch SPRG from LFN later in this section.

Informally (ignoring some technicalities), the definition of an SPRG with polynomial stretch is as follows. We say G is an SPRG (with polynomial stretch and outputs in $\{0,1\}^m$) if there is a randomized Boolean circuit SdSamp with the following properties:

1. Pseudorandomness over seeds from SdSamp, which says
$$\{G(\mathsf{seed}) \mid \mathsf{seed} \leftarrow \mathsf{SdSamp}(m)\} \approx_c U_m,$$
where U_m is the uniform distribution over $\{0,1\}^m$, and SdSamp is supported on $\{0,1\}^{m^{1-\Omega(1)}}$;
2. The randomized circuit SdSamp has size $m^{1-\Omega(1)}$; and
3. G can be written as a polynomial of total degree $d = O(1)$ over $\mathbb{Z}$ with locality $O(m^\tau)$ for arbitrarily small constant $\tau > 0$.

This relaxes the notion of a PRG in NC^0 with polynomial stretch in two main ways: (1) the seed does not have to be uniformly random (just sampleable with sublinear efficiency), and (2) the locality can be an arbitrarily small polynomial in m, as long as the *degree* of the polynomial over $\mathbb{Z}$ is bounded by a constant $d = O(1)$. We emphasize that our degree condition is over $\mathbb{Z}$, but we require pseudorandomness over $\{0,1\}^m$.

Comparison with Structured-Seed PRG*s in* [JLS21]. We note that the $i\mathcal{O}$ construction of [JLS21] relies on another notion of structured-seed PRG that is incomparable to ours. Most importantly, we require SdSamp to have circuit size $m^{1-\Omega(1)}$, while [JLS21] imposes no such requirement. This is because [JLS21] ultimately constructs sublinear size-succinct FE, which they then bootstrap to sublinear time-succinct FE assuming LWE [GKP+13, LPST16] before finally bootstrapping to $i\mathcal{O}$ [AJ15, BV18]. We do not rely on LWE, so SdSamp must itself have an efficient circuit.

Secondly, [JLS21] allows SdSamp to generate a public and a private seed, and requires that decompression be degree 2 in the private seed and degree $O(1)$ in the public seed. In our case, it suffices for decompression to have degree $O(1)$, so we do not need to work with a public seed. In the other direction, it will be important to us for technical reasons that the locality of G can be bounded by an arbitrarily small polynomial in its output length, whereas [JLS21] does not require any such restriction.

Finally, we construct SPRG assuming only LFN. It can also be trivially instantiated from a (polynomial-stretch) PRG in NC^0. In contrast, the [JLS21] construction of their notion of structured-seed PRG requires *both* a (polynomial-stretch) PRG in NC^0 and LPN over $\mathbb{Z}_p$.

2.1 Weakening the Polynomial-Stretch PRG in NC^0 in [JLS22]

Before describing why these weaker forms of pseudorandomness are sufficient to replace the polynomial-stretch PRG in NC^0 in [JLS22], we briefly summarize the overall template in [JLS22].

The starting point in [JLS22] is the notion of partially-hiding functional encryption (PHFE) [GVW15], which can be built from the DLIN assumption on symmetric bilinear groups of prime order [JLS19, GJLS21, Wee20]. This gives a special form of functional encryption (FE), where function keys can be given to functions that are degree-2 (over $\mathbb{Z}_p$) over the secret input SI, but allowed to be degree-$O(1)$ (or more generally, NC^1) over a *public* input PI that the FE scheme does not need to hide. If one could turn this into a sublinear time-succinct FE scheme for all polynomial size circuits (with sub-exponential security), then using known bootstrapping results [BV18, AJ15, KNT22], one gets $i\mathcal{O}$. By *time-succinct*, we mean that the time to generate a ciphertext (or more accurately, the size of the circuit generating the ciphertext) should be sublinear in the size of the circuit given in the function keys. More precisely, to handle function keys for circuits $C : \{0,1\}^n \to \{0,1\}^m$ of size s, we require the time to compute FE.Enc (more accurately, the circuit size) to be $s^{1-\Omega(1)} \cdot \mathsf{poly}(n, \lambda)$.

This is a stronger notion than *size-succinct* FE, which only requires that the size of the ciphertext (i.e., output length of FE.Enc) be sublinear, with no sublinear constraint on the time needed to generate the ciphertext. For size-succinct FE, we only know how to get $i\mathcal{O}$ from additionally assuming the learning with errors assumption (LWE) [LPST16, GKP+13], which we do not want to rely on.

Jain, Lin, and Sahai [JLS22] then define a cryptographic object called a Preprocessed Randomized Encoding (PRE), which exactly converts the PHFE as described above into a time-succinct FE for all polynomial size circuits, resulting in a construction of $i\mathcal{O}$. Roughly speaking, PRE splits up the computation of a given circuit C on an input $\mathbf{x}$ into 2 steps: (a) a preprocessing step, that needs to be time-succinct, generating $(\mathsf{PI}, \mathsf{SI})$ from $\mathbf{x}$, and (b) an encoding step, which is degree $(O(1), 2)$ in $(\mathsf{PI}, \mathsf{SI})$ (i.e., total degree $O(1)$ in PI and total degree 2 in SI), that outputs a randomized encoding of $C(\mathbf{x})$. One can directly plug this into a PHFE to get time-succinct FE for all polynomial-size circuits (assuming the PHFE encryption time is linear, which it is).

To build PRE, which is sufficient for $i\mathcal{O}$ as explained above, [JLS22] build it modularly from two objects that they construct: a Preprocessed Polynomial Encoding (PPE), and an Amortized Randomized Encoding (ARE). Roughly speaking, PPE preprocesses any $\mathbf{x}$ into $(\mathsf{PI}, \mathsf{SI})$ in such a way that any degree-$O(1)$ computation on $\mathbf{x}$ (over $\mathbb{Z}_p$) is turned into a degree-$(O(1), 2)$ computation in $(\mathsf{PI}, \mathsf{SI})$. Crucially, this preprocessing step is time-succinct. [JLS22] show how to build PPE directly from (standard) LPN over $\mathbb{Z}_p$ with polynomially many samples and any inverse polynomial error rate, by constructing a special-purpose homomorphic encryption scheme tailored to constant-degree computations.

Their last missing piece, ARE, generates a (binary) randomized encoding of computing a circuit C on input $\mathbf{x}$ (e.g., using Yao's garbled circuits [Yao86]) in such a way that the encoding algorithm has constant locality, i.e., is in NC^0. Since the encoding algorithm is in NC^0, taking the multilinear representation over $\mathbb{Z}$, this directly becomes a degree-$O(1)$ computation over $\mathbb{Z}$ and hence $\mathbb{Z}_p$. Plugging such an ARE into PPE directly gives PRE, as desired. (For simplicity,

we gloss over the amortization constraint here, which allows their composition with PPE to be time-succinct.)

One important property here is that Yao's garbled circuits needs pseudorandomness in two ways:

1. For computing the garbled tables in Yao's garbled circuits, the encoding and decoding requires computing a length-doubling PRG to preserve pseudorandomness of unused entries in the garbled tables. To retain $O(1)$-degree over $\mathbb{Z}$ in the encoding, [JLS22] assumes the existence of a polynomial-stretch PRG in NC^0, which, in particular, gives a linear-stretch PRG in NC^0.
2. The wire labels in the garbled circuit need to be random and hidden. Since PPE does not guarantee any function privacy, we need to hide this randomness in the input to PPE. To retain time-succinctness of PPE preprocessing, the randomized encoding uses a polynomial-stretch PRG by including the seed as part of the input to PPE. To retain $O(1)$-degree over $\mathbb{Z}$, [JLS22] assumes the existence of a polynomial-stretch PRG in NC^0 so that the encoding is still $O(1)$-degree over $\mathbb{Z}$. (For technical reasons, for time-succinctness, when using the PPE, we need the number of monomials to be arbitrarily close to linear in the output length of the randomized encoding.)

This is exactly the place where we can use weaker forms of pseudorandomness than a polynomial-stretch PRG in NC^0. To solve Item 1, we can directly use a linear-stretch PRG in NC^0, and to solve Item 2, we can exactly use our notion of a structured-seed PRG. As explained above, being degree $O(1)$ over $\mathbb{Z}$ is sufficient, and to retain time-succinctness, we need to make sure that PRE, and therefore ARE, can sample these seeds in a time-succinct way. This is exactly the constraint we have on seed sampling in an SPRG.

We briefly explain why an SPRG alone (i.e., without the standard linear-stretch PRG) is insufficient. For Yao's garbled circuit evaluation, we need composability of the linear-stretch PRG, as we feed the outputs of the the linear-stretch PRG as an input to the PRG in the next gate of the circuit. Unfortunately, having a structured seed ruins this composability, as sampling the seed from (unstructured) PRG output need not be degree $O(1)$. Alternatively, for the polynomial-stretch PRG, we only need to evaluate it once, so there is no need for composability; the seed sampling can just happen once at the PRE preprocessing level, as long as it is time-succinct.

More generally, taking a step back, we view our definition of SPRG as naturally fitting into the [JLS22] paradigm. Just as in [JLS22], we separate the computation (in this case, PRG evaluation) into 2 steps: (1) an efficient preprocessing step (i.e., in a time-succinct way), and (2) computing degree-$O(1)$ polynomials on top of the preprocessing. We exactly harness this flexibility that is built into the [JLS22] framework to build our SPRG.

We summarize our construction and how it compares with the construction of [JLS22] in Fig. 1.

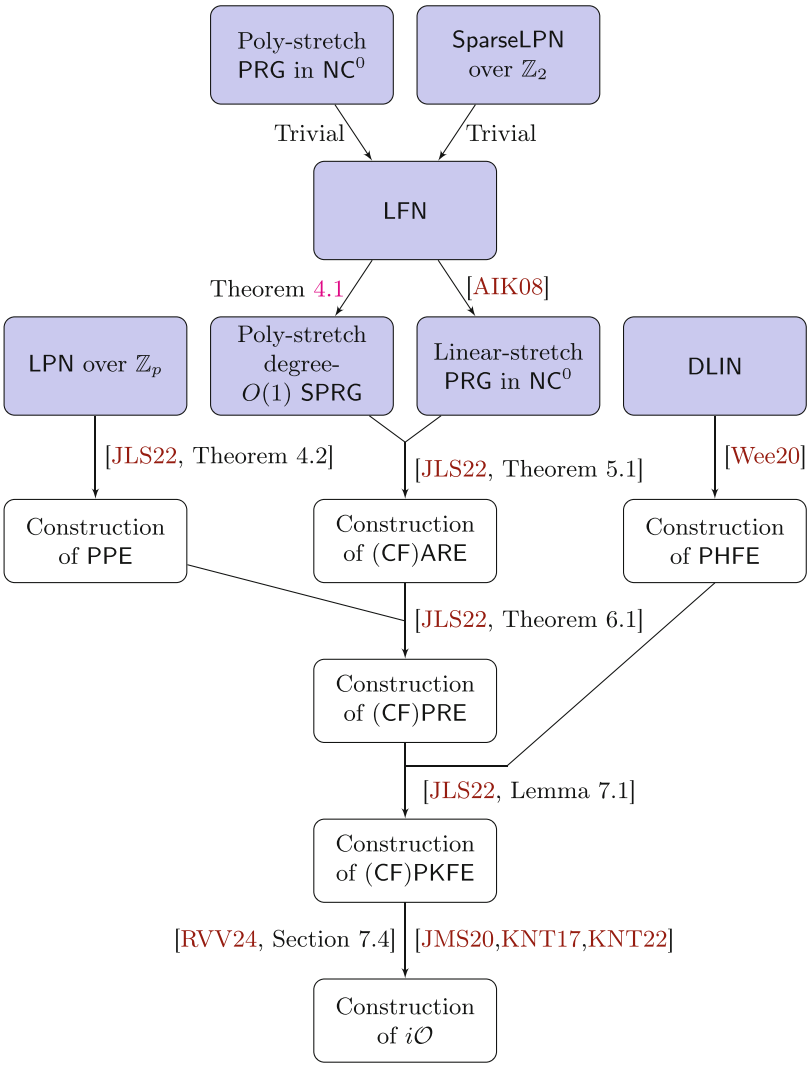


Fig. 1. Flowchart depicting our technical outline to get to $i\mathcal{O}$, following the framework of [JLS22]. The prefix CF stands for "combiner-friendly", and we inherit the notions of PPE (Preprocessed Polynomial Encoding), ARE (Amortized Randomized Encoding), PRE (Preprocessed Randomized Encoding), PHFE (Partially Hiding Functional Encryption), and PKFE (Public-Key Functional Encryption) from [JLS22]. Our observation is that [JLS22] ultimately needs two abstractions from their PRG in NC^0, namely a poly-stretch degree-$O(1)$ structured-seed PRG, and a linear-stretch PRG in NC^0. We show in this work that these two abstractions can also be instantiated assuming LFN, thus providing an alternate construction of $i\mathcal{O}$. Note that everything here needs to be sub-exponentially secure to get to $i\mathcal{O}$. We provide more details on the bootstrapping involved in the final arrow in Fig. 2, and refer the reader to the full version [RVV24] for details.

2.2 Our SPRG Construction from LFN

In this section, we explain how we construct the two weaker pseudorandom objects that we need, namely a linear-stretch PRG in NC^0 and a polynomial-stretch structured-seed PRG. It has been shown by [AIK08] that the SparseLPN assumption implies the existence of a linear-stretch PRG in NC^0. However, as the authors of [AIK08] mention, their techniques do not yield a PRG in NC^0 with superlinear stretch, let alone polynomial stretch. The proof by [AIK08] also works for the more general LFN assumption which is why we are able to rely on this assumption. For simplicity, we focus on the particular case of SparseLPN for the remainder of the technical overview, but we emphasize that everything below goes through in the exact same way for LFN.

As explained in Sect. 2.1, we observe that the polynomial-stretch PRG would only be needed to generate pseudorandom bits to use instead of randomness for Yao's garbled circuits [Yao86]; it does not matter whether the seed is uniformly random or has some structure. SparseLPN appears like a natural candidate for this functionality: given a secret vector $\mathbf{s} \in \mathbb{Z}_2^n$ and a sparse (hence, structured) error vector $\mathbf{e} \in \mathbb{Z}_2^m$, it expands $\mathbf{s}$ to $g_{\mathbf{A}}(\mathbf{s}, \mathbf{e}) := \mathbf{A}\mathbf{s} \oplus \mathbf{e} \in \mathbb{Z}_2^m$, where $\{g_{\mathbf{A}}\}_{\mathbf{A}}$ would comprise a candidate SPRG family. The map $\mathbf{s} \mapsto \mathbf{A}\mathbf{s}$ has locality $O(1)$ due to the row sparsity of $\mathbf{A}$, and hence it has degree $O(1)$ over $\mathbb{Z}$. (We remark that $\mathbf{s} \mapsto \mathbf{A}\mathbf{s}$ is linear over $\mathbb{Z}_2$, but this in and of itself does not imply anything nontrivial about its degree over $\mathbb{Z}$.) Similarly, the map $(\mathbf{A}\mathbf{s}, \mathbf{e}) \mapsto \mathbf{A}\mathbf{s} \oplus \mathbf{e}$ has locality 2 and hence also has degree $O(1)$ over $\mathbb{Z}$. It follows that $g_{\mathbf{A}}$ has locality $O(1)$ (and hence degree $O(1)$ over $\mathbb{Z}$).

However, there remains the conundrum of how the error vector should be handled. If the error vector is included as-is in the structured seed, the resulting "SPRG" would map $m+n$ to m bits and hence not even be expanding. Instead, we could try to sample $\mathbf{e}$ using a polynomially smaller number of uniformly random bits. However, by standard facts from the analysis of Boolean functions, it is impossible to generate samples from $\mathsf{Bern}(\eta)$ for $\eta = o(1)$ from any $O(1)$-degree or NC^0 function over $\mathbb{Z}$ from uniformly random bits, regardless of locality [O'D14, e.g., Exercise 1.11].

Instead, we approach this problem by compressing the error vector into some $\widetilde{\mathbf{e}} \in \{0,1\}^{m'}$, where $m' = m^{1-\Omega(1)}$, such that the decompression map $\widetilde{\mathbf{e}} \mapsto \mathbf{e}$ has degree $O(1)$ over $\mathbb{Z}$. Since $\mathbf{e}$ is polynomially sparse, we can use the beautiful idea given by [JLS21, JLS22] of using low-rank matrix decompositions. We now provide an overview of this technique.

Given the SparseLPN noise rate $\eta = m^{-\Omega(1)}$, we can set parameters ℓ, L such that $L \ll 1/\eta$ is a small polynomial in m and $\ell \cdot L^2 = m$. We can accordingly reorganize the entries of $\mathbf{e} \in \{0,1\}^m$ into ℓ matrices $\mathbf{M}_1, \ldots, \mathbf{M}_\ell \in \{0,1\}^{L \times L} \subseteq \mathbb{Z}^{L \times L}$. For each i, the entries in $\mathbf{M}_i$ are independently sampled Bernoulli random variables with probability η. Hence the expected number of nonzero entries in $\mathbf{M}_i$ is $L^2\eta \ll L$. In particular, the rank of $\mathbf{M}_i$ over $\mathbb{Z}$ will be at most $L' = O(L^2\eta)$. We can hence write each $\mathbf{M}_i = \mathbf{U}_i \cdot \mathbf{V}_i$ where $\mathbf{U}_i, \mathbf{V}_i^\top \in \mathbb{Z}^{L \times L'}$ (in fact, $\{0,1\}^{L \times L'}$), and finally output $\{(\mathbf{U}_i, \mathbf{V}_i)\}_{i \in [\ell]}$ as the compressed error $\widetilde{\mathbf{e}}$. Decompression can

be done in degree 2 over $\mathbb{Z}$, and the size of $\widetilde{\mathbf{e}}$ is $O(\ell \cdot L \cdot L') = O(\ell \cdot L^3 \eta) = O(m^{1-\Omega(1)})$.

This would appear to immediately give us a SPRG from $\mathsf{SparseLPN}$. The problem is that although this structured seed is polynomially smaller than m, the time taken to construct it could be polynomial in m (in particular, it would have to be at least $\Omega(m)$ just to read $\mathbf{e}$). Instantiating the [JLS22] scheme with this construction would yield a sublinear *size*-succinct FE scheme, where the ciphertexts are succinct but we do not impose a constraint on the time taken to generate them. This would have to first be bootstrapped to sublinear *time*-succinct FE [GKP+13,LPST16] assuming LWE, which can then be bootstrapped to $i\mathcal{O}$ [BV18,AJ15,KNT17,KNT22]. Ideally, we would want to construct $i\mathcal{O}$ without the need for LWE and the bootstrapping results of [GKP+13,LPST16].

To do this, we need to sample $\widetilde{\mathbf{e}}$ with a circuit of size $m^{1-\Omega(1)}$. In particular, we cannot even explicitly sample $\mathbf{e} \sim \mathsf{Bern}(\eta)^m$. Instead, we implicitly sample it as follows: we first sample its Hamming weight which will be $\mathsf{wt} = O(m\eta) = O(m^{1-\Omega(1)})$ with all but sub-exponentially small probability. Then, rather than sampling $\mathbf{e}$ in its entirety, we simply sample the list of locations where $\mathbf{e}$ is nonzero. This information is sufficient to directly construct the matrices $\{\mathbf{U}_i, \mathbf{V}_i\}_{i\in[\ell]}$. This will be doable in time $\mathsf{wt} \cdot \mathsf{poly}(\log m) = O(m^{1-\Omega(1)})$.

As in [JLS22], we need to take particular care to ensure that our algorithm is implementable with a sublinear-size *circuit*, rather than a RAM program (this is because the bootstrapping results of [BV18,AJ15,KNT17,KNT22] impose this requirement on the encryption of the FE, which is where our SPRG will sample its structured seed). Fortunately, this is not too difficult and follows using similar approaches to those used by [JLS22]; at a high level, our algorithm can be decomposed into sorting steps [AKS83] and $O(1)$-tape Turing machine computations [PF79], both of which are known to be achievable using circuits with low overheads.

We make some remarks comparing our use of the techniques by [JLS22] for compressing with low-rank matrices and doing this with a sublinear-time circuit, with how they were originally used in [JLS22]. Firstly, while we use these techniques to instantiate the SPRG (which is intended to replace the PRG in NC^0 used by [JLS22]), these techniques were originally used by [JLS22] to instantiate Preprocessed Polynomial Encodings (PPE) assuming LPN over $\mathbb{Z}_p$. We make black-box use of their PPE construction, and hence our construction implicitly relies on these techniques in two different places: our SPRG and the PPE of [JLS22].

Secondly, our use of these techniques is simpler than that of [JLS22]. This is because the sparse vector that we need to compress has very little structure: its entries are simply i.i.d. Bernoulli random variables. On the other hand, the vector that [JLS22] needs to compress is also polynomially sparse but is highly structured; it contains information about how LPN errors propagate through homomorphic evaluations in the special-purpose homomorphic encryption scheme that they use. Keeping track of these errors requires much more careful bookkeeping than we need for our purposes. Although technically their construction also only

requires the results of [PF79, AKS83], they crucially rely on the observation that these results can be used to efficiently make batched non-adaptive RAM queries to a database [JLS22, Lemma 4.6], whereas our construction does not need such complex functionality.

2.3 Our Use of FE Combiners

Let $\mathsf{SparseMat}(t, n, m)$ denote the set of matrices $\mathbf{A} \in \mathbb{Z}_2^{m \times n}$ whose rows all have sparsity exactly t. An "ideal" version of the $\mathsf{SparseLPN}$ assumption would say we have the sub-exponential indistinguishability

$$\begin{aligned} &\{(\mathbf{A}, \mathbf{u} = \mathbf{A}\mathbf{s} \oplus \mathbf{e}) \mid \mathbf{A} \leftarrow \mathsf{SparseMat}(t, n, m), \mathbf{s} \leftarrow \mathbb{Z}_2^n, \mathbf{e} \leftarrow \mathsf{Bern}(\eta)^m\} \\ \approx_c &\{(\mathbf{A}, \mathbf{u}) \mid \mathbf{A} \leftarrow \mathsf{SparseMat}(t, n, m), \mathbf{u} \leftarrow \mathbb{Z}_2^m\}. \end{aligned}$$

However, for $t = O(1)$, this is *false*. In particular, with at least $1/\mathsf{poly}(n)$ probability, there exist two identical rows of $\mathbf{A}$, implying there is an (efficiently computable) vector of sparsity 2 in the left kernel of $\mathbf{A}$. By left-multiplying this vector with $\mathbf{u}$, this can be used to break indistinguishability when $\eta = o(1)$.

For $\mathbf{A}$ that do not have such sparse vectors in the left kernel (more formally, when the dual distance of $\mathbf{A}$ is large; see Appendix A of the full version), there are no known attacks that work with better than sub-exponential advantage. For the negligible (but not sub-exponential) security regime, [AK23] gives an efficient sampler for sparse matrices $\mathbf{A}$ for which negligible security is plausible. Therefore, there is no issue with the above-mentioned template to instantiate negligible-secure sublinear-time (single-key) public-key FE. This allows for a simpler construction of public-key FE and its downstream applications in the negligible (but not sub-exponential) security regime.

However, to bootstrap to $i\mathcal{O}$, one needs sub-exponential security. Unfortunately, there is no known algorithm to efficiently sample these good $\mathbf{A}$ with plausible sub-exponential security.[6] Thus, our sparse LPN assumption has the flavor that there exist *not necessarily efficiently sampleable* distributions $\mathsf{GoodSparseMat}(t, n, m)$ and $\mathsf{BadSparseMat}(t, n, m)$ such that $\mathsf{SparseMat}(t, n, m)$ can be written as a mixture of $\mathsf{GoodSparseMat}(t, n, m)$ and $\mathsf{BadSparseMat}(t, n, m)$, with weight at least $\mu = \Omega(1)$ on $\mathsf{GoodSparseMat}(t, n, m)$, and that (sub-exponential) indistinguishability holds with respect to $\mathbf{A} \leftarrow \mathsf{GoodSparseMat}(t, n, m)$. The issue is that we only know how to efficiently sample

[6] The [AK23] sampler plausibly samples sub-exponentially secure $\mathbf{A}$ (i.e., with sufficient dual distance) with negligible probability. Hence we could also get a construction of "$i\mathcal{O}$ with public parameters" that is plausibly sub-exponentially secure with all but negligible probability over the selection of public parameters. However, we believe that our $i\mathcal{O}$ construction with combiners has the advantages of a) not requiring public parameters; b) being truly sub-exponentially secure; and c) being more robust to further cryptanalysis of $\mathsf{SparseLPN}$ beyond the linear test framework that might break the [AK23] sampler, for example. Moreover, we obtain all these benefits without requiring any additional assumptions.

from $\mathsf{SparseMat}(t, n, m)$, but we can claim sub-exponential security only when sampling from $\mathsf{GoodSparseMat}(t, n, m)$.

To address this problem, there are two natural approaches:

Idea 1 The first idea would be to combine SPRG outputs using the standard XOR approach: given sparse LPN matrices $\mathbf{A}_1, \mathbf{A}_2, \ldots, \mathbf{A}_t$, we could instead consider a "combined SPRG" that independently calls $\mathsf{SPRG.SdSamp}$ t times to obtain (structured) seeds $\mathbf{r}_1, \ldots, \mathbf{r}_t$, then at the decompression stage computes $\bigoplus_{i=1}^{t} \mathsf{SPRG.Eval}(\mathbf{A}_i, \mathbf{r}_i)$. This will be secure as long as at least one of the sparse LPN matrices $\mathbf{A}_1, \ldots, \mathbf{A}_t$ is "good."
The problem we now face is that if we want one of the sparse LPN matrices to be good with all but a sub-exponentially small probability, t will have to be $\lambda^{\Omega(1)}$, resulting in the degree of the SPRG decompression growing too fast.
On the other hand, if $t = O(1)$, then the SPRG decompression will still be constant degree. Jumping ahead, this observation will turn out to be helpful for us, even though this idea on its own will not directly achieve sub-exponential security.

Idea 2 Alternatively, we could instantiate polynomially many copies of our final FE scheme and plug these into an unconditional FE combiner [ABJ+19, JMS20].
The central problem with this approach is that the combiner of [JMS20] does not directly preserve sublinear time-succinctness of the FE. In their construction, their first step is to use $O(1)$-*nested* FE candidates [ABJ+19], which is a direct construction of an FE combiner for $O(1)$ candidates. Informally, for a constant $B = O(1)$, B-nesting FE candidates $\{\mathsf{FE}_i\}_{i\in[B]}$ refers to the following construction. To encrypt, set

$$\mathsf{FENest.Enc}(\mathbf{x}) = \mathsf{FE}_B.\mathsf{Enc}(\cdots(\mathsf{FE}_2.\mathsf{Enc}(\mathsf{FE}_1.\mathsf{Enc}(\mathbf{x})))\cdots).$$

To produce a function key $\mathsf{FENest.KeyGen}(\mathsf{MSK}, C)$, one releases SK_B, defined by the following iterative process: $\mathsf{SK}_1 = \mathsf{FE}_1.\mathsf{KeyGen}(\mathsf{MSK}_1, C)$, and for $i \in [B-1]$, $G_i = \mathsf{FE}_i.\mathsf{Dec}(\mathsf{SK}_i, \cdot)$, where $\mathsf{SK}_{i+1} = \mathsf{FE}_{i+1}.\mathsf{KeyGen}(\mathsf{MSK}_{i+1}, G_i)$. Since there is no constraint on the size of the decryption circuits, there is no guarantee that we get time succinctness.

However, it turns out that we can integrate these two ideas by making white-box use of the results of [JMS20]. Let's first recall the ideas underlying their combiner. Given candidate FE schemes $\{\mathsf{FE}_i\}_{i\in[t]}$, the combiner of [JMS20] comprises two steps:

1. They first use a naive 3-nesting-based approach to construct FE schemes $\{\mathsf{FE}_{i_1,i_2,i_3}\}_{i_1,i_2,i_3\in[t]}$. The only structure they need here is the existence of an index $i^* \in [t]$ such that the schemes $\{\mathsf{FE}_{i^*,i_2,i_3}\}, \{\mathsf{FE}_{i_1,i^*,i_3}\}, \{\mathsf{FE}_{i_1,i_2,i^*}\}$ are all secure. However, this is the part of their construction that does not preserve sublinear time-succinctness.

2. They then use these nested FE candidates to compute a transcript of an "input-local" semi-honest t-party secure multi-party computation (MPC) protocol, with up to $t-1$ corrupted parties. Roughly speaking, for $i \in [t]$, each party P_i gets an XOR secret share $\mathbf{x}_i$ of the input $\mathbf{x}$, and they together run the MPC protocol for outputting $C(\mathbf{x}_1 \oplus \cdots \oplus \mathbf{x}_t)$ for a given circuit C. Each bit of the transcript depends only on 3 parties (and their correlated randomness), and they use the nested FE candidates to compute each bit of the transcript, from which the MPC output can be recovered. As long as there is some $i^* \in [t]$ such that all 3-nested FE candidates that include i^* are secure, then they can argue security of the overall FE scheme. By instantiating this template with a special form of Yao's garbled circuits [Yao86, GS22, GIS18] and the [GMW87] MPC protocol, the efficiency can be made linear in the circuit size, which is sufficient to preserve time-succinctness. (They rely only on the existence of one-way functions, in particular by instantiating the correlated randomness model with PRFs.)
   Another way to view this step is that it bootstraps the 3-nesting construction, which is secure provided at least one of 3 FE schemes is secure, to a FE scheme which is secure provided at least one of $\mathsf{poly}(\lambda)$ FE schemes is secure. Crucially, this bootstrapping preserves succinctness.

Our observation is that Step 1 does not specifically need to use 3-nesting to construct the schemes $\{\mathsf{FE}_{i_1,i_2,i_3}\}$. Rather, all they need is:

1. the property that if at least one of $\mathsf{FE}_{i_1}, \mathsf{FE}_{i_2}, \mathsf{FE}_{i_3}$ is secure, then $\mathsf{FE}_{i_1,i_2,i_3}$ is secure; and
2. the property that the encryption of $\mathsf{FE}_{i_1,i_2,i_3}$ is sublinear time-succinct.

We formalize these properties through the notion of a *combiner-friendly* secret-key functional encryption scheme (CFSKFE). A CFSKFE scheme samples B common reference strings in a setup phase, and then uses all of the crs's in KeyGen, Enc and Dec. If at least one of the crs's results in a sub-exponentially secure FE scheme, then the CFSKFE instantiated with these three crs's should also be sub-exponentially secure. Jumping ahead, we remark that for us, the crs's will be the SparseLPN $\mathbf{A}$ matrices and a good crs will be one sampled from GoodSparseMat. Furthermore, setting $B = 3$ will suffice for us. To construct such a CFSKFE, we can simply use Idea 1 above, namely combine FE instances at the SPRG level, to instantiate the scheme $\mathsf{FE}_{i_1,i_2,i_3}$. Since there are only $O(1)$ many FE schemes being combined in this way, the degree of SPRG decompression will now remain $O(1)$.

Once we have such a CFSKFE, it can replace Step 1 in [JMS20], and can then be be bootstrapped to handle $\mathsf{poly}(\lambda)$ many crs's using Step 2 of [JMS20], while preserving sublinearity. Now we only need at least one of $\mathsf{poly}(\lambda)$ many crs's to be sub-exponentially secure, which will hold with all but sub-exponentially small probability.

To make this combiner statement modular and potentially useful to downstream works, in Theorem 7.7 of the full version, we state a more general version

of the FE combiner result given by [JMS20] that now preserves succinctness, as long as the underlying FE candidates have this "combiner-friendliness."

Since the construction of [JMS20] restricts attention to secret-key FE combiners, we ultimately obtain a (sub-exponentially) secure secret-key FE scheme that only supports single function queries. At a high level, our construction (like the construction of [JLS22]) does not support multiple function queries because the SPRG seeds would then be reused across multiple uses of Yao's garbled circuit construction [Yao86]. Finally, we bootstrap this construction to $i\mathcal{O}$ using results of [KNT17, KNT22]. We note that the other bootstrapping results we are aware of would not suffice for our purposes; the results of [BV18, AJ15] consider *public-key* FE, while the result of [BNPW20] considers secret-key FE but requires security with polynomially many function queries. It is plausible that the [JMS20] combiner can be directly made to work for public-key FE, but for simplicity, we use it as is.

We summarize our bootstrapping pipeline using combiners in Fig. 2.

3 Preliminaries

3.1 Notation

For any positive integer n, we let U_n denote the uniform distribution over strings in $\{0,1\}^n$. Let $\mathsf{Binom}(n,p)$ denote the binomial distribution that counts the number of successes in n independent Bernoulli trials with probability p. Wherever we work with polynomials in this paper, we assume they are represented as a list of monomial-coefficient pairs.

Throughout the paper, by a p.p.t. algorithm or adversary, we mean a non-uniform probabilistic polynomial time algorithm.

Definition 1 (Indistinguishability). *We say that two ensembles of distributions $\mathcal{X} = \{\mathcal{X}_\lambda\}_{\lambda\in\mathbb{N}}$ and $\mathcal{Y} = \{\mathcal{Y}_\lambda\}_{\lambda\in\mathbb{N}}$ are* indistinguishable *if for all p.p.t. adversaries $\mathcal{A}$, there exists a negligible function* negl *such that for all sufficiently large $\lambda \in \mathbb{N}$,*

$$\left|\Pr_{x\leftarrow\mathcal{X}_\lambda}[\mathcal{A}(1^\lambda,x)=1] - \Pr_{y\leftarrow\mathcal{Y}_\lambda}[\mathcal{A}(1^\lambda,y)=1]\right| \leq \mathsf{negl}(\lambda).$$

Moreover, we say two ensembles $\mathcal{X} = \{\mathcal{X}_\lambda\}_{\lambda\in\mathbb{N}}$ and $\mathcal{Y} = \{\mathcal{Y}_\lambda\}_{\lambda\in\mathbb{N}}$ are sub-exponentially indistinguishable *if there exists a real number $c > 0$ such that for all p.p.t. adversaries $\mathcal{A}$,*

$$\left|\Pr_{x\leftarrow\mathcal{X}_\lambda}[\mathcal{A}(1^\lambda,x)=1] - \Pr_{y\leftarrow\mathcal{Y}_\lambda}[\mathcal{A}(1^\lambda,y)=1]\right| \leq \exp(-\lambda^c).$$

As short hand, we sometimes use the notation $\approx_c$ to denote indistinguishability, where the parameters are clear from context.

3.2 Locality and Degree

For finite sets A and B and a function $f : A^{n_1} \to B^{n_2}$, we will use the notation

$$f(\mathbf{x}) = (f_1(\mathbf{x}), \cdots, f_{n_2}(\mathbf{x})),$$

where for all $j \in [n_2]$, $f_j : A^{n_1} \to B$. For $\mathbf{x} \in A^{n_1}$ and a subset $S \subseteq [n_1]$, we will let $\mathbf{x}|_S \in A^{|S|}$ denote the restriction of $\mathbf{x}$ to indices in S.

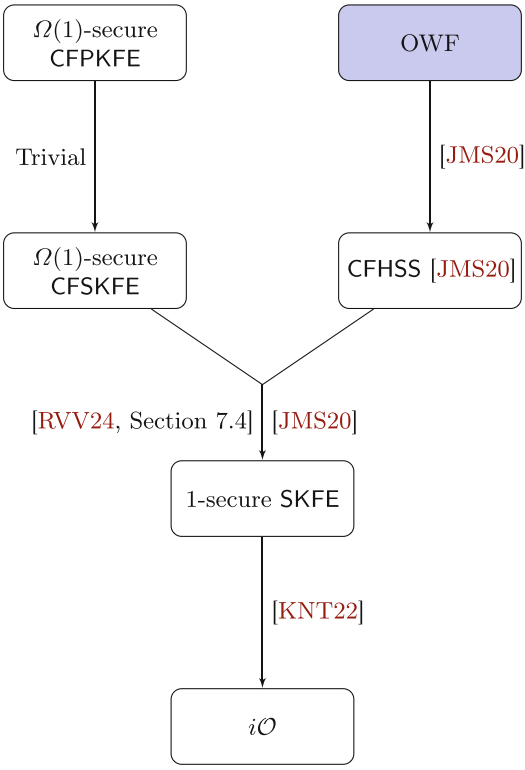


Fig. 2. Flowchart depicting our method for bootstrapping combiner-friendly PKFE (public-key functional encryption) to $i\mathcal{O}$. We rely on a white-box modification of the SKFE (secret-key functional encryption) combiner constructed by [JMS20], to construct a single-key SKFE that is sub-exponentially secure with all but sub-exponentially small probability. This can then be bootstrapped to $i\mathcal{O}$ as shown by [KNT22]. Here, 1-secure (single-key) SKFE is simply (single-key) SKFE; we call it 1-secure just to compare to the other objects. Also, CFHSS refers to a combiner-friendly homomorphic secret sharing, as needed in the unconditional SKFE combiner [JMS20]. Note that everything here needs to be sub-exponentially secure to get to $i\mathcal{O}$. In the negligible but not sub-exponential regime, one can efficiently sample (plausibly) secure $\mathbf{A}$ for SparseLPN efficiently using [AK23], so one can go straight to secure PKFE without any consideration of combiners or combiner-friendliness, as in Fig. 1.

Definition 2 (Locality). *For $n_1, n_2 \in \mathbb{N}$, finite sets A and B, and functions $f : A^{n_1} \to B^{n_2}$, we define the* locality *of f, $\mathsf{loc}(f)$, to be the maximum possible of input variables that any output coordinate depends on. More formally,*

$$\mathsf{loc}(f) := \max_{j \in [n_2]} \min \big\{ k \in \mathbb{N} : \exists S \subseteq [n_1], |S| = k, \\ \exists \widehat{f_j} : A^{|S|} \to B \text{ s.t. } \forall \mathbf{x} \in A^{n_1}, f_j(\mathbf{x}) = \widehat{f_j}(\mathbf{x}|_S) \big\}.$$

By the union bound, we immediately have the following useful lemma.

Lemma 1. *For $n_1, n_2, n_3 \in \mathbb{N}$, finite sets A_1, A_2, A_3, and functions $f : A_1^{n_1} \to A_2^{n_2}$ and $g : A_2^{n_2} \to A_3^{n_3}$, we have*

$$\mathsf{loc}(g \circ f) \leq \mathsf{loc}(g) \cdot \mathsf{loc}(f).$$

Definition 3 (Degree). *For a multivariate polynomial $f : \mathbb{Z}^n \to \mathbb{Z}$, we let $\deg(f)$ denote the total degree of f. For a multi-output multivariate polynomial $f : \mathbb{Z}^{n_1} \to \mathbb{Z}^{n_2}$, we let*

$$\deg(f) := \max_{j \in [n_2]} \deg(f_j).$$

We recall a standard bound on the degree of a composition of polynomials.

Lemma 2. *For $n_1, n_2, n_3 \in \mathbb{N}$ and polynomials $f : \mathbb{Z}^{n_1} \to \mathbb{Z}^{n_2}$ and $g : \mathbb{Z}^{n_2} \to \mathbb{Z}^{n_3}$, we have*

$$\deg(g \circ f) \leq \deg(g) \cdot \deg(f).$$

3.3 LPN and Sparse LPN

Before defining our version of the sparse learning parity with noise (LPN) assumption, we first define the learning parity with noise (LPN) assumption, as used in [JLS22]. Let $\mathsf{Bern}(\mathbb{Z}_q, \eta)$ denote the distribution that is 0 with probability $1 - \eta$ and a uniformly random non-zero element of $\mathbb{Z}_q$ with probability η. For $q = 2$, this corresponds exactly to $\mathsf{Bern}(\eta)$.

Definition 4. *We say that (sub-exponential)* LPN over large fields *is true if the following holds. There exists a constant $\delta \in (0, 1)$ such that for all constants $\beta_1, \beta_2 > 0$ such that $q = q(n)$ is a prime number of n^{β_1} bits and $m = m(n) = n^{\beta_2}$, the following two distributions are (sub-exponentially) indistinguishable:*

$$\big\{ (\mathbf{A}, \mathbf{b} = \mathbf{A}\mathbf{s} + \mathbf{e}) \mid \mathbf{A} \leftarrow \mathbb{Z}_q^{m \times n}, \mathbf{s} \leftarrow \mathbb{Z}_q^n, \mathbf{e} \leftarrow \mathsf{Bern}(\mathbb{Z}_q, n^{-\delta})^m \big\}_{n \in \mathbb{N}},$$
$$\big\{ (\mathbf{A}, \mathbf{u}) \mid \mathbf{A} \leftarrow \mathbb{Z}_q^{m \times n}, \mathbf{u} \leftarrow \mathbb{Z}_q^m \big\}_{n \in \mathbb{N}}.$$

We define LPN only with respect to large fields because that is our only use for it (more specifically, for the construction of Preprocessed Polynomial Encodings (PPE) in [JLS22]).

We now introduce our $\mathsf{SparseLPN}$ assumption. Let $\mathsf{SparseMat}_q(t, n, m)$ denote the set of all matrices $\mathbf{A} \in \mathbb{Z}_q^{m \times n}$ such that every row of $\mathbf{A}$ has exactly t non-zero entries. For sparsity $t = O(1)$, since a uniformly random row-sparse

matrix $\mathbf{A}$ is "bad" (i.e., has small dual distance) with noticeable probability $1/\mathsf{poly}(n)$, we only require the indistinguishability to hold with respect to some distribution of "good" sparse matrices $\mathsf{GoodSparseMat}_q(t, n, m)$ supported on (a subset) of $\mathsf{SparseMat}_q(t, n, m)$, as long as these good matrices are reasonably "dense" among the set of all sparse matrices.[7] Unfortunately, we do not know how to *efficiently* sample from such a $\mathsf{GoodSparseMat}_q(t, n, m)$ that has plausible sub-exponential security, as is needed to bootstrap to $i\mathcal{O}$ [BV18, AJ15, KNT22]. Instead, we will sample $\mathbf{A}$ from $\mathsf{SparseMat}_q(t, n, m)$ in our constructions and then use an unconditional FE combiner [ABJ+19, JMS20] to get true sub-exponential security. We note that this issue also comes up in [JLS22] if instantiating the PRG in NC^0 with Goldreich's PRGs [Gol00] (see [JLS22, Remark 3.1]). More explicitly, we will assume that we can write the uniform distribution over $\mathsf{SparseMat}_q(t, n, m)$ as a mixture of (not necessarily efficiently sampleable) distributions $\mathsf{GoodSparseMat}_q(t, n, m)$ and $\mathsf{BadSparseMat}_q(t, n, m)$, with the weight on $\mathsf{GoodSparseMat}$ being $\Omega(1)$.

In general, our construction does not use any particular property of the uniform distribution over $\mathsf{SparseMat}_q(t, n, m)$, and as such, the distribution over $\mathsf{SparseMat}_q(t, n, m)$ can be replaced with any efficiently sampleable distribution $\mathsf{EffSparseMat}_q(t, n, m)$ over sparse matrices that satisfies the above assumption. For the case of sub-exponential security, we set $\mathsf{EffSparseMat}_q(t, n, m) = \mathsf{SparseMat}_q(t, n, m)$ only for concreteness. Additionally, even though $\mathsf{GoodSparseMat}_q(t, n, m)$ is not explicit, our constructions will be explicit because they use $\mathsf{EffSparseMat}_q(t, n, m)$. (That being said, one can specify a plausible "explicit" distribution for $\mathsf{GoodSparseMat}_q(t, n, m)$, but we do not know how to efficiently sample from it; see Appendix A of the full version.) However, for negligible (but *not* sub-exponential) security in the case of $\mathbb{Z}_2$, then there is an efficiently computable candidate distribution for $\mathsf{GoodSparseMat}_2(t, n, m)$ [AK23], which we denote by $\mathsf{AKSparseMat}_2(t, n, m)$. We define SparseLPN in a generic way using $\mathsf{EffSparseMat}$, to allow us to instantiate the assumption in different ways depending on whether we want negligible or sub-exponential security.

Definition 5. *We say that the (sub-exponential)* SparseLPN *assumption over* $\mathbb{Z}_q$ *is true if the following holds: there exist constants* $t \in \mathbb{N}$, $\delta \in (0, 1)$, *and* $\epsilon > 0$, *an efficiently sampleable distribution* $\mathsf{EffSparseMat}_q(t, n, m)$ *supported on a subset of* $\mathsf{SparseMat}_q(t, n, m)$, *and (not necessarily efficiently sampleable) distributions* $\mathsf{GoodSparseMat}_q(t, n, m), \mathsf{BadSparseMat}_q(t, n, m)$ *such that for* $m = m(n) = n^{1+\epsilon}$ *and* $\eta = \eta(n) = n^{-\delta}$, *the following distributions are*

[7] As a motivating example, for $q = 2$, the probability that two rows of $\mathbf{A}$ are the same is at least $1/\mathsf{poly}(n)$ if the sparsity t is constant. This immediately gives a vector with Hamming weight 2 in the left kernel of $\mathbf{A}$, breaking security.

(sub-exponentially) indistinguishable:

$$\left\{(\mathbf{A}, \mathbf{b} = \mathbf{A}\mathbf{s} + \mathbf{e} \pmod q) \;\middle|\; \begin{array}{c}\mathbf{A} \leftarrow \mathsf{GoodSparseMat}_q(t, n, m), \\ \mathbf{s} \leftarrow \mathbb{Z}_q^n, \mathbf{e} \leftarrow \mathsf{Bern}(\mathbb{Z}_q, \eta)^m\end{array}\right\}_{n\in\mathbb{N}},$$
$$\{(\mathbf{A}, \mathbf{u}) \mid \mathbf{A} \leftarrow \mathsf{GoodSparseMat}_q(t, n, m), \mathbf{u} \leftarrow \mathbb{Z}_q^m\}_{n\in\mathbb{N}},$$

as well as the mixture condition that

$$\mathsf{EffSparseMat}_q(t, n, m) = \mu \cdot \mathsf{GoodSparseMat}_q(t, n, m) + (1 - \mu) \cdot \mathsf{BadSparseMat}_q(t, n, m),$$

for some $\mu = \mu(n)$ where $\mu = \Omega(1)$. Here, this equality refers to equality of distributions as a mixture distribution.

If q is unspecified, we take it to mean $q = 2$, but more generally, we allow $q = q(n)$ to be a function of n.

For simplicity, without loss of generality, we will assume that the noise rate η is the inverse of a power of 2 (with exponent $O(\log n)$). To summarize, we have two distinct instantiations of the SparseLPN assumption depending on the security regime and modulus:

- If we only require negligible security and are restricting attention to $q = 2$, then we could plausibly take $\mu = 1$ and instantiate $\mathsf{EffSparseMat}_2(t, n, m) = \mathsf{AKSparseMat}_2(t, n, m)$.
- If we require sub-exponential security, then we can plausibly take $\mu = \Omega(1)$ and instantiate $\mathsf{EffSparseMat}_q(t, n, m) = \mathsf{SparseMat}_q(t, n, m)$.
  Note that for $t \geq 3$, choosing $\mathsf{GoodSparseMat}_q(t, n, m)$ to be matrices with sufficiently large dual distance, which has density $1 - 1/\mathsf{poly}(n)$ within $\mathsf{SparseMat}_q(t, n, m)$, the best known p.p.t. distinguishing attacks have sub-exponential advantage, as does any linear test. As such, we view setting $\mu = \Omega(1)$ as reasonably mild.

We refer to Appendix A of the full version [RVV24] for more detailed cryptanalysis on the assumption.

3.4 Relaxing Sparse LPN to LFN

It turns out that we can really work with an even more general assumption than the SparseLPN assumption. The SparseLPN assumption informally says that a noisy version of $\mathbf{s} \mapsto \mathbf{A}\mathbf{s} \pmod 2$ is indistinguishable from random. However, neither our construction nor the results by [AIK08] relies on the fact that this map is linear over $\mathbb{Z}_2$; we only need its locality. Motivated by this, we can formulate a more general assumption as follows. Let $\mathsf{LocalFunction}(t, n, m)$ denote the collection of all functions $f : \mathbb{Z}_2^n \to \mathbb{Z}_2^m$ such that each output entry of f depends on at most t inputs.

Definition 6. *We say that the (sub-exponential)* local functions with noise *(*LFN*) assumption is true if the following holds: there exist constants* $t \in \mathbb{N}, \delta \in (0,1)$*, and* $\epsilon > 0$*, an efficiently sampleable distribution* $\mathsf{EffLocalFunction}(t,n,m)$ *supported on a subset of* $\mathsf{LocalFunction}(t,n,m)$*, and (not necessarily efficiently sampleable) distributions* $\mathsf{GoodLocalFunction}(t,n,m), \mathsf{BadLocalFunction}(t,n,m)$ *such that for* $m = m(n) = n^{1+\epsilon}$ *and* $\eta = \eta(n) = n^{-\delta}$*, the following distributions are (sub-exponentially) indistinguishable:*

$$\left\{(f, \mathbf{b} = f(\mathbf{s}) \oplus \mathbf{e}) \;\middle|\; \begin{array}{c} f \leftarrow \mathsf{GoodLocalFunction}(t,n,m), \\ \mathbf{s} \leftarrow \mathbb{Z}_2^n, \mathbf{e} \leftarrow \mathsf{Bern}(\eta)^m \end{array}\right\}_{n\in\mathbb{N}},$$
$$\{(f, \mathbf{u}) \mid f \leftarrow \mathsf{GoodLocalFunction}(t,n,m), \mathbf{u} \leftarrow \mathbb{Z}_2^m\}_{n\in\mathbb{N}},$$

as well as the mixture condition that

$$\begin{aligned}&\mathsf{EffLocalFunction}(t,n,m)\\ &\quad = \mu \cdot \mathsf{GoodLocalFunction}(t,n,m) + (1-\mu) \cdot \mathsf{BadLocalFunction}(t,n,m),\end{aligned}$$

for some $\mu = \mu(n)$ *where* $\mu = \Omega(1)$*. Here, this equality refers to equality of distributions as a mixture distribution.*

More generally, we say the μ-LFN *assumption is true to explicitly set* μ*, in case it is not set to be* $\Omega(1)$*.*

We remark that the above definition abuses notation and uses f to refer to both the function and a description of it; any local function has an efficient description.

Note that this assumption is simultaneously a generalization of $\mathsf{SparseLPN}$ (over $\mathbb{Z}_2$) and the existence of a polynomial-stretch PRG in NC^0: for $\mathsf{SparseLPN}$, the local function f is linear over $\mathbb{F}_2$, and for the poly-stretch PRGs in NC^0, there is no noise (i.e., $\eta = 0$), so there is an immediate reduction to LFN that adds noise. As such, this assumption is *weaker* than both $\mathsf{SparseLPN}$ and the existence of PRGs in NC^0. One other special case of our assumption that might be of interest is instantiating Goldreich's PRG [Gol00] with additional $\mathsf{Bern}(n^{-\delta})$ noise.

Finally, we remark that the LFN assumption can be viewed as a statement about Boolean t-local constraint satisfaction problems (CSPs), where the specific type of CSP can be baked in to the definition of $\mathsf{GoodLocalFunction}$. The assumption is that a random t-local CSP with n inputs and m clauses that is $(1-n^{-\delta})$-satisfiable is indistinguishable from a truly random CSP on n inputs and m clauses (i.e., the m output bits are chosen independently and uniformly at random). In the case of $\mathsf{SparseLPN}$, the analogous CSP problem is t-XOR. (Note that this assumption can only possibly hold if the CSP is balanced in the sense that every clause is 0 or 1 with equal probability for a uniformly random input in $\mathbb{Z}_2^n$; otherwise, examining the Hamming weight of the m output bits would provide a trivial distinguisher.)

4 Structured-Seed PRGs

The construction of $i\mathcal{O}$ by [JLS22] uses the existence of a sub-exponentially secure polynomial-stretch Boolean PRG in NC^0. Our central observation is that a weaker primitive suffices for the purposes of their construction, and that this primitive is achievable under the LFN assumption. As it turns out, this primitive is closely related to the *structured-seed PRGs* (SPRG) defined and constructed by [JLS21]. We provide definitions below in a way that captures our construction, and we provide a comparison with the definition and construction of [JLS21] afterwards.

Definition 7. *A structured seed PRG (*SPRG *for short) consists of the following p.p.t. algorithms:*

- $I \leftarrow \mathsf{IdSamp}(1^{m_{\mathsf{SPRG}}})$*: the generation algorithm takes as input the output length* m_{SPRG} *in unary. It outputs a function index* I*.*
- $\mathsf{seed} \leftarrow \mathsf{SdSamp}(m_{\mathsf{SPRG}})$*: the preprocessing algorithm takes as input the desired output length* m_{SPRG} *in binary and outputs a binary string* seed*. The reason that* m_{SPRG} *is given in binary is that we will require the size of* SdSamp *to be* $m_{\mathsf{SPRG}}^{1-\Omega(1)}$*; note that we abuse notation and do* not *require this algorithm to run in polynomial time in* $\log(m_{\mathsf{SPRG}})$*.*
- $\mathbf{s} \leftarrow \mathsf{Eval}(I, \mathsf{seed})$*: deterministically outputs a string* $\mathbf{s} \in \mathbb{Z}^{m_{\mathsf{SPRG}}}$*.*

Intuitively, seed can serve as a compressed version of the SPRG output such that decompression can be described by a low-degree polynomial. In other words, we would like Eval to be such that computing $\mathsf{Eval}(I, \mathsf{SdSamp}(m_{\mathsf{SPRG}}))$ works just as well as computing $\mathsf{PRG}(U_n)$ for some $n = m_{\mathsf{SPRG}}^{1-\Omega(1)}$, for the purposes of the $i\mathcal{O}$ construction of [JLS22]. We next state the properties capturing this intuition:

Definition 8. *We say a* SPRG *satisfies (perfect) correctness if we have* $\mathsf{Eval}(I, \mathsf{seed}) \in \{0,1\}^{m_{\mathsf{SPRG}}}$ *for all* I, seed *in the support of* $\mathsf{IdSamp}, \mathsf{SdSamp}$ *respectively. (Note that this is not obvious, since the polynomial computing* Eval *is over* $\mathbb{Z}$*.)*

Definition 9. *We say* SPRG *is (sub-exponentially)* $\mu(m_{\mathsf{SPRG}})$*-secure if there exist distributions* $\mathsf{GoodIdSamp}(1^{m_{\mathsf{SPRG}}})$ *and* $\mathsf{BadIdSamp}(1^{m_{\mathsf{SPRG}}})$ *such that both of the following are true:*

- *The following distributions are (sub-exponentially) indistinguishable:*

$$\left\{ (I, \mathsf{Eval}(I, \mathsf{seed})) \;\middle|\; \begin{array}{c} I \leftarrow \mathsf{GoodIdSamp}(1^{m_{\mathsf{SPRG}}}), \\ \mathsf{seed} \leftarrow \mathsf{SdSamp}(m_{\mathsf{SPRG}}) \end{array} \right\}_{m_{\mathsf{SPRG}} \in \mathbb{N}}$$

$$\left\{ (I, \mathbf{s}) \;\middle|\; \begin{array}{c} I \leftarrow \mathsf{GoodIdSamp}(1^{m_{\mathsf{SPRG}}}), \\ \mathbf{s} \leftarrow U_{m_{\mathsf{SPRG}}} \end{array} \right\}_{m_{\mathsf{SPRG}} \in \mathbb{N}}$$

- *We have* $\mu = \Omega(1)$ *and the following two distributions are identical:*
  - *Sample* $I \leftarrow \mathsf{IdSamp}(1^{m_{\mathsf{SPRG}}})$*.*

- *With probability* μ*, sample* $I \leftarrow \mathsf{GoodIdSamp}(1^{m_{\mathsf{SPRG}}})$*. With probability* $1-\mu$*, sample* $I \leftarrow \mathsf{BadIdSamp}(1^{m_{\mathsf{SPRG}}})$.

Definition 10. *We say that an* SPRG *satisfies* ϵ-sublinear efficiency *if* SdSamp *is computable by a uniformly efficiently generatable randomized circuit of size at most* $O(m_{\mathsf{SPRG}}^{1-\epsilon})$.

Definition 11. *We say that a* SPRG *satisfies* degree d and τ-local decompression *if each entry of the output of* $\mathsf{Eval}(I,\cdot)$ *is expressible as a uniformly efficiently generatable polynomial over* $\mathbb{Z}$ *satisfying the following two requirements:*

- *Its total degree (over* $\mathbb{Z}$*) is at most* d *(which we will later require to be* $O(1)$*).*
- *It depends on at most* $O(m_{\mathsf{SPRG}}^{\tau})$ *entries of* seed.

We now point out some of the differences between our definition and the definition used in [JLS21]:

- The most significant difference is that we require SdSamp to satisfy sublinear efficiency. In contrast, [JLS21] is not concerned with the efficiency of SdSamp, but instead requires only that its output seed be small. (At the core, this is because the construction of [JLS21] is only aiming for sublinear *size*-succinct functional encryption; this is then bootstrapped to sublinear *time*-succinct FE assuming LWE [LPST16, GKP+13]. However our construction, like the later construction of [JLS22], does not assume LWE and hence needs to achieve sublinear time-succinctness directly.)
- The construction of [JLS21] allows SdSamp to generate a public and a private seed such that security still holds when the distinguisher is given the public seed. The reason for this is that they require Eval to be degree 2 in the private seed.
  In our case, we have some more freedom because it suffices for Eval to have degree $O(1)$, hence we do not need to work with a public seed.
- On the other hand, the fact that Eval has low locality is important for our use case, whereas locality is not important for [JLS21].
- [JLS21] assumes the polynomial computing Eval is over $\mathbb{Z}_p$ rather than $\mathbb{Z}$, where p is a prime determined by other components of their $i\mathcal{O}$ construction. This is a minor difference; we could also have defined SPRG to work over $\mathbb{Z}_p$ rather than $\mathbb{Z}$, but elected to work with $\mathbb{Z}$ to emphasize that our construction does not depend on the prime p at all.

It should also be noted that our SPRG and its application to $i\mathcal{O}$ is not analogous to that of [JLS21]; it is closer to the application of an NC^0 PRG in [JLS22]. To illustrate this, note firstly that the construction of [JLS21] of SPRG assumes both LPN over $\mathbb{Z}_p$ and the existence of a polynomial-stretch PRG in NC^0. This is the only place in the $i\mathcal{O}$ construction of [JLS21] where LPN over $\mathbb{Z}_p$ is used. In contrast, the construction of [JLS22] and our modification of it will use LPN over $\mathbb{Z}_p$ elsewhere, namely to hide the circuit C and PRG seed with a special-purpose homomorphic encryption scheme which will homomorphically carry out Yao's garbling procedure [Yao86].

Secondly, the notion of SPRG that we need for our purposes is actually *weaker* than a polynomial-stretch PRG in NC^0. Indeed, we can directly construct a simple SPRG given a sub-exponentially secure PRG G in NC^0 from n bits to m_{SPRG} bits, provided $m_{\mathsf{SPRG}} = n^{1+\Omega(1)}$:

- IdSamp deterministically outputs $I = \perp$.
- SdSamp simply outputs a uniform $\mathbf{r} \leftarrow U_n$.
- $\mathsf{Eval}(I, \mathbf{r})$ just directly evaluates $G(\mathbf{r})$.

SdSamp runs in size n which is sublinear in m_{SPRG}, and Eval has locality $O(1)$ which implies that it satisfies low-degree and τ-local decompression with $\tau = 0$. Moreover, this construction is clearly 1-secure.

We now formally state the theorem implied by our construction:

Theorem 4.1. *Assume the (sub-exponential)* LFN *assumption holds for* n *and* $m := m_{\mathsf{SPRG}} = n^{1+\epsilon}$. *Then, there exists constants* $\nu > 0$ *and* $d \in \mathbb{N}$ *such that for any constant* $\tau > 0$, *there exists a perfectly correct (sub-exponentially)* $\Omega(1)$*-secure* SPRG *satisfying* ν*-sublinear efficiency with degree* d *and* τ*-local decompression.*

4.1 SPRG Construction Details

In essence, we will use LFN as our SPRG and we will compress the sparse error vector $\mathbf{e}$ using low-rank decompositions of sparse matrices, as in [JLS21, JLS22]. We begin by setting up some parameters and notation:

- Let $\gamma = \delta/(1+\epsilon)$. Note that the noise rate $\eta = n^{-\delta} = m_{\mathsf{SPRG}}^{-\gamma}$. We also write $\eta = 2^{-b}$ for $b \in \mathbb{N}$ bounded by $O(\log m_{\mathsf{SPRG}})$.
- Set a constant parameter $\alpha \in (\gamma/2, 3\gamma/4)$ such that $2\alpha - \gamma < \tau$.
- Let $L = \lceil m_{\mathsf{SPRG}}^{\alpha} \rceil$ and $\ell = \lceil m_{\mathsf{SPRG}}^{1-2\alpha} \rceil$. Note that $2m_{\mathsf{SPRG}} \geq \ell \cdot L^2 \geq m_{\mathsf{SPRG}}$.
- Let $\beta \in (0, \alpha - \gamma/2)$ be a constant, and let $\rho = m_{\mathsf{SPRG}}^{\beta}$ be a slack parameter. Additionally, let $L' = \lfloor L^2 m_{\mathsf{SPRG}}^{-\gamma} + \rho \cdot L m_{\mathsf{SPRG}}^{-\gamma/2} \rfloor$. Note that $L' = O(m_{\mathsf{SPRG}}^{2\alpha-\gamma} + m_{\mathsf{SPRG}}^{\beta+\alpha-\gamma/2}) = O(m_{\mathsf{SPRG}}^{2\alpha-\gamma}) = O(m_{\mathsf{SPRG}}^{\tau})$.
- Let ϕ be a canonical injective map that maps $\ell \cdot L^2 \geq m_{\mathsf{SPRG}}$ items into ℓ matrices $\{\mathbf{M}_i\}_{i \in [\ell]}$ of size $L \times L$. That is, for $j \in [\ell \cdot L^2]$, $\phi(j) = (j_1, (j_2, j_3))$, where item j is mapped to matrix $j_1 \in [\ell]$ and element $(j_2, j_3) \in [L] \times [L]$ of the matrix. As noted by [JLS22], this map can be computed with a circuit of size $\mathsf{poly}(\log(\ell \cdot L^2)) = \mathsf{poly}(\log m_{\mathsf{SPRG}})$, by first dividing $j \in [\ell \cdot L^2]$ by ℓ and setting the remainder as j_1. Then the quotient can be further divided by L, yielding a quotient and remainder which can be used as (j_2, j_3).
  For convenience, we also use ψ to denote the second half of this injective map, namely the part that maps $j \in [L^2]$ to $(j_2, j_3) \in [L] \times [L]$.

Our construction is described in Fig. 3.

SPRG Construction

$I \leftarrow \mathsf{IdSamp}(1^{m_{\mathsf{SPRG}}})$:

1. Sample $f \leftarrow \mathsf{EffLocalFunction}(t, n, m_{\mathsf{SPRG}})$.
2. Output $I = f$ (by this, we really mean a description of f).

$\mathsf{seed} \leftarrow \mathsf{SdSamp}(m_{\mathsf{SPRG}})$. We describe the procedure in pseudocode below, and defer a discussion of its implementation as a sublinear-time circuit to Section 4.2.

1. Sample errors $\mathbf{e} \leftarrow \mathsf{Bern}(\eta)^{\ell \cdot L^2}$. (We will actually sample from another distribution that is sub-exponentially close in statistical distance to $\mathsf{Bern}(\eta)^{\ell \cdot L^2}$, and we will also ensure that this distribution is supported on $\{0,1\}^{\ell \cdot L^2}$ for correctness.)
2. For every index $j \in [\ell]$, initialize a matrix $\mathbf{M}_j \in \{0,1\}^{L \times L} \subseteq \mathbb{Z}^{L \times L}$ with zero entries.
3. For each $j \in [\ell \cdot L^2]$, compute $\phi(j) = (j_1, (j_2, j_3))$ and set $\mathbf{M}_{j_1}[j_2, j_3] = \mathbf{e}_j$.
4. If there is any $j \in [\ell]$ such that the number of nonzero entries in $\mathbf{M}_j$ is outside the range $[L^2 m_{\mathsf{SPRG}}^{-\gamma} - \rho \cdot L m_{\mathsf{SPRG}}^{-\gamma/2}, L^2 m_{\mathsf{SPRG}}^{-\gamma} + \rho \cdot L m_{\mathsf{SPRG}}^{-\gamma/2}]$, return to Step 1 and resample $\mathbf{e}$. (By the way that we are implementing the sampling of $\mathbf{e}$, we will actually ensure that this condition will never be violated, so $\mathbf{e}$ will never have to be resampled.)
5. For each $j \in [\ell]$, compute matrices $\mathbf{U}_j, \mathbf{V}_j^\top \in \{0,1\}^{L \times L'} \subseteq \mathbb{Z}^{L \times L'}$ such that $\mathbf{M}_j = \mathbf{U}_j \cdot \mathbf{V}_j$ (where this matrix multiplication takes place over $\mathbb{Z}$).
6. Finally, sample $\mathbf{r} \leftarrow \{0,1\}^n$ and output $\mathsf{seed} = (\mathbf{r}, \{\mathbf{U}_j, \mathbf{V}_j\}_{j \in [\ell]})$.

$\mathbf{s} \leftarrow \mathsf{Eval}(I, \mathsf{seed})$:

1. Parse $I = f$ and $\mathsf{seed} = (\mathbf{r}, \{\mathbf{U}_j, \mathbf{V}_j\}_{j \in [\ell]})$.
2. For each $j \in [\ell]$, compute $\mathbf{M}_j = \mathbf{U}_j \cdot \mathbf{V}_j$.
3. Define $\mathbf{e} \in \{0,1\}^{m_{\mathsf{SPRG}}}$ as follows: for each $j \in [m_{\mathsf{SPRG}}]$, compute $\phi(j) = (j_1, (j_2, j_3))$ and set $\mathbf{e}_j = \mathbf{M}_{j_1}[j_2, j_3]$.
4. Output $\mathbf{s} = f(\mathbf{r}) \oplus \mathbf{e}$.

Fig. 3. Our SPRG construction.

Security: The main observation is the following lemma:

Lemma 3. *The probability that the number of nonzero entries in any* $\mathbf{M}_j$ *is outside the range* $[L^2 m_{\mathsf{SPRG}}^{-\gamma} - \rho \cdot L m_{\mathsf{SPRG}}^{-\gamma/2}, L^2 m_{\mathsf{SPRG}}^{-\gamma} + \rho \cdot L m_{\mathsf{SPRG}}^{-\gamma/2}]$ *is* $O(\exp(-m_{\mathsf{SPRG}}^{\Omega(1)}))$.

Proof. It suffices to show the result for a fixed $j \in [\ell]$; the conclusion would then follow from a union bound since $\ell = m_{\mathsf{SPRG}}^{O(1)}$. Following Claim 4.2 in [JLS22], this follows from a straightforward Chernoff bound. For each $j_2, j_3 \in [L]$, let X_{j_2,j_3} be 1 if $\mathbf{M}_j[j_2, j_3] = 1$ and 0 otherwise. Then the X_{j_2,j_3}'s are i.i.d. samples from

$\mathsf{Bern}(m_{\mathsf{SPRG}}^{-\gamma})$. Hence their sum has expectation $L^2 m_{\mathsf{SPRG}}^{-\gamma}$. A Chernoff bound then tells us that:

$$\begin{aligned}
&\Pr\left[\left|\sum_{j_2,j_3\in[L]} X_{j_2,j_3} - L^2 m_{\mathsf{SPRG}}^{-\gamma}\right| \geq \frac{\rho}{L m_{\mathsf{SPRG}}^{-\gamma/2}} \cdot L^2 m_{\mathsf{SPRG}}^{-\gamma}\right] \\
&\leq 2\exp\left(-\left(\frac{\rho}{L m_{\mathsf{SPRG}}^{-\gamma/2}}\right)^2 \cdot \frac{L^2 m_{\mathsf{SPRG}}^{-\gamma}}{3}\right) \\
&= 2\exp\left(-\frac{\rho^2}{3}\right),
\end{aligned}$$

which implies the conclusion since $\rho = m_{\mathsf{SPRG}}^{\beta}$. Note the Chernoff bound applies in this setting because $\frac{\rho}{L m_{\mathsf{SPRG}}^{-\gamma/2}} = O(m_{\mathsf{SPRG}}^{\beta-\alpha+\gamma/2}) \in (0,1)$. □

It follows that that the $\mathbf{e}$ used when constructing the $\mathbf{U}_j, \mathbf{V}_j$ matrices is statistically close to an honest sample from $\mathsf{Bern}(\eta)^{m_{\mathsf{SPRG}}}$ with statistical distance bounded by $O(\exp(-m_{\mathsf{SPRG}}^{\Omega(1)}))$. Security is now immediate from the LFN assumption; we will take $\mathsf{GoodIdSamp}$ to sample $f \leftarrow \mathsf{GoodLocalFunction}(t, n, m_{\mathsf{SPRG}})$ and $\mathsf{BadIdSamp}$ to sample $f \leftarrow \mathsf{BadLocalFunction}(t, n, m_{\mathsf{SPRG}})$.

Low-degree and τ-local decompression: We begin by observing that for any positive integer k, any function $F : \{0,1\}^k \to \{0,1\}$ can be computed by a multilinear polynomial over $\mathbb{Z}$ (which hence has total degree at most k). We now argue step by step:

- The ith entry of $f(\mathbf{r})$ is a function of t entries of $\mathbf{r}$. By our observation, this is expressible as a polynomial over $\mathbb{Z}$ evaluated on $\mathbf{r}$ with locality and total degree at most t.
- For each $j \in [m_{\mathsf{SPRG}}]$, the jth entry of $\mathbf{e}$ is $\mathbf{M}_{j_1}[j_2, j_3]$ where $\phi(j) = (j_1, (j_2, j_3))$. This is in turn equal to $(\mathbf{U}_{j_1} \cdot \mathbf{V}_{j_1})[j_2, j_3] = \sum_{i\in[L']} \mathbf{U}_{j_1}[j_2, i] \cdot \mathbf{V}_{j_1}[i, j_3]$. As a function of all the entries of the matrices $\mathbf{U}_{j_1}$ and $\mathbf{V}_{j_1}$, this has degree 2 and locality $O(L') = O(m_{\mathsf{SPRG}}^{\tau})$.
- Each entry of $f(\mathbf{r}) \oplus \mathbf{e}$ is the XOR of one entry from each of $f(\mathbf{r})$ and $\mathbf{e}$. By the aforementioned observation, this is expressible as a polynomial over $\mathbb{Z}$ evaluated on $f(\mathbf{r})$ and $\mathbf{e}$ with locality and total degree ≤ 2.

Putting these together and using Lemmas 1 and 2 implies that each entry of $\mathsf{Eval}(I, \mathsf{seed})$ is expressible as a polynomial over $\mathbb{Z}$ in the entries of seed with total degree at most $4t = O(1)$ and locality $O(m_{\mathsf{SPRG}}^{\tau})$. Note importantly that our bound $4t$ on the degree comes directly from the parameters in the LFN assumption and is hence independent of τ.

4.2 Sublinear-Time Seed Sampling

It remains to argue the sublinear efficiency of SdSamp. We begin with some lemmas about circuit implementability that will be helpful throughout this section.

Circuit Implementability Lemmas. Our circuit constructions build on two well-known primitives, which we state below. We then sketch the lemmas that we will need, and refer the reader to the full version [RVV24] for proofs. We begin by recalling a result about circuits for sorting [AKS83]:

Lemma 4 ([AKS83], as stated in Lemma 4.4 of [JLS22]). *Suppose we have N strings of size B bits and a comparator circuit $\psi : \{0,1\}^B \times \{0,1\}^B \to \{0,1\}$ of size T_ψ. Then these strings can be sorted with respect to the comparison function computed by ψ with a uniformly efficiently generatable circuit of size $O(N \cdot B \cdot T_\psi \cdot \mathsf{poly}(\log(N \cdot B \cdot T_\psi)))$.*

We also recall the following lemma that will enable us to work with constant-tape Turing machines instead of circuits:

Lemma 5 ([PF79], as stated in Lemma 4.5 of [JLS22]). *For any Turing machine M with $O(1)$ tapes running in time $T(n)$ on inputs of length n, there exists an efficiently generatable Boolean circuit family $\{\mathcal{C}_n\}_{n\in\mathbb{N}}$ where $\mathcal{C}_n$ takes n bits of input, produces the same output, and has $O(T(n) \cdot \mathsf{poly}(\log T(n)))$ gates.*

We now present some simple sampling lemmas that we will use in Sect. 4.2 to implicitly sample the error vector $\mathbf{e}$ in sublinear size.

Lemma 6. *Let N be a positive integer. Let $\mathcal{D}$ be any distribution supported on $\{0,1,\ldots,N-1\}$. Then for any $\epsilon > 0$, there exists a (randomized) circuit of size $O(N^{1+\epsilon})$ that generates a sample from $\mathcal{D}$ with statistical error $O(\exp(-N^{\Omega(1)}))$. (In other words, the distribution of the circuit's output is supported on $\{0,\ldots,N-1\}$ and has statistical distance $O(\exp(-N^{\Omega(1)}))$ from $\mathcal{D}$.)*

Moreover, if $\mathcal{D}$ is the uniform distribution on $\{0,1,\ldots,N-1\}$, there exists such a circuit of size only $O(N^\epsilon)$.

Additionally, if $p_i := \Pr_{X\sim\mathcal{D}}[X = i]$ is uniformly and efficiently computable (represented as a rational number) for all $i \in [0, N-1]$, then this circuit is uniformly and efficiently generatable.

Proof. Provided in the full version [RVV24, Lemma 4.5]. □

Finally, we show that polynomially sparse Hamming slices can be sampled with a sublinear-size circuit:

Lemma 7. *Let $\delta \in (0,1)$ be a constant and N, k be positive integers such that $k \le O(N^{1-\delta})$. Then there exists a (randomized) circuit of size $O(N^{1-\delta/4})$ that takes k as input and outputs a sample from a distribution over subsets of $\{0,1,\ldots,N-1\}$ of size at most k, that has statistical distance $O(\exp(-N^{\Omega(1)}))$ from the uniform distribution over all subsets of size exactly k.*

The set will be output as a list, padded with $\perp$ to some length $N'' = O(N^{1-\delta})$. (We do not require that the list be sorted, or that the order of the elements in the list is distributed in any specific way.)

Proof. Provided in the full version [RVV24, Lemma 4.7]. □

Implementation of SdSamp. We need to implement SdSamp using a circuit with size sublinear in m_{SPRG}, so we cannot afford to sample the Bernoulli error vector directly. Instead, we will first sample its Hamming weight for each of the ℓ buckets, and then sample the set of positions where it is 0.

To this end, let $\mathcal{D}$ be the distribution of $x \leftarrow \mathsf{Binom}(L^2, \eta) = \mathsf{Binom}(L^2, m_{\mathsf{SPRG}}^{-\gamma})$ conditioned on $x \in [L^2 m_{\mathsf{SPRG}}^{-\gamma} - \rho \cdot L m_{\mathsf{SPRG}}^{-\gamma/2}, L^2 m_{\mathsf{SPRG}}^{-\gamma} + \rho \cdot L m_{\mathsf{SPRG}}^{-\gamma/2}]$. By Lemma 3, this is $O(\exp(-m_{\mathsf{SPRG}}^{\Omega(1)}))$-close in statistical distance to $\mathsf{Binom}(L^2, m_{\mathsf{SPRG}}^{-\gamma})$. Moreover, let $\mathcal{D}'$ be the translated distribution $\mathcal{D} - \lceil L^2 m_{\mathsf{SPRG}}^{-\gamma} - \rho \cdot L m_{\mathsf{SPRG}}^{-\gamma/2} \rceil$. This is just for compatibility with Lemma 6.

For each $j \in [\ell]$, our goal is to independently sample matrices $\mathbf{U}_j, \mathbf{V}_j \in \mathbb{Z}^{L \times L'}$ such that the entries of $\mathbf{U}_j \cdot \mathbf{V}_j$ are independent samples from $\mathsf{Bern}(\eta)$ (up to sub-exponential statistical error). We do this as follows:

1. Using Lemma 6, sample $k' \leftarrow \mathcal{D}'$.
2. Compute $k = k' + \lceil L^2 m_{\mathsf{SPRG}}^{-\gamma} - \rho \cdot L m_{\mathsf{SPRG}}^{-\gamma/2} \rceil$. Note by definition of $\mathcal{D}$ that we must have $k \le L' = O(m_{\mathsf{SPRG}}^{2\alpha-\gamma}) = O(L^{2(1-\gamma/2\alpha)})$.
3. Hence using Lemma 7, we can sample a subset $S \subseteq \{0, 1, \ldots, L^2 - 1\}$ of size $c \le k$, padded on the right to length $L' \ge k$ with $\perp$ symbols.
   With all but sub-exponential probability, we will have $c = k$ and S will be a uniform sample from subsets of $\{0, 1, \ldots, L^2 - 1\}$ of size exactly k. Label the elements of S as $(s_1, s_2, \ldots, s_c)$.
4. For each $i \in [c]$, compute $(x_i, y_i) = \psi(s_i)$, so that $x_i, y_i \in [L]$. Let $\mathbf{u}_i \in \mathbb{Z}^L$ be 1 in position x_i and 0 everywhere else, and similarly let $\mathbf{v}_i \in \mathbb{Z}^L$ be 1 in position y_i and 0 everywhere else. Once again, we pad these arrays on the right to length L' with $\perp$ symbols.
5. Define $\mathbf{U}_j = \left[\mathbf{u}_1 \ldots \mathbf{u}_c \, \mathbf{0}^{L \times (L'-c)}\right]$ and $\mathbf{V}_j^\top = \left[\mathbf{v}_1 \ldots \mathbf{v}_c \, \mathbf{0}^{L \times (L'-c)}\right]$.

Once we have done this for each $j \in [\ell]$, we can just sample $\mathbf{r} \leftarrow U_n$ directly and output $(\mathbf{r}, \{\mathbf{U}_j, \mathbf{V}_j\}_{j \in [\ell]})$.

Correctness: First, we address the statistical errors incurred from calls to Lemmas 6 and 7. In the calls to Lemma 6, we are taking $N = L' - (L^2 m_{\mathsf{SPRG}}^{-\gamma} - \rho \cdot L m_{\mathsf{SPRG}}^{-\gamma/2}) = \Theta(\rho \cdot L m_{\mathsf{SPRG}}^{-\gamma/2}) = \Theta(m_{\mathsf{SPRG}}^{\beta+\alpha-\gamma/2}) = \Theta(m_{\mathsf{SPRG}}^{\Omega(1)})$. Hence the statistical error is $O(\exp(-m_{\mathsf{SPRG}}^{\Omega(1)}))$. In the calls to Lemma 7, we are taking $N = L^2 = \Theta(m_{\mathsf{SPRG}}^{2\alpha})$, so here also we incur statistical error is $O(\exp(-m_{\mathsf{SPRG}}^{\Omega(1)}))$. We make at most $\ell < m_{\mathsf{SPRG}}$ calls to each of these lemmas, so the total statistical error will remain sub-exponentially small.

Up to the statistical error mentioned above, we have sampled the Hamming weight $k \leftarrow \mathsf{Binom}(L^2, m_{\mathsf{SPRG}}^{-\gamma})$ and then sampled a uniformly random subset of $\{0, 1, \ldots, L^2 - 1\}$ of size k. This is clearly equivalent to including each element of $\{0, 1, \ldots, L^2 - 1\}$ in S independently with probability η each.

Finally, we argue that $\mathbf{U}_j \cdot \mathbf{V}_j$ is 1 in row x and column y if $\psi^{-1}(x, y) \in S$ and 0 otherwise. This will complete our argument. Indeed, we have $\mathbf{U}_j \cdot \mathbf{V}_j = \sum_{i=1}^{k} \mathbf{u}_i \mathbf{v}_i^\top$. The matrix $\mathbf{u}_i \mathbf{v}_i^\top$ is 1 in row x_i and column y_i and 0 everywhere else. Since $s_1, \ldots, s_c$ are distinct and hence the tuples (x_i, y_i) are distinct, our claim follows.

Even in the case where we encounter some statistical error when calling Lemmas 6 and 7, we will always sample a collection $(s_1, \ldots, s_c)$ of $c \leq L'$ distinct elements of $\{0, 1, \ldots, L^2 - 1\}$, for some c. It then follows that all entries of $\mathbf{U}_j \cdot \mathbf{V}_j$ will still be 0 or 1, which ensures perfect correctness of our SPRG construction.

Uniform Generatability: The only thing that needs to be addressed is the call to Lemma 6, for which we need to check that the histogram of $\mathcal{D}'$ can be efficiently computed. To do this, it clearly suffices to efficiently compute the histogram of $\mathsf{Binom}(L^2, m_{\mathsf{SPRG}}^{-\gamma})$; we can then apply the necessary conditioning to recover a histogram for $\mathcal{D}$, and finally translate it to obtain a histogram for $\mathcal{D}'$. This histogram is characterized by the following identity:

$$\begin{aligned}
\Pr\left[X = i \,|X \leftarrow \mathsf{Binom}(L^2, m_{\mathsf{SPRG}}^{-\gamma})\right] &= \binom{L^2}{i} \cdot \left(m_{\mathsf{SPRG}}^{-\gamma}\right)^i \cdot \left(1 - m_{\mathsf{SPRG}}^{-\gamma}\right)^{L^2-i} \\
&= \binom{L^2}{i} \cdot \left(2^{-b}\right)^i \cdot \left(1 - 2^{-b}\right)^{L^2-i} \\
&= \binom{L^2}{i} \cdot 2^{-bL^2} \cdot \left(2^b - 1\right)^{L^2-i}.
\end{aligned}$$

It is now clear that this is computable in time $\mathsf{poly}(m_{\mathsf{SPRG}})$; the binomial coefficients $\binom{L^2}{i}$ can all be computed in time $\mathsf{poly}(L) = \mathsf{poly}(m_{\mathsf{SPRG}})$ e.g., by iterating through Pascal's triangle. Since $b \leq O(\log m_{\mathsf{SPRG}})$, we have $2^b - 1 \leq \mathsf{poly}(m_{\mathsf{SPRG}})$. The value $\left(2^b - 1\right)^{L^2-i}$ can hence be computed exactly, e.g., via repeated squaring; note importantly that the bit length of this number will always be at most $L^2 \cdot O(\log m_{\mathsf{SPRG}}) = \mathsf{poly}(m_{\mathsf{SPRG}})$. Finally, the factor of 2^{-bL^2} can be easily handled by bit shifting, noting again that $bL^2 = \mathsf{poly}(m_{\mathsf{SPRG}})$.

Sublinear Efficiency: We first argue that for any fixed $j \in [\ell]$, $\mathbf{U}_j$ and $\mathbf{V}_j$ can be sampled with a randomized circuit of size $O(m_{\mathsf{SPRG}}^{2\alpha - \Omega(1)})$:

- The initial call to Lemma 6 sets $N = \Theta(m_{\mathsf{SPRG}}^{\beta+\alpha-\gamma/2}) = O(m_{\mathsf{SPRG}}^{2\alpha-\gamma})$. It follows by Lemma 6 that this can be done with a circuit of size $O(m_{\mathsf{SPRG}}^{2\alpha-\gamma/2})$.
- Calculating k from k' is a straightforward addition of integers that are at most L^2, so this is doable with a circuit of size $\mathsf{poly}(\log m_{\mathsf{SPRG}})$.
- The call to Lemma 7 takes $N = L^2 = O(m_{\mathsf{SPRG}}^{2\alpha})$, so it follows that this can be done with a circuit of size $O(L^{2(1-\gamma/(8\alpha))}) = O(m_{\mathsf{SPRG}}^{2\alpha-\gamma/4})$.
- Computing c and then $(x_i, y_i) = \psi(s_i)$ for all $i \in [c]$ is doable with a circuit of size $L' \cdot \mathsf{poly}(\log m_{\mathsf{SPRG}}) = O(m_{\mathsf{SPRG}}^{2\alpha-\gamma/2})$.
- Given an integer $j \in [L]$ as input, there exists a circuit of size $O(L \cdot \mathsf{poly}(\log L))$ to compute a vector in $\mathbb{Z}^L$ that is 1 in position j and 0 everywhere else. Indeed, this can be done on a two-tape Turing machine in $O(L \cdot \mathsf{poly}(\log L))$ time: one tape will store j and the second tape will store the output. The head on the second tape can pass over the output while decrementing the value on the first tape after each step, and write a 1 on the second tape if the first tape's value is exactly 0 and write a 0 otherwise. Our claim now follows from Lemma 5.

It follows that assembling the matrices $\mathbf{U}_j$ and $\mathbf{V}_j$ is doable with a circuit of total size $O(L' \cdot L \cdot \mathsf{poly}(\log L)) = O(m_{\mathsf{SPRG}}^{3\alpha-\gamma} \cdot \mathsf{poly}(\log m_{\mathsf{SPRG}})) \leq O(m_{\mathsf{SPRG}}^{2\alpha-\gamma/5})$.

Hence the total circuit size for sampling $\mathbf{U}_j, \mathbf{V}_j$ for all $j \in [\ell]$ is $O(\ell \cdot m_{\mathsf{SPRG}}^{2\alpha-\gamma/5}) = O(m_{\mathsf{SPRG}}^{1-\gamma/5})$. Finally, sampling $\mathbf{r} \leftarrow U_n$ and adding that to the output is doable with a circuit of size $O(n) = O(m_{\mathsf{SPRG}}^{1/(1+\epsilon)}) = O(m_{\mathsf{SPRG}}^{1-\Omega(1)})$. Note importantly that the size of our circuit is determined by $\gamma = \delta/(1+\epsilon)$ and ϵ, which come directly from the parameters in the LFN assumption and are hence independent of τ. This completes our proof that SdSamp is implementable with a sublinear-time circuit, and hence Theorem 4.1.

Acknowledgements. We thank Aayush Jain, Rachel Lin, and an anonymous TCC reviewer for suggesting the generalization from sparse LPN to LFN. We additionally thank Rachel Lin for answering our questions about the constructions by [JLS21, JLS22]. The first author was supported by an Akamai Presidential Fellowship. The second author's research was supported by NSF fellowship DGE-2141064 and by the grants of the third author. The third author's research was supported in part by DARPA under Agreement No. HR00112020023, NSF CNS-2154149, a grant from the MIT-IBM Watson AI, a grant from Analog Devices, a Microsoft Trustworthy AI grant, a Thornton Family Faculty Research Innovation Fellowship from MIT and a Simons Investigator Award. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the United States Government or DARPA.

References

[ABJ+19] Ananth, P., Badrinarayanan, S., Jain, A., Manohar, N., Sahai, A.: From FE combiners to secure MPC and back. In: Hofheinz, D., Rosen, A. (eds.) TCC 2019. LNCS, vol. 11891, pp. 199–228. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-36030-6_9

[ABSV15] Ananth, P., Brakerski, Z., Segev, G., Vaikuntanathan, V.: From selective to adaptive security in functional encryption. In: Gennaro, R., Robshaw, M. (eds.) CRYPTO 2015. LNCS, vol. 9216, pp. 657–677. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48000-7_32

[ABW10] Applebaum, B., Barak, B., Wigderson, A.: Public-key cryptography from different assumptions. In: Schulman, L.J. (edr.) Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010, Cambridge, Massachusetts, USA, 5-8 June 2010, pp. 171–180. ACM (2010)

[ADI+17] Applebaum, B., Damgård, I., Ishai, Y., Nielsen, M., Zichron, L.: Secure arithmetic computation with constant computational overhead. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017. LNCS, vol. 10401, pp. 223–254. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63688-7_8

[AIK08] Applebaum, B., Ishai, Y., Kushilevitz, E.: On pseudorandom generators with linear stretch in NC^0. Comput. Complex. **17**(1), 38–69 (2008)

[AJ15] Ananth, P., Jain, A.: Indistinguishability Obfuscation from compact functional encryption. In: Gennaro, R., Robshaw, M. (eds.) CRYPTO 2015. LNCS, vol. 9215, pp. 308–326. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-47989-6_15

[AK23] Applebaum, B., Kachlon, E.: Sampling graphs without forbidden subgraphs and unbalanced expanders with negligible error. SIAM J. Comput. **52**(6), 1321–1368 (2023)

[AKS83] Ajtai, M., Komlós, J., Szemerédi, E.: An o(n log n) sorting network. In: Johnson, D.S., et al. (eds.) Proceedings of the 15th Annual ACM Symposium on Theory of Computing, 25-27 April 1983, Boston, Massachusetts, USA, pp. 1–9. ACM (1983)

[AKV04] Abbot, T., Kane, D., Valiant, P.: On algorithms for Nash equilibria. Unpublished manuscript, pp. 1 (2004)

[AL18] Applebaum, B., Lovett, S.: Algebraic attacks against random local functions and their countermeasures. SIAM J. Comput. **47**(1), 52–79 (2018)

[Ale11] Alekhnovich, M.: More on average case vs approximation complexity. Comput. Complex. **20**(4), 755–786 (2011)

[AOW15] Allen, S.R., O'Donnell, R., Witmer, D.: How to refute a random CSP. In: Guruswami, V. (ed.) IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October 2015, pp. 689–708. IEEE Computer Society (2015)

[AS16] Ananth, P., Sahai, A.: Functional encryption for turing machines. In: Kushilevitz, E., Malkin, T. (eds.) TCC 2016. LNCS, vol. 9562, pp. 125–153. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49096-9_6

[BCGI18] Boyle, E., Couteau, G., Gilboa, N., Ishai, Y.: Compressing vector OLE. In: Lie, D., Mannan, M., Backes, M., Wang, X. (eds.) Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, 15-19 October 2018, pp. 896–912. ACM (2018)

[BGI+12] Barak, B., et al.: On the (Im)possibility of obfuscating programs. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 1–18. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-44647-8_1

[BNPW20] Bitansky, N., Nishimaki, R., Passelegue, A., Wichs, D.: From Cryptomania to Obfustopia through secret-key functional encryption. J. Cryptol. **33**(2), 357–405 (2020)

[BPR15] Bitansky, N., Paneth, O., Rosen, A.: On the cryptographic hardness of finding a Nash equilibrium. In: 2015 IEEE 56th Annual Symposium on Foundations of Computer Science, pp. 1480–1498. IEEE (2015)

[BV18] Bitansky, N., Vaikuntanathan, V.: Indistinguishability obfuscation from functional encryption. J. ACM **65**(6), 39:1–39:37 (2018)

[BZ14] Boneh, D., Zhandry, M.: Multiparty key exchange, efficient traitor tracing, and more from indistinguishability obfuscation. In: Garay, J.A., Gennaro, R. (eds.) CRYPTO 2014. LNCS, vol. 8616, pp. 480–499. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-44371-2_27

[CLP15] Chung, K.-M., Lin, H., Pass, R.: Constant-round concurrent zero-knowledge from indistinguishability obfuscation. In: Gennaro, R., Robshaw, M. (eds.) CRYPTO 2015. LNCS, vol. 9215, pp. 287–307. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-47989-6_14

[CLTV15] Canetti, R., Lin, H., Tessaro, S., Vaikuntanathan, V.: Obfuscation of probabilistic circuits and applications. In: Dodis, Y., Nielsen, J.B. (eds.) TCC 2015. LNCS, vol. 9015, pp. 468–497. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46497-7_19

[CM01] Cryan, M., Miltersen, P.B.: On pseudorandom generators in NC. In: Sgall, J., Pultr, A., Kolman, P. (eds.) Mathematical Foundations of Computer Science 2001, 26th International Symposium, MFCS 2001 Marianske Lazne, Czech Republic, 27-31 August 2001, Proceedings, LNCS, vol. 2136, pp. 272–284. Springer, Heidelberg (2001)

[CPP20] Canetti, R., Park, S., Poburinnaya, O.: Fully deniable interactive encryption. In: Micciancio, D., Ristenpart, T. (eds.) CRYPTO 2020. LNCS, vol. 12170, pp. 807–835. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-56784-2_27

[DIJL23] Dao, Q., Ishai, Y., Jain, A., Lin, H.: Multi-party homomorphic secret sharing and sublinear MPC from sparse LPN. In: Handschuh, H., Lysyanskaya, A. (eds.) Advances in Cryptology - CRYPTO 2023 - 43rd Annual International Cryptology Conference, CRYPTO 2023, Santa Barbara, CA, USA, 20-24 August 2023, Proceedings, Part II, LNCS, vol. 14082, pp. 315–348. Springer, Switzerland (2023). https://doi.org/10.1007/978-3-031-38545-2_11

[DJ24] Dao, Q., Jain, A.: Lossy cryptography from code-based assumptions. IACR Cryptol. ePrint Arch., pp. 175 (2024)

[Fei02] Feige, U.: Relations between average case complexity and approximation complexity. In: Reif, J.H. (ed.) Proceedings on 34th Annual ACM Symposium on Theory of Computing, 19-21 May 2002, Montréal, Québec, Canada, pages 534–543. ACM (2002)

[GGH+13] Garg, S., Gentry, C., Halevi, S., Raykova, M., Sahai, A., Waters, B.: Candidate indistinguishability obfuscation and functional encryption for all circuits. In: 54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA, pp. 40–49. IEEE Computer Society (2013)

[GIS18] Garg, S., Ishai, Y., Srinivasan, A.: Two-round MPC: information-theoretic and black-box. In: Beimel, A., Dziembowski, S. (eds.) TCC 2018. LNCS, vol. 11239, pp. 123–151. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-03807-6_5

[GJLS21] Gay, R., Jain, A., Lin, H., Sahai, A.: Indistinguishability obfuscation from simple-to-state hard problems: new assumptions, new techniques, and simplification. In: Canteaut, A., Standaert, F.-X. (eds.) EUROCRYPT 2021. LNCS, vol. 12698, pp. 97–126. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-77883-5_4

[GKP+13] Goldwasser, S., Kalai, Y., Popa, R.A., Vaikuntanathan, V. and Zeldovich, N.: Reusable garbled circuits and succinct functional encryption. In: Boneh, D., Roughgarden, T., Feigenbaum, J., (eds.) Symposium on Theory of Computing Conference, STOC 2013, Palo Alto, CA, USA, 1-4 June 2013, pp. 555–564. ACM (2013)

[GMW87] Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game or a completeness theorem for protocols with honest majority. In: Aho, A.V. (ed.) Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 1987, New York, New York, USA, pp. 218–229. ACM (1987)

[Gol00] Goldreich, O.: Candidate one-way functions based on expander graphs. IACR Cryptol. ePrint Arch., p. 63 (2000)

[GPS16] Garg, S., Pandey, O., Srinivasan, A.: Revisiting the cryptographic hardness of finding a Nash equilibrium. In: Robshaw, M., Katz, J. (eds.) CRYPTO 2016. LNCS, vol. 9815, pp. 579–604. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53008-5_20

[GS16] Garg, S., Srinivasan, A.: Single-key to multi-key functional encryption with polynomial loss. In: Hirt, M., Smith, A. (eds.) TCC 2016. LNCS, vol. 9986, pp. 419–442. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53644-5_16

[GS22] Garg, S., Srinivasan, A.: Two-round multiparty secure computation from minimal assumptions. J. ACM, **69**(5), 36:1–36:30 (2022)

[GVW15] Gorbunov, S., Vaikuntanathan, V., Wee, H.: Predicate encryption for circuits from LWE. In: Gennaro, R., Robshaw, M. (eds.) CRYPTO 2015. LNCS, vol. 9216, pp. 503–523. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48000-7_25

[HY20] Hubácek, P., Yogev, E.: Hardness of continuous local search: query complexity and cryptographic lower bounds. SIAM J. Comput. **49**(6), 1128–1172 (2020)

[IKOS08] Ishai, Y., Kushilevitz, E., Ostrovsky, R., Sahai, A.: Cryptography with constant computational overhead. In: Dwork, C. (ed.) Proceedings of the 40th Annual ACM Symposium on Theory of Computing, Victoria, British Columbia, Canada, 17-20 May 2008, pp. 433–442. ACM (2008)

[JLS19] Jain, A., Lin, H., Sahai, A.: Simplifying constructions and assumptions for IO. IACR Cryptol. ePrint Arch., p. 1252 (2019)

[JLS21] Jain, A., Lin, H., Sahai, A.: Indistinguishability obfuscation from well-founded assumptions. In: Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2021, pp. 60–73, New York, NY, USA (2021). Association for Computing Machinery

[JLS22] Jain, A., Lin, H.,D Sahai, A.: Indistinguishability obfuscation from LPN over $\mathbb{F}\_p$, dlin, and prgs in nc^0. In: Dunkelman, O., Dziembowski, S. (eds.) Advances in Cryptology - EUROCRYPT 2022 - 41st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Trondheim, Norway, May 30 - June 3, 2022, Proceedings, Part I, LNCS, vol. 13275, pp. 670–699. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-06944-4_23

[JMS20] Jain, A., Manohar, N., Sahai, A.: Combiners for functional encryption, unconditionally. In: Canteaut, A., Ishai, Y. (eds.) EUROCRYPT 2020. LNCS, vol. 12105, pp. 141–168. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-45721-1_6

[KMOW17] Kothari, P.K., Mori, R., O'Donnell, R., Witmer, D.: Sum of squares lower bounds for refuting any CSP. In: Hatami, H., McKenzie, P., King, V. (edis.) Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, 19-23 June 2017, pp. 132–145. ACM (2017)

[KNT17] Kitagawa, F., Nishimaki, R., Tanaka, K.: Indistinguishability obfuscation for all circuits from secret-key functional encryption. IACR Cryptol. ePrint Arch., p. 361 (2017)

[KNT22] Kitagawa, F., Nishimaki, R., Tanaka, K.: Obfustopia built on secret-key functional encryption. J. Cryptol. **35**(3), 19 (2022)

[KNTY19] Kitagawa, F., Nishimaki, R., Tanaka, K., Yamakawa, T.: Adaptively secure and succinct functional encryption: improving security and efficiency, simultaneously. In: Boldyreva, A., Micciancio, D. (eds.) CRYPTO 2019. LNCS, vol. 11694, pp. 521–551. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-26954-8_17

[KNY17] Komargodski, I., Naor, M., Yogev, E.: Secret-sharing for NP. J. Cryptol. **30**(2), 444–469 (2017)

[KRS15] Khurana, D., Rao, V., Sahai, A.: Multi-party key exchange for unbounded parties from indistinguishability obfuscation. In: Iwata, T., Cheon, J.H. (eds.) ASIACRYPT 2015. LNCS, vol. 9452, pp. 52–75. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48797-6_3

[KS20] Komargodski, I., Segev, G.: From Minicrypt to Obfustopia via private-key functional encryption. In: Coron, J.-S., Nielsen, J.B. (eds.) EUROCRYPT 2017. LNCS, vol. 10210, pp. 122–151. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-56620-7_5

[LM16] Li, B., Micciancio, D.: Compactness vs collusion resistance in functional encryption. In: Hirt, M., Smith, A. (eds.) TCC 2016. LNCS, vol. 9986, pp. 443–468. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53644-5_17

[LPST16] Lin, H., Pass, R., Seth, K., Telang, S.: Output-compressing randomized encodings and applications. In: Kushilevitz, E., Malkin, T. (eds.) TCC 2016. LNCS, vol. 9562, pp. 96–124. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49096-9_5

[MST06] Mossel, E., Shpilka, A., Trevisan, L.: On epsilon-biased generators in nc^0. Random Struct. Algorithms **29**(1), 56–81 (2006)

[O'D14] O'Donnell, R.: Analysis of Boolean Functions. Cambridge University Press (2014)

[PF79] Pippenger, N., Fischer, M.J.: Relations among complexity measures. J. ACM **26**(2), 361–381 (1979)

[RVV24] Ragavan, S., Vafa, N., Vaikuntanathan, V.: Indistinguishability obfuscation from bilinear maps and LPN variants. IACR Cryptol. ePrint Arch., pp. 856 (2024)

[SW14] Sahai, A., Waters, B.: How to use indistinguishability obfuscation: deniable encryption, and more. In: Shmoys, D.B. (ed) Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014, pp. 475–484. ACM (2014)

[Wee20] Wee, H.: Functional encryption for quadratic functions from k-lin, revisited. In: Pass, R., Pietrzak, K. (eds.) TCC 2020. LNCS, vol. 12550, pp. 210–228. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-64375-1_8

[WW24a] Waters, B., Wu, D.J.: Adaptively-sound succinct arguments for NP from indistinguishability obfuscation. IACR Cryptol. ePrint Arch., pp. 165 (2024)

[WW24b] Waters, B., Wu, D.J.: A pure indistinguishability obfuscation approach to adaptively-sound SNARGs for NP. IACR Cryptol. ePrint Arch., pp. 933 (2024)

[Yao86] Yao, A.C.: How to generate and exchange secrets (extended abstract). In: 27th Annual Symposium on Foundations of Computer Science, Toronto, Canada, 27-29 October 1986, pp. 162–167. IEEE Computer Society (1986)

Towards General-Purpose Program Obfuscation via Local Mixing

Ran Canetti[1,2(✉)], Claudio Chamon[1,2], Eduardo R. Mucciolo[1,2], and Andrei E. Ruckenstein[1,2]

[1] Boston University, Boston, USA
canetti@bu.edu
[2] University of Central Florida, Orlando, USA

Abstract. We explore the possibility of obtaining general-purpose obfuscation for all circuits by way of making only simple, local, functionality preserving random perturbations in the circuit structure. Towards this goal, we use the additional structure provided by reversible circuits, but no additional algebraic structure. Our approach is rooted in statistical mechanics and can be thought of as locally "thermalizing" a circuit while preserving its functionality.

We analyze the security of this approach in two steps. First, we provide arguments towards its security for a relatively simple task: obfuscating random circuits of bounded length. Next we show how to construct indistinguishability obfuscators for all (unbounded length) circuits given an obfuscator for random reversible circuits of bounded length. Here security is proven under a new assumption regarding the pseudorandomness of sufficiently-long random reversible circuits.

Our specific candidate obfuscators are very simple and relatively efficient: the obfuscated version of an n-wire, m-gate (reversible) circuit with security parameter κ has n wires and $O(\kappa m)$ gates. We hope that our initial exploration will motivate further study of this alternative path to program obfuscation (and, consequently, to cryptography in general).

1 Introduction

Program obfuscation [Had00, BGI+01, BGI+12], namely the ability to efficiently purturb a program in a way that preserves its functionality but hides "all other information" about the program, is an intriguing beast. At first, perturbing - or randomizing - the internal structure of a program may appear to be rather mundane and inconsequential. However, with the right formalization of "sufficiently perturbed", program obfuscation has proven to be immensely powerful.

As shown in [BGI+01, BGI+12], in general *any* polysize representation of a program, even a "perfectly randomized" one, gives significantly more com-

This work is supported by NSF grants 2428487 and 2428488, DARPA grants HR00112020021 and HR00112020023 (R.C.), DOE Grant DE-FG02-06ER46316 (C.C.), and a Grant from the Mass Tech Collaborative Innovation Institute (A.E.R.). R.C., C.C., and A.E.R. also acknowledge the Quantum Convergence Focused Research Program, funded by the Rafik B. Hariri Institute at Boston University.

E. Boyle and M. Mahmoody (Eds.): TCC 2024, LNCS 15367, pp. 37–70, 2025.
https://doi.org/10.1007/978-3-031-78023-3_2

putational power than black-box access to the function computed by the program. However, the more modest goal of perturbing the program just to the point of making the perturbed versions of any two equal-length, functionally equivalent programs indistinguishable is potentially obtainable and has proven to be immensely powerful. Indeed, while the ability to obfuscate general programs to that level (namely obtaining *Indistinguishability Obfuscation (IO)* [BGI+01, BGI+12]) does not imply any computational hardness in and of itself (Indeed, if P=NP then IO exists), IO for all circuits combined with the mere assumption that P≠NP implies public key encryption, trapdoor permutations, general secure multiparty computation, non-interactive zero knowledge, succinct non-interactive arguments, and deniable encryption to name only very few, see e.g. [SW14, GGHR14, BPW16]. When combined with lossy one way functions, it gives also fully homomorphic encryption, collision resistant hashing, and more [CLTV15].

The history of attempts at constructing general purpose program obfuscators, starting from the breakthrough works of [SW14, GGSW13], is intriguing as well.

In the "first generation" constructions such as [GGSW13, BGK+14, AB15] [GGH15] the obfuscated program typically follows the instruction structure of the the plaintext program without modification, while using algebraic structures to perform the instructions "homomorphically" while hiding them from an adversary who runs the program and sees the entire execution trace. However, the analyses of these first generation constructions was invariably incomplete, often by way of relying on an idealized version of a core primitive, and indeed explicit attacks have been demonstrated against many proposed instantiations of these candidates (e.g., [CGH+15, CVW18, CHVW19]).

The "second generation" constructions (starting from [BV15, AJS15, LPST16]) take a different approach: Rather than directly follow the steps of the input program, the obfuscated program is treated as a "compressed store" of "garbled programs", namely, obfuscated programs that are valid only for a single input. Given an input, the overall obfuscated program first gradually "uncompresses" the garbled program for that input, and then runs this garbled program to obtain the desired output. A number of more recent IO candidate constructions, including the breakthrough works of Jain, Lin and Sahai [JLS21] that provide the first IO schemes whose security is proven based on relatively well understood assumptions, as well as [GP21, WW21, DQV+21, KNT22, RVV24] and others, use that structure.

This two-stage structure is, however, a bit roundabout and results in prohibitively high space and time overhead relative to the complexity of the plaintext program, rendering general program obfuscation a purely theoretical primitive.

1.1 This Work

We explore a new approach to constructing general-purpose program obfuscation. The idea is very simple: Repeatedly perform random local perturbations of the given program, while guaranteeing that each perturbation preserves the overall functionality of the program. The overarching hope is that, while individual

perturbations can be easily undone, the aggregate effect of the perturbation process will be that of converging to a distribution over programs that hides (or even completely destroys) the structure of the original program—all while preserving functionality.

Of course, significantly more detail and structure are needed in order to turn this very high-level idea ideal into a concrete proposal. Here one must also keep in mind the long list of failed attempts at using such techniques to provide "white box security" for programs that are accessible to an adversary (see e.g. [Wik24]).

The structure we employ is that of reversible computation, where the number of state variables remains fixed throughout the computation, and each individual computational step can be reversed (namely, undone) in a single step. Specifically, we concentrate on reversible circuits, where state variables correspond to wires, and a computational step corresponds to a gate that applies a permutation to the current state.

It is stressed that, while reversible circuits are often studied because of their physical properties (say for energy efficiency or quantum computation [Ben73, Tof80]), here the motivation to consider reversible circuits is purely cryptographic. Specifically, we use the algebraic structure provided by the fact that reversible circuits consist of sequences of permutations to argue that appropriately chosen local perturbations of the circuit structure are likely to have a global effect that is hard to reverse and is likely to make the obfuscated versions of any two same-size, functionally equivalent programs indistinguishable. More specifically, our construction and analysis proceeds in two main steps:

- The first step describes a candidate scheme (or, rather, a meta-scheme) for obfuscating *bounded-length random circuits.* These are n-wire circuits that consist of m gates (where m is some fixed polynomial in the security parameter) and each gate is chosen independently at random from a fixed set of gates. While we only provide informal arguments for the security of this scheme, we do rigorously formulate a notion of security, Random Input and Output (RIO) obfuscation, that we conjecture to be satisfied by our scheme.
- The second step constructs an IO scheme for all circuits (not necessarily reversible), given any RIO obfuscator. We prove security of this construction under a new intractability assumption on the distribution of random reversible circuits.

We start the exposition of our results with a brief overview of reversible circuits, followed by an exposition of our intractability assumptions regarding the same. Next we review our definition of RIO obfuscation and how we use it to construct an IO scheme for all circuits. Finally, we sketch our candidate RIO obfuscator and the arguments for its security.

1.2 Reversible Circuits and Their Pseudorandomness Properties

Reversible circuits. Recall that reversible circuits have a fixed number, n, of wires (or, binary state variables), and each gate γ computes a permutation on the n-bit state. The permutation $\mathcal{P}_C$ computed by $C = \gamma_1 \dots \gamma_m$ is the composition of

the individual permutations, $\mathcal{P}_C = \mathcal{P}_{\gamma_m} \circ \ldots \circ \mathcal{P}_{\gamma_1}$, or, in other words, $C(x) = \gamma_m(\ldots\gamma_1(x)\ldots)$. We restrict our attention to Toffoli gates, namely gates of the form $\gamma_{i,j,k,f}(s_1 \ldots s_n) = (s'_1 \ldots s'_n)$ where $s_1 \ldots s_n$ is the old state, $s'_1 \ldots s'_n$ is the new state, i, j, k are distinct indices in $[n]$, $f : \{0,1\}^2 \to \{0,1\}$, $s'_i = s_i + f(s_j, s_k)$, and $s'_{i'} = s_{i'}$ for all $i' \neq i$ [Tof80].

We first argue that restricting attention to obfuscation of reversible circuits of the above form does not limit the generality of the treatment. Indeed, the set $\mathbb{B}_n$ of gates of the above form generates the alternating group $\mathbb{A}_{2^n}$ of even permutations over $\{0,1\}^n$ (see e.g. [CG75, Bro04]). Furthermore, any (non-reversible) circuit can be embedded in a reversible circuit while preserving both the functionality and the complexity of the original circuit (see e.g. [Tof80]).[1]

On the other hand, reversible circuits have some attractive properties which are essential for our treatment:

Limited Independence and Pseudorandomness. The model enables for a natural notion of *random circuits* of certain dimensions (say, numbers of wires and gates), which is efficiently samplable. Furthermore, the fact that all gates compute permutations makes it plausible that the permutation computed by a *random n-wire, m-gate circuit* has some randomness properties, and that the "level of randomness" increases monotonically with m. (Natural distributions over general Boolean circuits do not appear to exhibit such properties.) Indeed, the pseudorandomness of random reversible circuits has been the focus of much study over the past decades, with some very new and exciting progress.

Gowers [Gow96] shows that $\mathcal{C}_{n,m}$, the family of n-wire, m-gate circuits is ε-close to being strongly t-wise independent whenever $m = \Omega(n^3 t^3 \log(\varepsilon^{-1}))$. Hoory et al. [HMMR05] and later Brodsky and Hoory [HB05] improve this bound to $m = \Omega(n^3 t^2 + n^2 t \log(\varepsilon^{-1}))$. Very recently, He and O'Donnell [HO24] and Gretta, He and Pelecanos [GHP24] have further improved the bound to $m = \tilde{O}(nt \log(\varepsilon^{-1}))$. (We note that, while Gowers considered all $8!\binom{n}{3}$ permutations on 3 wires as base permutations, all other works mentioned above consider the same set $\mathbb{B}_n$ of base permutations considered here.)

Gowers conjectured that the family of permutations defined by m-gate reversible circuits on n wires might be pseudorandom (in the cryptographic sense) for some $m = poly(n)$.[2] In fact, his construction can be viewed as the "quintessential block cipher" where each base permutation is an independently

[1] More specifically, any circuit C with α input wires, β output wires, μ NAND gates and width ω can be transformed to a reversible circuit C' on $n = \alpha + \beta + \delta$ wires and m gates, where $n = O(\omega)$ and $m = O(\mu)$, and where $C'(x, y, 0^\delta) = (x, C(x) + y, 0^\delta)$ for any $x \in \{0,1\}^\alpha, y \in \{0,1\}^\beta$. While known constructions are only guaranteed to preserve functionality when some of the input wires are set to 0 and may thus not be sufficient for the purpose of program obfuscation. To address this issue, we give an "obfuscation compatible" transform with the additional guarantee that $C'(x, y, z) = (x, y, z)$ whenever $z \neq 0^\delta$ (see [CCMR24]).

[2] The conjecture is actually only implicit in [Gow96]. It is made explicit in Barak's survey [Bar17].

chosen "S-Box" and the key essentially specifies the schedule and ordering of S-boxes to be applied. Indeed, the main conceptual difference between the Gowers construction and modern block ciphers such as AES is the use of a key schedule that significantly reduces the overall key size. (AES and other block ciphers contain additional linear operations over the entire state; however as evidenced by the t-wise independence results mentioned above, the Gowers construction effectively approximates such operations as well.) The t-wise independence of AES and the pseudorandomness of the Gowers construction have also been studied in [LTV21,LPTV23,HO24].

Rerandomiability. Reversible circuits appear to be readily amenable to functionality-preserving rerandomization via local perturbations. We discuss this property at length later on, and only note at this point that all base permutations $\beta \in \mathbb{B}_n$ are inverses of themselves, namely $\beta\beta = I_n$, where I_n denotes the identity permutation on $\{0,1\}^n$. This also means that, for any circuit C on n wires, the circuit $C|C^\dagger$ computes I_n, where $C^\dagger$ has the gates of C in reverse order and $|$ denotes circuit concatenation. In fact, the set of n-gate reversible circuits with set $\mathbb{B}$ of base gates can be viewed as the Free Group $F_{\mathbb{B}}$ over alphabet $\mathbb{B}$. Furthermore, the operation of evaluating a circuit can be viewed as a group action of $F_{\mathbb{B}}$ on the Alternating group $\mathbb{A}_{2^n}$ of even permutations on $\{0,1\}^n$. The kernel of this action is the set of identity circuits and the cosets are the sets of functionally equivalent circuits. This algebraic structure provides a basis for our obfuscation scheme based on local perturbations, described in Sects. 1.5 and 6.

White-box Pseudorandomness. while the pseudorandomness properties of the permutations computed by random reversible circuits may be intriguing, they do not suffice for our needs. Here instead we are concerned with adversaries that have full access to the circuit description, and can mount attacks that combine the circuit's functionality and structure.

The good news about random reversible circuits is that their "internal structure" appears to be largely uncorrelated with their functionality, in the sense that even very large portions of a sufficiently long random circuit remain pseudorandom *even given oracle access to the overall circuit.* For instance, let m^* denote the number of gates that suffices for Gower's conjecture to hold with respect to some number of wires n and security parameter κ. (For notational simplicity we assume $n = \kappa$). Now, let $C \xleftarrow{R} \mathcal{C}_{n,m}$ for some $m > 2m^*$, and let C_i denote the circuit C without the m^*-gate sub-circuit that starts at gate i. It is easy to see that the following is implied by Gower's conjecture, for any i: Polytime adversaries that are given i, oracle access to C and a challenge $(m - m^*)$-gate circuit C', cannot tell significantly better than a random guess whether $C' = C_i$, or else C' is an independently chosen random circuit, i.e. $C' \xleftarrow{R} \mathcal{C}_{n,m-m^*}$.[3]

[3] Indeed, an algorithm A that guesses correctly for some i can be used to break Gower's conjecture: Given oracle access to an unknown function F, choose $P_0, P_1 \xleftarrow{R} \mathcal{C}_{n,i}, S_0, S_1 \xleftarrow{R} \mathcal{C}_{n,m-m^*-i}$ and $b \xleftarrow{R} \{0,1\}$, run A on input (i, P_0, S_0), and answer each oracle query x of A with $S_b(F(P_b(x))$. If $\mathcal{A}$ guesses b correctly then guess that F is taken from Gower's PRF, else guess that F is a random permutation.

Furthermore, it is only natural that this same property—pseudorandomness of large circuit segments—would extend also to sufficiently long random circuits with some *fixed functionality.* For instance, let $\mathcal{E}_{P,m}$ denote the set of all m-gate circuits that compute permutation P, let $C \xleftarrow{R} \mathcal{E}_{I_n,2m^*}$, and let $C_{[1,m^*]}$ denote the m^*-gate prefix of C. While $C_{[1,m^*]}$ is statistically far from a random n-wire, m^*-gate circuit, it appears plausible that the two distributions are indistinguishable.

By the same token, it seems plausible that

$$C : C \xleftarrow{R} \mathcal{E}_{I_n,2m^*} \overset{c}{\approx} C|C' : C \xleftarrow{R} \mathcal{C}_{n,m^*}; C' \xleftarrow{R} \mathcal{E}_{C^\dagger,m^*},$$

namely that a random $2m^*$-gate identity circuit is indistinguishable from a random m^*-gate circuit C followed by the inverse of another random m^*-gate circuit C' that's functionally equivalent to C. (We use $\mathcal{E}_{C,m}$ as a shorthand for $\mathcal{E}_{\mathcal{P}_C,m}$, where $\mathcal{P}_C$ is the permutation computed by circuit C.) Indeed, here we have two instances of the previous distribution, where the instances are correlated only via the permutation $\mathcal{P}_C$.[4]

Taking this logic a step further, let $\mathbf{C}$ be an arbitrary, potentially highly structured m-gate circuit, and let $C \xleftarrow{R} \mathcal{E}_{\mathbf{C},m'}$ be a random m'-gate circuit that is functionally equivalent to $\mathbf{C}$, where $m' \geq 2m^*m$. Then it is plausible that any $(m' - m^*)$-gate portion of C is indistinguishable from a random circuit of the same length. Furthermore, let $\mathbf{C}_1, \mathbf{C}_2$ be m_1-gate prefix and m_2-gate suffix of $\mathbf{C}$, $m_1 + m_2 = m$. Then it seems plausible that:

$$C : C \xleftarrow{R} \mathcal{E}_{\mathbf{C},2m^*m} \overset{c}{\approx} C_1|C_2 : \\ C_1 \xleftarrow{R} \mathcal{E}_{(\mathbf{C}_1|R),2m^*m_1}; C_2 \xleftarrow{R} \mathcal{E}_{(R^\dagger|\mathbf{C}_2),2m^*m_2}; R \xleftarrow{R} \mathcal{C}_{n,m^*},$$

namely that a random $2m^*m$-gate circuit that's functionally equivalent to $\mathbf{C}$ is indistinguishable from a random $2m^*m_1$-gate circuit C_1 that computes the permutation $\mathbf{C}_1|R$ for a random m^*-gate circuit R, followed by a random $2m^*m_2$-gate circuit C_2 that computes $R^\dagger|\mathbf{C}_2$. We call this assumption the **Split-Circuit Pseudorandomness (SCP)** assumption (see also Fig. 1).[5]

Discussion. We stress that the SCP assumption may not be efficiently falsifiable even if false. This is so since it considers indistinguishability of distributions

[4] One consequence of the correlation is that here m^* needs to be large enough not only to make Gower's conjecture work, but also to make sure that two random instantiations of the same permutation look sufficiently different from each other. However, this distinction appears to become moot when $m^* = \tilde{\Omega}(|\mathbb{B}_n|)$. See more discussion within.

[5] The constant 2 above is clearly arbitrary and was only used to underline the progression of the logic underlying the assumption. Also, the above formulation actually corresponds to a strong version of the SCP assumption, whereas a somewhat weaker version suffices for our treatment. See more details within.

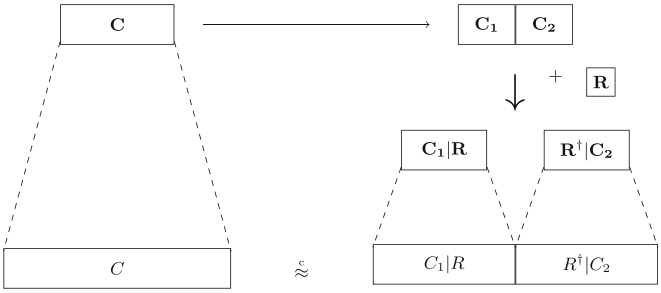


Fig. 1. The Split Circuit Pseudorandomness (SCP) assumption. Circuit $\mathbf{C}$ (top left) is an arbitrary n-wire, m-gate reversible circuit. Circuits $\mathbf{C_1}$ and $\mathbf{C_2}$ at the top right are the m_1-gate prefix and m_2-gate suffix of $\mathbf{C}$ (with $m_1 + m_2 = m$), and R is a random $m^{\#}$-gate circuit, where $m^{\#}$ depends only on n and the security parameter, while m is an arbitrarily large polynomial. Circuits $C_1|R$ and $R^{\dagger}|C_2$ at the bottom right are random $m^{\#}m_1$-gate and $m^{\#}m_2$-gate circuits that are functionally equivalent to $\mathbf{C_1}|R$ and $R^{\dagger}|\mathbf{C_2}$, respectively. The assumption states that the concatenation of these two circuits is computationally indistinguishable from a random $m^{\#}m$-gate circuit that's functionally equivalent to $\mathbf{C}$ (bottom left), in spite of the fact that each one of $C_1|R$ and $R^{\dagger}|C_2$, taken separately, computes a pseudorandom permutation.

which are not known to be efficiently samplable. In fact, many of these distributions are not even efficiently recognizable - e.g. we don't have a feasible way to know for sure that a given circuit computes even the identity permutation.

At the same time, this assumption is a fairly minimal instantiation of a more general intuition regarding the pseudorandomness of sufficiently long random circuits with fixed functionality. This intuition essentially states that there exist $n^*_\kappa, m^*_\kappa \in \text{poly}(\kappa)$ such that for any large enough value of the security parameter κ, any $m \geq m^*_\kappa$, and any fixed circuit $\mathbf{C} \in \mathcal{C}_{n^*_\kappa, m}$, a random $O(m^*_\kappa m)$-gate circuit C that is functionally equivalent to $\mathbf{C}$ essentially renders "all information on both the structure and functionality of short and medium range segments of $\mathbf{C}$" inaccessible to polytime observers, while keeping the overall functionality intact.

We note that the SCP assumption appears closely related - at least in spirit - to assumptions regarding the hardness of distinguishing between random strings with different Kolmogorov (respectively, MCSP) complexities (see e.g. [LP20, LP21, IRS22, BLMP23, ILW23]). While some initial connections are made within, further exploration and exploitation of these apparent connections may be of independent interest.

1.3 New Notions of Security for Circuit Obfuscation

Next, we sketch the definition of RIO obfuscation (which relaxes IO). We also define another variant, called random output (RO) obfuscation, which will be useful for presenting and analyzing our constructions. (As we'll see, RO obfuscation

for some circuit classes will provide stronger guarantees than IO for these classes; still, IO for all circuits and RO for all circuits will end up being equivalent.)

Let $\mathcal{C}_n$ denote the set of all n-wire reversible circuits. A transformation $O : \mathcal{C}_n \to \mathcal{C}_n$ is functionality-preserving if $O(C)$ and C are functionally equivalent for any $C \in \mathcal{C}_n$.

A functionality-preserving transformation $O : \mathcal{C}_n \to \mathcal{C}_n$, is a **random output indistinguishability (RO)** obfuscator for a set $\mathbb{C} \subseteq \mathcal{C}_n$ of circuits and inner-stretch function ξ if there exists an efficient "post-processing algorithm" π such that for any m-gate circuit $C \in \mathbb{C}$ we have:

$$O(C) \stackrel{c}{\approx} \pi(\widehat{C}) : \widehat{C} \stackrel{R}{\leftarrow} \mathcal{E}_{C,\xi(n,m)}.$$

It can be verified that if $\xi(n, m) = m$ then RO obfuscation coincides with standard indistinguishability obfuscation (IO). (In particular, in this case we can set $\pi = O$ without losing generality.) When $\xi(n, m) > m$, RO obfuscation for some classes of circuits becomes non-trivial to obtain even in situations where IO for these classes is trivial (e.g. when the input circuit C is the only one with the same size and functionality in that class). Furthermore, together with the SCP assumption, RO obfuscation with large inner-stretch (namely, when $\xi(n, m) = \Omega(m^* m)$) guarantees that both the structure and the functionality of any not-too-large portion of C are essentially lost. Still, observe that RO obfuscation for any class of circuits can be constructed from IO for all circuits, by appropriately padding the input circuit before obfuscating it.

A functionality-preserving transformation $O : \mathcal{C}_n \to \mathcal{C}_n$, is a **random input and output (RIO)** obfuscator with respect to $\mathcal{C}_{n,m}$ if the following two requirements hold:[6]

1. $(O(C), O(C)) : C \stackrel{R}{\leftarrow} C_{n,m} \stackrel{c}{\approx} (O(C), O(C')) : C \stackrel{R}{\leftarrow} C_{n,m}; C' \stackrel{R}{\leftarrow} \mathcal{E}_{C,m}$
2. For any "advice" function Z with poly-length output we have

$$O(C), Z(\mathcal{P}(C_1, C_2)) \stackrel{c}{\approx} O(C'), Z(\mathcal{P}(C_1, C_2))$$

where $C \stackrel{R}{\leftarrow} \mathcal{C}_{n,m}$, $C' \stackrel{R}{\leftarrow} \mathcal{E}_{C,m}$, $\mathcal{P}(C)$ denotes the permutation computed by circuit C, and C_1 and C_2 are the $m/2$-gate prefix and suffix of C, respectively.

The two requirements from an RIO obfuscator are incomparable and capture different security aspects: The first requirement makes sure that two obfuscated versions of the same random circuit C do not look "too much alike" relative to the obfuscated versions of two random circuits C, C' with the same functionality and length.

The second requirement makes sure that $O(C)$ remains indistinguishable from $O(C')$ even when given arbitrary polysize advice that's computed given the permutations computed by C_1 and C_2, the first and second halves of C.

[6] For simplicity we present here the definition only for the special case where there is no inner-stretch requirement and the input is uniform. A more general formulation appears within.

Note that in the left hand side distribution, the permutation computed by C_1 is the same as the permutation computed by the first half of the obfuscated circuit C. In contrast, in the right hand side experiment the permutation computed by the first half of the obfuscated circuit C' is different than the permutation computed by C_1.

It is stressed that neither of the two RIO requirements considers a distinguisher that has access to the input circuit C. This stands in sharp contrast to the case of IO (and RO) where the distinguisher sees both C and $O(C)$, making RIO potentially easier to obtain—not only than IO, but also than IO for random circuits.

1.4 From RIO to RO for All Circuits

We show:

Theorem 1 (Informal). *If there exist RIO obfuscators for $\mathcal{C}_{n,m^*}$, where n, m^* satisfy the SCP assumption, then there exists an RO obfuscator O with large inner-stretch for all circuits in $\mathcal{C}_n$. Furthermore, if C has m gates then $O(C)$ has poly$(m^*)m$ gates.*

For the construction, we first construct the following building blocks (See Fig. 2):

- A random identity generator (RIG), which is an RO obfuscator for the identity permutation with inner-stretch $2m^*$. This is done by choosing $C \xleftarrow{R} \mathcal{C}_{n,m^*}$, then sampling $C', C'' \xleftarrow{R} O(C)$ where O is an RIO obfuscator, and finally outputting $C'|C''^{\dagger}$. Security is proven using the RIO security of O and the SCP assumption.
- A gate obfuscator GO, namely an RO obfuscator for β, per each gate $\beta \in \mathbb{B}_n$. This can be done simply by sampling random identities using the previous step, until an identity circuit that starts with β is sampled. Then, remove the leading β gate (or alternatively replace it with an identity gate) and output the result.
- A procedure for "soldering" RO-obfuscated circuits, namely combining an RO obfuscator O_1 for a circuit C_1 and an RO obfuscator O_2 for a circuit C_2 into an RO obfuscator for the circuit $C_1|C_2$. The idea is again simple: Let $\tilde{C}_1 \xleftarrow{R} O_1(C_1)$, $\tilde{C}_2 \xleftarrow{R} O_2(C_2)$. Now, let $\tilde{C}_1 = C_{1,1}|C_{1,2}$ and $\tilde{C}_2 = C_{2,1}|C_{2,2}$, where $C_{1,2}$ and $C_{2,1}$ have m^*-gates each. Now, compute $G \xleftarrow{R} O(C_{1,2}|C_{2,1})$ where O is an RIO obfuscator, and output the circuit $C_{1,1}|G|C_{2,2}$. Security is proven based on the security properties of the building blocks, using the SCP assumption. (While the proof is conceptually straightforward, care has to be taken to the fact that several of the intermediate distributions are not efficiently samplable.)

Now, to obfuscate a circuit $C = \gamma_1 \dots \gamma_m$, first sample $\Gamma_i \xleftarrow{R} \mathrm{GO}(\gamma_i)$ for $i = 1..m$, and then solder the circuit pieces one by one: Let $C_1 = \Gamma_1$, and for $i = 2..m$ let C_i be the result of soldering C_{i-1} and Γ_i. Finally output C_m.

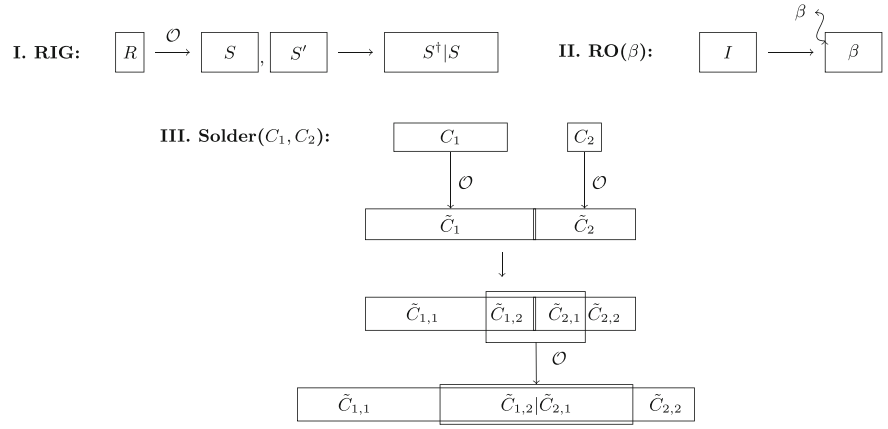


Fig. 2. The building blocks for constructing RO obfuscation for all reversible circuits from RIO obfuscation for bounded length random circuits. The first building block is random identity generators (RIGs), constructed by concatenating two RIO-obfuscated versions of a random circuit, one in reverse. The second building block is RO obfuscators for single gates, constructed by sampling a RIG with the desired first gate and removing that gate. The third building block is soldering RO-obfuscated versions of circuits C_1 and C_2 into an RO-obfuscated version of $C_1|C_2$ by concatenating the individual obfuscations and re-obfuscating the circuit segment around the seam. These basic building blocks are then iterated to solder obfuscated versions or arbitrarily long circuits.

The use of RO obfuscation with large inner-stretch for the intermediate steps in the obfuscation process (rather than, say, plain IO) is critical for this approach to work. In particular, we critically use the fact that, after each step, the intermediate circuit C_i has essentially lost "all polynomially accessible information" on its structure (i.e. on $\gamma_1 \dots \gamma_i$) *other than the overall functionality of* $\gamma_1 \dots \gamma_i$. This may be viewed as evidence for the power of RO obfuscation.

1.5 Constructing RIO Obfuscators

Reversible circuits admit a wide variety of *functionality preserving local perturbations*. For instance, given a circuit $C = \gamma_1 \dots \gamma_i \dots \gamma_{i+\ell} \dots \gamma_m$ one can replace a circuit segment $\gamma_i \dots \gamma_{i+\ell}$ with any circuit $C' = \gamma'_1 \dots \gamma'_{\ell'}$ that is functionally equivalent to $\gamma_i \dots \gamma_{i+\ell}$ (i.e. $\mathcal{P}_{C'} = \mathcal{P}_{\gamma_i \dots \gamma_{i+\ell}}$), obtaining a perturbed circuit $C'' = \gamma_1 \dots \gamma_{i-1}|C'|\gamma_{i+\ell+1} \dots \gamma_m$ that is functionally equivalent to C (i.e. $\mathcal{P}_{C''} = \mathcal{P}_C$). When ℓ, ℓ' are small enough (say, constants), it is possible to sample uniformly from all - or sufficiently many - ℓ'-gate circuits that are functionally equivalent to any given ℓ-gate circuit so as to make for effective randomization of that particular segment. It is thus tempting to explore the possibility that the space of functionally equivalent circuits within a given length is ergodic—namely that iterative replacements of randomly chosen small circuit segments with random functionally equivalent alternative segments may provide more global mix-

ing (and hence obfuscation) properties. Note that this mixing approach can be viewed as a recipe for generating random elements in a group presentation where the underlying alphabet is the set of base gates, and the set of generating words consist of the initial circuit, plus a sufficiently large set of identity circuits.

One drawback of a literal implementation of this idea is that much of the randomness in a random circuit can be effectively "factored out", say via efficiently computable cannonical representations of circuits. For instance, note that many pairs of gates $\beta, \beta' \in \mathbb{B}_n$ commute, namely $\mathcal{P}_{\beta\beta'} = \mathcal{P}_{\beta'\beta}$. (In fact, all but $O(1/n)$ of them do.) Thus applying the above process with segments of size up to $o(\sqrt{n})$ and $\ell' = \ell$ will end up only re-ordering commuting gates, almost always. However, such re-randomization is easily factored out by using a cannonical representation that fixes the order for each pair of commuting gates (say, starting from the left and using some lexicographic ordering of the gates).

A natural approach to get around the above "attack" is to consider circuit segments that are not consecutive: for instance, pick a random gate γ_i in the circuit and a random direction (left/right), and let γ_j be the nearest gate in that direction that "collides" (i.e., does not commutes) with γ_i. Then remove γ_i and γ_j, and replace them by a functionally equivalent sequence of gates (say, as in Fig. 3), placed anywhere between locations i and j. Such a strategy may appear harder to reverse, but it is again ultimately reversible (at least in and of itself) since it leaves behind clusters of "collision debris" gates that are relatively easy to identify.

A more general issue with naive realizations of local rerandomization of circuit segments is that, for most ℓ-gate circuits C, the set $\mathcal{E}_{C,\ell}$ is relatively small. (As we demonstrate within, this is in fact a general property that holds for all values of ℓ; but it is perhaps most prominent when ℓ is small.) This means that, when $\ell = \ell'$ the above process may again not provide sufficient randomization. On the other hand, when $\ell < \ell'$, the circuit would continually grow in size, which means that there is little hope to reach any stationary distribution—or to even to guarantee more basic mixing properties such as having each segment in the final circuit depend on all gates in the original circuit.

Furthermore, it is unlikely to be the case any two functionally equivalent circuits of the same size are connected via a "path", or sequence of polynomially many local transformations that are quaranteed to be functionality preserving. Indeed, if this were the case, then we would have a polysize witness for the fact that two circuits are functionally equivalent, implying NP=coNP. (Note that this rules out the very existence of such a sequence, not just the feasibility of finding one. This observation is a slight variant of a more general result by Goldwasser and Rothblum [GR14], which demonstrates, in a similar way, that perfect IO for all circuits implies NP = coNP.)

Still, these arguments leave open the possibility that a somewhat more nuanced or structured local perturbation process could actually provide sufficient "confusion and diffusion" so as to satisfy the relatively weak requirements of RIO obfuscation for random circuits that have sufficiently many gates so as to make the Gowers conjecture hold.

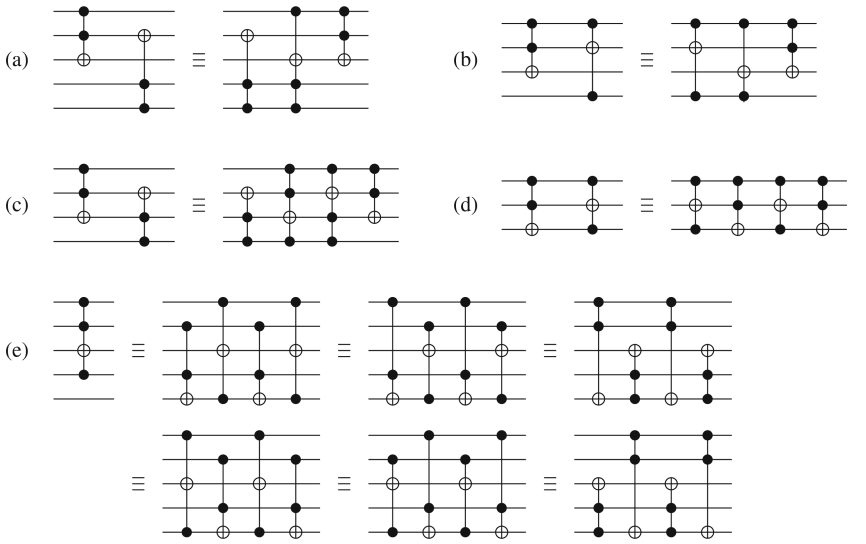


Fig. 3. Some possible replacements for the case of $\ell^{\text{OUT}} = 2$ (namely, colliding pairs of gates), for the special case where the control function is $\phi(a, b) = ab$ (namely, logical conjunction). A gate is depicted as a vertical line connecting several wires, where the control wires are identified by black dots and the active wire is identified via a circle. Panels (a) and (b) show possible replacements for one-headed collision, i.e. for the case where the active wire of one gate is also a control wire of the other gate. Panels (c) and (d) correspond to a two-head collision, when the active wires of both gates are also control wires of the other gate. Notice that in panels (a) and (c) the circuit on the right includes a 3-control gate. As shown in panel (e), this 3-control gate can be decomposed into four base gates, while using an additional wire (to be chosen out of the $n-4$ remaining wires in the circuit). Overall, in case (c) the figure depicts $6^2\binom{n-4}{2}$ replacement circuits.

We heuristically propose such a process. First, we formulate a representation of circuits that facilitates generalizing the above "colliding gates" method to identifying sets of nearby (albeit not necessarily consecutive) gates that form structured sub-circuits that are amenable to rerandomization.

Second, we split the mixing process into two stages. In the first, "inflationary" stage, the size ℓ^{OUT} of the sub-circuits to be replaced is a relatively small constant, and the size ℓ^{IN} of the replacement circuit is only slightly larger - just enough for effective re-randomization of the structure of the replaced sub-circuit while preserving its functionality. In the second, "kneading" stage, the size ℓ^{KND} of the replacement circuit is set to be identical to the size of the circuit to be replaced, and both are set to be significantly larger than ℓ^{IN}—say $\ell^{\text{KND}} = \Theta(\log \log n)$, where n is the number of wires.

In a nutshell, the rationale here is the following. The inflationary stage adds a significant amount of "random redundancy" to the circuit. (We measure the "level of redundancy" in a circuit by way of the "complexity gap", or the differ-

ence between the number of gates in the circuit and the number of gates in the smallest functionally equivalent circuit.) As noted above, this stage alone does not suffice since the complexity gap is concentrated in small sub-circuits of the overall circuit and may thus still be identifiable and removable with feasible computational overhead. Still, the structure of the replaced sub-circuits enables the kneading stage to spread the already-existing complexity gap over successively larger sub-circuits, thus making it computationally hard to localize and remove.

We provide more detailed rationale within. It is stressed however that the analysis is far from rigorous, and that the proposed process is merely an exploration meant to demonstrate the viability of the approach rather than well-analyzed candidate circuit obfuscator. We leave further analysis to future work.

1.6 Related Work

The randomizing power of permutation groups is not new to cryptography, with a prominent examples being the seminal work of Kilian that shows how to use Barrington's $\mathbb{S}_5$ representation of branching programs to randomize general NC^1 computations [Bar86, Kil88]. Kilian's randomization technique has been widely used, including in early candidate obfuscation schemes [CV13].

Alagic, Jeffery and Jordan [AJJ14] use the randomizing power of permutation groups (in the more restricted context of Braid permutations) to show unconditional "partial inditinguishability obfuscation" mechanisms for programs that are within the same equivalence class of a certain normal-form representation.

Chamon, Mucciolo and Ruckenstein [CMR22] study pseudorandomness properties of random reversible circuits, and provide evidence that as little as $m = O(n \log n)$ gates suffice for the family $\mathcal{C}_{n,m}$ to be an SPRP, when n is taken to be the security parameter.

Chamon et al. [CJMR22] use local perturbation techniques of a different flavor of the ones proposed here to construct a candidate "homomorphic pseudorandom permutation family" and use it as a basis for a symmetric homomorphic encryption scheme. It is stressed though that the security requirements needed in that application are significantly weaker than the ones needed for general program obfuscation, or even RIO obfuscation.

Finally, [CRMC23] takes a thermodynamic approach to circuit complexity, and in particular studies mixing of polynomial-sized reversible circuits of a given functionality through the iterative equilibration of concatenated short subcircuits described via local equilibrium distributions of reversible gates. In particular, that work uses the thermodynamics framework to argue that the set of functionally equivalent reversible circuits of some size is partitioned to sectors where each sector is ergodic with mixing time that's polynomial in the circuit size. In other words, that work suggests that viability of the local mixing approach as an obfuscation method reduces to the indistinguishability of random circuits from different sectors.

2 Reversible Boolean Circuits

This section recalls the model of reversible Boolean circuits and its relationship with standard Boolean circuits.

A reversible Boolean circuit C on n wires consists of a sequence of permutations $C = \gamma_1 \dots \gamma_m$ where each γ_i is a permutation on $\{0,1\}^n$, taken from a predetermined set B of *base permutations.* The permutation $\mathcal{P}_C$ computed by C is the composition of the individual permutations, $\mathcal{P}_C = \gamma_m \circ \dots \circ \gamma_1$, or in other words $C(x) = \gamma_m(\dots \gamma_1(x) \dots)$.

We concentrate on circuits where the base permutations consist of applying a Toffoli gate to three chosen wires, where a Toffoli gate is a permutation on $\{0,1\}^3$ of the form $\tau_\phi(a_1, a_2, a_3) = (a_1 + \phi(a_2, a_3), a_2, a_3)$ where $\phi : \{0,1\}^2 \to \{0,1\}$ is the *control function* of the gate. (We often refer to the three wires of a Toffoli gate as *pins*, where the first pin is *active* and the second and third pins are *non-active.*) That is, we consider the set of base permutations defined by

$$\mathbb{B}_n = \{\beta_{w_1,w_2,w_3,\phi} : w_1, w_2, w_3 \in [n]^3, w_2 \neq w_1 \neq w_3, w_2 \neq w_3, \phi : \{0,1\}^2 \to \{0,1\}\}$$

where $\beta_{w_1,w_2,w_3,\phi}(x_1 \dots x_n) = y_1 \dots y_n$ such that $(y_{w_1}, y_{w_2}, y_{w_3}) = \tau_\phi(x_{w_1}, x_{w_2}, x_{w_3})$, and $y_j = x_j$ for each $j \in [n] \setminus \{w_1, w_2, w_3\}$. (We note that, as defined above, $\mathbb{B}_n$ is actually a multi-set since $\beta_{w_1,w_2,w_3,\phi}$ and $\beta_{w'_1,w'_2,w'_3,\phi'}$ may well describe the same permutation. In fact, while there are 16 different control functions ϕ, there are roughly $8n^3$ different base permutations overall. For convenience we use the convention where only a single representative of each base permutation is used, i.e. $b_n \stackrel{\text{def}}{=} |\mathbb{B}_n| \approx 8n^3$. However this convention does not appear essential for the treatment.)

The natural evaluation of $C = \gamma_1 \dots \gamma_m$, where each $\gamma_i = \beta_{w_{1,i},w_{2,i},w_{3,i},\phi_i}$, on input $x = x_1, ..., x_n \in \{0,1\}^n$ is described iteratively as follows. For $j = 1..n$ we have $x_j^{(0)} = x_j$, and for each $i = 1..m$ we have $(x_1^{(i)} \dots x_n^{(i)} = \gamma_i(x_1^{(i-1)} \dots x_n^{(i-1)})$. The *value of wire j after gate i* is defined as $x_j^{(i)}$. It may be useful to envision reversible circuits as a sequence of n horizontal parallel wires, where each gate connects three wires, and where the computation proceeds from left to right.

Since all base permutations (or, gates) are even, reversible circuits can only compute even permutations on $\{0,1\}^n$. Still, considering only circuits of the above form does not limit the generality of the treatment. Indeed, the set $\mathbb{B}_n$ of gates generates all even permutations over $\{0,1\}^n$, namely the alternating group $\mathbb{A}_{2^n}$ (see e.g. [CG75, Bro04]).

Furthermore, any circuit C with α input wires, β output wires, μ NAND gates and width ω can be transformed to a reversible circuit C' on $n = \alpha + \beta + \delta$ wires and m gates, where $n = O(\omega)$ and $m = O(\mu)$, and where $C'(x, y, 0^\delta) = (x, C(x) + y, 0^\delta)$ for any $x \in \{0,1\}^\alpha, y \in \{0,1\}^\beta$ (see e.g. [Ben73, Tof80, Ben89, Bro04]). In the Appendix we show how to "harden" the standard transformation so as to guarantee that $C'(x, y, z) = (x, y, z)$ for $z \neq 0^\delta$, and how to use the hardened

transform to show that obfuscation of reversible circuits suffices for general-purpose obfuscation of all circuits[7].

Let $C^\dagger$ denote the natural inverse (or, "reverse") of circuit C. That is, if $C = \gamma_1, ..., \gamma_m$ then $C^\dagger = \gamma_m, ..., \gamma_1$. Indeed, note that $\mathcal{P}_{C|C^\dagger} = \mathcal{P}_{C^\dagger|C} = I_n$, where I_n denotes the identity permutation on $\{0,1\}^n$. This is so since the base permutations are the inverses of themselves, i.e. $\mathcal{P}_{\beta|\beta} = I_n$ for all base permutations β. (Here '|' denotes the natural concatenation, or composition, of gates or circuits.) Let $\mathcal{C}_{n,m}$ denote the set of all m-gate circuits on n wires, and let $\mathcal{C}_n = \bigcup_{m>0} \mathcal{C}_{n,m}$.

For a circuit $C = \gamma_1 \dots \gamma_m$, let $|C| = m$ denote the number of gates in C. For $i, l \in [m]$, let $C_{[i,l]} = \gamma_i \dots \gamma_{i+l(\mathrm{mod}\ m)}$ denote the l-gate segment of C that starts at the ith gate, taken modularily; in particular, $i < 0$ refers to $m - i$. We also use $C_{[i,*]}$ as a shorthand for $C_{[i+1,m-i]}$.

A note about asymptotics. Throughout we treat n, the number of wires, m, the number of gates, and the runtimes of adversaries as functions of (specifically, polynomials in) the security parameter κ. We will also be mostly interested in the regime where m is polynomial in n. While our treatment is mostly asymptotic in κ, a non-asymptotic treatment with concrete values can be naturally derived.

2.1 Reversible Circuits as a Free Group

The set $\mathcal{C}_n$ of circuits with the set $\mathbb{B}_n$ of base gates can be viewed as the free group $F_{\mathbb{B}_n}$ of reduced strings (namely, strings where any two consecutive identical characters are eliminated) over the alphabet $\mathbb{B}_n$, with the group operation being the standard concatenation followed by reduction. One can then define the following natural action of $F_{\mathbb{B}_n}$ on the alternating group $\mathbb{A}_{2^n}$ (namely the group of all even permutations on $\{0,1\}^n$): for a circuit $C \in F_{\mathbb{B}_n}$ and permutation $\pi \in \mathbb{A}_{2^n}$, let $Eval(C, \pi) = \pi' \circ \pi$ where $\pi' = \mathcal{P}_C$. That is, $Eval(C, \pi)$ returns the permutation that first computes π and then evaluates C on the result. It is easy to see that $Eval$ is a group action whose kernel is the set of all identity circuits, and where each coset consists of all functionally equivalent circuits. Viewed in this way, program obfuscation is the problem of efficiently generating a pseudorandom sample from the coset of a given circuit (restricted to some given length).

3 Hardness Assumptions

This section presents and motivates the hardness assumptions used in this work. We start off with a reminder of the standard definition of computational

[7] Note that not all 16 control functions are needed for completeness to hold. In fact, the functions $\phi(x, y) = xy$, $\phi(x, y) = x$, $\phi(x, y) = 1$ suffice. However, considering all 16 control functions will be convenient for our treatment. In particular, this way the value of the active wire of τ_ϕ for a random control function is uniformly distributed regardless of the values of the input wires. Furthermore, having the identity as a base permutation (with $\phi(x, y) = 0$) will be convenient as well. This set of permutations is also the one considered by Brodsky and Hoory [HMMR05, HB05].

indistinguishability, and a natural extension thereof. Let $\mathcal{A} = \{A_\kappa\}_{\kappa\in\mathbf{N}}$ and $\mathcal{B} = \{B_\kappa\}_{\kappa\in\mathbf{N}}$ be distribution ensembles. (More precisely, we think of each A_κ (resp., B_κ) as a sampling algorithm. The distribution is defined via the probability of obtaining each possible output value when running the algorithm on an input which is drawn uniformly from $\{0,1\}^\dagger$.) $\mathcal{A}$ and $\mathcal{B}$ are said to be computationally indistinguishable, denoted $\mathcal{A} \stackrel{c}{\approx} \mathcal{B}$, if there exists a negligible function $\varepsilon(\kappa)$ such that for any polysize family of distinguishing algorithms $\mathcal{D} = \{D_\kappa\}_{\kappa\in\mathbf{N}}$ and all large enough values of κ it holds that $\text{Prob}[D_\kappa(A_\kappa) = 1] - \text{Prob}[D_\kappa(B_\kappa) = 1] < \varepsilon(\kappa)$.

3.1 On the Distribution of Functionally Equivalent Reversible Circuits

We first take a moment to define a measure of complexity for reversible circuits and then use it to estimate the sizes and makeup of the clusters of functionally equivalent reversible circuits of a given length. This detour will be useful both as a basis for our hardness assumptions, and as a basis for the local perturbation mechanisms developed in Sect. 6.

For a permutation $P \in \mathbb{A}_{2^n}$, let $\mathcal{E}_{P,m}$ denote the set of all m-gate circuits that compute P, namely $\mathcal{E}_{P,m} = \{C \in \mathcal{C}_{n,m} : \mathcal{P}_C = P\}$. Slightly abusing notation, for a circuit C we let $\mathcal{E}_{C,m} = \mathcal{E}_{\mathcal{P}(C),m}$.

We would like to estimate the size of $\mathcal{E}_{C,m}$. Towards this, we define the **Computational Complexity** $\text{CC}(P)$ of a permutation P as the number of gates in the smallest circuit that computes P. Similarly, let $\text{CC}(C) = \text{CC}(\mathcal{P}_C)$ denote the number of gates in the smallest circuit that computes $\mathcal{P}_C$. The **complexity gap** of an m-gate circuit C is defined to be $\text{CG}(C) = m - \text{CC}(C)$.

While $\text{CC}(C)$ is clearly distinct from the Kolmogorov complexities of string representations of a circuit C, these notions have many similarities. For one, it is easy to see that $b^m > |\mathcal{E}_{P,m}| > b^{\frac{1}{2}(m-\text{CC}(P))}$ for any permutation P, where $b \approx 8n^3$ is the number of base permutations:

Claim. For any circuit $C \in \mathcal{C}_n$ and any m we have $b^m > |\mathcal{E}_{C,m}| > b^{\frac{1}{2}(\text{CG}(C))}$.

Proof. The upper bound is immediate. For the lower bound, observe that for any sequence of base permutations $\beta_1 \dots \beta_l$ where $l = (\text{CG}(C))/2$, the circuit $C\beta_1\beta_1 \dots \beta_l\beta_l$ is functionally equivalent to C.

Furthermore, for almost all circuits in $\mathcal{C}_{n,m}$ we have $b^m > |\mathcal{E}_{C,m}| > b^{m-\frac{\text{CC}(C)}{\log b}-o(1)}$:

Claim. For all but an ε-fraction of the circuits $C \in \mathcal{C}_{n,m}$ we have $|\mathcal{E}_{C,m}| > b^{m-\frac{\text{CC}(C)}{\log b}-\log(\varepsilon m \log b)}$.

Proof. Note that any string $\sigma = \{0,1\}^s$ can be interpreted as a description of a reversible circuit $\hat{\sigma}$ on n wires and m gates where $s = m \log b$. (Recall that $b \approx 8n^3$ is the number of base permutations.) Furthermore, for any such n, m,

the string σ is fully determined via a circuit δ of size $\mathrm{CC}(\hat{\sigma})$ that's functionally equivalent to $\hat{\sigma}$, plus the ordinal of $\hat{\sigma}$ among all m-gate circuits that are functionally equivalent to $\hat{\sigma}$. This means that $K(\sigma) \leq \mathrm{CC}(\hat{\sigma}) + \log(|\mathcal{E}_{\hat{\sigma},m}|)$, or equivalently that

$$|\mathcal{E}_{\hat{\sigma},m}| \geq 2^{K(\sigma)-\mathrm{CC}(\hat{\sigma})} = b^{\frac{1}{\log b}(K(\sigma)-\mathrm{CC}(\hat{\sigma}))} = b^{\frac{m}{s}(K(\sigma)-\frac{\mathrm{CC}(\hat{\sigma})}{\log b})}$$

where $K(\sigma)$ denotes the Kolmogorov complexity of σ. The bound follows by noting that $K(\sigma) > s - \log(\varepsilon s)$ for all but εs of the strings $\sigma \in \{0,1\}^s$.

Put together, Claims 3.1 and 3.1 says that, while for a random m-gate circuit C, the size of $\mathcal{E}_{C,m}$ is only moderate, the size of $\mathcal{E}_{C,m'}$ grows exponentially in m', for any given C.

Another conclusion from this state of affairs is that $|\{C \in \mathcal{C}_{n,m} : \mathrm{CC}(C) = m\}|b^{-m} \leq \mathrm{negl}(m)$, namely that the fraction of m-gate circuits whose computational complexity is m, out of all m-gate circuits, tends to zero rather quickly as m grows (see more discussion in [CRMC23].) This fact becomes handy in Sect. 6.

3.2 Hardness Assumptions Regarding Random Reversible Circuits

We present the hardness assumptions used in this work. The presentation builds on the motivation given in the Introduction. Further motivation is provided by presenting a sequence of gradually stronger assumptions culminating in the assumptions we actually use later on. This presentation will hopefully provide additional evidence for the viability of the main assumption (Assumption 4).

Limited Independence. We start by recalling the works that serve as the mathematical and intuitive basis for our analysis. Intrigued by the potential pseudorandomness of random reversible circuits, Gowers [Gow96] showed that $\mathcal{C}_{n,m}$, the family of m-gate permutations on n wires, is ε-close to being strongly t-wise independent for any $t < 2^n$ and $m = O(n^3t^3\log(\varepsilon^{-1}))$. That is, for any sequence of distinct values $x_1 \ldots x_t \in \{0,1\}^n$, and for $C \xleftarrow{R} \mathcal{C}_{n,m}$, the statistical distance between $C(x_1)\ldots C(x_t)$ and a random sequence of distinct values $r_1 \ldots r_t$, is at most ε. Hoory et al. [HMMR05] and later Brodsky and Hoory [HB05] improve this bound to $m = O(n^3t^2 + n^2t\log(\varepsilon^{-1}))$. Very recently, He and O'Donnell [HO24] and Gretta, He and Pelecanos [GHP24] have further improved the bound to $m = \tilde{O}(nt\log(\varepsilon^{-1}))$. (We note that, while Gowers considered all $8!\binom{n}{3}$ permutations on 3 wires as base permutations, all other works mentioned above consider the same set $\mathbb{B}_n$ of base permutations considered here.)

Pseudorandomness. Gowers conjectured that the family of permutations defined by m-gate reversible circuits on n wires might be pseudorandom (in the cryptographic sense) for some $m = poly(n)$. This construction can be viewed as the "quintessential block cipher" where each base permutation is an independently chosen "S-Box" and the key essentially specifies the schedule of which S-boxes to use. Indeed, the main difference between the Gowers construction and modern block ciphers such as AES is the key schedule that significantly reduces the

key size. (AES and other block ciphers contain additional linear operations over the entire state; however as evidenced by the k-wise independence results mentioned above, the Gowers construction effectively approximates such operations as well.) The k-wise independence of AES and the pseudorandomness of the Gowers construction have been studied in [LTV21,LPTV23,HO24]. We adopt this conjecture as a starting point for our investigation:

Definition 1 (Strong Pseudorandom Permutations (SPRPs)). *An ensemble $F = \{F_\kappa\}_{\kappa\in\mathbf{N}}$ of circuit families, where the family $F_\kappa \subset \mathcal{C}_{n_\kappa}$ consists of circuits on n_κ wires, is a* **strong pseudorandom permutation family** *if there exists a negligible function $\nu(\kappa)$ such that for any family of polynomial-size adversaries $\mathcal{A} = \{A_\kappa\}_{\kappa\in\mathbf{N}}$, and all large enough value of κ we have*

$$\text{Prob}[A_\kappa^{C,C^\dagger} = 1 : C \xleftarrow{R} F_\kappa] - \text{Prob}[A_\kappa^{P,P^{-1}} = 1 : P \xleftarrow{R} \mathbb{A}_{2^{n_\kappa}}] < \nu(\kappa). \quad (1)$$

Here $\mathbb{A}_{2^n}$ denotes the set of even permutations on the set $\{0,1\}^n$ and $\text{poly}(\kappa)$ denotes the set of polynomials in κ. Next we state Gowers' conjecture from the Introduction):

Assumption 1 (Polysize random reversible circuits are SPRPs [Gow96]). There exist $n^*_\kappa, m^*_\kappa \in \text{poly}(\kappa)$ such that the ensemble $F = \{F_\kappa\}_{\kappa\in\mathbf{N}}$ where $F_\kappa = \mathcal{C}_{n^*_\kappa,m^*_\kappa}$ is an SPRP.

We note that, while our analysis remains valid for any polynomial values of n^*_κ, m^*_κ, the assumption does not appear to be easy to refute even for relatively shallow circuits with $n^*_\kappa = \Theta(\kappa)$ and $m^*_\kappa = \tilde{\Theta}(\kappa)$. Additional argumentation for the viability of this assumption for the case where $m^*_\kappa = \tilde{\Theta}(n^*_\kappa)$ appears in [CMR22].

Pseudorandomness of correlated SPRPs. As a first step towards presenting our main assumption regarding pseudorandomness of split random circuits with fixed functionality, we demonstrate that a milder form of that assumption actually follows from a mild extension of Assumption 1. Rather than considering only the family of all circuits of a given lenth, the extension considers circuits that are sufficiently long prefixes of a sufficiently long random circuit that computes some fixed permutation. Specifically, let $\mathbf{Q} = \{\mathbf{Q}_\kappa\}_{\kappa\in\mathbf{N}}$ with $Q_\kappa \in \mathcal{C}_{n^*_\kappa,m_\kappa}$ be an ensemble of circuits, and let $C \xleftarrow{R} \mathcal{E}_{Q_\kappa,m}$ be a random m-gate circuit that computes Q_κ, where $m \geq m_\kappa m^*_\kappa$ for a "long enough cushion" m^*_κ, akin to the number of gates needed to obtain pseudorandomness in Assumption 1. We assume that, for any ℓ such that $m^*_\kappa \leq \ell \leq (m_\kappa - 1)m^*_\kappa$, the ℓ-gate prefix of C is an SPRP:

Assumption 2 (Prefixes of Random Circuits with Fixed Functionality are SPRPs). There exist $n^*_\kappa, m^*_\kappa \in \text{poly}(\kappa)$ such that for any ensemble $\mathbf{Q} = \{\mathbf{Q}_\kappa\}_{\kappa\in\mathbf{N}}$ of circuits with $Q_\kappa \in \mathcal{C}_{n^*_\kappa,m_{Q_\kappa}}$ for $m_{Q_\kappa} \in \text{poly}(\kappa)$, and any m_κ, ℓ_κ such that $m_\kappa \geq m_{Q_\kappa} m^*_\kappa$ and $m^*_\kappa \leq \ell_\kappa \leq m_\kappa - m^*_\kappa$, the ensemble $\{G_\kappa\}_{\kappa\in\mathbf{N}}$ where $G_\kappa = \{C_{[1,\ell_\kappa]} : C \in \mathcal{E}_{Q_\kappa,m_\kappa}\}$ is an SPRP.

We note that circuits drawn from G_κ (for some fixed Q_κ) are in general statistically far from random ℓ_κ-gate circuits.[8] Still, it appears that Assumption 2 is only a mild generalization of Assumption 1.

An immediate consequence from Assumption 2 is that, for any ensemble of fixed circuits $\{Q_\kappa\}_{\kappa\in\mathbf{N}}$ where $Q_\kappa \in \mathcal{C}_{n^*_\kappa, m_{Q_\kappa}}$, polysize adversaries can distinguish between the following two cases only with negligible advantage.

Oracle Access to a Prefix and Remainder of a Random Circuit for Q_κ: The adversary has oracle access to C_1, C_2 (and their inverses), where $C = C_1|\mathcal{C}_2$ is a random m_κ-gate circuit for $Q_\kappa \in \mathbb{A}_{2^{n^*(\kappa)}}$, where $|C_1| = \ell_\kappa$, and where $n^*_\kappa, m^*_\kappa, m_\kappa, \ell_\kappa$ satisfy the length requirements of Assumption 2.

Oracle Access to two SPRPs that Jointly Compute Q_κ: Let $\{Q_{1,\kappa}, Q_{2,\kappa}\}_{\kappa\in\mathbf{N}}$ be an ensemble of pairs of circuits where $Q_\kappa = Q_{1,\kappa}|Q_{2,\kappa}$, and let $F = \{F_\kappa\}_{\kappa\in\mathbf{N}}$ be an SPRP ensemble where $\mathcal{F}_\kappa \subseteq \mathcal{C}_{n^*}$. The adversary has oracle access to P_1, P_2 (and their inverses), where $P_1 = Q_{1,\kappa}|C$ and $P_2 = C^\dagger|Q_{2,\kappa}$, and $C \xleftarrow{R} F_\kappa$.

That is:

Claim. Let $n^*_\kappa, m^*_\kappa \in \text{poly}(\kappa)$ and $\mathbf{Q} = \{\mathbf{Q}_\kappa\}_{\kappa\in\mathbf{K}}$ be as in Assumption 2 with $Q_\kappa = Q_{1,\kappa}|Q_{2,\kappa}$, and let $F = \{F_\kappa\}_{\kappa\in\mathbf{N}}$ where $F_\kappa \subset \mathcal{C}_{n^*_\kappa}$ be an SPRP. Then for any m_κ, ℓ_κ s.t. $m_{Q_\kappa} m^*_\kappa \le m_\kappa$ and $m^*_\kappa \le \ell_\kappa \le m_\kappa - m^*_\kappa$ there exists a negligible function $\nu(\kappa)$ such that for any family of polynomial-size adversaries $\mathcal{A} = \{A_\kappa\}_{\kappa\in\mathbf{N}}$, and all large enough value of κ we have

$$\begin{aligned}\text{Prob}[A_\kappa^{C_1, C_1^\dagger, C_2, C_2^\dagger} = 1 : C \in \mathcal{E}_{Q_\kappa, m_\kappa}; C_1 = C_{[1,\ell_\kappa]}, C_2 = C_{[\ell_\kappa, *]}] - \\ \text{Prob}[A_\kappa^{P_1, P_1^{-1}, P_2, P_2^{-1}} = 1 : C \xleftarrow{R} F_\kappa; P_1 = Q_{1,\kappa}|C; P_2 = C^\dagger|Q_{2,\kappa}] < \nu(\kappa).\end{aligned} \tag{2}$$

Proof. Since $\{F_\kappa\}_{\kappa\in\mathbf{N}}$ is an SPRP ensemble then so is the ensembles $\{P_{1,\kappa}|C : C \xleftarrow{R} F_\kappa\}_{\kappa\in\mathbf{N}}$. It follows that:

$$\text{Prob}[A_\kappa^{C_1, C_1^\dagger} = 1 : C \in \mathcal{E}_{Q_\kappa, m_\kappa}; C_1 = C_{[1,\ell_\kappa]}] - \text{Prob}[A_\kappa^{P_1, P_1^\dagger} = 1 : P_1 \xleftarrow{R} F_\kappa] < \nu(\kappa).$$

The claim follows by observing that oracle access to the last two oracles in (2), namely either $C_2, C_2^\dagger$ in the left hand side experiment or P_2, P_2^{-1} in the

[8] As a simple example, compare a random n-wire, $2m$-gate circuit R that computes the identity permutation I_n to a circuit $C_1|C_2$ where C_1 is a random m-gate circuit and C_2 is a random m-gate circuit such that $C_1|C_2$ computes I_n. Observe that R_1, the m-gate prefix of R, is more likely to compute a permutation that's computed by many m-gate circuits, or in other words a permutation with smaller circuit complexity than C_1. (Indeed, let $\alpha \in \mathcal{C}_{n,m}$. Then $\Pr[C_1 = \alpha] = b^{-m}$ (where b is the number of gates on n wires), whereas $\Pr[R_1 = \alpha]$ is the number of m-gate circuits R_2 such that $\mathcal{P}_{\alpha|R_2} = I_n$ divided by the number of $2m$-gate identity circuits, namely $|\mathcal{E}_{\alpha,m}|/|\mathcal{E}_{I_n,2m}|$. By Claim 3.1, for most α the latter probability is proportional to $b^{-\text{CC}(\alpha)}$.).

right hand side experiment, can be emulated given oracle access to the first two oracles in that experiment and advice in the form of a polysize circuit C_{Q_κ} that computes Q_κ. (Specifically, let O_1, O_2, O_3, O_4 denote the four oracles. Then, $O_3(x) = C_{Q_\kappa}(y)$ where $y = O_2(x)$. Similarly, $O_4(x) = O_1(y)$ where $y = C^\dagger_{Q_\kappa}(x)$.)

Pseudorandomness of Split Random Ciruits with Fixed Functionality. We now turn to considering observers that, rather than only having oracle access to the permutations in (2), have access to a *random circuit* (of a certain size) that computes each permutation. Clearly, having access to a polysize circuit that computes a permutation provides significantly more "computational power" than oracle access to the permutation (for one, the permutation is now easily distinguishable from a random permutation). Still, intuitively, the added power provided by *sufficiently long* random circuits that compute the two permutations in question (either $\mathcal{P}_{C_1}, \mathcal{P}_{C_2}$ or alternatively P_1, P_2) should not be of any help in distinguishing (2). This intuition is formalized in the next assumption, which states that for any ensemble of fixed permutations $\{Q_\kappa\}_{\kappa\in\mathbf{N}}$, which are defined by way of an ensemble of pairs of polysize circuits $\{P_{1,\kappa}, P_{1,\kappa}\}_{\kappa\in\mathbf{N}}$ where $P_{i,\kappa} \in C_{n_\kappa, m_{i,\kappa}}$, $i = 1, 2$, and $\mathcal{P}_{P_{1,\kappa}|P_{1,\kappa}} = Q_\kappa$, polysize adversaries can distinguish between the following distributions only with negligible advantage.

- A circuit of the form $\widehat{C}_1|\widehat{C}_2$ where $\widehat{C}_1$ is a random $\ell_{1,\kappa}$-gate circuit that computes $\mathcal{P}_{P_{1,\kappa}|C}$, where $C \xleftarrow{R} F_\kappa$ for an SPRP ensemble $\{F_\kappa\}_{\kappa\in\mathbf{N}}$, where $\ell_{1,\kappa}$ is larger than $(m_{1,\kappa} + |C|)$ by a sufficiently large margin, $\widehat{C}_2$ is a random $\ell_{2,\kappa}$-gate circuit that computes $\mathcal{P}_{C^\dagger|P_{2,\kappa}}$ and $\ell_{2,\kappa}$ is larger than $(m_{2,\kappa} + |C|)$ by a sufficiently large margin.
- A random $(\ell_{1,\kappa} + \ell_{2,\kappa})$-gate circuit $\widehat{C}$ that computes Q_κ.

A bit more formally:

Assumption 3 (Split Pseudorandom Circuits are Pseudorandom (SPCP)). For any SPRP ensemble $F = \{F_\kappa\}_{\kappa\in\mathbf{N}}$ where $F_\kappa \subset \mathcal{C}_{n_\kappa, m_\kappa}$ there exist $m^\#_\kappa \in \text{poly}(\kappa)$ such that for any ensemble of pairs of circuits $\mathbf{Q} = \{\mathbf{P_1}_{,\kappa}, \mathbf{P_2}_{,\kappa}\}_{\kappa\in\mathbf{N}}$ where $P_{i,\kappa} \in C_{n_\kappa, m_{i,\kappa}}$, and any $\ell_{1,\kappa}, \ell_{2,\kappa}$ where $\ell_{i,\kappa} \geq m_{i,\kappa} m^\#_\kappa$, $i = 1, 2$, we have:

$$\{\widehat{C}_1|\widehat{C}_2 : C \xleftarrow{R} F_\kappa; \widehat{C}_1 \xleftarrow{R} \mathcal{E}_{(P_{1,\kappa}|C),\ell_{1,\kappa}}; \widehat{C}_2 \xleftarrow{R} \mathcal{E}_{(C^\dagger|P_{2,\kappa}),\ell_{2,\kappa}}\}_{\kappa\in\mathbf{N}} \overset{c}{\approx} \{\widehat{C} : \widehat{C} \xleftarrow{R} \mathcal{E}_{(P_{1,\kappa}|P_{2,\kappa}),\ell_{1,\kappa}+\ell_{2,\kappa}}\}_{\kappa\in\mathbf{N}}. \quad (3)$$

In the present work we only need a restricted variant of this assumption, where F is the family of all m^*_κ gate circuits from Assumption 1. Still, the more general statement appears to more closely match the intuition for the nature of the hardness.

Finally, we combine Assumptions 1 and 3 to one:

Assumption 4 (Split Circuit Pseudorandomness (SCP):). There exist n^*_κ, $m^*_\kappa \in \text{poly}(\kappa)$ that satisfy Assumption 1, as well as $m^\#_\kappa \in poly(\kappa)$ that satisfies Assumption 3 with respect to the SPRP in Assumption 1.

We also consider a somewhat stronger variant of the SCP assumption, where $m^*_\kappa = m^\#_\kappa$. To see why this variant is stronger, consider again the case of comparing a random n_κ-wire, $m^\#_\kappa$ gate identity circuit $R \xleftarrow{R} \mathcal{E}_{I_{n_\kappa} m^\#_\kappa}$ to the split version $C'|C''^\dagger : C \xleftarrow{R} \mathcal{C}_{n_\kappa, m^*_\kappa}; C', C'' \xleftarrow{R} \mathcal{E}_{C, m^\#_\kappa}$, and recall that, when $m^*_\kappa = m^\#_\kappa$, the split version tends to be skewed towards circuits C whose computational complexity is higher than that of R_1, the m^*_κ-gate prefix of R, or in other words $|\mathcal{E}_{C,m^*_\kappa}| < |\mathcal{E}_{R_1,m^*_\kappa}|$. (See the exposition in Footnote 8.) This also means that C'' is likely to be "more similar" to C' than R_2 to R_1, making distinguishing R from $C'|C''^\dagger$ potentially easier than distinguishing R_1 from C' alone. When $m^\#_\kappa$ grows relative to m^*_κ, this discrepancy tapers off and $\mathrm{CC}(C)$ (which is at most m^*_κ) eventually drops below $\mathrm{CC}(R_1)$ (which keep growing with $m^\#_\kappa$). Furthermore, the discrepancy between $|\mathcal{E}_{C,m^*_\kappa}|$ and $|\mathcal{E}_{R_1,m^*_\kappa}|$ is prominent only when $m^*_\kappa < b$. When $m^*_\kappa \gg b$, e.g. $m^*_\kappa = \Omega(n^4)$, we have that $|\mathcal{E}_{C,m^*_\kappa}|$ is sufficiently large so as to make the discrepancy moot.[9]

On the other hand, we note that this somewhat stronger assumption enables demonstrating that a weaker variant of RIO obfuscation suffices for obtaining full-fledged obfuscation for all circuits.

Assumption 5 (Strong Split Circuit Pseudorandomness (SSCP):). Assumption 4 holds with $m^*_\kappa = m^\#_\kappa$.

4 Notions of Obfuscation for Reversible Circuits

A (randomized) transformation $O : \mathcal{C}_n \to \mathcal{C}_n$ on reversible circuits has **stretch** σ if for any $C \in \mathcal{C}_{n,m}$ we have $O(C) \in \mathcal{C}_{n,m+\sigma(n,m)}$. O is said to be **functionality preserving** on a set $\mathbb{C}$ of circuits if $\mathcal{P}_{O(C)} = \mathcal{P}_C$ for any $C \in \mathbb{C}$. An **obfuscator** $\mathcal{O} = \{O_\kappa\}_{\kappa\in\mathcal{N}}$ for $\mathbb{C} = \{\mathbb{C}_\kappa\}_{\kappa\in\mathbf{N}}$ is an ensemble of transformations on reversible circuits where O_κ is functionality preserving on $\mathbb{C}_\kappa$. We start by recalling the standard definition of Indistinguishability obfuscation (IO):

Definition 2 (Indistingiushability Obfuscation (IO):). *An obfuscator $\mathcal{O} = \{O_\kappa\}_{\kappa\in\mathcal{N}}$ is an* **indistingusihability obfuscator (IO)** *for $\mathbb{C} = \{\mathbb{C}_\kappa\}_{\kappa\in\mathbf{N}}$ if for any ensemble of pairs of circuits $\{C_{0,\kappa}, C_{1,\kappa}\}_{\kappa\in\mathbf{N}}$ that are equal size (i.e., $C_{0,\kappa}, C_{1,\kappa} \in \mathbb{C}_\kappa \cap \mathcal{C}_{n_\kappa,m_\kappa}$ for some n_κ, m_κ) and functionally equivalent (i.e. $\mathcal{P}_{C_{0,\kappa}} = \mathcal{P}_{C_{1,\kappa}}$ for all κ), we have*

$$\{\mathcal{O}_\kappa(C_{0,\kappa})\}_{\kappa\in\mathbf{N}} \overset{c}{\approx} \{\mathcal{O}_\kappa(C_{1,\kappa})\}_{\kappa\in\mathbf{N}}.$$

An alternative and equivalent formulation of this definition requires that $\mathcal{O}_\kappa(C) \approx_c R_C$ for any circuit $C \in \mathbb{C}_\kappa$, where R_C is a circuit drawn from some (not necessarily efficiently computable) **reference distribution** that depends only on $\mathcal{P}_C$ and the size of C:

[9] Observe that the computational complexity of a random n=wire, m=gate circuit C is at most $\tilde{\Theta}(m/n^2)$. Indeed, it is easy to verify that a each gate γ_i cancels out with an earlier identical gate $\gamma_j = \gamma_i$ for some $j < i$ with probability $\Theta(n^{-2})$. By Claim 3.1, this means that if $C \xleftarrow{R} \mathcal{C}_{n,n^4}$ then $|\mathcal{E}_{C,n^4}| > b^{n^2}$.

Definition 3 (IO - alternative formulation:). *An obfuscator* $\mathcal{O} = \{O_\kappa\}_{\kappa\in\mathcal{N}}$ *is an* **indistingusihability obfuscator (IO)** *for* $\mathbb{C} = \{\mathbb{C}_\kappa\}_{\kappa\in\mathbf{N}}$ *if there exists a (not necessarily polytime) sampling algorithm* $\mathcal{D}$ *such that for any ensemble* $C = \{C_\kappa\}_{\kappa\in\mathbf{N}}$ *of circuits such that* $C_\kappa \in \mathbb{C}_\kappa \cap \mathcal{C}_{n_\kappa,m_\kappa}$ *we have:*

$$\{O_\kappa(C_\kappa)\}_{\kappa\in\mathbf{N}} \overset{c}{\approx} \{R : R \overset{R}{\leftarrow} \mathcal{D}(\kappa, m_\kappa, \mathcal{P}_{C_\kappa})\}_{\kappa\in\mathbf{N}}.$$

Claim. An obfuscator satisfies Definition 3 for an ensemble $\mathbb{C}$ of circuits iff it satisfies Definition 2 for $\mathbb{C}$. □

This alternative formulation provides a stepping stone towards presenting two new notions of obfuscation that will be key to our construction and analysis: Random Output (RO) and Random Input (RI) obfuscation.

4.1 Random Output Obfuscators

In the rest of this work we will be mostly interested in obfuscators where the output distribution $\mathcal{D}$ is of a particular form. Ideally, we would have liked to require that the distribution $\mathcal{D}(n, m, P)$ be the uniform distribution over $\mathcal{E}_{P,m'}$ for some $m' \geq m$. However, this may be over-restrictive, as it may set an unnecessarily high bar for obfuscation schemes. (For instance, the local random perturbations technique of Sect. 6 may well be a secure obfuscation scheme for random circuits even if it outputs circuits that are distinguishable from random ones.) We thus settle for the following relaxation: We allow distributions $\mathcal{D}(\kappa, m, P)$ where the output circuit is the result of applying some polytime post-processing algorithm to a circuit $\widehat{C}$ drawn uniformly from $\mathcal{E}_{P,m'}$ for some $m' \geq m$. Indeed, on the one hand this relaxation allows obfuscation mechanisms that fall short of generating circuits that match a specific distribution, and on the other hand it still guarantees that $\pi(\widehat{C})$ is independent of the original circuit C, other than having access to $P = \mathcal{P}(C)$.

In some cases we will make the additional requirement that the post-processing algorithm be applied separately to different segments of $\widehat{C}$, e.g. $\pi = (\pi_1, \pi_2)$ where $\pi(\widehat{C}_1|\widehat{C}_2) = \pi_1(\widehat{C}_1)|\pi_2(\widehat{C}_2)$. This additional requirement will be used, together with Assumption 4, to argue that that a certain segment of an obfuscated circuit is "computationally independent" even from the overall functionality of the circuit.

Definition 4 (Random Output Obfuscators). *An IO obfuscator* $\mathcal{O} = \{O_\kappa\}_{\kappa\in\mathcal{N}}$ *for* $\mathbb{C} = \{\mathbb{C}_\kappa\}_{\kappa\in\mathbf{N}}$ *is a* **Random Output Indistinguishability (RO)** *obfuscator with inner-stretch function* $\xi : \mathbf{N}^3 \to \mathbf{N}$ *and post-processing algorithm* $\pi : \mathcal{C}_{n_\kappa} \to \mathcal{C}_{n_\kappa}$ *if for any ensemble* $\{C_\kappa\}_{\kappa\in\mathbf{N}}$ *of circuits where* $C_\kappa \in \mathbb{C}_\kappa \cap \mathcal{C}_{n_\kappa}$ *we have:*[10]

$$\{O_\kappa(C_\kappa)\}_{\kappa\in\mathbf{N}} \overset{c}{\approx} \{\pi(\widehat{C}) : \widehat{C} \overset{R}{\leftarrow} \mathcal{E}_{C_\kappa,\xi(\kappa,n_\kappa,|C_\kappa|)}\}_{\kappa\in\mathbf{N}}.$$

[10] Note that the overall stretch of $\mathcal{O}$ is the composition of the inner-stretch function ξ and the stretch of the post-processing algorithm π. That is, if $\pi : \mathcal{C}_{n_\kappa,m'_\kappa} \to \mathcal{C}_{n_\kappa,\tau(\kappa,n,m'_\kappa)}$ then the stretch of $\mathcal{O}$ is $\sigma(\kappa, n_\kappa, m_\kappa) = \tau(\kappa, n_\kappa, \xi(\kappa, n_\kappa, m_\kappa))$.

The inner-stretch function ξ captures the "effective stretch" of the obfuscator. That is, if $\mathcal{O}$ has inner stretch ξ and $\tilde{C} = O_\kappa(C)$, where $C \in \mathcal{C}_{n,m}$, then $\tilde{C}$ provides "effectively the same obfuscation guarantees" as would a random circuit in $\mathcal{E}_{C,\xi(\kappa,n,m)}$. This is so in spite of the fact that $\mathcal{O}$ might have longer stretch, and $\tilde{C}$ might not look random at all. In particular, note that RO obfuscation where $\xi(\kappa, n_\kappa, m_\kappa) - m_\kappa = \Omega(\kappa)$ provides a meaningful security guarantee (and may be challenging to obtain) even when the input circuit is the only circuit with the same functionality and length in the class $\mathbb{C}$. (In particular recall that, by Claim 3.1, for each $C_\kappa \in \mathbb{C}_\kappa \cap \mathcal{C}_{n_\kappa,m_\kappa}$ we have that the size of $\mathcal{C}_{C_\kappa,\xi(\kappa,n_\kappa,m_\kappa)}$ is exponential in κ.) In contrast, plain IO is meaningless in such cases.

Separable RO obfuscators. The following variant of RO obfuscators will be useful for our soldering-based construction. An RO obfuscator $\mathcal{O}$ is called m_κ**-left-separable** if:

1. The computational complexity of the m_κ-gate prefix of obfuscated circuits is not too high: for any C, $\mathrm{CC}((O_\kappa(C))_{[1,m_\kappa]}) \leq m_\kappa/2$.
2. The post-processing algorithm is of the form $\pi = (\pi_1, \pi_2)$ where $\pi(C) = \pi_1(C_{[1,m_\kappa]})|\pi_2(C_{[m_\kappa,*]})$.

Right-separable obfuscators are defined analogously. (Formally, obfuscator $\mathcal{O}$ is m_κ**-right-separable** if $\mathcal{O}^\dagger$ is left-separable, where $f^\dagger(C) = (f(C^\dagger))^\dagger$ for a function $f : \mathcal{C} \to \mathcal{C}$.) An m_κ**-separable** obfuscator is both m_κ-left-separable and m_κ-right-separable.

Observe that if $\mathcal{O} = \{O_\kappa\}_{\kappa\in\mathcal{N}}$ is an m_κ-left-separable RO obfuscator then $\mathcal{O}^\dagger = \{O^\dagger_\kappa\}_{\kappa\in\mathcal{N}}$ is an m'_κ-right-separable RO obfuscator (and vice versa).

4.2 Random Input and Output Obfuscators

Here we consider obfuscators (namely, functionality preserving transformations on circuits) where security is required only with respect to circuits drawn from a specific distribution. Furthermore, in contrast with IO where security must hold against an observer who sees both the plaintext circuit and the obfuscated circuit, here the observer sees only one or more obfuscated circuits, plus some limited auxiliary information on the plaintext circuit. More specifically, we consider two alternative (and incomparable) security requirements, made with respect to a circuit C chosen from some base distribution $\mathcal{R}_\kappa$ over $\mathcal{C}_{n_\kappa,2m_\kappa}$, and an output distribution $\mathcal{D}(\kappa, 2m_\kappa, \mathcal{P}_C)$:

1. Two obfuscated versions C should not look "too much alike" compared to two independent draws from the underlying distribution $\mathcal{D}(\kappa, 2m_\kappa, \mathcal{P}_C)$. In other words, the observer should not be able to distinguish between two obfuscated versions of C and two draws from $\mathcal{D}(\kappa, 2m_\kappa, \mathcal{P}_C)$.
2. An obfuscated version of C should hide the "midway functionality" of C, namely the permutation computed by the first m_κ-gate block of of C. More specifically, the observer should not be able to distinguish between an obfuscated version of C and a circuit drawn from $\mathcal{D}(\kappa, 2m_\kappa, \mathcal{P}_C)$, even when given

circuits $\widehat{C}_1, \widehat{C}_2$ computed as follows. Let Z_1, Z_2 be two fixed circuits (which are tantamount to an "auxiliary input"), let $C = C_1|C_2$ where $|C_1| = m_\kappa$, and let $\widehat{C}_1 \stackrel{R}{\leftarrow} \mathcal{E}_{(Z_1|C_{[1,m]}),\lambda_\kappa|Z_1|)}$, $\widehat{C}_2 \stackrel{R}{\leftarrow} \mathcal{E}_{(C_{[m,*]}|Z_2),\lambda_\kappa|Z_2|}$ and sufficiently large "leeway" λ_κ. (That is, $\widehat{C}_1$ is a sufficiently long random circuit that's functionally equivalent to $Z_1|C_{[1,m]}$. Similarly, $\widehat{C}_2$ is a sufficiently long random circuit that's functionally equivalent to $C_{[m,*]}|Z_2$.) The rationale here is that $\widehat{C}_1$ and $\widehat{C}_2$ essentially give the observer only the ability to evaluate $Z_1|C_{[1,m]}$ and $C_{[m,*]}|Z_2$ (and their inverses) on inputs of its choice.

More formally:

Definition 5 (Random Input (RI) Obfuscators). *$\mathcal{O} = \{O_\kappa\}_{\kappa\in\mathcal{N}}$ is* **Random Input (RI)** *obfuscator for $n_\kappa, 2m_\kappa$, input distribution ensemble $\mathcal{R} = \{\mathcal{R}_\kappa\}_{\kappa\in\mathbf{N}}$ where $\mathcal{R}_\kappa \subseteq \mathcal{C}_{n_\kappa,2m_\kappa}$, and output distribution $\mathcal{D}$, if:*

I.

$$\left\{ \begin{array}{l} (C_1, C_2) : C \stackrel{R}{\leftarrow} \mathcal{R}_{n_\kappa,2m_\kappa}; \\ C_1, C_2 \stackrel{R}{\leftarrow} \mathcal{O}_\kappa(C) \end{array} \right\}_{\kappa\in\mathbf{N}} \stackrel{c}{\approx} \left\{ \begin{array}{l} (\widehat{C}_1, \widehat{C}_2) : C \stackrel{R}{\leftarrow} \mathcal{R}_\kappa; \\ \widehat{C}_1, \widehat{C}_2 \stackrel{R}{\leftarrow} \mathcal{D}(\kappa, n_\kappa, 2m_\kappa, \mathcal{P}_C) \end{array} \right\}_{\kappa\in\mathbf{N}}.$$

II. There exists a leeway function $\lambda_\kappa \in poly(\kappa)$ such that for any two circuit ensembles $\mathbf{Z_1} = \{\mathbf{Z_{1,\kappa}}\}_{\kappa\in\mathbf{N}}, \mathbf{Z_2} = \{\mathbf{Z_{2,\kappa}}\}_{\kappa\in\mathbf{N}}$ with $Z_i \in \mathcal{C}_{n_\kappa,m_{i,\kappa}}$ for some $m_{i,\kappa}$, $i = 1, 2$, and any $\lambda \geq \lambda_\kappa$ we have:

$$\left\{ \begin{array}{l} (O_\kappa(C), \widehat{C}_1, \widehat{C}_2) : \\ C \stackrel{R}{\leftarrow} \mathcal{R}_\kappa; \\ \widehat{C}_1 \stackrel{R}{\leftarrow} \mathcal{E}_{\mathcal{P}_{(C_{[1,m_\kappa]}|Z_{1,\kappa})},m_{1,\kappa}\lambda}; \\ \widehat{C}_2 \stackrel{R}{\leftarrow} \mathcal{E}_{(\mathcal{P}_{Z_{2,\kappa}|C_{[m_\kappa,*]}}),m_{2,\kappa}\lambda} \end{array} \right\}_{\kappa\in\mathbf{N}} \stackrel{c}{\approx} \left\{ \begin{array}{l} (\widehat{C}, \widehat{C}_1, \widehat{C}_2) : \\ C \stackrel{R}{\leftarrow} \mathcal{R}_\kappa; \widehat{C} \stackrel{R}{\leftarrow} \mathcal{D}(\kappa, n_\kappa, 2m_\kappa, \mathcal{P}_C); \\ \widehat{C}_1 \stackrel{R}{\leftarrow} \mathcal{E}_{\mathcal{P}_{(C_{[1,m_\kappa]}|Z_{1,\kappa})},m_{1,\kappa}\lambda}; \\ \widehat{C}_2 \stackrel{R}{\leftarrow} \mathcal{E}_{(\mathcal{P}_{Z_{2,\kappa}|C_{[m_\kappa,*]}}),m_{2,\kappa}\lambda} \end{array} \right\}_{\kappa\in\mathbf{N}}.$$

Definition 6 (Random Input & Output (RIO) Obfuscators). *An RI obfuscator $\mathcal{O} = \{O_\kappa\}_{\kappa\in\mathcal{N}}$ for n_κ, m_κ is* **Random Input Output (RIO)** *with inner-stretch function $\xi : \mathbf{N^3} \to \mathbf{N}$ and post-processing algorithm $\pi : \mathcal{C}_{n_\kappa} \to \mathcal{C}_{n_\kappa}$ if its output distribution $\mathcal{D}$ is of the form $\mathcal{D}(\kappa, n_\kappa, m_\kappa, P) = \pi(C)$ for $C \stackrel{R}{\leftarrow} \mathcal{E}_{P,\xi(\kappa,n_\kappa,m_\kappa)}$.*

Requirements (I) and (II) appear to be incomparable. Furthermore, each use of RIO obfuscators within our construction needs only one of the two requirements, with respect to a specific input distribution. This means that in principle one could have two different constructions of RIO obfuscation, where each construction is geared towards realizing only one of the two requirements. Still, the rationale for the validity of the obfuscation algorithm described in Sect. 6 applies in the same way to both properties (see discussion there).

5 From RIO Obfuscation to RO for All Circuits

This section presents the construction of RO obfuscators for all circuits from RIO obfuscators. More specifically, Let $n^*_\kappa, m^*_\kappa, m^\#_\kappa$ be length functions that satisfy Assumption 4. Our starting point is two obfuscators, O_1 and O_2, such that:

- O_1 is an RIO obfuscator that satisfies property I with respect to the uniform input distribution $C \overset{\text{R}}{\leftarrow} \mathcal{C}_{n^*_\kappa, m^*_\kappa}$, with inner-stretch $\xi(\kappa, n^*_\kappa, m^*_\kappa) = m^\#_\kappa$ and with post-processing algorithm π.
- O_2 is an an RIO obfuscator that satisfies property II with respect to the input distribution $C = \pi(C')|\pi^\dagger(C'') : C', C'' \overset{\text{R}}{\leftarrow} \mathcal{C}_{n^*_\kappa, m^*_\kappa}$ and leeway $\lambda_\kappa \leq m^\#_\kappa$.

That is, we show:

Theorem 2. *Let $n^*_\kappa, m^*_\kappa, m^\#_\kappa$ be length functions that satisfy Assumption 4. If there exist algorithms O_1, π, O_2 such that:*

- *O_1 satisfies property I of RIO obfuscation for input distribution ensemble $\{C : C \overset{R}{\leftarrow} \mathcal{C}_{n^*_\kappa, m^*_\kappa}\}_{\kappa \in \mathbf{N}}$, with inner-stretch $\xi(\kappa, n^*_\kappa, m^*_\kappa) = m_\kappa \geq m^\#_\kappa$ and post-processing algorithm π,*
- *O_2 is an an RIO obfuscator that satisfies property II with respect to the input distribution $C = \pi(C')|\pi^\dagger(C'') : C', C'' \overset{R}{\leftarrow} \mathcal{C}_{n^*_\kappa, m^*_\kappa}$ and leeway $\lambda_\kappa \leq m^\#_\kappa$.*

then there exists an RO obfuscator $\mathcal{O}$ for all reversible circuits. Furthermore, the inner-stretch of $\mathcal{O}$ for m-gate circuits is $\Omega(m^\#_\kappa m)$.

An overview of the obfuscation algorithm appears in the Introduction. We present the construction and its analysis in four steps. First, we show how to construct RO obfuscators for the identity function, with some specific parameters. (We call such obfuscators *pseudrandom identity generators.*)

Next we use random identity generators to construct RO obfuscators for single gate circuits.

Nest we show how to use RIO obfuscators with the above parameters to combine, or "solder" obfuscated circuits to obtain obfuscated versions of the concatenation of these circuits.

Next we combine the last two steps to construct full-fledged RO obfuscation for all reversible circuits.

The Appendix of [CCMR24] demonstrates how indistinguishability obfuscator for all Boolean circuits can be obtained using an indistinguishability obfuscator for all reversible circuits.

5.1 Random Identity Generators

Random identity generators (RIGs) are separable RO obfuscators for the identity permutation with specific parameters: Let I_{n_κ} denote the identity permutation on n_κ wires. An (n_κ, m_κ)-RIG is an m_κ-separable RO obfuscator for I_{n_κ} with inner-stretch $\xi(\kappa, n_\kappa, 1) \geq 2m_\kappa$.

In other words, an RIG is a sampling algorithm that, given κ, generates circuits that are indistinguishable from $\pi(C)$, where C is a random circuit with n_κ wires and $2m_\kappa$ gates that computes the identity permutation, and π is a post-processing algorithm. Furthermore, π is of the form $\pi = (\pi_1, \pi_2)$ where π_1 is applied to $C_{[1,m_\kappa]}$ and π_2 is applied to $C_{[m_\kappa,*]}$, and the computational complexities of both $C_{[1,m_\kappa]}$ and $C_{[m_\kappa,*]}$ are less than $m_\kappa/2$,

Definition 7 (Random Identity Generators). *An algorithm $\{G_k\}_{\kappa\in\mathcal{N}}$ is an (n_κ, m_κ)-**RIG** if it is an m_κ-separable RO obfuscator for $\{I_{n_\kappa}\}_{\kappa\in\mathbf{N}}$, with inner-stretch $\xi(\kappa, n_\kappa, 1) \geq 2m_\kappa$.*

Let $n^*_\kappa, m^*_\kappa, m^\#_\kappa$ be length functions that satisfy Assumption 4. We construct an $(n^*_\kappa, 2m_\kappa)$-RIG G_κ given an obfuscator O that satisfies property I of RIO obfuscation (see Definition 5) for uniformly chosen inputs in $\mathcal{C}_{n^*_\kappa,m^*_\kappa}$, with inner-stretch ξ such that $\xi(\kappa, n^*_\kappa, m^*_\kappa) = m_\kappa$ where $m_\kappa \geq m^\#_\kappa$. The construction is straightforward:

1. Sample $C \xleftarrow{R} \mathcal{C}_{n^*_\kappa,m^*_\kappa}$
2. Sample $C', C'' \xleftarrow{R} O_\kappa(C)$
3. Output $C'|C''^\dagger$.

We show:

Claim. Let $n^*_\kappa, m^*_\kappa, m^\#_\kappa$ be length functions that satisfy Assumption 4, and let $O = \{O_\kappa\}_{\kappa\in\mathcal{N}}$ satisfy property I of RIO obfuscation for input distribution $\mathcal{R}_\kappa = \mathcal{C}_{n^*_\kappa,m^*_\kappa}$, and with inner-stretch $\xi(\kappa, n^*_\kappa, m^*_\kappa) = m_\kappa$ where $m_\kappa \geq m^\#_\kappa$. Then $G = \{G_k\}_{\kappa\in\mathcal{N}}$ described above is an (n^*_κ, m_κ)-RIG.

Proof. We show that G_κ is an m_κ-separable RO obfuscator for the identity function $\{I_{n^*_\kappa}\}_{\kappa\in\mathbf{N}}$, with inner-stretch $\xi(\kappa, n^*_\kappa, 1) = 2m_\kappa$, and with post-processing algorithm $\pi' = (\pi, \pi^\dagger)$. That is, we show:

$$\{C : C \xleftarrow{R} G_\kappa\}_{\kappa\in\mathbf{N}} = \{C'|C''^\dagger : C \xleftarrow{R} \mathcal{C}_{n^*_\kappa,m^*_\kappa}; C', C'' \xleftarrow{R} O_\kappa(C),\}_{\kappa\in\mathcal{N}} \overset{c}{\approx} \quad (6)$$

$$\{\pi(C')|\pi(C'')^\dagger : C \xleftarrow{R} \mathcal{C}_{n^*_\kappa,m^*_\kappa}; C', C'' \xleftarrow{R} \mathcal{E}_{C,m_\kappa}\}_{\kappa\in\mathcal{N}} \overset{c}{\approx} \quad (7)$$

$$\{\pi(\hat{I}_{[1,m_\kappa]})|(\pi(\hat{I}_{[m_\kappa,*]}))^\dagger : \hat{I} \xleftarrow{R} \mathcal{E}_{I_{n^*_\kappa},2m_\kappa}\}_{\kappa\in\mathcal{N}}. \quad (8)$$

Indistinguishability of experiment (6) and experiment (7) follows directly from the RIO security of O (property I). Indistinguishability of experiment (7) and experiment (8) follows from Assumption 4. Indeed, by Assumption 1, $\{C : C \xleftarrow{R} \mathcal{C}_{n^*_\kappa,m^*_\kappa}\}_{\kappa\in\mathbf{N}}$ is an SPRP. Since $|C'| = |C| = m_\kappa \geq m^\#_\kappa$, we can use Assumption 3 to conclude that:

$$\{C'|C'' : C \xleftarrow{R} \mathcal{C}_{n^*_\kappa,m^*_\kappa}; C', C'' \xleftarrow{R} \mathcal{E}_{C,m_\kappa}\}_{\kappa\in\mathcal{N}} \overset{c}{\approx} \{\widehat{J}_{[1,m_\kappa]}|(\widehat{J}_{[m_\kappa,*]})^\dagger : \widehat{J} \xleftarrow{R} \mathcal{E}_{I_{n^*_\kappa},2m_\kappa}\}_{\kappa\in\mathcal{N}}. \quad (9)$$

Now, an algorithm A_κ that distinguishes between experiments (7) and (8) can be used to distinguish between the two distributions in (9): Given a circuit

$C \in \mathcal{C}_{n,2m_\kappa}$, output $A_\kappa(\pi(C_{[1,m_\kappa]})|\pi^\dagger(C_{[m_\kappa,*]}))$. Observe that if C was drawn from the l.h.s. distribution in (9) then A_κ's input is drawn from (7) and if C was drawn from the r.h.s. distribution then A_κ's input is drawn from (8). The claim follows by transitivity of computational indistinguishability, along with verifying that G_κ is indeed both m_κ-right-separable and m_κ-left-separable.

Directly Generating Random Identities? We note that there may well be other ways to construct RIGs, other than using an RIO obfuscator that satisfies property I. Indeed, functionality-reserving obfuscation may not be needed at all; instead one might opt to "jointly generate" two circuits that are functionally equivalent and look sufficiently random otherwise. In fact we are not aware of any "barrier" to having statistically secure RIGs.

5.2 RO Obfuscation of Single Gates

Next we show how to use a random identity generator G to construct RO obfuscators of single gates, namely RO obfuscators $\mathrm{GO} = \{\mathrm{GO}_\kappa\}_{\kappa\in\mathbf{N}}$ for the set $\mathbb{C} = \{\mathbb{B}_\kappa\}_{\kappa\in\mathbf{N}}$, where $\mathbb{B}_\kappa$ is the set of base permutations on κ wires. That is, given any base permutation $\beta \in \mathbb{B}_\kappa$, algorithm $\mathrm{GO}_\kappa(\beta)$ samples circuits that are indistinguishable from $\pi(C)$ for a random circuit $C \xleftarrow{R} \mathcal{E}_{\beta,m_\kappa}$ for some $m_\kappa \in \mathrm{poly}(\kappa)$ and post-processing algorithm π with length and separability requirements that are similar to those of random identity generators (RIGs): GO_κ should have inner-stretch ξ where $\xi(\kappa,\kappa,m_\kappa) = 2m_\kappa$ with $m_\kappa \geq m^*_\kappa$; furthermore, it should be m_κ-separable.

Definition 8 (Gate Obfuscators.). *An algorithm* $\mathrm{GO} = \{\mathrm{GO}_k\}_{\kappa\in\mathcal{N}}$ *is an* m_κ**-gate obfuscator** *if, for any* $\beta \in \mathbb{B}_\kappa\}$*, we have that* $\mathrm{GO}_\kappa(\beta_\kappa)$ *is an* m_κ*-separable RO obfuscator for* β_κ*, with inner-stretch* $\xi(\kappa,\kappa,1) \geq 2m_\kappa$*.*

The construction is simple: $\mathrm{GO}_\kappa(\beta)$ keeps sampling identity circuits using G_κ until the first gate in the generated circuit is β. Once this happens, GO replaces that first gate with the identity gate β_I and outputs the resulting circuit. Note that in order for $\mathrm{GO}_\kappa(\beta)$ to terminate in polynomial time we need to further assume that the circuits generated by G_κ start with β with polynomial probability. The random identity generators constructed in this work satisfy this property unconditionally.

Claim. Let $\{G_\kappa\}_{\kappa\in\mathbf{N}}$ be an (κ, m_κ)-random identity generator such that $\mathrm{Prob}[C_{[1,1]} = \beta : C \xleftarrow{R} G_k] \in poly(\kappa)$ for all $\beta \in \mathbb{B}_\kappa$. Then GO is an m_κ-gate-obfuscator.

Proof. To see that $\mathrm{GO}_\kappa(\beta)$ is an m_κ-separable RO obfuscator for β, let $\pi = (\pi_1, \pi_2)$ be the post-processing algorithm guaranteed by Definition 7, such that

$$\{C \xleftarrow{R} G_\kappa\}_{\kappa\in\mathbf{N}} \overset{c}{\approx} \{\pi_1(C_{[1,m_\kappa]})|\pi_2(C_{[m_\kappa,*]}) : C \xleftarrow{R} \mathcal{E}_{I_\kappa,2m_\kappa}\}_{\kappa\in\mathbf{N}}. \tag{10}$$

Consider the post-processing algorithm $\pi = (\pi'_1, \pi_2)$ where $\pi'_1(C) = \beta_I | \pi_1(\beta | C)$. We argue that

$$\{\beta_I | C_{[1,*]} : C \overset{R}{\leftarrow} G_\kappa \text{ s.t. } C_{[1,1]} = \beta\}_{\kappa \in \mathbf{N}} \overset{c}{\approx} \tag{11}$$
$$\{\pi'_1(C_{[1,m_\kappa]}) | \pi_2(C_{[m_\kappa,*]}) : C \overset{R}{\leftarrow} \mathcal{E}_{\beta,2m_\kappa} \text{s.t. } C_{[1,1]} = \beta\}_{\kappa \in \mathbf{N}}.$$

Indeed, an algorithm A_κ that distinguishes between the two distributions in (11) can be used to distinguish between the two distributions in (10): given a circuit C, if $C_{[1,1]} = \beta$, output $A_\kappa(\beta_I | C_{[1,*]})$; else, output a random bit. Observe that if C was drawn from the l.h.s. distribution in (10) then, whenever $C_{[1,1]} = \beta$, we have that $C_{[1,*]}$ is drawn from the l.h.s. distribution in (11). If C was drawn from the r.h.s. distribution in (10) then, whenever $C_{[1,1]} = \beta$, we have that $C_{[1,*]}$ is drawn from the r.h.s. distribution in (11).

We note that both the efficiency and security of GO can be significantly improved with little effort: Once the first base permutation $\beta' = (w'_1, w'_2, w'_3, \phi)$ in the sampled circuit has the same control function ϕ as the given $\beta = (w_1, w_2, w_3, \phi)$, can remove β' and then"rotate" the remaining circuit so that the wires w'_1, w'_2, w'_3 will become w_1, w_2, w_3. That is, if the sampled circuit is of the form $\beta' | C$ then output the circuit C' that is the result of renaming the wires in C via the permutation $\sigma = (w_1, w'_1)(w_2, w'_2)(w_3, w'_3)$ on $[n]$. This way, the random identity generator needs to be run at most 16 times in expectation (assuming that the control function of the first gate is distributed uniformly). The expected number of samples needed can be further reduced (for "nice" post-processing functions) by noting that any circular shift of an identity circuit is an identity circuit.

5.3 Soldering Obfuscated Circuits

Next we show how to combine (or, "solder") obfuscated circuits to obtain obfuscated versions of the concatenation of these circuits. Specifically, let $n^*_\kappa, m^*_\kappa, m^\#_\kappa$ satisfy Assumption 4 and let $\mathbb{C}_1 = \{\mathbb{C}_{1,\kappa}\}_{\kappa \in \mathbf{N}}, \mathbb{C}_2 = \{\mathbb{C}_{2,\kappa}\}_{\kappa \in \mathbf{N}}$ be ensembles of sets of circuits such that $C_{i,k} \in \mathcal{C}_{n^*_\kappa, m_{i,\kappa}}$ for $i = 1, 2$. Consider the following building blocks, with respect to some $m_\kappa \geq \max(m^*_\kappa, m^\#_\kappa)$:

- an m_κ-right-separable RO obfuscator RO_1 for ensemble $\mathbb{C}_1$, with post-processing algorithm $\pi_1 = (\pi_{1,1}, \pi_{1,2})$ and inner-stretch ξ_1 such that $\xi_1(\kappa, n^*_\kappa, m) \geq m_\kappa m$,
- an m_κ-left-separable RO obfuscator RO_2 for ensemble $\mathbb{C}_2$, with post-processing algorithm $\pi_2 = (\pi_{2,1}, \pi_{2,2})$ and inner-stretch ξ_2 such that $\xi_2(\kappa, n^*_\kappa, m) \geq m_\kappa m$,
- an RIO obfuscator O that satisfies Property II with leeway function $\lambda_\kappa \leq m_\kappa$, for auxiliary circuits $C^\dagger_{1,\kappa}, C^\dagger_{2,\kappa}$, and for input distribution ensemble:

$$\{(\pi_{1,2}(\widehat{C}_{1,2}) | \pi_{2,1}(\widehat{C}_{2,1})) : \tag{12}$$
$$C_{1,2}, C_{2,1} \overset{R}{\leftarrow} \mathcal{C}_{n^*_\kappa, m^*_\kappa}; \widehat{C}_{1,2} \overset{R}{\leftarrow} \mathcal{E}_{C_{1,2}, m_\kappa}; \widehat{C}_{2,1} \overset{R}{\leftarrow} \mathcal{E}_{C_{2,1}, m_\kappa}\}_{\kappa \in \mathbf{N}}. \tag{13}$$

We use RO_1, RO_2, O to construct an RO obfuscator $\mathrm{RO}_{1|2}$ for the ensemble $\mathbb{C} = \{\mathbb{C}_\kappa\}_{\kappa\in\mathbf{N}}$, where each circuit $C \in \mathbb{C}_\kappa$ is of the form $C = C_1|C_2$ where $C_1 \in \mathbb{C}_{1,\kappa}$ and $C_2 \in \mathbb{C}_{2,\kappa}$. Given a circuit $C = C_1|C_2 \in \mathbb{C}_\kappa$, obfuscator $\mathrm{RO}_{1|2,\kappa}$ proceeds as follows (see also Fig. 4):

1. Sample $\tilde{C}_1 \xleftarrow{R} \mathrm{RO}_{1,\kappa}(C_1)$ and $\tilde{C}_2 \xleftarrow{R} \mathrm{RO}_{2,\kappa}(C_2)$.
2. Let $\tau_{i,j}$ denote the stretch of the post-processing algorithm $\pi_{i,j}$, let $t_{i,j} = \tau_{i,j}(\kappa, n^*_\kappa, m_\kappa)$, and let $C_{1,1} = (\tilde{C}_1)_{[1,-t_{1,2}]}, C_{1,2} = (\tilde{C}_1)_{[-t_{1,2},*]}, C_{2,1} = (\tilde{C}_2)_{[1,t_{2,1}]}$, $C_{2,2} = (\tilde{C}_2)_{[t_{2,1},*]}$.
   Sample $G \xleftarrow{R} O_\kappa(C_{1,2}|C_{2,1})$.
3. Output $C_{1,1}|G|C_{2,2}$.

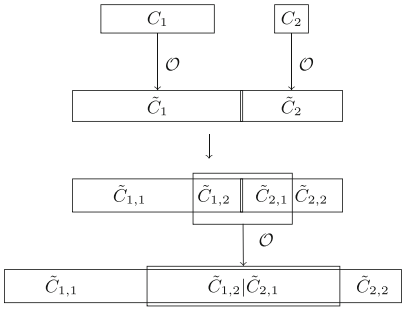


Fig. 4. Soldering RO-obfuscated circuits: The operation of obfuscator $\mathrm{RO}_{1|2}$ given circuits C_1 and C_2.

Claim. Let $n^*_\kappa, m^*_\kappa, m^\#_\kappa$ satisfy Assumption 4, and let $m_\kappa \geq m^\#_\kappa$. For $i = 1, 2$, let $\mathbf{C}_i = \{C_{i,\kappa}\}_{\kappa\in\mathbf{N}}$ be a circuit ensemble where $C_{i,k} \in \mathcal{C}_{n^*_\kappa, m_{i,\kappa}}$, and let $\mathrm{RO}_i = \{\mathrm{RO}_{i,\kappa}\}_{\kappa\in\mathbf{N}}$ be an RO obfuscator for $\mathbf{C}_i$ with inner-stretch ξ_i such that $\xi_i(\kappa, n^*_\kappa, m) = m_\kappa m$ and with post-processing algorithm $\pi_i = (\pi_{i,1}, \pi_{i,2})$; furthermore, RO_1 is m_κ-right-separable and RO_2 is m_κ-left-separable. Let O be an RIO obfuscator functionξ_3 and with post-processing algorithm π_3, for the input distribution ensemble in (12). Then $\mathrm{RO}_{1|2}$ defined above is an RO obfuscator for the circuit ensemble $\{C_{1,\kappa}|C_{2,\kappa}\}_{\kappa\in\mathbf{N}}$, with inner-stretch function $\xi(\kappa, n^*_\kappa, m) = m_\kappa(m-2) + \xi_3(\kappa, n^*_\kappa, 2m_\kappa)$ and post-processing algorithm

$$\pi(C) = \pi_1(C_{[1,\xi_1(\kappa,n^*_\kappa,m_{1,\kappa})-m_\kappa]})|$$
$$\pi_3(C_{[\xi_1(\kappa,n^*_\kappa,m_{1,\kappa})-m_\kappa+1,\xi_3(\kappa,n^*_\kappa,2\,m^*_\kappa)]})|\pi_2(C_{[-(\xi_2(\kappa,n^*_\kappa,m_{2,\kappa})-m_\kappa),*]}).$$

Furthermore, if RO_1 is m_κ-left-separable then so is $\mathrm{RO}_{1|2}$. If RO_2 is m_κ-right-separable then so is $\mathrm{RO}_{1|2}$.

See proof in [CCMR24].

5.4 RO for All Circuits

The RO obfuscator for all circuits combines a single gate obfuscator GO with the soldering process in the natural way. Specifically, consider the append-and-solder obfuscator AS that, to obfuscate an n-wire, m-gate circuit $C = \gamma_1 \dots \gamma_m$ with security parameter κ, proceeds as follows:

1. Let $n^*_\kappa, m^*_\kappa, m^\#_\kappa$ satisfy Assumption 4. Without loss of generality assume that $n = n^*_\kappa$. (If $n < n^*_\kappa$ then embed the circuit in n^*_κ wires. If $n > n^*_\kappa$ then proceed with the smallest $\kappa' > \kappa$ such that $n \leq n^*(\kappa')$.)
2. Let GO be a (n^*_κ, m_κ)-gate obfuscator for $m_\kappa \geq \max(m^*_\kappa, m^\#_\kappa)$. For each gate γ_i, $i = 1 \dots m$, let $\Gamma_i \xleftarrow{R} \mathrm{GO}(\gamma_i)$ be a $2m_\kappa$-gate circuit such that $\mathcal{P}_{\Gamma_i} = \gamma_i$.
3. Solder the circuits $\Gamma_1 \dots \Gamma_m$ one by one, using an RIO obfuscator O for the input distribution ensemble in (12). That is:
   (a) Let $C_1 = \Gamma_1$.
   (b) For $i = 2..m$, let $C_i = (C_{i-1})_{[1,-t_{1,\kappa}]} | O_\kappa((C_{i-1})_{[-t_{1,\kappa},*]} | (\Gamma_i)_{[1,t_{2,\kappa}]}) | (\Gamma_i)_{[t_{2,\kappa},*]}$ be the result of soldering C_{i-1} and Γ_i, where $t_{1,\kappa}$ and $t_{2,\kappa}$ are the lengths of the left and right margins for soldering, namely $\pi_1 : \mathcal{C}_{n^*,m_\kappa} \to \mathcal{C}_{n^*_\kappa,t_{1,\kappa}}$ and $\pi_2 : \mathcal{C}_{n^*,m_\kappa} \to \mathcal{C}_{n^*_\kappa,t_{2,\kappa}}$, where $\pi = (\pi_1, \pi_2)$ is the post-processing algorithm of GO_κ.
4. Output C_m.

It follows from Claim 5.3 that AS is an m_κ-separable RO obfuscator for all reversible circuits, with inner-stretch $\xi(\kappa, n, m) \geq m_\kappa m$. When GO is instantiated via the RIG and RIO described in Sects. 5.2 and 5.1 above, Theorem 2 follows from Claims 5.1 and 5.2.

Furthermore, observe that the stretch of AS grows only linearly in m. Specifically, it follows from Claim 5.3 that $|C_i| = |C_{i-1}| + \sigma_2(\kappa, n^*_\kappa, t_{1,\kappa} + t_{2,\kappa})$, where $\sigma_2(\kappa, n^*_\kappa, m^*_\kappa)$ is the overall stretch of the RIO obfuscator used in the soldering operation. When instantiating the construction with the single-gate obfuscator and random identity generator described in Sects. 5.2 and 5.1, based on an RIO obfuscator with stretch $\sigma_1(\kappa, n^*, m^*_\kappa)$, we obtain $|C_m| \leq m\sigma_2((\kappa, n_\kappa, 2\sigma_1(\kappa, n_\kappa, m^*_\kappa))$, where n^*_κ, m^*_κ are length functions that satisfy Assumption 4.

Finally, straightforward hybrids argument demonstrates that the security level of AS decreases only linearly in the number of gates. That is, to guarantee distinguishing probability of at most ϵ between an obfuscated m-gate circuit C and a circuit drawn from $D_{\mathcal{P}_C,|C|}$, it suffices to use building blocks (RIO and GO obfuscators) with security $\Omega(\epsilon/m)$.

6 Constructing RIO Obfuscators

This section presents a general approach for constructing RIO obfuscators, along with a family of candidate RIO obfuscators that may be a viable basis for RO (and in particular IO) obfuscators for all circuits as in Theorem 2.

This section, as well as the open problems section and the appendix have removed from this version due to page limits. These sections appear in [CCMR24].

Acknowledgements. We thank Luowen Qian for participating in early stages of this research, and the TCC'24 reviewers for their insightful comments. R.C. also thanks Nir Bitansky, Shafi Goldwasser and Omer Paneth for very helpful discussions.

References

[AB15] Applebaum, B., Brakerski, Z.: Obfuscating circuits via composite-order graded encoding. In: Dodis, Y., Nielsen, J.B. (eds.) TCC 2015. LNCS, vol. 9015, pp. 528–556. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46497-7_21

[AJJ14] Alagic, G., Jeffery, S. and Jordan, S.P.: Circuit obfuscation using braids. In: Flammia, S.T., Harrow, A.W. (eds.) 9th Conference on the Theory of Quantum Computation, Communication and Cryptography, TQC 2014, 21-23 May 2014, Singapore, LIPIcs, vol. 27 , pp. 141–160. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2014)

[AJS15] Ananth, P., Jain, A., Sahai, A.: Achieving compactness generically: indistinguishability obfuscation from non-compact functional encryption. IACR Cryptol. ePrint Arch., pp. 730 (2015)

[Bar86] Barrington, D.A.: Bounded-width polynomial-size branching programs recognize exactly those languages in NC^1. In: Hartmanis, J. (ed.) Proceedings of the 18th Annual ACM Symposium on Theory of Computing, 28-30 May 1986, Berkeley, California, USA, pp. 1–5. ACM (1986)

[Bar17] Barak, B.: The complexity of public-key cryptography. In: Tutorials on the Foundations of Cryptography. ISC, pp. 45–77. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-57048-8_2

[Ben73] Bennett, C.H.: Logical reversibility of computation. IBM J. Res. Dev. **17**, 525–532 (1973)

[Ben89] Bennett, C.H.: Time/space trade-offs for reversible computation. SIAM J. Comput. **18**(4), 766–776 (1989)

[BGI+01] Barak, B., et al.: On the (Im)possibility of obfuscating programs. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 1–18. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-44647-8_1

[BGI+12] Boaz et al.: On the (Im)possibility of obfuscating programs. J. ACM, **59**(2), 6:1–6:48 (2012)

[BGK+14] Barak, B., Garg, S., Kalai, Y.T., Paneth, O., Sahai, A.: Protecting obfuscation against algebraic attacks. In: Nguyen, P.Q., Oswald, E. (eds.) EUROCRYPT 2014. LNCS, vol. 8441, pp. 221–238. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-642-55220-5_13

[BLMP23] Ball, M., Liu, Y., Mazor, N., Pass, R.: Kolmogorov comes to cryptomania: on interactive Kolmogorov complexity and key-agreement. In: 2023 IEEE 64th Annual Symposium on Foundations of Computer Science (FOCS), pp. 458–483 (2023)

[BPW16] Bitansky, N., Paneth, O., Wichs, D.: Perfect structure on the edge of chaos. In: Kushilevitz, E., Malkin, T. (eds.) TCC 2016. LNCS, vol. 9562, pp. 474–502. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49096-9_20

[Bro04] Brodsky, A.: Reversible circuit realizations of Boolean functions. In: Levy, J.-J., Mayr, E.W., Mitchell, J.C. (eds.) TCS 2004. IIFIP, vol. 155, pp. 67–80. Springer, Boston, MA (2004). https://doi.org/10.1007/1-4020-8141-3_8

[BV15] Bitansky, N., Vaikuntanathan, V.: Indistinguishability obfuscation from functional encryption. IACR Cryptol. ePrint Arch., pp. 163 (2015)

[CCMR24] Canetti, R., Chamon, C., Mucciolo, E., Ruckenstein, A.: Towards general-purpose program obfuscation via local mixing. Cryptology ePrint Archive, Paper 2024/006 (2024)

[CG75] Coppersmith, D., Grossman, E.: Generators for certain alternating groups with applications to crytography. SIAM J. Appl. Math. **29**(4), 624–627 (1975)

[CGH+15] Coron, J.-S., et al.: Zeroizing without low-level zeroes: new MMAP attacks and their limitations. In: Gennaro, R., Robshaw, M. (eds.) CRYPTO 2015. LNCS, vol. 9215, pp. 247–266. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-47989-6_12

[CHVW19] Chen, Y., Hhan, M., Vaikuntanathan, V., Wee, H.: Matrix PRFs: constructions, attacks, and applications to obfuscation. In: Hofheinz, D., Rosen, A. (eds.) TCC 2019. LNCS, vol. 11891, pp. 55–80. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-36030-6_3

[CJMR22] Chamon, C., Jakes-Schauer, J., Mucciolo, E.R., Ruckenstein, A.E.: Encrypted operator computing: an alternative to fully homomorphic encryption. CoRR, abs/2203.08876 (2022)

[CLTV15] Canetti, R., Lin, H., Tessaro, S., Vaikuntanathan, V.: Obfuscation of probabilistic circuits and applications. In: Dodis, Y., Nielsen, J.B. (eds.) TCC 2015. LNCS, vol. 9015, pp. 468–497. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46497-7_19

[CMR22] Chamon, C., Mucciolo, E.R., Ruckenstein, A.E.: Quantum statistical mechanics of encryption: reaching the speed limit of classical block ciphers. Ann. Phys. **446**, 169086 (2022)

[CRMC23] Chamon, C., Ruckenstein, A.E., Mucciolo, E.R., Canetti, R.: Circuit complexity and functionality: a thermodynamic perspective. arXiv preprint arXiv:2309.05731, 2023

[CV13] Canetti, R., Vaikuntanathan, V.:D Obfuscating branching programs using black-box pseudo-free groups. IACR Cryptol. ePrint Arch., pp. 500 (2013)

[CVW18] Chen, Y., Vaikuntanathan, V., Wee, H.: GGH15 beyond permutation branching programs: proofs, attacks, and candidates. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018. LNCS, vol. 10992, pp. 577–607. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-96881-0_20

[DQV+21] Devadas, L., Quach, W., Vaikuntanathan, V., Wee, H., Wichs, D.: Succinct LWE sampling, random polynomials, and obfuscation. In: Nissim, K., Waters, B. (eds.) TCC 2021. LNCS, vol. 13043, pp. 256–287. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-90453-1_9

[GGH15] Gentry, C., Gorbunov, S., Halevi, S.: Graph-induced multilinear maps from lattices. In: Dodis, Y., Nielsen, J.B. (eds.) TCC 2015. LNCS, vol. 9015, pp. 498–527. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46497-7_20

[GGHR14] Garg, S., Gentry, C., Halevi, S., Raykova, M.: Two-round secure MPC from indistinguishability obfuscation. In: Lindell, Y. (ed.) TCC 2014. LNCS, vol. 8349, pp. 74–94. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-642-54242-8_4

[GGSW13] Garg, S., Gentry, C., Sahai, A., Waters, B.: Witness encryption and its applications. In: Symposium on Theory of Computing Conference, STOC 2013, Palo Alto, CA, USA, 1-4 June 2013, pp. 467–476 (2013)

[GHP24] Gretta, L., He, W., Pelecanos, A.: More efficient k-wise independent permutations from random reversible circuits via log-sobolev inequalitie, manuscript (2024)

[Gow96] Gowers, W.T.: An almost m-wise independed random permutation of the cube. Comb. Probab. Comput. **5**, 119–130 (1996)

[GP21] Gay, R., Pass, R.: Indistinguishability obfuscation from circular security. In: Khuller, S., Williams, V.V. (eds.) STOC 2021: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, 21-25 June 2021, pp. 736–749. ACM (2021)

[GR14] Goldwasser, S., Rothblum, G.N.: On best-possible obfuscation. J. Cryptol. **27**(3), 480–505 (2014)

[Had00] Hada, S.: Zero-knowledge and code obfuscation. In: Okamoto, T. (ed.) ASIACRYPT 2000. LNCS, vol. 1976, pp. 443–457. Springer, Heidelberg (2000). https://doi.org/10.1007/3-540-44448-3_34

[HB05] Brodsky, A., Hoory, S.: Simple permutations mix even better (2005)

[HMMR05] Hoory, S., Magen, A., Myers, S., Rackoff, C.: Simple permutations mix well. Theor. Comput. Sci. **348**(2):251–261 (2005). Automata, Languages and Programming: Algorithms and Complexity (ICALP-A 2004)

[HO24] He, W., O'Donnell, R.: Pseudorandom permutations from random reversible circuits (2024)

[ILW23] Ilango, R., Li, J., Williams, R.R.: Indistinguishability obfuscation, range avoidance, and bounded arithmetic. Electron. Colloquium Comput. Complex. TR23-038 (2023)

[IRS22] Ilango, R., Ren, H., Santhanam, R.: Robustness of average-case meta-complexity via pseudorandomness. In: Leonardi, S., Gupta, A. (eds.) STOC 2022: 54th Annual ACM SIGACT Symposium on Theory of Computing, Rome, Italy, 20-24 June 2022, pp. 1575–1583. ACM (2022)

[JLS21] Jain, A., Lin, H., Sahai, A.: Indistinguishability obfuscation from well-founded assumptions. In: Khuller, S., Williams, V.V. (eds.) STOC 2021: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, 21-25 June 2021, pp. 60–73. ACM (2021)

[Kil88] Kilian, J.: Founding cryptography on oblivious transfer. In: Simon, J. (ed.) Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA, pp. 20–31. ACM (1988)

[KNT22] Kitagawa, F., Nishimaki, R., Tanaka, K.: Obfustopia built on secret-key functional encryption. J. Cryptol. **35**(3), 19 (2022)

[LP20] Liu, Y., Pass, R.: On one-way functions and Kolmogorov complexity. In: Irani, S. (ed.) 61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, 16-19 November 2020, pp. 1243–1254. IEEE (2020)

[LP21] Liu, Y., Pass. R.: Cryptography from sublinear-time average-case hardness of time-bounded Kolmogorov complexity. In: Khuller, S., Williams, V.V. (eds.) STOC 2021: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, 21-25 June 2021, pp. 722–735. ACM (2021)

[LPST16] Lin, H., Pass, R., Seth, K., Telang, S.: Indistinguishability obfuscation with non-trivial efficiency. In: Cheng, C.-M., Chung, K.-M., Persiano, G., Yang, B.-Y. (eds.) PKC 2016. LNCS, vol. 9615, pp. 447–462. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49387-8_17

[LPTV23] Liu, T., Pelecanos, A., Tessaro, S., Vaikuntanathan, V.: Layout graphs, random walks and the t-wise independence of SPN block ciphers. In: Handschuh, H., Lysyanskaya, A. (eds.) Advances in Cryptology - CRYPTO 2023 - 43rd Annual International Cryptology Conference, CRYPTO 2023, Santa Barbara, CA, USA, August 20-24, 2023, Proceedings, Part III, LNCS, vol. 14083, pp. 694–726. Springer, Cham (2023). https://doi.org/10.1007/978-3-031-38548-3_23

[LTV21] Liu, T., Tessaro, S., Vaikuntanathan, V.: The t-wise independence of substitution-permutation networks. In: Malkin, T., Peikert, C. (eds.) CRYPTO 2021. LNCS, vol. 12828, pp. 454–483. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-84259-8_16

[RVV24] Ragavan, S., Vafa, N. and Vaikuntanathan, V.: Indistinguishability obfuscation from bilinear maps and LPN variants. Theory of Cryptography Conference (TCC) (2024)

[SW14] Sahai, A., Waters, B.: How to use indistinguishability obfuscation: deniable encryption, and more. In: Shmoys, D.B. (ed.) STOC 2014, New York, NY, USA, May 31 - June 03, 2014, pp. 475–484. ACM (2014)

[Tof80] Toffoli, T.: Reversible computing. In: de Bakker, J., van Leeuwen, J. (eds.) ICALP 1980. LNCS, vol. 85, pp. 632–644. Springer, Heidelberg (1980). https://doi.org/10.1007/3-540-10003-2_104

[Wik24] Wikipedia contributors: White-box cryptography — Wikipedia, the free encyclopedia (2024). [Online; accessed 24-September-2024]

[WW21] Wee, H., Wichs, D.: Candidate obfuscation via oblivious LWE sampling. In: Canteaut, A., Standaert, F.-X. (eds.) EUROCRYPT 2021. LNCS, vol. 12698, pp. 127–156. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-77883-5_5

Rate-1 Arithmetic Garbling From Homomorphic Secret Sharing

Pierre Meyer(✉), Claudio Orlandi, Lawrence Roy, and Peter Scholl

Aarhus University, Aarhus, Denmark
{pierre.meyer,orlandi,peter.scholl}@cs.au.dk

Abstract. We present a new approach to garbling arithmetic circuits using techniques from homomorphic secret sharing, obtaining constructions with high rate that support free addition gates. In particular, we build upon non-interactive protocols for computing distributed discrete logarithms in groups with an easy discrete-log subgroup, further demonstrating the versatility of tools from homomorphic secret sharing. Relying on distributed discrete log for the Damgård-Jurik cryptosystem (Roy and Singh, *Crypto '21*), whose security follows from the decisional composite residuosity assumption (DCR), we get the following main results:

- **Two ciphertexts per multiplication, from IND-CPA security of Damgård-Jurik.** Assuming the Damgård-Jurik cryptosystem is semantically secure (which follows from DCR), there is a garbling scheme for circuits with B-bounded integer arithmetic using only two ciphertexts per multiplication. The total bit-size of the resulting garbled circuit is:

  $$(n + 2s_\times + 2D_\times) \cdot (\zeta + 1) \cdot \log N$$

  where n is the number of inputs, $s_\times$ is the number of multiplications, $D_\times$ is the multiplicative depth of the circuit, N is an RSA modulus and $N^{\zeta-1}$ is roughly the bound B on the magnitude of wire values in the computation.
- **One ciphertext per multiplication, from KDM security of Damgård-Jurik.** Assuming the Damgård-Jurik encryption scheme remains secure given encryption of the key and its inverse, the construction achieves rate-1. The total bit-size of the resulting garbled circuit is:

  $$(n + s_\times + 1) \cdot (\zeta + 1) \cdot \log N$$

  where the parameters are as above, except $N^{\zeta-2}$ is the magnitude bound.

As a side result, we show that our scheme based on IND-CPA security achieves rate 3/5 for levelled circuits.

1 Introduction

Garbled circuits are a tool commonly used in protocols for secure two-party computation and other cryptographic applications. Starting with the work of

E. Boyle and M. Mahmoody (Eds.): TCC 2024, LNCS 15367, pp. 71–97, 2025.
https://doi.org/10.1007/978-3-031-78023-3_3

Yao [Yao82], most constructions of garbled circuits are in the setting of Boolean circuits: they transform a description of a circuit $C : \{0,1\}^n \to \{0,1\}$ into a garbled circuit $\widehat{C}$, together with some input encoding information $\boldsymbol{L} = \{L_{i,0}, L_{i,1}\}_{i=1}^n$. For an input x, an evaluator can use the garbled circuit together with the subset of encoded inputs $\boldsymbol{L}^x := \{L_{i,x_i}\}_i$ to learn $C(x)$, but nothing else about the circuit or the input (besides some structure of C). Yao's construction, and most subsequent works, incur a multiplicative overhead $O(\lambda)$ in the security parameter λ, meaning that the size of $\widehat{C}$ is $O(\lambda)$ times larger than the number of gates in the original circuit C.

To garble a function containing arithmetic operations such as integer addition and multiplication, traditionally one would have to first express these operations as Boolean circuits and then garble the resulting circuit. Known circuits for ℓ-bit integer multiplication have size $O(\ell \log \ell)$, and more commonly $O(\ell^2)$, which adds a large overhead to the size of the garbled circuit for these types of operations. This changed with the work of Applebaum, Ishai and Kushilevitz [AIK11], who gave the first direct methods for garbling arithmetic computations. They presented constructions in the model of arithmetic circuits for *bounded integer computation*, where there exists a (possibly exponential) bound $B \in \mathbb{N}$ such that for any set of admissible inputs, no wire value in the computation exceeds B in absolute value.

Their main construction builds upon information-theoretic techniques for garbling low-depth arithmetic circuits, and combines this with the learning with errors assumption to support arbitrary polynomial-size circuits. They give an alternative construction (with larger overhead) based solely on one-way functions, which combines classical Boolean garbled circuits with the Chinese remainder theorem.

Rate of Arithmetic Garbling. An important efficiency metric of a garbling scheme is the *rate*, which measures its bandwidth efficiency. In the model of B-bounded integer computation, where the bit-length of wire values is $\ell = \log(2B+1)$, for a circuit with n inputs and $|C|$ gates and garbled circuit of size $\mathsf{size}(\widehat{C})$ bits, the rate is roughly captured by the quantity

$$\mathsf{rate} = \frac{(|C| + n)\ell}{\mathsf{size}(\widehat{C}) + \mathsf{size}(\boldsymbol{L})}$$

We focus on measuring the rate as the circuit size tends towards infinity faster than the number of inputs, with worst-case circuits consisting of e.g. predominantly multiplication gates. We are also interested in the asymptotic behaviour of the rate as ℓ tends to ∞.

Using Yao's scheme, one can build an arithmetic garbling scheme with rate $O(1/(\lambda \log \ell))$ via asymptotically efficient (yet impractical) multiplication algorithms, or $O(1/(\lambda\ell))$ with schoolbook multiplication, relying only on the existence of one-way functions. The LWE-based construction of [AIK11] achieves constant rate for constant-degree polynomials, while for general circuits obtains rate $O(1/\lambda_{\mathsf{LWE}})$, where λ_{LWE} is the underlying LWE dimension. In very recent

work, Heath [Hea24] gave the first construction of arithmetic garbling with rate $O(1/\lambda)$ that solely relies on "Minicrypt"-style assumptions, in this case a circular correlation-robust hash. Another recent work of Li and Liu [LL24] obtains rate $O(1/(\lambda\ell^{0.5}))$ in the random oracle model, as well as a construction with rate $O(\log \ell/(\lambda\ell))$ that supports free addition, matching the asymptotic efficiency of a construction from [BMR16], whilst additionally supporting improved bit decomposition gates.

The first *constant-rate* arithmetic garbling scheme for bounded integer computations was constructed by Ball, Li, Lin and Liu [BLLL23], under the decisional composite residuosity (DCR) assumption. Their construction uses the Damgård-Jurik [DJ01] extension of Paillier encryption [Pai99] to construct a more efficient key extension gadget than the LWE-based one of [AIK11], such that each garbled addition or multiplication gate requires only a constant number of Damgård-Jurik ciphertexts. When the integer bit-length is sufficiently large, Damgård-Jurik ciphertexts have rate $1 - \varepsilon$, which leads to a constant-rate garbling scheme.

1.1 Related Work

Goldwasser et al. [GKP+13] constructed reusable garbled circuits based on the subexponential learning with errors assumption, which was later improved to be *compact* by Boneh et al. [BGG+14]. Compactness means that the size of the garbled circuit only scales polynomially with the depth of the circuit and not its size (assuming the cleartext circuit is known to the evaluator). This leads to a garbling scheme for Boolean or arithmetic circuits with rate *better than 1*. However, as observed in [BLLL23], the dependence on the circuit depth and security parameter are both very large (n^6) and the construction relies on heavy machinery including fully homomorphic encryption and attributed-based encryption, using subexponentially hard LWE.

The work of [BLLL23] additionally includes constructions of arithmetic garbling for modulo p computations, instead of bounded integers. They present constructions based on LWE and DCR, however, both with a low rate in $\tilde{O}(1/\lambda)$. Li and Liu [LL24] also presented constructions based on LWE and DCR, and in particular showed how to obtain free addition under DCR, albeit with worse communication for multiplication gates than [BLLL23]. Both of these works also present constructions that support bit-decomposition gadgets for mixing Boolean and arithmetic computations in the same circuit.

Using bilinear maps, [FMS19] gave a construction of arithmetic garbling for inner products with constant rate. Their construction can be bootstrapped to polynomial-sized circuits using the CRT-based compiler of [AIK11], however, this results in a poor rate.

Recently, Hazay and Yang [HY24] investigated the feasibility of maliciously secure garbling, building on the semi-honest construction of [BLLL23].

1.2 Our Contributions

We introduce a new approach to garbling arithmetic circuits using techniques from *homomorphic secret sharing* [BGI16, OSY21, RS21]. Unlike prior constructions [AIK11, BLLL23], which rely on the linearly homomorphic properties of Damgård-Jurik or LWE encryption, we additionally exploit methods for computing *distributed discrete logarithms* in groups with an easy discrete-log subgroup. Through this, we depart from the approach of combining an information-theoretic randomized encoding with a key-extension gadget [AIK11, BLLL23], instead giving direct constructions of arithmetic garbling with high rate.

We present two main constructions with security based on the Damgård-Jurik encryption scheme: (1) a construction with rate 1, based on the key-dependent message (KDM) security of Damgård-Jurik, and (2) a construction with rate 1/2, based on the decisional composite residuosity assumption (i.e. the IND-CPA security of Damgård-Jurik). The former result is the first rate-1 construction of arithmetic garbled circuits that does not rely on heavy tools such as attribute-based encryption. Both constructions also enjoy free addition gates.

The asymptotic efficiency of our constructions is compared with prior works in Table 1. We also give concrete examples for $\ell \approx \log N$ bits, where N is the RSA modulus, The concrete values for the rate of the [BLLL23] constructions come from [BLLL23, Table 3], which uses an integer bound of $B \approx 4000$ bits and a 4096-bit Paillier modulus N. Notice that our two constructions achieve concrete rates of 1/4 and 1/6 using the same parameters. As ℓ grows larger, the

Table 1. Summary of bounded integer arithmetic garbling schemes for ℓ-bit integers. N is an RSA modulus, $c > 1$ is a constant, ε is a constant approaching zero as $\ell \to \infty$. "Strong DCR" is a variant of DCR combined with a DDH-like assumption with small exponent

Construction	Security	Rate	Free addition
[AIK11]	LWE	$\Theta(1/\lambda_{\mathsf{LWE}})$	✗
Yao+[HVDH21]	OWF	$\Theta(1/\lambda \log \ell)$	✗
[BMR16, LL24]	RO	$\Theta(\log \ell/\lambda\ell)$	✓
[Hea24]	CCR hash	$\Theta(1/\lambda)$	✗
[BLLL23] (e.g. $\ell \approx \log N$)	DCR	$\Theta(1/c)$ 1/36	✗
[BLLL23] (e.g. $\ell \approx \log N$)	strong DCR	$\Theta(1/c)$ 1/24	✗
Section 4.1 (e.g. $\ell \approx \log N$)	DCR + KDM	$1 - \varepsilon$ 1/4	✓
Section 4.2 (e.g. $\ell \approx \log N$)	DCR	$1/2 - \varepsilon$ 1/6	✓

rate of our schemes quickly approaches 1 and 1/2, for instance when $\ell \approx 12000$ and $\log N \approx 2000$, we obtain respective rates 1/2 and 3/10. Meanwhile, the constructions from [BLLL23] have rate 1/12 at best.

2 Technical Overview

Given $x \in \mathbb{Z}$, we call "subtractive shares of x", denoted $\langle x \rangle_1$ and $\langle x \rangle_0$, values which satisfy $\langle x \rangle_1 - \langle x \rangle_0 = x$. We further use $\langle x \rangle$ to mean "either $\langle x \rangle_1$ or $\langle x \rangle_0$, as clear from context". In particular, if the evaluator and the garbler hold $\langle x \rangle_1$ and $\langle x \rangle_0$ respectively, then having parties compute $f(\langle x \rangle)$ means having them compute $f(\langle x \rangle_1)$ and $f(\langle x \rangle_0)$ respectively.

In Sect. 2.1, we present a simple, warm-up construction that shows how to build high-rate arithmetic garbled circuits using efficient distributed discrete logarithm (DDLog) algorithms. In Sect. 2.2, we describe an optimised scheme, which achieves rate-$(1-\varepsilon)$ if the DDLog's underlying encryption scheme is secure against *key-dependent messages* (KDM-secure). In Sect. 2.3, we instantiate this optimised construction using the Damgård-Jurik cryptosystem (and associated DDLog). In Sect. 2.4, we provide a rate-$\frac{1-\varepsilon}{2}$ garbling scheme, relying only on semantic security (and not KDM-security) of Damgård-Jurik.

2.1 Arithmetic Garbled Circuits Through the Lense of Homomorphic Secret Sharing

We build arithmetic garbled circuits following the definitional template introduced by Applebaum, Ishai, and Kushilevitz [AIK11]. The garbler (who knows the circuit but not its inputs) generates a global key $\boldsymbol{k}$ and a wire-dependent key $\boldsymbol{K}_x$ for each wire x. Each wire x is then associated with the label $\boldsymbol{L}_x = \boldsymbol{k} \cdot x + \boldsymbol{K}_x$. Note that this can also be interpreted as a secret-sharing $\langle \boldsymbol{k} \cdot x \rangle$ where $\langle \boldsymbol{k} \cdot x \rangle_1 = \boldsymbol{L}_x$ is known to the evaluator and $\langle \boldsymbol{k} \cdot x \rangle_0 = \boldsymbol{K}_x$ is known to the garbler. We will exploit this duality to use techniques originating in the *homomorphic secret-sharing (HSS)* domain to construct our garbling schemes.

Jumping ahead, the evaluator must be able to retrieve the values corresponding to the labels of the output wires. To allow this, we let the garbler sample $\boldsymbol{k}$ as a vector whose first coordinate is 1: it now follows that for each wire x, $\boldsymbol{L}_x.\mathsf{fst} = x + \boldsymbol{K}_x.\mathsf{fst}$. For each output value z, including $\boldsymbol{K}_z.\mathsf{fst}$ in the garbled circuit therefore allows the evaluator to retrieve z from $\boldsymbol{L}_z$. As it turns out, defining $\boldsymbol{k}$ like this also helps us with the gate-by-gate evaluation.

The wire-dependent key $\boldsymbol{K}_x$ is instead generated uniformly at random for each input x, and then the garbler proceeds to define the other keys in a gate-by-gate fashion. Considering the "HSS viewpoint", our task can then be viewed as having garbler and evaluator non-interactively convert subtractive shares $\langle \boldsymbol{k} \cdot x \rangle$ and $\langle \boldsymbol{k} \cdot y \rangle$ into subtractive shares $\langle \boldsymbol{k} \cdot g(x, y) \rangle$ for any given gate $g \in \{+, -, \times\}$.

If g is an addition gate with output $z = x + y$, the garbler can simply define their share $\boldsymbol{K}_z$ as $\boldsymbol{K}_z \coloneqq \boldsymbol{K}_x + \boldsymbol{K}_y$. This allows the evaluator to compute their

share $\boldsymbol{L}_z = \boldsymbol{L}_x + \boldsymbol{L}_y$, which equals $\boldsymbol{k} \cdot (x + y) + (\boldsymbol{K}_x + \boldsymbol{K}_y)$. From the HSS viewpoint, this works since $\langle x + y \rangle = \langle x \rangle + \langle y \rangle$.

If g is a multiplication gate with output $z = x \cdot y$, we will exploit the following identity:

$$\begin{aligned} \boldsymbol{k} \cdot z = \boldsymbol{k}x \cdot y &= (\boldsymbol{k}x + \boldsymbol{K}_x) \cdot (y + \boldsymbol{K}_y.\mathsf{fst}) - \boldsymbol{k}x \cdot (\boldsymbol{K}_y.\mathsf{fst}) \qquad (1) \\ &\quad - y \cdot \boldsymbol{K}_x - \boldsymbol{K}_x \cdot (\boldsymbol{K}_y.\mathsf{fst}) \\ &= \boldsymbol{L}_x \cdot (\boldsymbol{L}_y.\mathsf{fst}) - \boldsymbol{k}x \cdot (\boldsymbol{K}_y.\mathsf{fst}) - y \cdot \boldsymbol{K}_x - \boldsymbol{K}_x \cdot (\boldsymbol{K}_y.\mathsf{fst}) \qquad (2) \end{aligned}$$

The terms $\boldsymbol{L}_x \cdot (\boldsymbol{L}_y.\mathsf{fst})$ and $\boldsymbol{K}_x \cdot (\boldsymbol{K}_y.\mathsf{fst})$ can be computed locally by the evaluator and garbler respectively, but neither can compute $\boldsymbol{k}x \cdot (\boldsymbol{K}_y.\mathsf{fst})$ or $y \cdot \boldsymbol{K}_x$. We therefore let the garbler provide appropriate encodings of $(\boldsymbol{K}_y.\mathsf{fst})$ and $\boldsymbol{K}_x$ as part of the garbled circuit, so that the evaluator can compute shares of these terms, by solving the *distributed discrete logarithm problem (DDLog)*.

Share Conversion from Distributed Discrete Logarithm. Without going into details, for now we assume the existence of a deterministic algorithm DDLog, that both the garbler and the evaluator can locally execute, satisfying the following property (for some appropriately chosen encryption scheme):

> Let $c = \mathsf{Enc}_{\mathsf{pk}}(A)$ be an encryption of A with decryption key sk, and $\langle \mathsf{sk} \cdot B \rangle$ a sharing (over $\mathbb{Z}$) of $\mathsf{sk} \cdot B$, then $\mathsf{DDLog}(c^{\langle \mathsf{sk} \cdot B \rangle}) = \langle \mathsf{sk} \cdot AB \rangle$ is a subtractive share (over $\mathbb{Z}$) of $\mathsf{sk} \cdot AB$.
> In particular, if c is an encryption of sk^{-1} (with the inverse taken with respect to the plaintext modulus) then it holds that $\mathsf{DDLog}(c^{\langle \mathsf{sk} \cdot B \rangle})$ is a substractive share (over $\mathbb{Z}$) of B.

(The reader already familiar with homomorphic secret sharing literature may pause here, as it is more usual for $\mathsf{DDLog}(\mathsf{Enc}_{\mathsf{pk}}(A)^{\langle \mathsf{sk} \cdot B \rangle})$ to be a share of AB, not $\mathsf{sk} \cdot AB$.)

Garbling a Multiplication Gate. We first specify how $\boldsymbol{k}$ is defined: the garbler samples a keypair $(\mathsf{sk}, \mathsf{pk})$ associated with a DDLog-compatible encryption scheme, then sets $\boldsymbol{k} \leftarrow (1, \mathsf{sk})$. To garble a multiplication gate with input wire keys $\boldsymbol{K}_x, \boldsymbol{K}_y$, the garbler computes the following ciphertexts and adds them to the garbled circuit[1]:

$$c \xleftarrow{\$} \mathsf{Enc}_{\mathsf{pk}}(\mathsf{sk}^{-1}),\ c_y \xleftarrow{\$} \mathsf{Enc}_{\mathsf{pk}}(\boldsymbol{K}_y.\mathsf{fst}),\ \text{and}\ \boldsymbol{c}_x \xleftarrow{\$} \mathsf{Enc}_{\mathsf{pk}}(\boldsymbol{K}_x)\ .$$

Using this, the garbler and evaluator can do the following:

1. With their shares $\langle \mathsf{sk} \cdot x \rangle$ (namely, $\boldsymbol{K}_x.\mathsf{snd}$ and $\boldsymbol{L}_x.\mathsf{snd}$) and ciphertext c_y encrypting $\boldsymbol{K}_y.\mathsf{fst}$, run DDLog to obtain shares $\langle \mathsf{sk} \cdot x \cdot \boldsymbol{K}_y.\mathsf{fst} \rangle$.
2. Using shares $\langle \mathsf{sk} \cdot x \cdot \boldsymbol{K}_y.\mathsf{fst} \rangle$ and c encrypting sk^{-1}, run DDLog to get shares $\langle x \cdot \boldsymbol{K}_y.\mathsf{fst} \rangle$.

[1] Note that the first two ciphertexts encrypt scalars, while the last encrypts a tuple.

3. Concatenate the resulting shares $\langle x \cdot \boldsymbol{K}_y.\mathsf{fst}\rangle$ and $\langle \mathsf{sk} \cdot x \cdot \boldsymbol{K}_y.\mathsf{fst}\rangle$, to get $\langle \boldsymbol{k}x \cdot \boldsymbol{K}_y.\mathsf{fst}\rangle$.

Similarly, they can:

4. Use their shares $\langle \mathsf{sk} \cdot y\rangle$ (namely, $\boldsymbol{K}_y.\mathsf{snd}$ and $\boldsymbol{L}_y.\mathsf{snd}$) and the encryption $\boldsymbol{c}_x$ of $\boldsymbol{K}_x$, and run DDLog to compute shares $\langle \mathsf{sk} \cdot y \cdot \boldsymbol{K}_x\rangle$.
5. Use shares $\langle \mathsf{sk} \cdot y \cdot \boldsymbol{K}_x\rangle$ and the encryption of sk^{-1}, the parties compute shares $\langle y \cdot \boldsymbol{K}_x\rangle$.

If the garbler and the evaluator define $\boldsymbol{K}_z$ and $\boldsymbol{L}_z$ respectively by offsetting $\boldsymbol{K}_x \cdot (\boldsymbol{K}_y.\mathsf{fst})$ and $\boldsymbol{L}_x \cdot (\boldsymbol{L}_y.\mathsf{fst})$ by their shares of $\boldsymbol{k}x \cdot \boldsymbol{K}_y.\mathsf{fst}$ (step 3) and $y \cdot \boldsymbol{K}_x$ (step 5), then, by Eq. (2), the invariant "$\boldsymbol{L}_z = \boldsymbol{k} \cdot z + \boldsymbol{K}_z$" is maintained.

Summarising, the garbled circuit contains one globally reusable ciphertext (c, the encryption of sk^{-1}), one ciphertext $\boldsymbol{c}_x$ per left input-wire to a multiplication, one ciphertext c_y per right input-wire to a multiplication, and one mask $\boldsymbol{K}_z.\mathsf{fst}$ per output. Overall, provided the encryption scheme has rate-1, the garbling scheme has rate close to 1/2 (provided the secret key is small enough compared to the ring over which the circuit's arithmetic is performed[2]) and achieves free addition.

2.2 Achieving Rate-1

We now show how the above (sketch of a) construction can be adapted to achieve rate-1, while preserving free addition. There are two key ideas. The first is an alternative identity to Eq. (2) which breaks the asymmetry between the left and right wires of multiplication gates, which means all ciphertexts are of the same form. The second is exploiting linear homomorphism of the underlying encryption scheme in order to halve the number of ciphertexts: In the previous construction, the garbled circuit includes one ciphertext for each input wire to a multiplication gate. In the following, the garbled circuit only contains one ciphertext for each multiplication gate (which this time depends on the output wire of the gate), as well as one ciphertext per input wire.

In the rate-1 construction, $k \leftarrow \mathsf{sk}$, L_x and K_x (for each wire x) are now scalars satisfying $k \cdot x = L_x - K_x$. We handle addition gates exactly as before. To understand how we support multiplications, consider Eq. (4):

$$\begin{aligned} \mathsf{sk}^2 \cdot z = \mathsf{sk} \cdot x \cdot \mathsf{sk} \cdot y &= (\mathsf{sk} \cdot x + K_x) \cdot (\mathsf{sk} \cdot y + K_y) - \mathsf{sk} \cdot x \cdot K_y \qquad (3) \\ &\quad - \mathsf{sk} \cdot y \cdot K_x - K_x \cdot K_y \\ &= L_x \cdot L_y - \mathsf{sk} \cdot x \cdot K_y - \mathsf{sk} \cdot y \cdot K_x - K_x \cdot K_y \qquad (4) \end{aligned}$$

Including encryptions of K_x and K_y in the garbled circuit allows the garbler and evaluator to use DDLog to compute shares $\langle \mathsf{sk} \cdot x \cdot K_y\rangle$ and $\langle \mathsf{sk} \cdot y \cdot K_x\rangle$ since they hold shares $\langle \mathsf{sk} \cdot x\rangle$ (namely, L_x and K_x) and $\langle \mathsf{sk} \cdot y\rangle$ (namely, L_y and K_y). In

[2] We need *authenticated* shares of wire values (*i.e.* $\langle \mathsf{sk} \cdot w\rangle$) to fit in the plaintext space.

turn, using Eq. (4), the garbler and the evaluator can compute shares $\langle \mathsf{sk}^2 \cdot z \rangle$ (recall that the garbler can compute $K_x \cdot K_y$ and the evaluator can compute $L_x \cdot L_y$). Finally, including an encryption of sk^{-1} in the garbled circuit allows the parties to use their shares $\langle \mathsf{sk}^2 \cdot z \rangle$ and DDLog to compute shares $\langle \mathsf{sk} \cdot z \rangle$: these shares define K_z and L_z. Furthermore, if z is an output gate, the parties can use DDLog once more, again with the encryption of sk^{-1}, to compute shares $\langle z \rangle$; as before, if the garbler publishes its share of z then the evaluator can reconstruct the value of z from the label L_z.

One question remains: how do we obtain the encryptions of K_x and K_y for each multiplication gate $z = x \times y$, without publishing two ciphertexts per gate? By definition, $K_{x+y} \coloneqq K_x + K_y$, and therefore if the encryption scheme supports linear homomorphism, the parties can use encryptions of K_x and K_y to generate an encryption of K_{x+y}. So, it is enough for the garbler to send an encryption of K_w for each w which is either an input to the circuit or the output of a multiplication gate: the parties can then proceed in a gate-by-gate fashion to reconstruct all the necessary ciphertexts.

2.3 Concrete Instantiation Using the Damgård-Jurik encryption scheme [DJ01]

Recall that, in the Damgård-Jurik cryptosystem [DJ01], the public key is an RSA modulus $N = p \cdot q$ (where p and q are large primes) while the secret key is its Euler totient $\mathsf{sk} = (p-1) \cdot (q-1)$. The scheme is parameterised by an integer $\zeta \geq 2$: the plaintext space is $\mathbb{Z}/N^\zeta\mathbb{Z}$, while the ciphertext space is $\mathbb{Z}/N^{\zeta+1}\mathbb{Z}$. The encryption scheme therefore has rate $1/(1+1/\zeta)$. The scheme is also linearly homomorphic, and Roy and Singh [RS21] showed there exists a DDLog algorithm satifying the requirements from Sect. 2.1)[3], so it can be used to instantiate the arithmetic garbled circuit described in Sect. 2.2. Because we need to fit (shares of) $\mathsf{sk} \cdot w$ in the plaintext space (of size N^ζ) for each wire w (and $\mathsf{sk} = (p-1) \cdot (q-1) \leq p \cdot q = N$), we can set parameters so the arithmetic is performed over a ring of size $N^{\zeta-1}$. However, we cannot use this entire plaintext space, due to a failure probability in DDLog when the underlying plaintexts are too large. This means we get a garbling scheme for B-bounded integer computation, where $B \approx N^{\zeta-1}/2^\kappa$ and κ is a statistical correctness parameter (ensuring correctness with probability $1 - 2^{-\kappa}$ per gate). Overall, the garbling scheme of Sect. 2.2 has rate $(\zeta-1)/(\zeta+1)$. We can therefore achieve rate-$(1-\varepsilon)$ for an arbitrarily small constant ε.

Figure 1 summarises how the garbler and the evaluator perform multiplications in the rate-1 garbling scheme of Sect. 2.2 as instantiated with Damgård-Jurik (which we assume to be KDM-secure) and Roy-Singh's DDLog. For clarity, the figure shows the roles of the two parties as symmetric, but recall that we are instantiating a garbling scheme, where the garbler can perform all of their

[3] That is, if c is an encryption of y under public key N, and $\langle \mathsf{sk} \cdot x \rangle$ is a share of $\mathsf{sk} \cdot x$, then $\mathsf{DDLog}(c^{\langle \mathsf{sk} \cdot x \rangle})$ is a share of $\mathsf{sk} \cdot xy$.

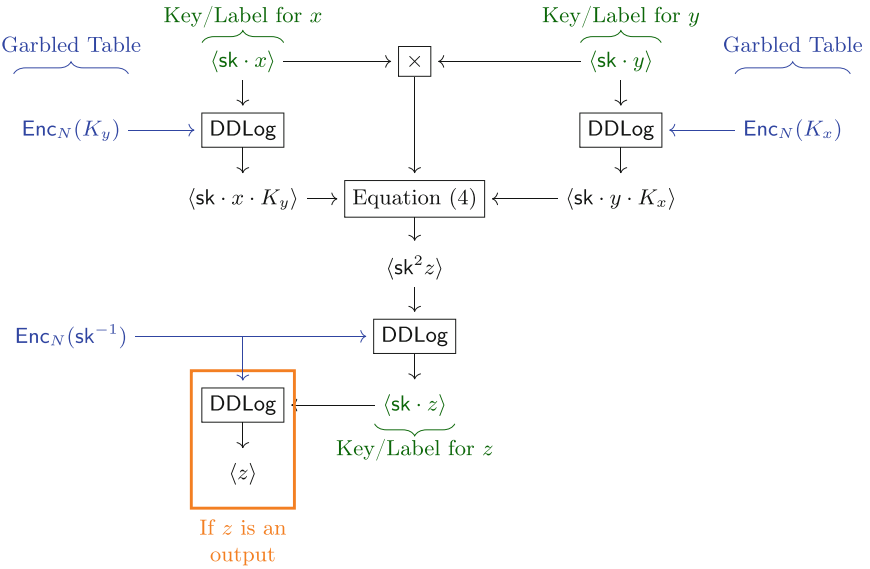


Fig. 1. Operations performed by the garbler (resp. evaluator) to convert their keys (resp. labels) for wires x and y into a key (resp. label) for wires $z = x \cdot y$. This requires KDM-security of Damgård-Jurik encryption; (sk, N) is the secret/public key pair.

garbling operations first, send the garbled circuit to the evaluator, who then evaluates the gates.

2.4 Removing the Circular-Security Assumption

The security of the scheme in Sect. 2.3 relies on assuming security of the underlying encryption scheme against *key-dependent message* attacks (KDM-security). Most visibly, this is because the garbled circuit contains an encryption of the inverse of the secret key, but also because the evaluator receives encryptions of shares of the form $\mathsf{Enc}_{\mathsf{pk}}(K_x = \langle \mathsf{sk} \cdot x \rangle_0)$ as well as receiving their share $L_x = \langle \mathsf{sk} \cdot x \rangle_1$ of this value. The evaluator may know x, and x would allow computing $\mathsf{Enc}_{\mathsf{pk}}(\mathsf{sk})$ from $\mathsf{Enc}_{\mathsf{pk}}(K_x)$ because $(L_x - K_x)/x = \mathsf{sk}$ and Enc is additively homomorphic. Our scheme can be adapted to instead rely on semantic security (provably implied by the *decisional composite residuosity* assumption, DCR), but at the cost of reducing the rate to $1/2 - \varepsilon$.

Removing the Need for $\mathsf{Enc}_N(\mathsf{sk}^{-1})$. Without circular security, we can no longer use an encryption of sk^{-1} to convert authenticated wire shares $\langle \mathsf{sk} \cdot w \rangle$ into plain shares $\langle w \rangle$ (which are required for output wires). Instead, following [RS21], we will use an alternative decryption key for Damgård-Jurik, which allows to directly obtain shares $\langle w \rangle$ from DDLog without an encryption of sk^{-1}:

> Let $d \leftarrow \mathsf{sk} \cdot (\mathsf{sk}^{-1} \bmod N^{\zeta})$ and c be a Damgård-Jurik encryption of y under public key N, then $\mathsf{DDLog}(c^{\langle d \cdot x \rangle}) = \langle x \cdot y \rangle$ is a share of $x \cdot y$.

Next, to use this in garbling, we will use a third multiplication identity:[4]

$$z = x \cdot y = \langle x \rangle_1 \cdot \langle y \rangle_1 - x \cdot \langle y \rangle_0 - y \cdot \langle x \rangle_0 - \langle x \rangle_0 \cdot \langle y \rangle_0 \tag{5}$$

Assuming the parties' shares can be derived from their key or label for x, the parties can locally compute the respective values $\langle x \rangle_0 \cdot \langle y \rangle_0$ and $\langle x \rangle_1 \cdot \langle y \rangle_1$. To obtain the missing cross-terms, the garbler can provide $\mathsf{Enc}_N(\langle w \rangle_0)$ for each wire w (or rather, exploiting linear homomorphism as in Sect. 2.2, one ciphertext for each input or output of a multiplication gate is enough), allowing use of DDLog to obtain shares of the necessary products.

The problem, however, is that we still need to get shares of $d \cdot w$ for each wire w. Trying to generate these shares in the same way as before runs again into a circularity issue, as an identity of the form

$$d \cdot z = \langle d \cdot x \rangle_1 \cdot \langle y \rangle_1 - dx \cdot \langle y \rangle_0 - y \cdot \langle d \cdot x \rangle_0 - \langle d \cdot x \rangle_0 \cdot \langle y \rangle_0$$

requires the garbler to provide an encryption of $\langle d \cdot x \rangle_0$. To bypass this problem, we instead generate a sequence of keypairs, one per multiplicative level of the circuit, and exploit the idea of *key-switching*.

Key-Switching. Let D be the multiplicative depth of the circuit. For each $i \in [0, D]$, we have the garbler generate a keypair (N_i, sk_i) (and set d_i accordingly). The idea behind key-switching is to allow the parties to convert shares of $d_i \cdot w$ into shares of $d_{i+1} \cdot w$ (then recursively of $d_{i+2} \cdot w$, *etc.*.). This can simply be done by having the garbler provide an encryption of d_{i+1} under keypair (N_i, sk_i). Then, the parties can compute shares:

$$\langle d_{i+1} \cdot w \rangle \leftarrow \mathsf{DDLog}(c_i^{\langle d_i \cdot w \rangle}), \text{ where } c_i \xleftarrow{\$} \mathsf{Enc}_{N_i}(d_{i+1}) \ .$$

Thanks to this trick, it is now possible to build an arithmetic garbling scheme without resorting to KDM-style assumptions, at the price of achieving a lower rate $\frac{1}{2} - \varepsilon$. Getting to the final construction requires taking care of more technical issues, which we explain in detail in Sect. 4.2.

3 Preliminaries

3.1 Arithmetic Garbled Circuits

3.1.1 Arithmetic Circuits with Bounded Integer Computation. We use a model of arithmetic circuits consisting of addition, subtraction and (fan-in two) multiplication gates, all computed over $\mathbb{Z}$. We work in the *bounded integer computation* model, where there exists a bound $B = B(\lambda) \in \mathbb{N}$ that bounds the magnitude of circuit wire values. For a circuit C with n input wires, we say that an input vector $\boldsymbol{x} \in \mathbb{Z}^n$ is *admissible with respect to bound* B if the inputs, outputs and every intermediate wire value obtained during the evaluation of C are in the range $[-B, B]$.

[4] Recall, we use subtractive share notation, where $\langle x \rangle$ means that $x = \langle x \rangle_1 - \langle x \rangle_0$.

3.1.2 Arithmetic Garbling. We define arithmetic garbled circuits, adapting the definition of [BLLL23].

Definition 1 (B-bounded) Arithmetic Garbled Circuit). *A garbling scheme for a family of circuits classes $\mathcal{C} = \{\mathcal{C}_\lambda\}_{\lambda\in\mathbb{N}^\star}$ for B-bounded integer computation with label space $\mathcal{L} = \mathcal{L}(\lambda)$ is a pair of* p.p.t. *algorithms $\mathsf{AGC} = (\mathsf{AGC.Garble}, \mathsf{AGC.Eval})$ with the following syntax and properties:*

- $\mathsf{Garble}(1^\lambda, C)$: *On input a security parameter 1^λ and a circuit $C \in \mathcal{C}_\lambda$ with n inputs,* Garble *outputs n key pairs $(\boldsymbol{k}_0^i, \boldsymbol{k}_1^i)_{i\in[n]} \in \mathcal{L}$ and a garbled circuit $\widehat{C}$.*
- $\mathsf{Eval}((\boldsymbol{L}_i)_{i\in[n]}, \widehat{C})$: *On input n input labels $(\boldsymbol{L}_i)_{i\in[n]} \in \mathcal{L}^n$, and a garbled circuit $\widehat{C}$,* Eval *outputs a value $y \in [0, B]$.*
- **Correctness.** *A garbling scheme is* correct *if there exists a negligible function* negl *such that for all $\lambda \in \mathbb{N}$, every circuit $C \in \mathcal{C}_\lambda$ with n inputs, and every $x_1, \ldots, x_n \in \mathcal{X}^n$ where $\mathcal{X}$ is the set of admissible inputs of C induced by bound B, the following holds:*

$$\Pr\left[\mathsf{Eval}((\boldsymbol{L}_i)_{i\in[n]}, \widehat{C}) =_{\mathbb{Z}} C(x_1, \ldots, x_n) \colon \begin{array}{r}((\boldsymbol{k}_0^i, \boldsymbol{k}_1^i)_{i\in[n]}, \widehat{C}) \xleftarrow{\$} \mathsf{Garble}(1^\lambda, C) \\ \boldsymbol{L}_i \leftarrow \boldsymbol{k}_0^i \cdot x_i + \boldsymbol{k}_1^i\end{array}\right] \leq \mathsf{negl}(\lambda) \ .$$

- **Privacy.** *A garbling scheme is* secure *if there exist a* p.p.t. *simulator $\mathcal{S}$ such that for all sequence of circuits $\{C_\lambda\}_{\lambda\in\mathbb{N}}$, where $C_\lambda \in \mathcal{C}_\lambda$ with $n = n(\lambda)$ inputs, and every admissible (with respect to B and C_λ) sequence of inputs $\{(x_{1,\lambda}, \ldots, x_{n,\lambda})\}_{\lambda\in\mathbb{N}}$, the following holds:*

$$\left\{\mathcal{S}(1^\lambda, C_\lambda, y) \colon y \leftarrow C_\lambda(x)\right\} \overset{c}{\approx} \left\{((\boldsymbol{L}_i)_{i\in[n]}, \widehat{C}_\lambda) \colon \begin{array}{r}((\boldsymbol{k}_0^i, \boldsymbol{k}_1^i)_{i\in[n]}, \widehat{C}_\lambda) \xleftarrow{\$} \mathsf{Garble}(1^\lambda, C_\lambda) \\ \boldsymbol{L}_i \leftarrow \boldsymbol{k}_0^i \cdot x_i + \boldsymbol{k}_1^i \\ y \leftarrow C_\lambda(x)\end{array}\right\} \ .$$

To measure the efficiency of an arithmetic garbled circuit, we define its *rate* as follows.

Definition 2 (Rate of Arithmetic Garbled Circuit). *Let $\mathcal{C}$ be a class of arithmetic circuits, let* AGC *be an arithmetic garbling scheme for $\mathcal{C}$ with B-bounded computation, and let $\ell = \log(2B + 1)$. The* rate *of* AGC *for $\mathcal{C}$ is the quantity:*

$$\mathsf{rate} = \liminf_{C\in\mathcal{C}} \min_x \frac{(|C| + n)\ell}{\mathsf{size}(\widehat{C}) + \sum_i \mathsf{size}(\boldsymbol{k}_0^i \cdot x_i + \boldsymbol{k}_1^i)},$$

where the minimum is taken over all admissible inputs to the circuit $\mathcal{C}$, and the limit infimum is taken over all circuits in $\mathcal{C}$ (in the sense of the limit of a net, as we require $\mathcal{C}$ to form a directed set when ordered by inclusion).

We are most interested in the case where the circuit size is dominated by the number of computation gates (including outputs, but as opposed to inputs).[5] That is, we will typically be interested in classes of circuits $\mathcal{C}_f$ that contain only circuits where where the number of inputs $n \leq f(|C|)$ for a sublinear function f. Note that the rate is a worst-case quantity that does not account for optimizations like free addition gates.

Previously, [BLLL23] define rate to be roughly the inverse of the above quantity, meaning that typically rate ≥ 1. Our definition is more consistent with the standard concept of the rate of an encryption scheme, which is at most 1.

When AGC supports arbitrarily large values of ℓ, we are also interested in the asymptotic behaviour of the rate as ℓ tends to ∞.

3.2 Damgård-Jurik Cryptosystem

The Damgård-Jurik cryptosystem [DJ01] is a generalisation of the Paillier cryptosystem [Pai99] (which, with the notation of Fig. 2, can be seen as the special case $\zeta = 1$).

Definition 3 (Decision Composite Residuosity Assumption (DCR), [Pai99]). *Let* $\mathsf{RSA.Gen}$ *be a polynomial-time algorithm which, on input a security parameter* λ*, outputs* (N, p, q) *where* p *and* q *are* λ*-bit primes and* $N = pq$*. Let* λ *be a security parameter. We say that the* Decision Composite Residuosity (DCR) *problem is hard relative to modulus-sampling algorithm* $\mathsf{RSA.Gen}$ *if*

$$\left\{ (N, x) \colon \begin{matrix} (N, p, q) \xleftarrow{\$} \mathsf{RSA.Gen} \\ x \xleftarrow{\$} (\mathbb{Z}/N^2\mathbb{Z})^\times \end{matrix} \right\} \overset{c}{\approx} \left\{ (N, x^N \bmod N) \colon \begin{matrix} (N, p, q) \xleftarrow{\$} \mathsf{RSA.Gen} \\ x \xleftarrow{\$} (\mathbb{Z}/N^2\mathbb{Z})^\times \end{matrix} \right\} .$$

Theorem 4 (Damgård-Jurik Cryptosystem [DJ01]). *For any choice of the parameter* $\zeta \geq 2$*, the construction of Fig. 2 is a CPA-secure linearly homomorphic encryption scheme if and only if the DCR assumption holds.*

[5] As an alternative to considering only circuits with more gates than inputs, we could allow the garbling scheme to use "compressed inputs". That is, if (as in in both of our constructions) the labels can be defined by fixing $\boldsymbol{k}_0^i$ to be the same for all i, and sampling each $\boldsymbol{k}_1^i$ pseudorandomly, then there are methods of generating the input wire labels using only sublinear communication, such as the succinct VOLE of [ARS24]. To reflect this, we could then modify the rate definition to not charge for the size of the input wire labels.

LHE Damgård-Jurik Cryptosystem [DJ01]

`Requires:`

- $\mathsf{RSA.Gen}(\cdot)$ is an RSA modulus generation algorithm which, on input a security parameter 1^λ, samples $(p, q) \xleftarrow{\$} [2^{\ell(\lambda)-1}, 2^{\ell(\lambda)}]$ (where ℓ is some polynomial chosen appropriately in order for the cryptosystem to achieve λ bits of security) then computes $N \leftarrow p \cdot q$, and outputs (N, p, q).
- $\zeta \geq 2$ is a constant defining the plaintext size.
- Functions $\exp\colon (\mathbb{Z}/N^\zeta\mathbb{Z})^+ \to 1 + N(\mathbb{Z}/N^{\zeta+1}\mathbb{Z})$ and $\log\colon 1 + N(\mathbb{Z}/N^{\zeta+1}\mathbb{Z}) \to (\mathbb{Z}/N^\zeta\mathbb{Z})^+$ are defined by the following expressions, as in [RS21]:

$$\exp(x) = \sum_{k=0}^{\zeta} \frac{(Nx)^k}{k!} \quad \text{and} \quad \log(1+Nx) = \sum_{k=1}^{\zeta} \frac{(-N)^{k-1}x^k}{k}$$

$\mathsf{DJ.KeyGen}(1^\lambda)$:

1. Run $(N, p, q) \xleftarrow{\$} \mathsf{RSA.Gen}(1^\lambda)$
2. Compute $\varphi \leftarrow (p-1)\cdot(q-1)$
3. Output (N, φ)

$\mathsf{DJ.Enc}_{N,\zeta}(x)$:

1. Sample $r \xleftarrow{\$} (\mathbb{Z}/N^{\zeta+1}\mathbb{Z})^\times$
2. Compute $c \leftarrow r^{N^\zeta} \cdot \exp(x)$
3. Output c

$\mathsf{DJ.Eval}_{N,\zeta}(c_1, c_2, \alpha)$:

// Homomorph. eval. $(x, y) \mapsto \alpha \cdot x + y$

Compute and output $c \leftarrow c_1^\alpha \cdot c_2$

$\mathsf{DJ.Dec}_{N,\zeta,\varphi}(c)$:

1. Compute ψ as the multiplicative inverse of φ in $\mathbb{Z}/N^\zeta\mathbb{Z}$
2. Compute and output $x \leftarrow \psi \cdot \log(c^\varphi)$

Fig. 2. The Damgård-Jurik cryptosystem.

3.3 IND-KDM Security

Informally, an encryption scheme is KDM-secure if it remains secure when the plaintext messages are a function of the secret key.

Definition 5 Key-Dependent Message Security[BRS03], Special Case). *A public-key encryption scheme* $\mathsf{PKE} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$*, whose private-key and message spaces are denoted* $\mathcal{K}$ *and* $\mathcal{M}$ *respectively, is* secure in the presence of key-dependent messages with respect to function class $\mathcal{F} \subseteq \mathcal{K} \to \mathcal{M}$ *(or*

simply $\mathcal{F}$-KDM-secure*) if for every* p.p.t. *algorithm* $\mathcal{A}$ *with oracle queries*

$$\left| \Pr\left[\mathcal{A}^{\mathcal{O}^{\mathsf{KDM}}_{\mathcal{F},\mathsf{sk},0}}(1^\lambda,\mathsf{pk}) = 1 \colon (\mathsf{sk},\mathsf{pk}) \xleftarrow{\$} \mathsf{KeyGen}(1^\lambda)\right] - \Pr\left[\mathcal{A}^{\mathcal{O}^{\mathsf{KDM}}_{\mathcal{F},\mathsf{sk},1}}(1^\lambda,\mathsf{pk}) = 1 \colon (\mathsf{sk},\mathsf{pk}) \xleftarrow{\$} \mathsf{KeyGen}(1^\lambda)\right] \right| \leq \mathsf{negl}(\lambda) \ ,$$

where oracles $\mathcal{O}^{\mathsf{KDM}}_{\mathcal{F},\mathsf{sk},0}$ *and* $\mathcal{O}^{\mathsf{KDM}}_{\mathcal{F},\mathsf{sk},1}$ *are defined in Fig. 3.*

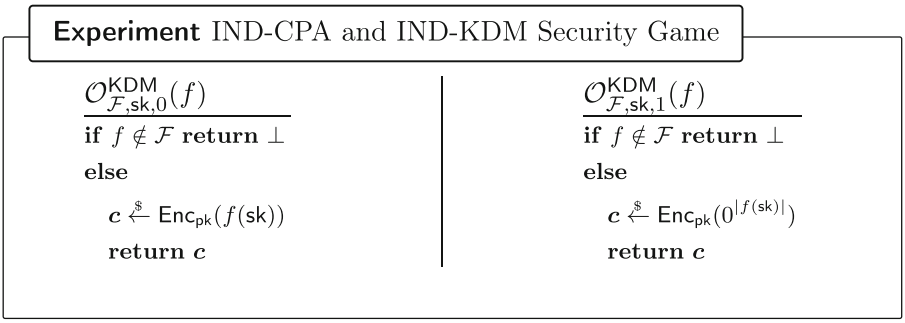


Fig. 3. Oracles used IND-KDM (Definition 5) security.

Note that in all generality, KDM-security requires we allow the plaintexts to be functions of multiple secret keys (*i.e.* $\mathcal{F} \subseteq \mathcal{K}^\ell \to \mathcal{M}$, where ℓ is some parameter, polynomial in the security parameter) and the encryption to be performed under any one of those keys. For our purposes however, it suffices to consider the special case $\ell = 1$. In fact, whenever we use KDM-secure encryption in this paper, it will suffice for the encryption scheme to be secure given encryptions of linear combinations of the key, its inverse, and all constants[6].

3.4 Distributed Discrete Logarithm

We recall in Fig. 4 the definition of distributed discrete logarithm function DDLog introduced by Roy and Singh [RS21]. The properties of this function which we use in this paper are stated in lemmata 6, 7, and 8. Note that these are immediate corollaries/reformulations of [RS21, Theorem 18] and [RS21, Lemma 19]. Throughout, we use the symmetric convention for the mod operator. Namely, "$\cdot \bmod B$" takes values in $(-B/2; B/2]$. We recall in Fig. 4 the definition of distributed discrete logarithm function DDLog introduced by Roy and Singh [RS21]. The properties of this function which we use in this paper are stated in lemmata 6, 7, and 8. Note that these are immediate corollaries/reformulations of [RS21,

[6] Note that if $\mathcal{F}$ includes all constant functions from $\mathcal{K}$ to $\mathcal{M}$, then $\mathcal{F}$-KDM security implies CPA security.

Theorem 18] and [RS21, Lemma 19]. Throughout, we use the symmetric convention for the mod operator. Namely, the function "$\cdot \bmod B$" takes values in $(-B/2; B/2]$.

DDLOG Damgård-Jurik Distance Function [RS21]

$\mathsf{DDLog}(N, h)$:

Compute and output $z \leftarrow \log\left(\frac{h}{h \bmod N}\right)$

Fig. 4. [RS21]'s distributed discrete logarithm for the Damgård-Jurik cryptosystem [DJ01].

Lemma 6. *For all* $(N, \varphi) \in \mathsf{Supp}(\mathsf{DJ.KeyGen})$*, for all exponents* $\zeta \geq 1$*, and for all ciphertexts* $c \in (\mathbb{Z}/N^{\zeta+1}\mathbb{Z})^\times$ *and shares (over* $\mathbb{Z}$*)* $\langle x\varphi\rangle_0, \langle x\varphi\rangle_1$ *of some* $x \in \mathbb{Z}$ *times* φ*, we have*

$$\mathsf{DDLog}(c^{\langle x\varphi\rangle_1}) - \mathsf{DDLog}(c^{\langle x\varphi\rangle_0}) \equiv \mathsf{DJ.Dec}_{N,\zeta,\varphi}(c) \cdot x \cdot \varphi \mod N^\zeta.$$

We refer to the full version of this paper [MORS24] for the proof of Lemma 6.

Lemma 7. *Let* $\zeta \geq 1$*. For all* $(N, \varphi) \in \mathsf{Supp}(\mathsf{DJ.KeyGen})$*, let* $\nu = N^{-\zeta} \bmod \varphi$*. For all ciphertexts* $c \in (\mathbb{Z}/N^{\zeta+1}\mathbb{Z})^\times$*, all* $x \in \mathbb{Z}$*, and all shares* $(\langle x\rangle_0, \langle x\rangle_1)$ *and* $(\langle x\nu\rangle_0, \langle x\nu\rangle_1)$ *of* x *and* $x\nu$*, we have*

$$\left[\mathsf{DDLog}(c^{\langle x\rangle_1 - N^\zeta \cdot \langle x\nu\rangle_1}) - \mathsf{DDLog}(c^{\langle x\rangle_0 - N^\zeta \cdot \langle x\nu\rangle_0}) \equiv \mathsf{DJ.Dec}_{N,\zeta,\varphi}(c) \cdot x \mod N^\zeta.\right.$$

We refer to the full version of this paper [MORS24] for the proof of Lemma 7.

Lemma 8. *For all moduli* $M > 1$ *and all modulo* M *shares* $\langle x\rangle_0, \langle x\rangle_1 \in \mathbb{Z}/M\mathbb{Z}$ *of some* $x \in \mathbb{Z}$*, we have*

$$\Pr_{r \xleftarrow{\$} \mathbb{Z}/M\mathbb{Z}}\left[(\langle x\rangle_1 + r) \bmod M - (\langle x\rangle_0 + r) \bmod M = x\right] = \max\left(1 - \frac{|x|}{N^\zeta}, 0\right).$$

Proof. This is essentially [RS21, Lemma 19]. □

4 Arithmetic Garbled Circuits

4.1 From KDM Security of Damgård-Jurik

We have already provided a high-level technical overview of this construction in Sect. 2.2, and we therefore proceed below with the formal description of our garbling scheme (Fig. 5) and its proof of correctness and security.

There is only one difficulty which is not addressed by the technical overview, and this is the reason why our garbling scheme only supports bounded-integer computation, and not full-blown modular arithmetic. By default, computing a DDLog allows the parties to generate shares modulo some value, but these shares need to then be converted into shares over $\mathbb{Z}$ before they can be used as input to another DDLog operation. If the shared value is small enough (which we guarantee by restricting ourselves to B-bounded computation) and the shares are (pseudo)randomised (which we guarantee by having garbler and evaluator offset their shares of each wire value using the same PRF key), then taking these shares modulo B yields correct shares over $\mathbb{Z}$. We note that this trick was previously used in [OSY21].

Theorem 9 (AGC with one Ciphertext per Multiplication from KDM-Secure Damgård-Jurik, via DDLog). *Let λ be a security parameter. Let $\zeta \geq 2$, assume the KDM-security*[7] *of the Damgård-Jurik encryption scheme with*

Arithmetic Garbling AGC from HSS (KDM-secure Damgård-Jurik variant)

Requires:

- $\mathsf{DJ} = (\mathsf{DJ.KeyGen}, \mathsf{DJ.Enc}, \mathsf{DJ.Dec})$ is the Damgård-Jurik cryptosystem of Fig. 2.
- DDLog is the algorithm of Fig. 4.
- $(\mathsf{PRF}_{N,\zeta,\lambda})_{N,\zeta,\lambda\in\mathbb{N}^3}$ is a family of pseudorandom functions s.t. $\mathsf{PRF}_{N,\zeta,\lambda} \colon \{0,1\}^\lambda \times ([N^\zeta] \times [0,3]) \to [N^\zeta]$.
- ord is an ordering of the gates of the circuit (*i.e.* a bijection between the set of gates and $[1,s]$, where s is the number of gates).

The Garbling Scheme (gate-by-gate description): Initially, the garbler samples $k_{\mathsf{PRF}} \xleftarrow{\$} \{0,1\}^\lambda$ and $(N,\varphi) \xleftarrow{\$} \mathsf{DJ.KeyGen}(1^\lambda)$, sets $k \leftarrow \varphi$, and computes $c \leftarrow \mathsf{DJ.Enc}_N(\varphi^{-1})$ (inversion is performed in $\mathbb{Z}/N^\zeta\mathbb{Z}$); for each input wire x, the garbler samples $K_x \xleftarrow{\$} [N^\zeta]$, computes $c_x \leftarrow \mathsf{DJ.Enc}_N(K_x)$. The label associated with input x is $L_x \leftarrow k \cdot x + K_x \in [N^\zeta]$.

Fig. 5. Arithmetic garbled circuit with one ciphertext per multiplication (and free addition) from KDM-security of Damgård-Jurik (gate-by-gate description).

[7] More precisely, the theorem assumes KDM-security with respect to the class containing the following functions: the inverse function, all linear combinations, and all constant functions. Note that because the Damgård-Jurik cryptosystem is linearly homomorphic, linear combination are well-defined.

Addition/Substraction gate $z = x \pm y$: For addition (resp. substraction) gates, at garbling time, given key pairs (k, K_x) and (k, K_y) as well as corresponding ciphertexts c_x and c_y, set $K_z \leftarrow K_x + K_y$ (resp. $K_z \leftarrow K_x - K_y$), compute the ciphertext $c_z \leftarrow c_x \cdot c_y$ (resp. $c_z \leftarrow c_x / c_y$) (c_z will be stored in order to compute later keys and tables, but is not included in the garbled circuit), and produce the key pair (k, K_z). At evaluation time, given the labels L_x and L_y, compute the label $L_z \leftarrow L_x + L_y$ (resp. $L_z \leftarrow L_x - L_y$) and the ciphertext $c_z \leftarrow c_x \cdot c_y$ (resp. $c_z \leftarrow c_x / c_y$).

Multiplication gate $z = x \cdot y$: At garbling time, given key pairs (k, K_x) and (k, K_y) as well as the corresponding ciphertexts c_x and c_y, compute the key pair (k, K_z), where K_z is computed as follows:

$$\begin{aligned} \mathsf{sh}_z^G &\leftarrow -\mathsf{DDLog}(c_x^{K_y}) - \mathsf{DDLog}(c_y^{K_x}) \\ &\quad - K_x \cdot K_y + \mathsf{PRF}_{N,\zeta,\lambda}(k_{\mathsf{PRF}}, \mathsf{ord}(z) \| 0) \bmod N^\zeta \\ K_z &\leftarrow \mathsf{DDLog}(c^{\mathsf{sh}_z^G}) + \mathsf{PRF}_{N,\zeta,\lambda}(k_{\mathsf{PRF}}, \mathsf{ord}(z) \| 1) \bmod N^\zeta \end{aligned}$$

Then produce the ciphertext $c_z \xleftarrow{\$} \mathsf{Enc}_N(K_z)$ (this ciphertext will be included in the garbled circuit). At evaluation time, given labels L_x and L_y as well as the corresponding ciphertexts c_x and c_y, compute the label L_z as follows:

$$\begin{aligned} \mathsf{sh}_z^E &\leftarrow L_x \cdot L_y - \mathsf{DDLog}(c_x^{L_y}) - \mathsf{DDLog}(c_y^{L_x}) \\ &\quad + \mathsf{PRF}_{N,\zeta,\lambda}(k_{\mathsf{PRF}}, \mathsf{ord}(z) \| 0) \bmod N^\zeta \\ L_z &\leftarrow \mathsf{DDLog}(c^{\mathsf{sh}_z^E}) + \mathsf{PRF}_{N,\zeta,\lambda}(k_{\mathsf{PRF}}, \mathsf{ord}(z) \| 1) \bmod N^\zeta \end{aligned}$$

Output: For each output z, the garbler computes $\mathsf{out}_z^G \leftarrow \mathsf{DDLog}(c^{K_z}) + \mathsf{PRF}_{N,\zeta,\lambda}(k_{\mathsf{PRF}}, \mathsf{ord}(z) \| 2) \bmod N^\zeta$ and outputs:

- The garbled circuit including: N, c, c_z for each mult. z, and out_z^G for each output z.
- The input encoding information: k, and K_x for each input x.

The evaluator runs $\mathsf{out}_z^E \leftarrow \mathsf{DDLog}(c^{L_z}) + \mathsf{PRF}_{N,\zeta,\lambda}(k_{\mathsf{PRF}}, \mathsf{ord}(z) \| 3) \bmod N^\zeta$ and outputs $(\mathsf{out}_z^E - \mathsf{out}_z^G)$ for each output z.

Fig. 5. (*continued*)

parameter ζ, and let $\ell(\lambda)$ be the bit-length of the corresponding RSA modulus. Let $\varepsilon > 0$. For every $B \leq 2^{(\zeta-2-\varepsilon)\cdot\ell(\lambda)}$, the construction of Fig. 5 is an arithmetic garbling scheme for B-bounded integer computation. Moreover, if a circuit C has n inputs, m outputs, and $s_\times$ multiplication gates then its garbled circuit has bit-size

$$(n + s_\times + 1) \cdot \underbrace{(\zeta+1)\cdot\ell(\lambda)}_{\substack{\text{size of a DJ} \\ \text{ciphertext}}} + \underbrace{m \cdot \zeta \cdot \ell(\lambda)}_{\text{decoding material}}$$

while each input label is $2\zeta \cdot \ell(\lambda)$ bits.

We refer to the full version of this paper [MORS24] for the proof of Theorem 9.

4.2 From CPA Security of Damgård-Jurik

In Sect. 4.2.1 we describe (in a "overview" style, introducing one technique at a time) the techniques which allow us to remove the need for KDM security assumptions. We refer to Sect. 4.2.2 for a formal treatment, with the resulting construction described in Fig. 7.

4.2.1 Techniques to Remove Circular Security. As discussed in Sect. 2.4, the main idea behind removing the dependence on KDM-style assumptions is to use the powerful idea of *key-switching i.e.*, for each layer of multiplications in the circuit the garbler will generate a different Damgård-Jurik key-pair, and only encrypt (functions of) the secret-key in layer $i+1$ under the public key of layer i.

Armed with this idea, there are still a few technical issues to address before we are ready to provide our formal construction. Also, for the construction in this section, keys and labels are defined in such a way as to maintain the invariant

$$\boldsymbol{L}_w = (1, d_i) \cdot w - \boldsymbol{K}_w,$$

where i is the multiplicative depth of wire w.

For each addition gate $z = x + y$, where x, y, z are at depth i_1, i_2, j (with $0 \leq i_1 < j \leq D-1$, $0 \leq i_2 < j \leq D-1$), the parties (1) decompose their labels/keys for x and y as $\langle x\rangle \| \langle d_{i_1} \cdot x\rangle$ and $\langle y\rangle \| \langle d_{i_2} \cdot y\rangle$, (2) perform key-switches to generate $\langle d_j \cdot x\rangle$ and $\langle d_j \cdot y\rangle$, (3) set their label/key for z to be $(\langle x\rangle + \langle y\rangle) \| (\langle d_j \cdot x\rangle + \langle d_j \cdot y\rangle)$. By linearity, these keys and labels indeed satisfy $\boldsymbol{L}_z = (1, d_j) \cdot z - \boldsymbol{K}_z$.

For each multiplication gate $z = x \cdot y$, where x, y, z are at depth i_1, i_2, j (with $0 \leq i_1 < j \leq D-1$, $0 \leq i_2 < j \leq D-1$), we exploit the identity from Eq. (5) restated here for convenience:

$$z = \langle x\rangle_1 \cdot \langle y\rangle_1 - x \cdot \langle y\rangle_0 - y \cdot \langle x\rangle_0 - \langle x\rangle_0 \cdot \langle y\rangle_0 \tag{6}$$

1. *Generate* $\langle z \rangle$. The garbler includes in the garbled circuit encryptions of its shares under the public key of the final layer D i.e., ciphertexts $c_x \xleftarrow{\$} \mathsf{Enc}_{N_D}(\langle x \rangle_0)$ and $c_y \xleftarrow{\$} \mathsf{Enc}_{N_D}(\langle y \rangle_0)$, and the parties (1) perform key switches to generate $\langle d_D \cdot x \rangle$ and $\langle d_D \cdot y \rangle$, (2) compute $\langle x \cdot \langle y \rangle_0 \rangle \leftarrow \mathsf{DDLog}(c_y^{\langle d_D \cdot x \rangle})$ and $\langle y \cdot \langle x \rangle_0 \rangle \leftarrow \mathsf{DDLog}(c_x^{\langle d_D \cdot y \rangle})$, (3) use Eq. (6) to compute shares of z.
2. *Generate* $\langle d_j \cdot z \rangle$. Assume that for $i \in [0, D-1]$, the garbler provided ciphertexts $c'_i \leftarrow \mathsf{Enc}_{N_i}(d_{i+1}^2)$ (these ciphertexts are to be globally reused for every multiplication gate). The garbler performs key-switches to convert $\langle d_{i_1} \cdot x \rangle_0$ to $\langle d_j \cdot x \rangle_0$, and then additionally provides the ciphertext $\mathsf{ct}_x \xleftarrow{\$} \mathsf{Enc}_{N_j}(\langle d_j \cdot x \rangle_0)$. The parties do the following:
   (a) perform key switching to compute $\langle d_{j-1} \cdot x \rangle$, $\langle d_j \cdot x \rangle$, $\langle d_{j-1} \cdot y \rangle$, and $\langle d_j \cdot y \rangle$; // Note that $j > 0$.
   (b) compute $\langle (d_j)^2 \cdot x \rangle \leftarrow \mathsf{DDLog}((c'_{j-1})^{\langle d_{j-1} \cdot x \rangle})$ and $\langle (d_j)^2 \cdot y \rangle \leftarrow \mathsf{DDLog}((c'_{j-1})^{\langle d_{j-1} \cdot y \rangle})$;
   (c) perform $(D-j)$ key-switches using $(c_i)_{i=j}^{D}$ to convert $\langle (d_j)^2 \cdot x \rangle$ to $\langle d_D \cdot d_j \cdot x \rangle$;[8]
   (d) perform $(D-j)$ key-switches using $(c_i)_{i=j}^{D}$ to convert $\langle (d_j)^2 \cdot y \rangle$ to $\langle d_D \cdot d_j \cdot y \rangle$;
   (e) compute $\langle d_j x \cdot \langle y \rangle_0 \rangle \leftarrow \mathsf{DDLog}(c_y^{\langle d_D \cdot d_j x \rangle})$ and $\langle y \cdot \langle d_j \cdot x \rangle_0 \rangle \leftarrow \mathsf{DDLog}(\mathsf{ct}_x^{\langle d_D \cdot y \rangle})$;
   (f) use the identity of Eq. (7), $\langle x \rangle$, $\langle y \rangle$, $\langle d_j x \cdot \langle y \rangle_0 \rangle$, and $\langle y \cdot \langle d_j \cdot x \rangle_0 \rangle$ to compute $\langle d_j \cdot z \rangle$.

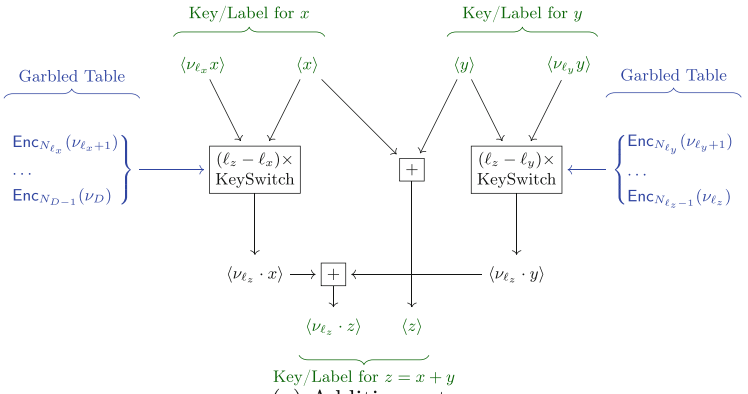
$$d_j \cdot z = \langle d_j \cdot x \rangle_1 \cdot \langle y \rangle_1 - d_j x \cdot \langle y \rangle_0 - y \cdot \langle d_j \cdot x \rangle_0 - \langle d_j \cdot x \rangle_0 \cdot \langle y \rangle_0 \quad (7)$$

With this approach, the garbled circuit contains two ciphertext per multiplication gate (that is, c_x and ct_z for $z = x \cdot y$), plus an additional $2D$ ciphertexts used to perform key switches (that is, the encryptions of d_{i+1} and d_{i+1}^2 under (N_i, sk_i) for $i \in [0, D-1]$).

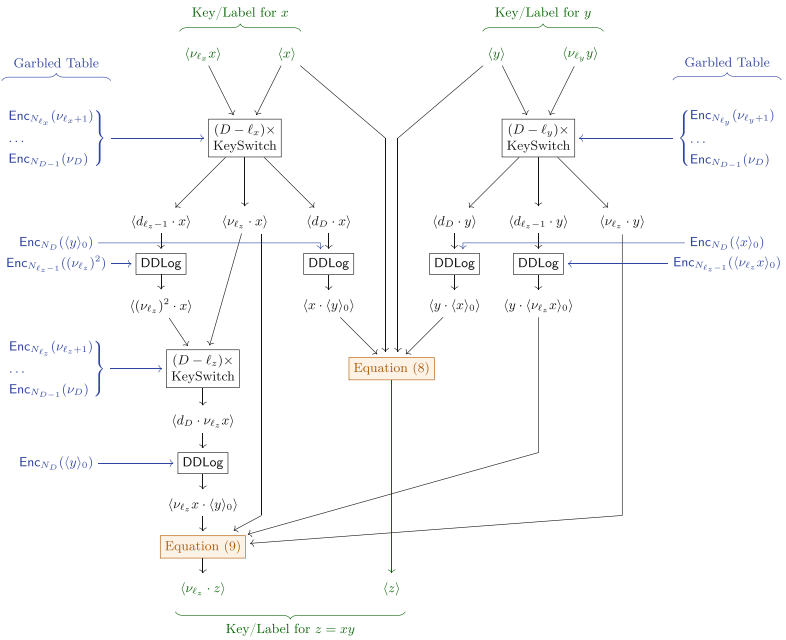
There is a problem however with the scheme as presented. Because the d_i can, in all generality, be as large as N^ζ, and we need authenticated shares of the wire values to fit in the plaintext space, it seems that we can only perform arithmetic over $\mathbb{F}_2$. Fortunately, there is a simple fix to this last problem.

Replace $\langle d \cdot X \rangle$ with $\langle \nu \cdot X \rangle$. Roy and Singh [RS21, Section 4.3] introduced a trick in order to bypass the above problem. If (N, sk) is a keypair for the Damgård-Jurik encryption scheme, and we define $d \leftarrow \mathsf{sk} \cdot (\mathsf{sk}^{-1} \bmod N^\zeta)$ and $\nu \leftarrow N^{-\zeta} \bmod \mathsf{sk}$, then by the Chinese remainders theorem $d \equiv 1 - N^\zeta \nu \bmod (\mathsf{sk} \cdot N^\zeta)$. Since N^ζ is public, this allows parties to locally convert ν-authenticated shares into d-authenticated ones: $\langle d \cdot X \rangle \leftarrow \langle X \rangle - N^\zeta \cdot \langle \nu \cdot X \rangle$. Crucially, since $\nu \leq \mathsf{sk} \leq N$, we can encrypt ν-authenticated shares of wire values as large as $N^{\zeta-1}$. In turn the garbling scheme has rate $(\zeta - 1)/(2\zeta + 2)$ (assuming that the multiplicative depth of the circuit is significantly smaller than its size).

[8] Note that $\langle (d_j)^2 \cdot x \rangle$ can be seen as $\langle d_j \cdot (d_j \cdot x) \rangle$: the leading d_j is the one being "switched out for d_D".



(a) Addition gates.



Key Identities

$$z = \langle x \rangle_1 \cdot \langle y \rangle_1 - x \cdot \langle y \rangle_0 - y \cdot \langle x \rangle_0 - \langle x \rangle_0 \cdot \langle y \rangle_0 \quad (8)$$

$$\nu_{\ell_z} \cdot z = \langle \nu_{\ell_z} \cdot x \rangle_1 \cdot \langle y \rangle_1 - \nu_{\ell_z} x \cdot \langle y \rangle_0 - y \cdot \langle \nu_{\ell_z} \cdot x \rangle_0 - \langle \nu_{\ell_z} \cdot x \rangle_0 \cdot \langle y \rangle_0 \quad (9)$$

(b) Multiplication gates.

Fig. 6. Operations performed by the garbler (resp. evaluator) to convert their keys (resp. labels) for wires x (at multiplicative depth ℓ_x) and y (at multiplicative depth ℓ_y) into a key (resp. label) for wire $z = f(x, y)$ (at multiplicative depth ℓ_z), where $f \in \{+, \times\}$. This only relies on CPA-security of Damgård-Jurik encryption.

Concretely, key-switches are now performed as follows, given $\langle x \rangle$, $\langle \nu \cdot x \rangle$, $c_i \stackrel{\text{def}}{=} \mathsf{Enc}_{N_i}(\nu_{i+1})$:

1. compute $\langle d_i \cdot x \rangle \leftarrow \langle x \rangle - N_i^{\zeta} \langle \nu \cdot x \rangle$;
2. compute $\langle \nu_{i+1} \cdot x \rangle \leftarrow \mathsf{DDLog}(c_i^{\langle d_i \cdot x \rangle})$;
3. optionally, use $\langle x \rangle$ again to compute $\langle d_{i+1} \cdot x \rangle$ and continue key-switching recursively.

We summarise in Figs. 6a and 6b how garbler and evaluator now perform additions and multiplications.

4.2.2 Formal Construction, Theorem Statement, and Proof.

Theorem 10 (AGC with two Ciphertexts per Multiplication from CPA-secure Damgård-Jurik, via DDLog). *Let $\varepsilon > 0$. Let λ be a security parameter. Let $\zeta \geq 2$, assume the CPA-security of the Damgård-Jurik encryption*

Arithmetic Garbling AGC from HSS (CPA-secure Damgård-Jurik variant)

Requires:

- $\mathsf{DJ} = (\mathsf{DJ.KeyGen}, \mathsf{DJ.Enc}, \mathsf{DJ.Dec})$ is the Damgård-Jurik cryptosystem of Fig. 2 and DDLog is the algorithm of Fig. 4.
- $(\mathsf{PRF}_{N,\zeta,\lambda})_{(N,\zeta,\lambda)\in\mathbb{N}^3}$ is a family of pseudorandom functions with outputs in $[N^{\zeta}]$.
- ord is an ordering of the gates of the circuit (*i.e.* a bijection between the set of gates and $[1, s]$, where s is the number of gates).

The Garbling Scheme (gate-by-gate description): Initially, the garbler samples $k_{\mathsf{PRF}} \xleftarrow{\$} \{0,1\}^{\lambda}$ and $(N_\ell, \varphi_\ell) \xleftarrow{\$} \mathsf{DJ.KeyGen}(1^{\lambda})$ then sets $\nu_\ell \leftarrow N^{-\zeta} \bmod \varphi_\ell$ for each $\ell \in [0, D]$ (where D is the multiplicative depth of the circuit). For each $\ell \in [0, D-1]$, the garbler computes $c_\ell \leftarrow \mathsf{DJ.Enc}_{N_\ell}(\nu_{\ell+1})$ and $c'_\ell \leftarrow \mathsf{DJ.Enc}_{N_\ell}(\nu^2_{\ell+1})$; for each input wire x, the garbler samples $\boldsymbol{K}_x \xleftarrow{\$} [N_0^{\zeta}]^2$. The label associated with input x is $\boldsymbol{L}_x \leftarrow (1, \nu_0) \cdot x + \boldsymbol{K}_x \in [N_0^{\zeta}]^2$.

Addition gate $z = x + y$: Let $\ell_x, \ell_y, \ell_z \in [0, D]$ be the multiplicative depths of wires x, y, and z respectively (and observe that $\ell_z = \max(\ell_x, \ell_y)$). Let w, W such that $\{w, W\} = \{x, y\}$, $\ell_w = \min(\ell_x, \ell_y)$ and $\ell_W = \max(\ell_x, \ell_y)$.

Fig. 7. Arithmetic garbled circuit with two ciphertexts per multiplication (and free addition) from CPA-security of Damgård-Jurik (gate-by-gate description).

Addition – Garbling

At garbling time, given $((1, \nu_{\ell_x}), \boldsymbol{K}_x)$ and $((1, \nu_{\ell_y}), \boldsymbol{K}_y)$ as well as corresponding ciphertexts c_x and c_y:

1. **Key switching to lift $\langle \boldsymbol{d_{\ell_w} \cdot w} \rangle$ to $\langle \boldsymbol{d_{\ell_z} \cdot w} \rangle$.** Set $\langle w \rangle_0 \leftarrow \boldsymbol{K}_w.\mathsf{fst}$ and $\langle \nu_{d_w} \cdot w \rangle_0 \leftarrow \boldsymbol{K}_w.\mathsf{snd}$, then, for each $\ell \in [\ell_w, \ell_z - 1]$, compute $\langle \nu_{\ell+1} \cdot w \rangle_0$ and $\langle d_\ell \cdot w \rangle_0$ as follows:

$$\begin{aligned} \langle d_\ell \cdot w \rangle_0 &\leftarrow \langle w \rangle_0 - N_\ell^\zeta \cdot \langle \nu_\ell \cdot w \rangle_0 \\ \langle \nu_{\ell+1} \cdot w \rangle_0 &\leftarrow \mathsf{DDLog}((c_\ell)^{\langle d_\ell \cdot w \rangle_0}) \\ &\quad + \mathsf{PRF}_{N,\zeta,\lambda}(k_{\mathsf{PRF}}, (\mathsf{ord}(w), 0, \ell, 1)) \bmod N^\zeta \ . \end{aligned}$$

2. Homomorphically compute the ciphertext $c_z \leftarrow c_x \cdot c_y$ (to be stored, but not given out as part of the garbled table).
3. Compute $\boldsymbol{K}_z \leftarrow (\boldsymbol{K}_w.\mathsf{fst} + \boldsymbol{K}_W.\mathsf{fst},\ \langle \nu_{\ell_z} \cdot w \rangle_0 + \boldsymbol{K}_W.\mathsf{snd})$ and output $(\nu_{\ell_z}, \boldsymbol{K}_z)$.

Addition – Evaluation

At evaluation time, given labels $\boldsymbol{L}_x$ and $\boldsymbol{L}_y$, compute the label $\boldsymbol{L}_z$ as follows:

1. **Key switching to lift $\langle \boldsymbol{d_{\ell_w} \cdot w} \rangle$ to $\langle \boldsymbol{d_{\ell_z} \cdot w} \rangle$.** Set $\langle w \rangle_1 \leftarrow \boldsymbol{L}_w.\mathsf{fst}$ and $\langle \nu_{\ell_w} \cdot w \rangle_1 \leftarrow \boldsymbol{L}_w.\mathsf{snd}$, then, for each $\ell \in [\ell_w, \ell_z - 1]$, compute $\langle \nu_{\ell+1} \cdot w \rangle_1$ and $\langle d_\ell \cdot w \rangle_1$ as follows:

$$\begin{aligned} \langle d_\ell \cdot w \rangle_1 &\leftarrow \langle w \rangle_1 - N_\ell^\zeta \cdot \langle \nu_\ell \cdot w \rangle_1 \\ \langle \nu_{\ell+1} \cdot w \rangle_1 &\leftarrow \mathsf{DDLog}((c_\ell)^{\langle d_\ell \cdot w \rangle_1}) \\ &\quad + \mathsf{PRF}_{N,\zeta,\lambda}(k_{\mathsf{PRF}}, (\mathsf{ord}(w), 0, \ell, 1)) \bmod N^\zeta \ . \end{aligned}$$

2. Compute and store $c_z \leftarrow c_x \cdot c_y$.
3. Compute and output the label $\boldsymbol{L}_z \leftarrow (\boldsymbol{L}_w.\mathsf{fst} + \boldsymbol{L}_W.\mathsf{fst},\ \langle \nu_{\ell_z} \cdot w \rangle_1 + \boldsymbol{L}_W.\mathsf{snd})$.

Multiplication gate $\boldsymbol{z = x \cdot y}$: Let $\ell_x, \ell_y, \ell_z \in [0, D]$ be the mult. depths of wires x, y, and z respectively ($\ell_z = \max(\ell_x, \ell_y) + 1$).

Fig. 7. (*continued*)

Multiplication – Garbling

At garbling time, given $((1,\nu_{\ell_x}),\boldsymbol{K}_x)$ and $((1,\nu_{\ell_y}),\boldsymbol{K}_y)$, as well as corresponding ciphertexts c_x and c_y, compute $\boldsymbol{K}_z$ as follows:

1. For $w \in \{x,y\}$, set $\langle w\rangle_0 \leftarrow \boldsymbol{K}_w.\mathsf{fst}$ and $\langle \nu_{\ell_w}\cdot w\rangle_0 \leftarrow \boldsymbol{K}_w.\mathsf{snd}$, then, for each $\ell \in [\ell_w, D-1]$, compute $\langle \nu_{\ell+1}\cdot w\rangle_0$ and $\langle d_\ell \cdot w\rangle_0$ as follows:

$$\begin{aligned}\langle d_\ell \cdot w\rangle_0 &\leftarrow \langle w\rangle_0 - N_\ell^\zeta \cdot \langle \nu_\ell \cdot w\rangle_0\\ \langle \nu_{\ell+1}\cdot w\rangle_0 &\leftarrow \mathsf{DDLog}((c_\ell)^{\langle d_\ell\cdot w\rangle_0})\\ &\quad + \mathsf{PRF}_{N,\zeta,\lambda}(k_{\mathsf{PRF}},(\mathsf{ord}(w),1,\ell)) \bmod N^\zeta\ .\end{aligned}$$

   Then, compute $\mathsf{ct}_x \xleftarrow{\$} \mathsf{DJ.Enc}_{N_{\ell_z-1}}(\langle \nu_{\ell_z}\cdot x\rangle_0)$.
2. Compute $\langle z\rangle_0 \leftarrow \big(-\mathsf{DDLog}((c_x)^{\langle d_D\cdot y\rangle_0}) - \mathsf{DDLog}((c_y)^{\langle d_D\cdot x\rangle_0})$
   $-\boldsymbol{K}_x.\mathsf{fst}\cdot\boldsymbol{K}_y.\mathsf{fst}$
   $+\mathsf{PRF}_{N,\zeta,\lambda}(k_{\mathsf{PRF}},(\mathsf{ord}(w),2))\big) \bmod N^\zeta$.
3. Compute $\langle \nu_{\ell_z}^2\cdot x\rangle_0 \leftarrow \mathsf{DDLog}((c'_{\ell_z-1})^{\langle d_{\ell_z-1}\cdot x\rangle_0}) + \mathsf{PRF}_{N,\zeta,\lambda}(k_{\mathsf{PRF}},(\mathsf{ord}(w),3)) \bmod N^\zeta$ then compute $(\langle d_\ell\cdot\nu_{\ell_z}\cdot x\rangle_0)_{\ell=\ell_z+1}^{D}$ recursively as:

$$\langle d_{\ell_z}\cdot\nu_{\ell_z}\cdot x\rangle_0 \leftarrow \langle \nu_{\ell_z}\cdot x\rangle_0 - N_\ell^\zeta\cdot\langle \nu_{\ell_z}^2\cdot x\rangle_0$$

   For $\ell \in [\ell_z, D-1]$,

$$\begin{aligned}\langle d_{\ell+1}\cdot\nu_{\ell_z}\cdot x\rangle_0 \leftarrow &\big(\langle \nu_{\ell_z}\cdot x\rangle_0 - N_\ell^\zeta\cdot\mathsf{DDLog}((c'_\ell)^{\langle d_\ell\cdot\nu_{\ell_z}\cdot x\rangle_0})\\ &+\mathsf{PRF}_{N,\zeta,\lambda}(k_{\mathsf{PRF}},(\mathsf{ord}(w),4,\ell)) \bmod N^\zeta\big)\ .\end{aligned}$$

   Compute:

$$\begin{aligned}\langle \nu_{\ell_z}\cdot z\rangle_0 \leftarrow &\big(-\mathsf{DDLog}(c_y^{\langle d_D\cdot\nu_{\ell_z}\cdot x\rangle_0}) - \mathsf{DDLog}(\mathsf{ct}_x^{\langle d_{\ell_z-1}\cdot y\rangle_0})\\ &-\langle \nu_{\ell_z}\cdot x\rangle_0\cdot(\boldsymbol{K}_y.\mathsf{fst})\\ &+\mathsf{PRF}_{N,\zeta,\lambda}(k_{\mathsf{PRF}},(\mathsf{ord}(w),5)) \bmod N^\zeta\big)\ .\end{aligned}$$

   Finally, compute $c_z \xleftarrow{\$} \mathsf{DJ.Enc}_{N_D}(\boldsymbol{K}_z.\mathsf{fst})$.
4. Produce key $\boldsymbol{K}_z \leftarrow (\langle z\rangle_0, \langle \nu_{\ell_z}\cdot z\rangle_0)$ and garbled table (c_z,ct_x).

Fig. 7. (*continued*)

Multiplication – Evaluation

At evaluation time, given labels $\boldsymbol{L}_x$ and $\boldsymbol{L}_y$ as well as corresponding ciphertexts c_x, c_y, and ct_x, compute the label L_z as follows:

1. For $w \in \{x, y\}$, set $\langle w \rangle_1 \leftarrow \boldsymbol{L}_w.\mathsf{fst}$ and $\langle \nu_{\ell_w} \cdot w \rangle_1 \leftarrow \boldsymbol{L}_w.\mathsf{snd}$, then, for each $\ell \in [\ell_w, D-1]$, compute $\langle \nu_{\ell+1} \cdot w \rangle_1$ and $\langle d_\ell \cdot w \rangle_1$ as follows:

$$\begin{aligned} \langle d_\ell \cdot w \rangle_1 &\leftarrow \langle w \rangle_1 - N_\ell^\zeta \cdot \langle \nu_\ell \cdot w \rangle_1 \\ \langle \nu_{\ell+1} \cdot w \rangle_1 &\leftarrow \mathsf{DDLog}((c_\ell)^{\langle d_\ell \cdot w \rangle_1}) \\ &\quad + \mathsf{PRF}_{N,\zeta,\lambda}(k_{\mathsf{PRF}}, (\mathsf{ord}(w), 1, \ell)) \bmod N^\zeta \ . \end{aligned}$$

2. Compute sh_z as follows:

$$\begin{aligned} \langle z \rangle_1 \leftarrow \big(& \boldsymbol{L}_x.\mathsf{fst} \cdot \boldsymbol{L}_y.\mathsf{fst} - \mathsf{DDLog}((c_x)^{\langle d_D \cdot y \rangle_1}) \\ & - \mathsf{DDLog}((c_y)^{\langle d_D \cdot x \rangle_1}) \\ & + \mathsf{PRF}_{N,\zeta,\lambda}(k_{\mathsf{PRF}}, (\mathsf{ord}(w), 2, \ell)) \bmod N^\zeta \big) \ . \end{aligned}$$

3. Compute $\langle \nu_{\ell_z}^2 \cdot x \rangle_1 \leftarrow \mathsf{DDLog}((c'_{\ell_z - 1})^{\langle d_{\ell_z - 1} \cdot x \rangle_1}) + \mathsf{PRF}_{N,\zeta,\lambda}(k_{\mathsf{PRF}}, (\mathsf{ord}(w), 3)) \bmod N^\zeta$ then compute $(\langle d_\ell \cdot \nu_{\ell_z} \cdot x \rangle_1)_{\ell=\ell_z+1}^{D}$ recursively as:

$$\langle d_{\ell_z} \cdot \nu_{\ell_z} \cdot x \rangle_1 \leftarrow \langle \nu_{\ell_z} \cdot x \rangle_1 - N_\ell^\zeta \cdot \langle \nu_{\ell_z}^2 \cdot x \rangle_1$$

For $\ell \in [\ell_z, D-1]$,

$$\begin{aligned} \langle d_{\ell+1} \cdot \nu_{\ell_z} \cdot x \rangle_1 \leftarrow \big(& \langle \nu_{\ell_z} \cdot x \rangle_1 - N_\ell^\zeta \cdot \mathsf{DDLog}((c'_\ell)^{\langle d_\ell \cdot \nu_{\ell_z} \cdot x \rangle_1}) \\ & + \mathsf{PRF}_{N,\zeta,\lambda}(k_{\mathsf{PRF}}, (\mathsf{ord}(w), 4, \ell)) \bmod N^\zeta \big) \ . \end{aligned}$$

Compute:

$$\begin{aligned} \langle \nu_{\ell_z} \cdot z \rangle_1 \leftarrow \big(& \langle \nu_{\ell_z} \cdot x \rangle_1 \cdot (\boldsymbol{L}_y.\mathsf{fst}) - \mathsf{DDLog}(c_y^{\langle d_D \cdot \nu_{\ell_z} \cdot x \rangle_1}) \\ & - \mathsf{DDLog}(\mathsf{ct}_x^{\langle d_{\ell_{z-1}} \cdot y \rangle_1}) \\ & + \mathsf{PRF}_{N,\zeta,\lambda}(k_{\mathsf{PRF}}, (\mathsf{ord}(w), 5)) \bmod N^\zeta \big) \ . \end{aligned}$$

4. Produce label $\boldsymbol{L}_z \leftarrow (\langle z \rangle_1, \langle \nu_{\ell_z} \cdot z \rangle_1)$.

Fig. 7. (*continued*)

Output: For each output value z, the garbler outputs:

- The garbled circuit, comprised of $(N_\ell)_{\ell\in[0,D]}$, $(c_\ell)_{\ell\in[0,D]}$, c_z for each input or multiplication z, ct_x for each x which is the left input of a multiplication, and $\mathsf{out}_z^G := \boldsymbol{K}_z.\mathsf{fst}$ for each output z.
- Its state (used to compute the input labels), comprised of $(N_\ell, \nu_\ell)_{\ell\in[0,D]}$, and $\boldsymbol{K}_x$ for each input x.

The evaluator outputs $(\mathsf{out}_z^E - \mathsf{out}_z^G)$ for each output z, where $\mathsf{out}_z^E := \boldsymbol{L}_z.\mathsf{fst}$.

Fig. 7. (*continued*)

scheme with parameter ζ, and let $\ell(\lambda)$ be the bit-length of the corresponding RSA modulus. For every $B \le 2^{(\zeta-1-\varepsilon)\cdot\ell(\lambda)}$, the construction of Fig. 7 is an arithmetic garbling scheme for B-bounded integer computation. Moreover, if a circuit C has n inputs, m outputs, $s_\times$ multiplication gates, and has multiplicative depth D, then its garbled circuit has bit-size

$$(n + 2s_\times + 2D) \cdot \underbrace{(\zeta+1)\cdot\ell(\lambda)}_{\substack{\text{size of a DJ}\\ \text{ciphertext}}} + \underbrace{m\cdot\zeta\cdot\ell(\lambda)}_{\substack{\text{decoding}\\ \text{material}}}$$

while each input label is $2\zeta\cdot\ell(\lambda)$ bits.

We refer to the full version [MORS24] for the proof of Theorem 10.

Acknowledgments. This research was supported by: the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme under grant agreement number 803096 (SPEC); the Danish Independent Research Council under Grant-IDs DFF-3103-00077B (CryptoDigi) and DFF-0165-00107B (C3PO); and the DARPA SIEVE program (contract HR001120C0085 "FROMAGER"). Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of DARPA. Distribution Statement "A" (Approved for Public Release, Distribution Unlimited).

References

[AIK11] Applebaum, B., Ishai, Y., Kushilevitz, E.: How to garble arithmetic circuits. In: Ostrovsky, R. (ed.) 52nd FOCS, pp. 120–129. IEEE Computer Society Press (2011)

[ARS24] Abram, D., Roy, L., Scholl, P.: Succinct homomorphic secret sharing. In: Joye, M., Leander, G. (eds.) Advances in Cryptology – EUROCRYPT 2024, pp. 301–330. Springer, Cham, 2024. https://doi.org/10.1007/978-3-031-58751-1_11

[BGG+14] Boneh, D., Gentry, C., Gorbunov, S., Halevi, S., Nikolaenko, V., Segev, G., Vaikuntanathan, V., Vinayagamurthy, D.: Fully key-homomorphic encryption, arithmetic circuit ABE and compact garbled circuits. In: Nguyen, P.Q., Oswald, E. (eds.) EUROCRYPT 2014. LNCS, vol. 8441, pp. 533–556. Springer, Berlin, Heidelberg (2014). https://doi.org/10.1007/978-3-642-55220-5_30

[BGI16] Boyle, E., Gilboa, N., Ishai, Y.: Breaking the circuit size barrier for secure computation under DDH. In: Robshaw, M., Katz, J. (eds.) CRYPTO 2016. LNCS, vol. 9814, pp. 509–539. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53018-4_19

[BLLL23] Ball, M., Li, H., Lin, H., Liu, T.: New ways to garble arithmetic circuits. In: Hazay, C., Stam, M. (eds.) EUROCRYPT 2023. Part II, volume 14005 of LNCS, pp. 3–34. Springer, Cham (2023). https://doi.org/10.1007/978-3-031-30617-4_1

[BMR16] Ball, M., Malkin, T., Rosulek, M.: Garbling gadgets for Boolean and arithmetic circuits. In: Weippl, E.R., Katzenbeisser, S., Kruegel, C., Myers, A.C., Halevi, S. (eds.) ACM CCS 2016, pp. 565–577. ACM Press (2016)

[BRS03] Black, J., Rogaway, P., Shrimpton, T.: Encryption-scheme security in the presence of key-dependent messages. In: Nyberg, K., Heys, H. (eds.) SAC 2002. LNCS, vol. 2595, pp. 62–75. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-36492-7_6

[DJ01] Damgård, I., Jurik, M.: A generalisation, a simplification and some applications of Paillier's probabilistic public-key system. In: Kim, K. (ed.) PKC 2001. LNCS, vol. 1992, pp. 119–136. Springer, Berlin, Heidelberg (2001). https://doi.org/10.1007/3-540-44586-2_9

[FMS19] Fleischhacker, N., Malavolta, G., Schröder, D.: Arithmetic garbling from bilinear maps. In: Sako, K., Schneider, S., Ryan, P.Y.A. (eds.) ESORICS 2019. LNCS, vol. 11736, pp. 172–192. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-29962-0_9

[GKP+13] Goldwasser, S., Kalai, Y., Popa, R.A., Vaikuntanathan, V., Zeldovich, N.: Reusable garbled circuits and succinct functional encryption. In: Boneh, D., Roughgarden, T., Feigenbaum, J. (eds.) 45th ACM STOC, pp. 555–564. ACM Press (2013)

[Hea24] Heath, D.: Efficient arithmetic in garbled circuits. In: Joye, M., Leander, G. (eds.) EUROCRYPT 2024. Part V, volume 14655 of LNCS, pp. 3–31. Springer, Cham (2024). https://doi.org/10.1007/978-3-031-58740-5_1

[HVDH21] Harvey, D., Van Der Hoeven, J.: Integer multiplication in time o(nlog\, n). Ann. Math. **193**(2), 563–617 (2021)

[HY24] Hazay, C., Yang, Y.: Toward malicious constant-rate 2PC via arithmetic garbling. In: Joye, M., Leander, G. (eds.) EUROCRYPT 2024. Part V, volume 14655 of LNCS, pp. 401–431. Springer, Cham (2024). https://doi.org/10.1007/978-3-031-58740-5_14

[LL24] Li, H., Liu, T.: How to garble mixed circuits that combine Boolean and arithmetic computations. In: Joye, M., Leander, G. (eds.) EUROCRYPT 2024. Part VI, volume 14656 of LNCS, pp. 331–360. Springer, Cham (2024). https://doi.org/10.1007/978-3-031-58751-1_12

[MORS24] Meyer, P., Orlandi, C., Roy, L., Scholl, P.: Rate-1 arithmetic garbling from homomorphic secret-sharing. Cryptology ePrint Archive, Report 2024/820 (2024)

[OSY21] Orlandi, C., Scholl, P., Yakoubov, S.: The rise of paillier: homomorphic secret sharing and public-key silent OT. In: Canteaut, A., Standaert, F.-X. (eds.) EUROCRYPT 2021. Part I, volume 12696 of LNCS, pp. 678–708. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-77870-5_24

[Pai99] Paillier, P.: Public-key cryptosystems based on composite degree residuosity classes. In: Stern, J. (ed.) EUROCRYPT'99. LNCS, vol. 1592, pp. 223–238. Springer, Berlin, Heidelberg (1999). https://doi.org/10.1007/3-540-48910-x_16

[RS21] Roy, L., Singh, J.: Large message homomorphic secret sharing from DCR and applications. In: Malkin, T., Peikert, C. (eds.) CRYPTO 2021. LNCS, vol. 12827, pp. 687–717. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-84252-9_23

[Yao82] Yao, A.C.: Protocols for secure computations (extended abstract). In: 23rd FOCS, pp. 160–164. IEEE Computer Society Press, November (1982)

Key-Homomorphic and Aggregate Verifiable Random Functions

Giulio Malavolta(✉)

Bocconi University, Milan, Italy
giulio.malavolta@hotmail.it

Abstract. A verifiable random function (VRF) allows one to compute a random-looking image, while at the same time providing a unique proof that the function was evaluated correctly. VRFs are a cornerstone of modern cryptography and, among other applications, are at the heart of recently proposed proof-of-stake consensus protocols. In this work we initiate the formal study of *aggregate VRFs*, i.e., VRFs that allow for the aggregation of proofs/images into a small digest, whose size is *independent* of the number of input proofs/images, yet it still enables sound verification. We formalize this notion along with its security properties and we propose two constructions: The first scheme is conceptually simple, concretely efficient, and uses (asymmetric) bilinear groups of prime order. Pseudorandomness holds in the random oracle model and aggregate pseudorandomness is proven in the algebraic group model. The second scheme is in the standard model and it is proven secure against the learning with errors (LWE) problem.

As a cryptographic building block of independent interest, we introduce the notion of *key homomorphic VRFs*, where the verification keys and the proofs are endowed with a group structure. We conclude by discussing several applications of key-homomorphic and aggregate VRFs, such as distributed VRFs and aggregate proof-of-stake protocols.

1 Introduction

A verifiable random function (VRF) is a keyed function that satisfies the following cryptographic properties:

- *Pseudorandomness:* The evaluation of the function at any point is computationally indistinguishable from uniform, provided that the distinguisher is not given the secret key.
- *Verifiability:* For any given image, there exists a unique proof of correct evaluation of the function that can be publicly verified, without revealing any additional information about the secret key.

VRFs were first introduced in the work of Micali, Rabin and Vadhan [43], and have been subject of an intense research effort. Over the years, a large amount of VRF constructions have emerged [3,17,24,29,32,33,35,42], improving on the

E. Boyle and M. Mahmoody (Eds.): TCC 2024, LNCS 15367, pp. 98–129, 2025.
https://doi.org/10.1007/978-3-031-78023-3_4

original proposal in terms of computational assumptions, and concrete efficiency. This progress was accompanied by a surge in applications built on top of VRFs, such as randomness generation [17,32], e-cash [1,2], key-escrow [36], zero-knowledge [41], network security [28], and more recently proof-of-stake [15,27] and distributed consensus [31] protocols. Given the large applicability of this primitive, it is safe to say that VRFs have a central role in modern cryptography. However, somewhat surprisingly, *aggregation properties* of VRFs have remained considerably understudied, and we lack even a good set of definitions for their precise security properties. This is in stark contrast with the situation for aggregate signatures [5], where aggregation algorithms and the practical benefits of this primitive have received a lot of attention in the recent years. The goal of this work is to fill this gap in our current understanding, and systematically study the aggregation properties of VRFs.

1.1 Our Contributions

In this work, we initiate the formal study of aggregate VRFs. Our contributions can be summarized as follows.

(a) Definitions. We formally define the notion of aggregate VRFs and propose new security definitions for uniqueness, binding, and pseudorandomness (Sect. 3.3), in the presence of maliciously generated keys. The precise notions turn out to be surprisingly subtle to identify.

(b) Pairing-Based Aggregate VRF. We present a new VRF construction based on asymmetric bilinear pairings (Sect. 4), which satisfies the strong notion of unique proofs (unconditionally), and pseudorandomness assuming the hardness of a variant of the bilinear Diffie-Hellman problem, in the random oracle model. Our scheme is conceptually simple and *concretely* efficient: All algorithms perform only a handful of group operations and the verification is dominated by the computation of three pairings. We propose an algorithm to aggregate VRF images and proofs (Sect. 4.2) and we show that it preserves pseudorandomness and (aggregate) uniqueness, in the algebraic group model [23].

(c) Lattice-Based Aggregate VRF. We present an aggregate VRF construction using general-purpose cryptographic tools (in the full version), that we can prove secure assuming the hardness of the standard learning with errors (LWE) problem. Our proof is in the standard model and we view this construction as our main technical contribution.

(d) Key-Homomorphic VRF. Along the way, we introduce the notion of key-homomorphic VRF (Sect. 3.2), which may be of independent interest. A key-homomorphic VRF is a standard VRF, where the key space $\mathcal{K}$ is endowed with a group structure, with $\oplus$ denoting the group operation. Given a point x, two images $y_0 = \mathsf{Eval}(k_0, x)$ and $y_1 = \mathsf{Eval}(k_1, x)$, and two proofs π_0 and π_1, there exists an efficient procedure that computes

$$y = \mathsf{Eval}(k_0 \oplus k_1, x) \quad \text{and} \quad \pi = \mathsf{Prove}(k_0 \oplus k_1, x).$$

We show that our pairing-based VRF is key-homomorphic for linear function and we consider several extensions of this base construction, such as a threshold variant (Sect. 4.4) and an anonymous version (Sect. 4.3).

(e) Applications. To substantiate the usefulness of our newly defined primitives, we describe how key-homomorphic and aggregate VRFs yield protocols for several applications (Sect. 5). For instance, we show how key-homomorphic VRFs imply a simple distributed VRFs, key homomorphic pseudorandom functions [6] with an additional verification property, and aggregate proof-of-stake consensus protocols.

1.2 Technical Outline

We give a brief technical outline of the main ideas behind our work. We start by describing a construction of key-homomorphic VRF and how it naturally leads to aggregation properties. We then zoom in the security of aggregate VRFs, and subsequently propose an alternative construction with security against the LWE assumption. We conclude by sketching one possible application of aggregate and key-homomorphic VRFs.

Key-Homomorphic VRF. Our starting point is a simple VRF based on asymmetric bilinear pairings of prime order p, with generators (g_1, g_2) and a uniformly sampled group element $S = g_1^s$, that we assume to be available to all participants. We stress that, while technically our scheme requires a common reference string to store S, the setup is completely transparent, and can be sampled with public coins by hashing into the group. A secret-verification key pair for our VRF is defined to be

$$k \leftarrow\$ \mathbb{Z}_p \text{ and } \mathsf{vk} = S^k.$$

On the other hand, the evaluation at point x is computed by pairing:

$$y = e\left(g_1, \mathcal{H}(x)\right)^k$$

where $\mathcal{H}$ is a hash function (modeled as a random oracle) that maps bitstrings into $\mathbb{G}_2$ elements. To prove that the evaluation was done correctly, one can simply compute $\pi = \mathcal{H}(x)^k$, which can be efficiently verified to be a valid proof, by checking

$$y \stackrel{?}{=} e\left(g_1, \pi\right) \quad \text{and} \quad e\left(\mathsf{vk}, \mathcal{H}(x)\right) \stackrel{?}{=} e\left(S, \pi\right).$$

It can be easily verified that both the proof and the image are uniquely determined by the verification key, which implies that the construction satisfies uniqueness. We can also show that the VRF is pseudorandom, with an argument similar to [7], with the crucial difference that we require a different variant of the bilinear Diffie-Hellman assumption. One interesting property of this scheme

is that it is *key homomorphic* for linear functions. To see why, it suffices to observe that

$$\begin{aligned}\mathsf{Eval}(k_0, x) \cdot \mathsf{Eval}(k_1, x) &= e\left(g_1, \mathcal{H}(x)\right)^{k_0} \cdot e\left(g_1, \mathcal{H}(x)\right)^{k_1} \\ &= e\left(g_1, \mathcal{H}(x)\right)^{k_0 + k_1} \\ &= \mathsf{Eval}\left(k_0 + k_1, x\right)\end{aligned}$$

and similarly

$$\begin{aligned}\mathsf{Prove}(k_0, x) \cdot \mathsf{Prove}(k_1, x) &= \mathcal{H}(x)^{k_0} \cdot \mathcal{H}(x)^{k_1} \\ &= \mathcal{H}(x)^{k_0 + k_1} \\ &= \mathsf{Prove}\left(k_0 + k_1, x\right)\end{aligned}$$

and the claim follows by linearity.

Aggregation. The scheme described above suggests a natural aggregation algorithm for VRF images and proofs: Simply apply the group operation to the images and proofs. By the key-homomorphism of the construction, the aggregate is still perfectly verifiable. Unfortunately this proposal suffers from *adaptive attacks*, that is, an attacker could sample maliciously one of the keys included in the aggregate, in such a way that it cancels out the contribution of some honest key. To make things worse, this simple solution is not even *collision resistant*, since any attacker can find two different set of preimages that result in the same aggregate image. We view these two attacks as strong indication that a different solution is needed.

Before describing our actual solution, let us give a somewhat more detailed account of the properties that we consider in this work for aggregate VRFs. The first notion that we consider is that of (i) *aggregate pseudorandomness*, which guarantees that any image of the aggregate VRF is still pseudorandom, so long as at least one of the parties involved is honest. This property is useful for protocols where the aggregate VRF is used as a random beacon, and we want it to be unbiasable by dishonest parties. The second property that we consider is that of (ii) *aggregate binding*, which states that it must be computationally hard to find an aggregate proof for an invalid set of images. This property mimics the collision resistance of a regular hash function, and it is important for scenarios where proofs are aggregated, but we want to prevent the adversary from changing the corresponding VRF images after the fact. Finally, we enforce the notion of (iii) *aggregate uniqueness*, which states that for an aggregate image, there should still exist a *unique* aggregate proof. I.e., the result of the aggregate algorithm is still a VRF.

Give this premise, let us now describe our aggregation algorithm. Instead of taking a simple product, we aggregate proofs and images by computing

$$\tilde{y} = \prod_{i=1}^{n} y_i^{r_i} \quad \text{and} \quad \tilde{\pi} = \prod_{i=1}^{n} \pi_i^{r_i}$$

where $r_1, \ldots, r_n$ are random coefficients computed as a deterministic function of the input. We are then able to show that this aggregation algorithm satisfies both of the desired properties, in the algebraic group model. We specifically mention here that, while this approach is inspired by the work of [4], our analysis is completely different, due to the fact that we are proving a decision property, rather than a search property.

Aggregate VRF from LWE. The construction as described above suffers from two drawbacks: It is proven secure in the random oracle model, and furthermore it is broken by quantum attacks. As a contribution of independent interest, we show a construction of an aggregate VRF (satisfying all of the properties outlined above) from general-purpose tools, that we can prove secure assuming the standard LWE assumption. Our scheme compiles any (non-aggregate) VRF and uses three main ingredients:

- *Function-Binding Hash:* A tree-based hash function Hash, where the root of the tree is statistically bound to any *node* (including intermediate ones) of the tree. The node is fixed at setup time, and the hash satisfies *mode indistinguishability*, in the sense that setups for different nodes are computationally indistinguishable. This notion is new to our work, and generalizes recent hash functions [8,22,34] beyond one-bit predicates. As a contribution of independent interest, we show how to build such a hash function using rate-1 FHE [9,26].
- *Somewhere Extractable Batch Arguments:* Somewhere extractable batch arguments (SE-BARG) for NP for batches of n-many NP statements. The somewhere extractability property says that there exists an index $i \in \{1, \ldots, n\}$ (fixed at setup time), such that, given a trapdoor, one can extract the witness for the i-th statement for the BARG. Furthermore, the index i is computationally hidden. Such SE-BARGs can be built from a variety of assumptions [11–13,37,46], including LWE.
- *Compute-and-Compare Obfuscation:* A compute and compare program, consists of a function f and a target y, and accepts an input x if $f(x) = y$. It is known how to obfuscate such functionalities, provided that y has enough min-entropy conditioned on f, assuming LWE [30,47].

Armed with these tools, we can proceed to outline our aggregation algorithms. To compress images $y_1, \ldots, y_n$ of a given (non-aggregate) VRF, we apply the function

$$\tilde{C} \circ \mathsf{Hash}(y_1, \ldots, y_n)$$

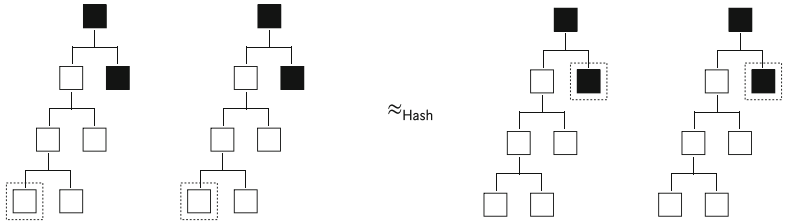
where $\tilde{C}$ is the obfuscation of a dummy (always rejecting) circuit. On the other hand, to aggregate proofs, we compute *two separate BARGs* (σ_0, σ_1) that certify that for each committed y_i, there exists a valid VRF proof π_i. Verification simply checks the validity of the two BARGs. Aggregate binding follows naturally from the collision resistance of Hash, and we discuss the proof of aggregate pseudorandomness and aggregate uniqueness in the following.

Aggregate Pseudorandomness via Randomness Extraction. We show that the construction is pseudorandom, as long as at least one of the input y_i is also pseudorandom. First, we switch the function Hash in binding mode for the i-th index. In the next hybrid, we switch y_i to be uniform and independently sampled. At this point, it is tempting to conclude that, since Hash is statistically bound to y_i, then its output is uniformly distributed. Unfortunately, this intuition is *flawed*, since the other inputs $y_{j\neq i}$ are chosen adversarially, and may bias the output of $\mathsf{Hash}(y_1, \ldots, y_n)$. In other words, we need to argue that Hash acts as a *randomness extractor* for seed-dependent sources. This is in general not possible, as shown by strong impossibility results on randomness extractors for sources that may depend on the random seed [16].

This is where the obfuscation $\tilde{C}$ enters into the picture. To solve this conundrum, we proceed in two steps: First (i) we switch $\tilde{C}$ to be the obfuscation of the circuit that *extracts* y_i from $\mathsf{Hash}(y_1, \ldots, y_n)$. By the extractability of the hash function Hash, this can be implemented efficiently, given a trapdoor. The effect of this step is to "clean up" the output of $\mathsf{Hash}(y_1, \ldots, y_n)$ by adversarial $y_{j\neq i}$. Next, (ii) we open up the construction of compute-and-compare obfuscation [30,47] to show that, if $f(x)$ has high entropy, then $\tilde{C}(x)$ is statistically close to uniform. In other words, $\tilde{C}$ is a good randomness extractor. This allows us to conclude that the output of the aggregate is pseudorandom.

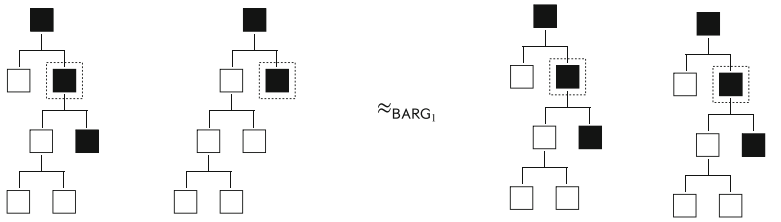
Aggregate Uniqueness via Proof Switching. We show that it is hard to compute valid proofs for an aggregate $y^* \neq \tilde{y}$, where $\tilde{y}$ is honestly computed. For the case of our construction, this means that the two aggregate must correspond to two different roots of the hash tree. Naively, one could hope to reduce this property directly to the underlying VRF, by extracting two valid proofs π_i and π_i^* for two different images y_i and y_i^*. However, the fact that we have two different roots, does not mean that we have a mismatch for exactly the i-th leaf! We have to worry about the fact that the BARG extraction may "miss" the leaf where the mismatch starts. Our strategy to rule this out is to track down the mismatch by iteratively moving towards it, but alternating between the two BARGs, to make sure that we can still extract from one, while arguing indistinguishability for the other. A pictorial description of the process is given in Fig. 1.

In more details, say that we set both BARGs to be binding for the left-most leaf. This allows us to extract two root-to-leaf paths (path_0 and path_1) which, for the sake of this overview, we always assume to be identical (it can be shown by a reduction to collision-resistance that this condition cannot be violated). As we discussed earlier, these root-to-leaf path must lead to a different root than the honestly computed one. Our first observation is that the extracted root-to-leaf paths may be identical to the one in the honest tree in the leafs, but they must diverge at some point (as otherwise they would result in the same root). Let N be the first differing node; in the first hybrid, we make the hash function Hash binding to N. This allows us to freely change the index of the two BARGs without worrying about this node having a different value, since it is now statistically bound to the root. We then change the first BARG to be



(a) Step I: Making the hash tree statistically bound to the differing node.

(b) Step II: Change the root-to-leaf path extracted by the first BARG.



(c) Step III: Change the root-to-leaf path extracted by the second BARG.

Fig. 1. Example of a step of the extraction procedure for a small tree. Black nodes indicate nodes where the two extracted paths (represented in the picture) differ from the honest path. A dotted line around a node indicates that the hash function is statistically bound to that node.

binding on the left-most leaf in the sub-tree spanned by N, and we can use the other BARG to extract the path while we argue indistinguishability. We then switch the roles of the BARGs and apply this operation again. At this point the node N must still differ from the honestly computed one, however it must also

be the case that one of the *children* of N must differ from the honest root-to-leaf path! That is, this argument has allowed us to move one level lower in the tree. Applying this argument iteratively, we will eventually reach the case where the differing node is a leaf, in which case we can appeal to the uniqueness of the base (non-aggregate) VRF, and conclude our proof.

Application: Aggregate Proof of Stake. To exemplify the applicability of aggregate VRFs, we briefly discuss how VRF can be used in proof-of-stake (PoS) protocols to reduce the storage cost by aggregating VRF proofs. A protocol based on similar principles was recently proposed in [21], and we recall here the main ideas. In existing PoS protocols, such as [15,27], users upload their VRF verification key to the blockchain. In each epoch, each user can evaluate the VRF on the slot number (which will play the role as a unique identifier for the epoch) to compute the corresponding image. If the image is less than (a function of) their wealth, then that user is identified as the designated party to generate the new block. Such party can then compute a proof to convince all the other parties that the VRF was evaluated correctly. This way, it can be publicly verified that the user was indeed the winner of the round.

Using a VRF that has aggregation properties, one can compress all VRF proofs into a *single one*, without affecting the ability of the participants to verify that the blockchain was correctly computed: By the aggregate uniqueness property, it is hard for any adversary to compute an aggregate proof for a false set of values. This means that there is no need to store a VRF proof for each epoch, instead this information can be compressed into a *single group element*.

1.3 Related Work

As mentioned before, VRF is a well-studied topic in cryptography, both from a theoretical and practical angle. Thus, many constructions [3,10,17–19,24,29,32,33,35,38,42] have emerged over the years. However, to the best of our knowledge, no scheme that satisfies key-homomorphism (as defined in this work) was known prior to our work. In [40] the notion of aggregation is considered in the context of VRF, except that it is in the *secret key settings*, i.e., the aggregation algorithm requires the knowledge of the secret key of the VRF. This is in contrast with our work, where aggregation is a public procedure. The notion of publicly aggregate VRF was concurrently and independently studied in [14], where a construction from bilinear parings is also proposed. Their scheme is quite different from the one we present, and in particular it does not satisfy the key-homomorphic property (as defined in our work). On the other hand, their construction satisfies the stronger notions of composable security and forward secrecy.

We also mention that the notion of key-homomorphic VRF was discussed (but not formalized) in [21] although the existence of a construction was left as an open problem. In their work, they implicitly construct a key-homomorphic VRF with a small domain and a small co-domain (i.e., the VRF is only defined on a polynomial number of inputs), which suffices for their application.

A related notion is that of verifiable unpredictable functions (VUF), which offers the weaker property of unpredictability, as opposed to pseudorandomness. Although a VUF can be generically transformed into a VRF, e.g., by hashing the image with a random oracle, this transformation does not preserve key homomorphism. Therefore, a key-homomorphic VUF does not immediately imply a key-homomorphic VRF. We also mention that aggregate VUF have been studied in the literature [5,39], but their definitions are not applicable to the notion of VRF, due to the weaker security requirement. Besides the definitional mismatch, building aggregate VRFs is much more complex than the case of VUF, since aggregation must preserve (i) the pseudorandomness and the (ii) uniqueness of the aggregate, whereas none of these properties is relevant in the context of VUFs/signatures. Finally, we mention that key-homomorphic PRFs have been studied [6], and we view our work as a continuation of this line of research, answering a natural question on adding verifiability to key-homomorphic primitives.

1.4 Open Problems

We hope that our work will motivate researchers to further study this cryptographic object, both from a foundational and practical perspective. For instance, it would be interesting to find a construction of key-homomorphic VRFs secure in the standard model, or from post-quantum assumptions. Furthermore, on the theoretical side, an interesting question is whether one can construct a key-homomorphic/aggregate VRF (perhaps satisfying weaker security notions) in the plain model, i.e., without any trusted setup. We also leave open the formalization of the security of aggregate VRFs in the presence of a malicious aggregator. On the other end of the spectrum, an interesting open question is finding more applications of this primitive, and rigorously analyze its security in the context of more complex protocols.

2 Preliminaries

Throughout this work, we write λ to denote the security parameter. We say a function f is negligible in the security parameter λ if $f = \lambda^{-\omega(1)}$. We say an algorithm is efficient if it runs in probabilistic polynomial time (PPT) in the length of its input. For a distribution D, we write $x \leftarrow\!\$\ S$ to denote that x is sampled uniformly at random from D. We say that two distributions D_0 and D_1 are computationally indistinguishable if there exists a negligible function μ such that for all $\lambda \in \mathbb{N}$ and for all PPT algorithms $\mathcal{A}$ it holds that

$$|\Pr[1 \leftarrow \mathcal{A}(x) : x \leftarrow\!\$\ D_0] - \Pr[1 \leftarrow \mathcal{A}(x) : x \leftarrow\!\$\ D_1]| = \mu(\lambda).$$

2.1 Bilinear Groups

Let $\mathcal{G} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, e) \leftarrow\$ \mathsf{GroupGen}(1^\lambda)$ be a generator of an asymmetric bilinear group generated by g_1 and g_2 of prime order p, with an efficiently computable pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$. We also assume that it is efficient to test whether a given element g is a member of the group $\mathbb{G}_1$, which is sometimes referred to as the *certified group* settings, see e.g. [44] for details. We recall the following variant of the Bilinear Diffie-Hellman (BDH) assumption.

Definition 1 (BDH Assumption). *We say that a bilinear group generator* $\mathsf{GroupGen}$ *is BDH-hard, if the following distributions are computationally indistinguishable*

$$\left(g_1, g_2, g_1^a, g_1^b, g_2^a, g_2^b, g_2^c, e(g_1, g_2)^{\frac{bc}{a}}\right) \approx \left(g_1, g_2, g_1^a, g_1^b, g_2^a, g_2^b, g_2^c, e(g_1, g_2)^r\right)$$

where $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, e) \leftarrow\$ \mathsf{GroupGen}(1^\lambda)$ *and* $(a, b, c, r) \leftarrow\$ \mathbb{Z}_p^4$.

We also define two less standard variants of this assumption, where the adversary is given additional extra group elements in its view. In Sect. 6, we show that both assumptions hold unconditionally in the generic group model (GGM).

Definition 2 (Augmented BDH Assumption). *We say that a bilinear group generator* $\mathsf{GroupGen}$ *is ABDH-hard, if the following distributions are computationally indistinguishable*

$$\begin{aligned}&\left(g_1, g_2, g_1^a, g_1^b, g_2^a, g_2^b, g_2^c, e(g_1, g_2)^{\frac{c}{a}}, e(g_1, g_2)^{\frac{c^2}{a}}, e(g_1, g_2)^{\frac{bc}{a}}\right)\\ &\approx \left(g_1, g_2, g_1^a, g_1^b, g_2^a, g_2^b, g_2^c, e(g_1, g_2)^{\frac{c}{a}}, e(g_1, g_2)^{\frac{c^2}{a}}, e(g_1, g_2)^r\right)\end{aligned}$$

where $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, e) \leftarrow\$ \mathsf{GroupGen}(1^\lambda)$ *and* $(a, b, c, r) \leftarrow\$ \mathbb{Z}_p^4$.

Definition 3 (External-Inverted BDH Assumption). *We say that a bilinear group generator* $\mathsf{GroupGen}$ *is XIBDH-hard, if the following distributions are computationally indistinguishable*

$$\left(g_1, g_2, g_1^a, g_1^{ab}, g_1^{ac}, g_1^{b/c}, g_2^d, g_2^{dc}\right) \approx \left(g_1, g_2, g_1^a, g_1^{ab}, g_1^{ac}, g_1^r, g_2^d, g_2^{dc}\right)$$

where $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, e) \leftarrow\$ \mathsf{GroupGen}(1^\lambda)$ *and* $(a, b, c, d, r) \leftarrow\$ \mathbb{Z}_p^5$.

3 Definitions

We recall the notion of a verifiable random function (VRF) [43]. Informally, a VRF is a keyed function

$$\mathrm{VRF} : \mathcal{K} \times \{0,1\}^\lambda \to \mathcal{Y}$$

that behaves like a random function, and simultaneously allows one to verify the validity of its outputs.

3.1 Verifiable Random Functions

We formally define the notion of a VRF, we adopt the standard definition [43], except for two syntactical modifications: First, we require the verification key to be a deterministic and *bijective* function of the secret key. This is needed for our definition of aggregate pseudorandomness (Definition 9). Furthermore, we allow for a one-time trusted setup.

Definition 4 (VRF). *A verifiable random function (VRF) is a tuple of polynomial-time algorithms with the following syntax.*

- $\mathsf{Setup}(1^\lambda)$*: On input the security parameter* 1^λ*, the setup algorithm returns the common reference string* crs*.*
- $\mathsf{Gen}(\mathsf{crs})$*: On input the common reference string* crs*, the generation algorithm samples a secret key* k *and returns* $k \in \mathcal{K}$ *and the verification key* $\mathsf{vk} = \mathsf{VKey}(k)$*, where* VKey *is a bijective function.*
- $\mathsf{Eval}(k, x)$*: On input the secret key* $k \in \mathcal{K}$ *and some point* $x \in \{0,1\}^\lambda$*, the evaluation algorithm returns an image* $y \in \mathcal{Y}$*.*
- $\mathsf{Prove}(k, x)$*: On input the secret key* $k \in \mathcal{K}$ *and some point* $x \in \{0,1\}^\lambda$*, the prover algorithm returns a proof* $\pi \in \mathcal{P}$*.*
- $\mathsf{Verify}(\mathsf{vk}, x, y, \pi)$*: On input the verification key* vk*, some point* $x \in \{0,1\}^\lambda$*, an image* $y \in \mathcal{Y}$*, and a proof* $\pi \in \mathcal{P}$*, the verification algorithm returns a bit* $b \in \{0,1\}$ *denoting acceptance or rejection.*

For correctness, we require that for all $\lambda \in \mathbb{N}$, all crs in the support of $\mathsf{Setup}(1^\lambda)$, all key pairs (k, vk) in the support of the $\mathsf{Gen}(\mathsf{crs})$ algorithm, and all points $x \in \{0,1\}^\lambda$ it holds that

$$\mathsf{Verify}(\mathsf{vk}, x, \mathsf{Eval}(k, x), \mathsf{Prove}(k, x)) = 1.$$

Next, we recall the security requirements of a VRF. The first property demands that it should be impossible to find a valid proof π^* for an invalid image y^*.

Definition 5 (Uniqueness). *A VRF satisfies* uniqueness *if for* $\lambda \in \mathbb{N}$*, all* crs *in the support of* $\mathsf{Setup}(1^\lambda)$*, all verification keys* vk*, all points* $x \in \{0,1\}^\lambda$*, and all images* $(y_0, y_1) \in \mathcal{Y}^2$ *and proofs* $(\pi_0, \pi_1) \in \mathcal{P}^2$ *it holds that*

$$1 = \mathsf{Verify}(\mathsf{vk}, x, y_0, \pi_0) = \mathsf{Verify}(\mathsf{vk}, x, y_1, \pi_1) \implies y_0 = y_1.$$

Finally, we recall the standard notion of pseudorandomness, which requires that the adversary should not be able to distinguish the output of a VRF from a uniformly sampled function, even given access to an oracle that computes images an proofs on arbitrary points chosen by the distinguisher.

Definition 6 (Pseudorandomness). *A VRF satisfies* pseudorandomness *if there exists a negligible function* μ *such that for* $\lambda \in \mathbb{N}$ *and all admissible PPT adversaries* $\mathcal{A}$ *it holds that*

$$\left| \frac{1}{2} - \Pr\left[1 \leftarrow \mathsf{ExpRandom}_{\mathcal{A}}(1^\lambda)\right] \right| = \mu(\lambda)$$

where the experiment $\mathsf{ExpRandom}_{\mathcal{A}}$ *is defined as follows.*

- *The challenger samples a reference string* $\mathsf{crs} \leftarrow\!\$\ \mathsf{Setup}(1^\lambda)$ *and a key pair* $(k, \mathsf{vk}) \leftarrow\!\$\ \mathsf{Gen}(\mathsf{crs})$ *and sends* $(\mathsf{crs}, \mathsf{vk})$ *to* $\mathcal{A}$.
- $\mathcal{A}$ *can query adaptively and at any time an oracle on input* x_i *and receives back from the challenger a pair* (y_i, π_i) *where* $y_i \leftarrow \mathsf{Eval}(k, x_i)$ *and* $\pi_i \leftarrow \mathsf{Prove}(k, x_i)$.
- *At any point (including between evaluation queries)* $\mathcal{A}$ *can query a challenge input* x^*. *The challenger flips a coin* $b \leftarrow\!\$\ \{0,1\}$ *and computes*

$$\begin{cases} y^* \leftarrow \mathsf{Eval}(k, x^*) & \text{if } b = 0 \\ y^* \leftarrow\!\$\ \mathcal{Y} & \text{if } b = 1 \end{cases}.$$

- *In the end of the experiment,* $\mathcal{A}$ *outputs a guess* b^*. *The experiment returns* 1 *if and only if* $b^* = b$.

Furthermore, we say that $\mathcal{A}$ *is admissible if* $x^* \notin Q$, *where* Q *denotes the set of points queried to the oracle as defined above.*

3.2 Key Homomorphism

We define the central object of this work, namely the notion of *key-homomorphic* VRF (KH-VRF). A KH-VRF is defined like a standard VRF with some additional structural properties, which allows one to meaningfully combine the images and the proofs computed under different keys. To formally define this notion, we will endow the key, the image, and the proof domain with a group structure, which allows us to state this additional property.

Definition 7 (Key Homomorphism). *A VRF is* key homomorphic *if* $(\mathcal{K}, \oplus)$, $(\mathcal{Y}, \otimes)$, $(\mathcal{P}, \otimes)$ *are groups and for all* $(k_0, k_1) \in \mathcal{K}^2$ *and all* $x \in \{0,1\}^\lambda$ *it holds that*

- $\mathsf{Eval}(k_0, x) \otimes \mathsf{Eval}(k_1, x) = \mathsf{Eval}(k_0 \oplus k_1, x)$, *and*
- $\mathsf{Prove}(k_0, x) \otimes \mathsf{Prove}(k_1, x) = \mathsf{Prove}(k_0 \oplus k_1, x)$.

Looking ahead at our instantiation in Sect. 4, the group $(\mathcal{K}, \oplus)$ will consist of the additive group $\mathbb{Z}_p$, for some prime p, whereas $(\mathcal{Y}, \otimes)$ and $(\mathcal{P}, \otimes)$ will be two multiplicative groups of order p.

3.3 Aggregation

Given the above defined homomorphic property, it is natural to derive an aggregation Agg algorithm that allows us to compress n VRFs into a single one, while at the same time outputting a proof of validity for the aggregate. We formally define the correctness of the algorithm in the following. Importantly, we define the algorithms to be *deterministic* so that running the same aggregation twice leads to exactly the same output.

Definition 8 (Aggregation). *A VRF is* aggregatable *if there exists a triple of deterministic polynomial-time algorithms with the following syntax.*

- $\mathsf{Agg}(x, \{\mathsf{vk}_i, y_i\}_{i=1\ldots n})$*: On input a point* $x \in \{0,1\}^\lambda$*,* n *verification keys* $(\mathsf{vk}_1, \ldots, \mathsf{vk}_n)$ *and images* $(y_1, \ldots, y_n) \in \mathcal{Y}^n$*, the aggregation algorithm outputs an aggregate key* $\tilde{\mathsf{vk}}$ *and an aggregate image* $\tilde{y}$*.*
- $\mathsf{AggProve}(x, \{\mathsf{vk}_i, \pi_i\}_{i=1\ldots n})$*: On input a point* $x \in \{0,1\}^\lambda$*,* n *verification keys* $(\mathsf{vk}_1, \ldots, \mathsf{vk}_n)$*, and proofs* $(\pi_1, \ldots, \pi_n) \in \mathcal{P}^n$*, the aggregate proof algorithm outputs an aggregate proof* $\tilde{\pi}$*.*
- $\mathsf{AggVerify}(\{\mathsf{vk}_i\}_{i=1\ldots n}, x, \tilde{y}, \tilde{\pi})$*: On input* n *verification keys* $(\mathsf{vk}_1, \ldots, \mathsf{vk}_n)$*, a point* $x \in \{0,1\}^\lambda$*, and an aggregate image-proof pair* $(\tilde{y}, \tilde{\pi})$ *the aggregate verification algorithm outputs a bit* $b \in \{0,1\}$ *denoting acceptance or rejection.*

The correctness of the aggregation algorithm requires that the aggregated image-proof pair correctly verifies, provided that the inputs are all correctly formed. More formally, we require that for all $\lambda \in \mathbb{N}$, all polynomials $n = n(\lambda)$, all crs in the support of $\mathsf{Setup}(1^\lambda)$, all key pairs (k_i, vk_i) in the support of the $\mathsf{Gen}(\mathsf{crs})$ algorithm, and all points $x \in \{0,1\}^\lambda$ it holds that

$$\mathsf{AggVerify}\left(\{vk_i\}_{i=1\ldots n}, x, \tilde{y}, \tilde{\pi}\right) = 1,$$

where

$$(\tilde{\mathsf{vk}}, \tilde{y}) \leftarrow \mathsf{Agg}\left(x, \{\mathsf{vk}_i, \mathsf{Eval}(k_i, x)\}_{i=1\ldots n}\right)$$

and

$$\tilde{\pi} \leftarrow \mathsf{AggProve}\left(x, \{\mathsf{vk}_i, \mathsf{Prove}(k_i, x)\}_{i=1\ldots n}\right).$$

Next we present formal definitions for the security of aggregate VRFs. In all of our definitions, we model the aggregation process as performed by an honest party, and therefore we always assume that the aggregation algorithms are run *honestly*. We leave the formalization of security in the presence of a malicious aggregator as ground for future work.

Aggregate Pseudorandomness. A desirable property for an aggregate VRF is that the output of the aggregation algorithm should still satisfy pseudorandomness, even if some of the keys are controlled by the adversary. It turns out that formalizing the correct version of this property is surprisingly subtle.

To understand our definitional choice, consider the following natural (but flawed) attempt at a formal definition: Given the adversary oracle access to the challenge VRF, we allow the adversary to additionally return some verification keys of its choice $\{\mathsf{vk}_i\}_{i=1\ldots n}$, along with the corresponding image-proof pairs $\{y_i, \pi_i\}_{i=1\ldots n}$. The challenger then checks whether the proofs verify, and if so it computes either

$$y^* \leftarrow \mathsf{Agg}\left(x, \mathsf{vk}, y, \{\mathsf{vk}_i, y_i\}_{i=1\ldots n}\right) \quad \text{or} \quad y^* \leftarrow\!\!\$\ \mathcal{Y}.$$

While this is a perfectly well-defined notion, we argue that this definition is *too weak*, since it does not allow the adversary to set its key depending on the honest keys. To see why this is the case, consider the simplified case where the aggregation of images is simply their sum $\tilde{y} = \sum_i y_i$. The adversary may

launch an *adaptive attack* by setting its own verification key vk^* depending on the verification key of the challenger vk, in such a way that $y^* = -y$. This way, the contribution of y to the aggregate $\tilde{y}$ is canceled out, and the output of the aggregation algorithm is perfectly predictable by the adversary, even before seeing the answer to the challenge query. Nevertheless, this attacker would not be able to win the experiment as outlined above, because it would not be able to compute y^* *before* seeing y, and thus cannot send it as part of the query to the challenge oracle. This is a strong indication that one needs a stronger definition of security for aggregate pseudorandomness.

To obviate this problem, our idea is to make the life of the attacker easier, by simply delegating to the challenger the task of computing the image of a verification key sent by the adversary. In general, this is not an efficient procedure, but we can bypass this obstacle by letting the challenger run in unbounded time. Note that the output of this procedure is always well-defined, since the function VKey is bijective, and therefore its inverse is uniquely determined. We present the formal definition below.

Definition 9 (Aggregate Pseudorandomness). *A VRF satisfies* aggregate pseudorandomness *if there exists a negligible function μ such that for $\lambda \in \mathbb{N}$ and all admissible PPT adversaries $\mathcal{A}$ it holds that*

$$\left| \frac{1}{2} - \Pr\left[1 \leftarrow \mathsf{ExpAggRandom}_{\mathcal{A}}(1^\lambda)\right] \right| = \mu(\lambda)$$

where the experiment $\mathsf{ExpAggRandom}_{\mathcal{A}}$ is defined as follows.

- *The challenger samples a reference string $\mathsf{crs} \leftarrow\!\!\$\ \mathsf{Setup}(1^\lambda)$ and a key pair $(k, \mathsf{vk}) \leftarrow\!\!\$\ \mathsf{Gen}(\mathsf{crs})$ and sends $(\mathsf{crs}, \mathsf{vk})$ to $\mathcal{A}$.*
- *$\mathcal{A}$ can query adaptively and at any time an oracle on input x_i and receives back from the challenger a pair (y_i, π_i) where $y_i \leftarrow \mathsf{Eval}(k, x_i)$ and $\pi_i \leftarrow \mathsf{Prove}(k, x_i)$.*
- *At any point (including between evaluation queries) $\mathcal{A}$ can query a challenge input x^* along with some challenge keys $\{\mathsf{vk}_i\}_{i=1\ldots n}$. The challenger (inefficiently) computes $k_i \leftarrow \mathsf{VKey}^{-1}(\mathsf{vk}_i)$ and sets $y_i \leftarrow \mathsf{Eval}(k_i, x^*)$. Then it flips a coin $b \leftarrow\!\!\$\ \{0,1\}$ and computes*

$$\begin{cases} y^* \leftarrow \mathsf{Agg}\left(x^*, \mathsf{vk}, \mathsf{Eval}(k, x^*), \{\mathsf{vk}_i, y_i\}_{i=1\ldots n}\right) & \text{if } b = 0 \\ y^* \leftarrow\!\!\$\ \mathcal{Y} & \text{if } b = 1 \end{cases}.$$

- *In the end of the experiment, $\mathcal{A}$ outputs a guess b^*. The experiment returns 1 if and only if $b^* = b$.*

Furthermore, we say that $\mathcal{A}$ is admissible if $x^* \notin Q$, where Q denotes the set of points queried to the oracle as defined above.

Aggregate Binding. We also require that no attacker should be able to find two different pre-images for a given aggregate. This property is useful in settings where one stores an aggregate proof, and wants to prevent changes the

corresponding images after the fact. Given that the size of the aggregate is independent of n, we can only hope to achieve a computational guarantee, where finding collision is only computationally hard. We present a formal definition below. We remark that, since the aggregation algorithm is deterministic, there is no need to provide the aggregate as an input to the adversary explicitly, since it can be recomputed given the information that the adversary already possesses.

Definition 10 (Aggregate Binding). *A VRF satisfies* aggregate binding *if there exists a negligible function μ such that for $\lambda \in \mathbb{N}$ and all PPT adversaries $\mathcal{A}$ it holds that*

$$\Pr\left[1 \leftarrow \mathsf{ExpAggBind}_{\mathcal{A}}(1^\lambda)\right] = \mu(\lambda)$$

where the experiment $\mathsf{ExpAggBind}_{\mathcal{A}}$ is defined as follows.

- *The challenger samples a reference string $\mathsf{crs} \leftarrow\!\!\$\ \mathsf{Setup}(1^\lambda)$ and sends it to $\mathcal{A}$.*
- *$\mathcal{A}$ returns a point x^*, n verification keys $\mathsf{vk}_1, \ldots, \mathsf{vk}_n$, two tuples $(y_1, \ldots, y_n)$ and $(y_1^*, \ldots, y_n^*)$, and proofs $(\pi_1, \ldots, \pi_n)$.*
- *The challenger computes*

$$\tilde{\pi} \leftarrow \mathsf{AggProve}\left(x^*, \{\mathsf{vk}_i, \pi_i\}_{i=1\ldots n}\right) \quad \text{and} \quad \tilde{y} \leftarrow \mathsf{Agg}\left(x^*, \{\mathsf{vk}_i, y_i^*\}_{i=1\ldots n}\right)$$

and outputs 1 if and only if:

$$\left\{\mathsf{Verify}(\mathsf{vk}_i, x^*, y_i, \pi_i) \stackrel{?}{=} 1\right\}_{i=1\ldots n} \quad \text{and} \quad \mathsf{AggVerify}\left(\{vk_i\}_{i=1\ldots n}, x, \tilde{y}, \tilde{\pi}\right) \stackrel{?}{=} 1$$

and furthermore $(y_1, \ldots, y_n) \neq (y_1^*, \ldots, y_n^*)$.

Aggregate Uniqueness. Finally, we require that an aggregate key must also be a valid VRF key, and in particular it must satisfy uniqueness. We formalize this property below.

Definition 11 (Aggregate Uniqueness). *A VRF satisfies* aggregate uniqueness *if for $\lambda \in \mathbb{N}$, all polynomials $n = n(\lambda)$, all crs in the support of $\mathsf{Setup}(1^\lambda)$, all verification keys vk_i, and all points $x \in \{0,1\}^\lambda$, it holds that $\tilde{\mathsf{vk}}$ satisfies uniqueness (Definition 5), where*

$$(\tilde{\mathsf{vk}}, \tilde{y}) \leftarrow \mathsf{Agg}\left(x, \{\mathsf{vk}_i, \mathsf{Eval}(k_i, x)\}_{i=1\ldots n}\right).$$

4 Key-Homomorphic VRFs from Bilinear Groups

We describe our main construction for a KH-VRF. We begin by introducing a simple scheme with key-homomorphic properties, then we show how to extend it to achieve several extra properties.

4.1 Base Scheme

We present our base KH-VRF in the following. The construction assumes the existence of a hash function $\mathcal{H}$ where

$$\mathcal{H} : \{0,1\}^\lambda \to \mathbb{G}_2$$

is modelled as a random oracle. The algorithms are specified below.

- $\mathsf{Setup}(1^\lambda)$: Sample a bilinear group

$$(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, e) \leftarrow\!\$\ \mathsf{GroupGen}(1^\lambda)$$

  along with an integer $s \leftarrow\!\$\ \mathbb{Z}_p$. Then compute $S = g_1^s$ and set the crs to (g_1, g_2, S).
- $\mathsf{Gen}(\mathsf{crs})$: Sample $k \leftarrow\!\$\ \mathbb{Z}_p$ and set vk to S^k. Return (k, vk).
- $\mathsf{Eval}(k, x)$: Compute and output

$$y = e\left(g_1, \mathcal{H}(x)\right)^k .$$

- $\mathsf{Prove}(k, x)$: Compute and output $\pi = \mathcal{H}(x)^k$.
- $\mathsf{Verify}(\mathsf{vk}, x, y, \pi)$: Accept if $\mathsf{vk} \in \mathbb{G}_1$ and

$$e\left(\mathsf{vk}, \mathcal{H}(x)\right) \stackrel{?}{=} e\left(S, \pi\right) \quad \text{and} \quad y \stackrel{?}{=} e\left(g_1, \pi\right).$$

We remark that, although we described the setup algorithm as sampling the secret integer s explicitly, an alternative (but equivalent) sampling procedure would be to derive S by hashing into the group $\mathbb{G}_1$. This way, the setup can be done in a completely *transparent* manner, i.e., using public random coins.

It is easy to check that the scheme is correct by simply substituting the variables

$$e\left(\mathsf{vk}, \mathcal{H}(x)\right) = e\left(g_1, \mathcal{H}(x)\right)^{ks} = e\left(g_1^s, \mathcal{H}(x)^k\right) = e\left(S, \pi\right)$$

and

$$y = e\left(g_1, \mathcal{H}(x)\right)^k = e\left(g_1, \mathcal{H}(x)^k\right) = e\left(g_1, \pi\right).$$

To establish that the scheme is also key-homomorphic, it is enough to observe that for all polynomials $n = n(\lambda)$ it holds that

$$\prod_{i=1}^{n} \mathsf{Eval}(k_i, x) = \prod_{i=1}^{n} e\left(g_1, \mathcal{H}(x)\right)^{k_i} = e\left(g_1, \mathcal{H}(x)\right)^{\sum_{i=1}^n k_i} = \mathsf{Eval}\left(\sum_{i=1}^{n} k_i, x\right)$$

and similarly

$$\prod_{i=1}^{n} \mathsf{Prove}(k_i, x) = \prod_{i=1}^{n} \mathcal{H}(x)^{k_i} = \mathcal{H}(x)^{\sum_{i=1}^n k_i} = \mathsf{Prove}\left(\sum_{i=1}^{n} k_i, x\right).$$

Furthermore, the scheme satisfies uniqueness, since π is uniquely determined by S, vk, and x.

Analysis. We state the main theorem of this section, namely that the base scheme is secure if the BDH assumption holds.

Theorem 1. *If the group generator* $\mathsf{GroupGen}$ *is BDH-hard, then the construction as described above satisfies pseudorandomness, in the random oracle model.*

Proof. The pseudorandomness of our VRF follows from a reduction to the BDH assumption (Definition 1). Recall that the assumption stipulates that the following distributions are computationally indistinguishable

$$\left(g_1, g_2, g_1^a, g_1^b, g_2^a, g_2^b, g_2^c, e(g_1,g_2)^{\frac{bc}{a}}\right) \approx \left(g_1, g_2, g_1^a, g_1^b, g_2^a, g_2^b, g_2^c, e(g_1,g_2)^r\right)$$

where $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, e) \leftarrow\$ \mathsf{GroupGen}(1^\lambda)$ and $(a,b,c,r) \leftarrow\$ \mathbb{Z}_p^4$. By a variable renaming, this is equivalent to distinguishing between

$$\left(g_1, g_2, g_1^s, g_1^{ks}, g_2^s, g_2^{ks}, g_2^z, e(g_1,g_2)^{kz}\right) \approx \left(g_1, g_2, g_1^s, g_1^{ks}, g_2^s, g_2^{ks}, g_2^z, e(g_1,g_2)^r\right).$$

On input a sample from either distribution, the reduction sets $S = g_1^s$ and $\mathsf{vk} = g_1^{ks}$, then it activates the adversary $\mathcal{A}$. Let Q be the number of queries of $\mathcal{A}$ to the random oracle, the reduction guesses an index $i^* \in \{1, \ldots, Q\}$ and simulates the oracle as follows: For $i \neq i^*$, on input x_i the reduction programs the random oracle to

$$\mathcal{H}(x_i) = (g_2^s)^{\rho_i} \text{ where } \rho_i \leftarrow\$ \mathbb{Z}_p.$$

On the other hand, on input x_{i^*} the reduction sets

$$\mathcal{H}(x_{i^*}) = g_2^z.$$

Next, we show how the reduction simulates the queries to the VRF evaluation oracle. Without loss of generality, we assume that the adversary already queried the input x_i to the random oracle (otherwise, the reduction can perform the query itself before answering to $\mathcal{A}$). If $x_i = x_{i^*}$, the reduction aborts and returns a random bit. Otherwise, the reduction computes

$$\pi_i = \left(g_2^{ks}\right)^{\rho_i} \quad \text{and} \quad y_i = e(g_1, \pi)$$

by fetching the appropriate value of ρ_i. On input the challenge query x^*, the reduction checks if $x^* = x_{i^*}$ and returns a random bit if this is not the case. Otherwise, it computes

$$y^* = \left\{e(g_1,g_2)^{kz}, e(g_1,g_2)^r\right\}$$

i.e., it simply returns the last group element from the challenge tuple. The reduction returns whatever bit $\mathcal{A}$ returns.

First, observe that the reduction is efficient, as it only performs basic group operations. Consider the case where the reduction guesses correctly the query to

the random oracle that contains the challenge input x^*. In this case, the reduction perfectly simulates the queries to the oracle offered by the pseudorandomness experiment, since

$$\pi_i = \left(g_2^{ks}\right)^{\rho_i} = \left(g_2^{s\rho_i}\right)^k = \mathcal{H}(x_i)^k$$

and $s\rho_i$ is uniformly distributed in $\mathbb{Z}_p$. Finally, note that the case where the last group element is uniform perfectly simulates the experiment where $b = 1$, whereas in the other case

$$y^* = e(g_1, g_2)^{kz} = e(g_1, g_2^z)^k = e\left(g_1, \mathcal{H}(x^*)\right)^k$$

the view is identical for the case where $b = 0$. Since the reduction guesses correctly with probability at least $1/Q$, which is inverse polynomial, the success probability of the reduction in distinguishing the two distributions is at most a polynomial factor smaller than to that of $\mathcal{A}$. This contradicts the BDH assumption and concludes our proof.

4.2 Aggregation

The base scheme described above enjoys useful structural properties that make it amenable to aggregation. A naive proposal to aggregate VRF would be to simply compute the product of the images and proofs, i.e.,

$$\tilde{y} = \prod_{i=1}^{n} y_i = \prod_{i=1}^{n} e\left(g_1, \mathcal{H}(x)\right)^{k_i} \quad \text{and} \quad \tilde{\pi} = \prod_{i=1}^{n} \pi_i = \prod_{i=1}^{n} \mathcal{H}(x)^{k_i}.$$

This is syntactically correct, since it corresponds to the output of the Eval and Prove algorithms on with key $\tilde{k} = \sum_{i=1}^{n} k_i$. Unfortunately this simple proposal suffers from many issues: (i) First of all, it does not achieve aggregate pseudorandomness, since it is vulnerable to *rogue key attacks*. To see why, it suffices to observe that, on input an honest key $\mathsf{vk} = S^k$, an attacker could set their key to $\mathsf{vk}^{-1} = S^{-k}$ (which is efficiently computable). This way, the output of the aggregate on any input x is simply

$$e\left(g_1, \mathcal{H}(x)\right)^k \cdot e\left(g_1, \mathcal{H}(x)\right)^{-k} = 1$$

which is clearly not pseudorandom. This particular problem can be fixed by adding non-interactive zero-knowledge proof (NIZK) [20] to the public key, however the resulting VRF would no longer be key-homomorphic. Moreover, even with the NIZK in place, (ii) this construction does not satisfy aggregate binding. To illustrate this last point, observe that it is easy to compute an x^* and a pair $(y_1, \ldots, y_n) \neq (y_1^*, \ldots, y_n^*)$ such that

$$\mathsf{Agg}\left(x^*, \{\mathsf{vk}_i, y_i\}_{i=1\ldots n}\right) = \prod_{i=1}^{n} y_i = \prod_{i=1}^{n} y_i^* = \mathsf{Agg}\left(x^*, \{\mathsf{vk}_i, y_i^*\}_{i=1\ldots n}\right)$$

by simple linear algebra. In the following, we propose a different aggregation function that simultaneously resolves both limitations.

Aggregate VRF. Our approach is inspired by the work of [4], although our construction and analysis turns out to be substantially different. We assume the existence of two hash functions

$$\tilde{\mathcal{G}} : \{0,1\}^\lambda \times \mathbb{G}_1^n \times \mathbb{G}_T^n \to \{0,1\}^\lambda \quad \text{and} \quad \tilde{\mathcal{H}} : \{0,1\}^\lambda \to \mathbb{Z}_q^n.$$

modeled as random oracles. Loosely speaking, the first random oracle will be used to sample a random seed, which is included in the proof, whereas the second random oracle will be used to sample the coefficients for a linear combination of the VRF images. Since the output of this oracle is unpredictable to the adversary, we will be able to show that no (efficient) adversary can cancel out the contributions of the honest keys. Our aggregation algorithms are described below.

- $\mathsf{Agg}(x, \{\mathsf{vk}_i, y_i\}_{i=1\ldots n})$: Compute $R \leftarrow \tilde{\mathcal{G}}(x, \mathsf{vk}_1, \ldots, \mathsf{vk}_n, y_1, \ldots, y_n)$ and $(r_1, \ldots, r_n) \leftarrow \tilde{\mathcal{H}}(R)$, then return $\tilde{y} = \prod_{i=1}^n y_i^{r_i}$.
- $\mathsf{AggProve}(x, \{\mathsf{vk}_i, \pi_i\}_{i=1\ldots n})$: Compute $R \leftarrow \tilde{\mathcal{G}}(x, \mathsf{vk}_1, \ldots, \mathsf{vk}_n, e(g_1, \pi_1), \ldots, e(g_1, \pi_n))$ along with $(r_1, \ldots, r_n) \leftarrow \tilde{\mathcal{H}}(R)$, then return $(\tilde{\pi} = \prod_{i=1}^n \pi_i^{r_i}, R)$.
- $\mathsf{AggVerify}(\{\mathsf{vk}_i\}_{i=1\ldots n}, x, \tilde{y}, (\tilde{\pi}, R))$: Compute $(r_1, \ldots, r_n) \leftarrow \tilde{\mathcal{H}}(R)$ and define $\tilde{\mathsf{vk}} = \prod_{i=1}^n \mathsf{vk}_i^{r_i}$. Return

$$1 \stackrel{?}{=} \mathsf{Verify}(\tilde{\mathsf{vk}}, x, \tilde{y}, \tilde{\pi}) \quad \text{and} \quad y \stackrel{?}{\neq} 1.$$

Aggregate correctness follows immediately from the key homomorphic property of our VRF, since the pair $(\tilde{y}, \tilde{\pi})$ is a valid image-proof pair under the key $\tilde{k} = \sum_{i=1}^n k_i \cdot r_i$, and furthermore $y_i = e(g_1, \pi_i)$. Furthermore, $y = 1$ only with negligible probability. Next we show that the construction satisfies aggregate pseudorandomness.

The Algebraic Group Model. Before delving into the proof, we recall the basics of the algebraic group model (AGM) [23]. In the AGM, all algorithms are treated as algebraic algorithms. That is, along with every group element that they return, they are required to add an *explanation* of such element as a linear combination of the input group elements. We define the notion of algebraic algorithms below.

Definition 12 (Algebraic Algorithms). *An algorithm $\mathcal{A}$ is* algebraic *if, for every group element $h \in \mathbb{G}$ that it outputs, it also returns the corresponding list of algebraic coefficients. That is*

$$(h, c_1, \ldots, c_t) \leftarrow\!\$\ \mathcal{A}(h_1, \ldots, h_t) \text{ such that } h = \prod_{i=1}^t h_i^{c_i}.$$

For an adversary $\mathcal{A}$ that has access to oracles during its runtime, we impose the above restriction to all group elements that it sends to the oracle. Similarly, all group elements that A receives from the oracle are treated as new inputs to $\mathcal{A}$.

We also recall the Schwartz-Zippel lemma, which is going to be useful in our analysis.

Lemma 1 (Schwartz-Zippel). *Let $P \in \mathbb{F}[X_1, \ldots, X_n]$ be a non-zero n-variate polynomial over a field $\mathbb{F}$ of degree d. Let $r_1, \ldots, r_n$ be selected uniformly at random from $\mathbb{F}$, then*

$$\Pr\left[P(r_1, \ldots, r_n) = 0\right] \leq \frac{d}{|\mathbb{F}|}.$$

Aggregate Pseudorandomness. We are now in the position to show that our construction satisfies aggregate pseudorandomness in the AGM and in the random oracle model.

Theorem 2. *If the group generator* $\mathsf{GroupGen}$ *is ABDH-hard, then the construction as described above satisfies aggregate pseudorandomness against algebraic adversaries, in the random oracle model.*

Proof. The proof of aggregate pseudorandomnes consists of a reduction against the ABDH assumption (Definition 2). Recall that, up to a variable renaming, the assumption states that the following distributions are computationally indistinguishable

$$\left(g_1, g_2, g_1^s, g_1^{ks}, g_2^s, g_2^{ks}, g_2^z, e(g_1, g_2)^{\frac{z}{s}}, e(g_1, g_2)^{\frac{z^2}{s}}, e(g_1, g_2)^{kz}\right)$$
$$\approx \left(g_1, g_2, g_1^s, g_1^{ks}, g_2^s, g_2^{ks}, g_2^z, e(g_1, g_2)^{\frac{z}{s}}, e(g_1, g_2)^{\frac{z^2}{s}}, e(g_1, g_2)^{r}\right).$$

On input a sample from either distribution, the reduction sets $S = g_1^s$ and $\mathsf{vk} = g_1^{ks}$, then it activates the adversary $\mathcal{A}$. The reduction simulates the composition of random oracles $\tilde{\mathcal{H}} \circ \tilde{\mathcal{G}}$ by lazy sampling. More specifically, for any input query q to $\tilde{\mathcal{G}}$, the reduction samples a random output for $\tilde{\mathcal{H}}$ on input $\tilde{\mathcal{G}}(q)$ and records the pair $(q, \tilde{\mathcal{H}}(\tilde{\mathcal{G}}(q)))$ as the output of the combined oracle. Note that such a simulation fails when one of the outputs of $\tilde{\mathcal{G}}$ (as defined in the simulation) was queried to $\tilde{\mathcal{H}}$ before the corresponding input is queried to $\tilde{\mathcal{G}}$. However, this happens with negligible probability by a standard birthday bound. Let Q be the number of queries of $\mathcal{A}$ to the random oracle $\mathcal{H}$, the reduction guesses an index $i^* \in \{1, \ldots, Q\}$ and simulates the oracle as follows: For $i \neq i^*$, on input x_i the reduction programs the random oracle to

$$\mathcal{H}(x_i) = (g_2^s)^{\rho_i} \text{ where } \rho_i \leftarrow\!\$\ \mathbb{Z}_p.$$

On the other hand, on input x_{i^*} the reduction sets

$$\mathcal{H}(x_{i^*}) = g_2^z.$$

For the VRF evaluation oracle, we assume without loss of generality that the adversary already queried the input x_i to the random oracle. The reduction proceeds as follows: If $x_i = x_{i^*}$, the reduction aborts and returns a random bit. Otherwise, the reduction computes

$$\pi_i = \left(g_2^{ks}\right)^{\rho_i} \text{ and } y_i = e(g_1, \pi)$$

by fetching the appropriate value of ρ_i. The challenge query of the (algebraic) adversary is interpreted by the reduction as

$$(x^*, \mathsf{vk}_1, \ldots, \mathsf{vk}_n) \text{ and } \left\{P_i \text{ such that } g_1^{P_i(1,s,ks,z)} = \mathsf{vk}_i\right\}_{i=1\ldots n}$$

where P_i is a multilinear polynomial in the variables $(1, s, ks, z)$, in its coefficient embedding. For notational convenience, we group the coefficients corresponding to the variables $\rho_1, \ldots, \rho_Q$ (which are also formal variables from the point of view of the adversary) in the constant term. This is always possible since the reduction samples ρ_i itself and therefore knows the corresponding integer in the plain. The reduction checks if $x^* = x_{i^*}$ and returns a random bit if this is not the case. Else it defines P_0 as

$$P_0 = (0, 0, 1, 0)$$

and computes

$$y^* = \prod_{i=0}^{n} e(g_1, g_2)^{r_i P_i(z/s, z, \{kz, r\}, z^2/s)}.$$

where $r_0, \ldots, r_n$ is the output of the random oracle $\tilde{\mathcal{H}} \circ \tilde{\mathcal{G}}$ on input $(x^*, \mathsf{vk}, \mathsf{vk}_1, \ldots, \mathsf{vk}_n)$. Finally, the reduction returns whatever bit $\mathcal{A}$ returns.

Note that, in case the reduction guesses correctly the query to the random oracle that contains the challenge input x^*, the answers to the oracle queries of the adversary are perfectly simulated by the reduction. This follows using the same argument as in Thm. 1. Furthermore, observe that the reduction is efficient, since it only performs basic group operations. In particular, the reduction is supplied by the challenger all group elements needed to compute the evaluation at the challenge point y^*.

In case where the challenge tuple contains the element $e(g_1, g_2)^{kz}$, we argue that the reduction perfectly simulates the view of the adversary with the bit $b = 0$. To see why this is the case, it suffices to observe that

$$
\begin{aligned}
y^* &= \prod_{i=0}^{n} e(g_1, g_2)^{r_i P_i(z/s, z, kz, z^2/s)} \\
&= e(g_1, g_2)^{r_0 kz} \prod_{i=1}^{n} e(g_1, g_2)^{r_i P_i(z/s, z, kz, z^2/s)} \\
&= e(g_1, \mathcal{H}(x^*))^{r_0 k} \prod_{i=1}^{n} e(g_1, \mathcal{H}(x^*))^{r_i P_i(1/s, 1, k, z/s)} \\
&= e(g_1, \mathcal{H}(x^*))^{r_0 k} \prod_{i=1}^{n} e(g_1, \mathcal{H}(x^*))^{r_i \mathsf{DLog}_S(\mathsf{vk}_i)} \\
&= e(g_1, \mathcal{H}(x^*))^{r_0 k} \prod_{i=1}^{n} e(g_1, \mathcal{H}(x^*))^{r_i \mathsf{VKey}^{-1}(\mathsf{vk}_i)} \\
&= \mathsf{Agg}\left(x^*, \mathsf{vk}, e(g_1, \mathcal{H}(x^*))^k, \left\{\mathsf{vk}_i, e(g_1, \mathcal{H}(x^*))^{k_i}\right\}_{i=1\ldots n}\right)
\end{aligned}
$$

which is precisely what the attacker is expecting. On the other hand, in case the challenge tuple contains the element $e(g_1, g_2)^r$, we argue that the output y^* computed by the reduction is uniform, and therefore perfectly simulates the case where $b = 1$, except with negligible probability. Let

$$\tilde{P} = \sum_{i=0}^{n} r_i P_i,$$

then, since $\mathbb{Z}_p$ is a field, the y^* computed by the reduction is uniform if and only if the third coefficient of $\tilde{P}$ is non-zero. Since the tuple $r_0, \ldots, r_n$ is sampled uniformly by the reduction *after* the polynomials $P_0, \ldots, P_n$ are fixed, by Lmm. 1, this happens with probability at most $1/p$, which is negligible.

The above argument shows that the reduction succeeds perfectly in case of a correct guess and otherwise returns a random bit. Since this happens with probability at least $1/Q$, the success probability of the reduction in distinguishing the two distributions is at most a polynomial factor smaller than to that of $\mathcal{A}$. This contradicts the ABDH assumption and concludes our proof.

Aggregate Uniqueness. It can be easily shown that the construction satisfies aggregate uniqueness, unconditionally.

Theorem 3. *The construction as described above satisfies aggregate uniqueness.*

Proof. The proof follows immediately by observing that the aggregate verification key

$$\tilde{\mathsf{vk}} = \prod_{i=1}^{n} \mathsf{vk}_i^{r_i} = S^{\sum_{i=1}^{n} k_i \cdot r_i} = S^{\tilde{k}}$$

is a well-formed VRF key, and thus aggregate uniqueness follows from the uniqueness of the base construction.

Aggregate Binding. In the following we show that the scheme satisfies aggregate binding. In fact, we show a slightly stronger statement, namely that aggregate binding holds against *unbounded* adversaries that make a polynomial number of queries to the random oracle.

Theorem 4. *The construction as described above satisfies aggregate binding in the random oracle model.*

Proof. Given the aggregate secret key $\tilde{\mathsf{vk}}$, the image-proof pair is unique, thus the only way to break the aggregate uniqueness is to find a collision in the images. That is, for any given set of verification keys $\mathsf{vk}_1, \ldots, \mathsf{vk}_n$, the adversary must find a point x^* and two tuples $(y_1, \ldots, y_n) \neq (y_1^*, \ldots, y_n^*)$ such that

$$\mathsf{Agg}\left(x^*, \{\mathsf{vk}_i, y_i\}_{i=1\ldots n}\right) = \mathsf{Agg}\left(x^*, \{\mathsf{vk}_i, y_i^*\}_{i=1\ldots n}\right).$$

In the following we argue that this happens only with negligible probability, over the random choice of $\tilde{\mathcal{G}}$ and $\tilde{\mathcal{H}}$. Recall that the aggregation algorithm computes

$$(r_1, \ldots, r_n) \leftarrow \tilde{\mathcal{H}}\left(\tilde{\mathcal{G}}(x, \mathsf{vk}_1, \ldots, \mathsf{vk}_n, y_1, \ldots, y_n)\right).$$

and outputs $\tilde{y} = \prod_{i=1}^{n} y_i^{r_i}$. First, note that we can equivalently analyze the experiment where $\tilde{\mathcal{H}} \circ \tilde{\mathcal{G}}$ is simulated by lazy sampling (see the proof of Thm. 2), with only a negligible loss in the advantage. Second, observe that, except for the tuple $(y_1, \ldots, y_n) = (1, \ldots, 1)$, the output of the aggregation algorithm is uniformly distributed in $\mathbb{G}_1$. It follows that, for any (possibly unbounded) algorithm making a polynomial number of queries to the random oracle, the probability of finding a collision is negligible (in λ) by a standard birthday bound.

4.3 Anonymity

The notion of anonymity, as defined in [25] requires that the verification key of the VRF can be updated to a new verification key

$$\mathsf{vk}' \leftarrow \mathsf{Update}(\mathsf{vk}, k; \rho)$$

where ρ denotes the random coins used by the Update algorithm, in such a way that the updated key vk' is not linkable to the original key vk. Furthermore, there exists an updated proof algorithm UProve that, on input the secret key k and the random coins ρ used during the update procedure, produces a valid proof π' to correctly verify the image of the VRF at any point. We define more formally the anonymity property below. Note that, compared with the syntax of [25], we define a more permissive variant, where the update algorithm is allowed to take as input the secret key as well. Nevertheless, this variant still suffices for all applications presented in [25].

Definition 13 (Anonymity). *A VRF satisfies* anonymity *if there exists a negligible function μ such that for $\lambda \in \mathbb{N}$ and all admissible PPT adversaries $\mathcal{A}$ it holds that*

$$\left|\frac{1}{2} - \Pr\left[1 \leftarrow \mathsf{ExpAnon}_{\mathcal{A}}(1^\lambda)\right]\right| = \mu(\lambda)$$

where the experiment $\mathsf{ExpAnon}_{\mathcal{A}}$ *is defined as follows.*

- *The challenger samples a reference string* $\mathsf{crs} \leftarrow\!\!\$\ \mathsf{Setup}(1^\lambda)$ *and two key pairs*

$$(k_0, \mathsf{vk}_0) \leftarrow\!\!\$\ \mathsf{Gen}(\mathsf{crs}) \text{ and } (k_1, \mathsf{vk}_1) \leftarrow\!\!\$\ \mathsf{Gen}(\mathsf{crs})$$

  and sends $(\mathsf{crs}, \mathsf{vk}_0, \mathsf{vk}_1)$ *to* $\mathcal{A}$.
- $\mathcal{A}$ *outputs a challenge input* x^*.
- *The challenger flips a coin* $b \leftarrow\!\!\$\ \{0,1\}$ *and updates the verification key* $\mathsf{vk}^* \leftarrow \mathsf{Update}(\mathsf{vk}_b, k_b)$. *Denote by* ρ *the random coins of the* Update *procedure, the challenger computes*

$$y^* \leftarrow \mathsf{Eval}(k_b, x^*) \quad \text{and} \quad \pi^* \leftarrow \mathsf{UProve}(k_b, \rho, x^*)$$

  and sends $(\mathsf{vk}^*, y^*, \pi^*)$ *to* $\mathcal{A}$.
- *In the end of the experiment,* $\mathcal{A}$ *outputs a guess* b^*. *The experiment returns* 1 *if and only if* $b^* = b$.

Update Algorithm. Next, we describe an update algorithm suitable for our construction. We stress here that the updated key belongs to a different domain than the original key, and in particular it no longer satisfies the same homomorphic properties. Nevertheless, this property can still be useful in settings where one performs the homomorphic computation *before* updating the key.

On input a verification key $\mathsf{vk} = S^k = g_1^{ks}$, the update algorithm Update samples a uniform $\kappa \leftarrow\!\!\$\ \mathbb{Z}_p$ and computes the updated key vk' as

$$\mathsf{vk}' = \left(V = g_1^{s\kappa}, T = g_1^{k/\kappa}\right).$$

The updated proof of correct evaluation π' at point x is computed as $\pi' = \mathcal{H}(x)^\kappa$ and is verified by checking the equations

$$y \stackrel{?}{=} e\left(T, \pi'\right) \text{ and } e\left(S, \pi'\right) \stackrel{?}{=} e\left(V, \mathcal{H}(x)\right).$$

Both equations are satisfied with probability 1 since

$$e\left(T, \pi'\right) = e\left(g_1^{k/\kappa}, \mathcal{H}(x)^\kappa\right) = e(g_1, \mathcal{H}(x))^k = y$$

and

$$e\left(V, \mathcal{H}(x)\right) = e\left(g_1, \mathcal{H}(x)\right)^{s\kappa} = e\left(S, \pi'\right).$$

Furthermore, for any given V and T, the value of π' is uniquely determined.

Analysis. The following theorem shows that the update algorithm as defined above satisfies anonymity.

Theorem 5. *If the group generator* GroupGen *is XIBDH-hard, then the construction as described above satisfies anonymity in the random oracle model.*

Proof. Observe that the view of the adversary in the experiment with the challenge bit fixed to $b = 0$ consists of the tuple

$$\left(g_1^s, g_1^{sk_0}, g_1^{sk_1}, g_1^{s\kappa}, g_1^{k_0/\kappa}, g_2^z, g_2^{z\kappa}, e(g_1, g_2^{z\kappa})\right)$$

where $g_2^z = \mathcal{H}(x^*)$. By the XIBDH Assumption, this distribution is computationally indistinguishable from

$$\left(g_1^s, g_1^{sk_0}, g_1^{sk_1}, g_1^{s\kappa}, g_1^{\delta}, g_2^z, g_2^{z\kappa}, e(g_1, g_2^{z\kappa})\right)$$

where $\delta \leftarrow\$ \mathbb{Z}_p$. Another invocation of the XIBDH Assumption allows us to argue that this is computationally indistinguishable from

$$\left(g_1^s, g_1^{sk_0}, g_1^{sk_1}, g_1^{s\kappa}, g_1^{k_1/\kappa}, g_2^z, g_2^{z\kappa}, e(g_1, g_2^{z\kappa})\right)$$

which is exactly the view in the experiment with the challenge bit fixed to $b = 1$.

4.4 Threshold

We briefly outline how our KH-VRF immediately implies a t-out-of-n threshold VRF, exploiting the linearity of the key to combine the construction with a linear secret-sharing scheme. This transformation is analogous to [6] and therefore we only provide a sketch of the scheme here, and we refer the reader to [6] for more details. In the threshold variant, we sample a random key $k \leftarrow\$ \mathbb{Z}_p$ and we interpret it as the constant coefficient of a degree $t-1$ univariate polynomial P, where the rest of the coefficients are also sampled uniformly from $\mathbb{Z}_p$. The share of the i-th party consists of the evaluation of the polynomial $k_i = P(i)$ at point i. For any point $x \in \{0,1\}^\lambda$, each party publishes

$$y_i = e(g_1, \mathcal{H}(x))^{k_i} \quad \text{and} \quad \pi_i = \mathcal{H}(x)^{k_i}.$$

Given t such share, it is possible to publicly reconstruct a valid image-proof pair, by running Lagrange interpolation in the exponent, i.e., computing

$$y = \prod_{i=1}^{t} y_i^{L_i(0)} \quad \text{and} \quad \pi = \prod_{i=1}^{t} \pi_i^{L_i(0)}$$

where L_i is the i-th Lagrange polynomial.

5 Applications

We discuss several motivating applications for the notion of key-homomorphic and aggregate VRFs. In favor of a more intuitive explanation, we keep the following discussion at a high-level and we leave the precise formalization of the security properties and the protocol description as ground for future work. We stress that the purpose of this discussion is to demonstrate the usefulness and the versatility of the notion of our VRFs, but we do not claim that our solutions are superior in *all aspects* compared to existing protocols.

5.1 Distributed VRF Without a Trusted Dealer

As a warm-up application, we show how an aggregate VRF imply a simple and efficient construction of an n-out-of-n VRF where the key is distributed across n parties. The key generation is completely non-interactive and performed locally by all parties, which do not be even be aware of each other's existence. In particular, no trusted dealer is needed to distribute the shares of the key. We outline the protocol below.

- *Local Key Generation:* On input the common reference string crs, each party simply samples a key-pair locally

$$(k, \mathsf{vk}) \leftarrow\!\!\$\ \mathsf{Gen}(\mathsf{crs})$$

  and outputs vk.
- *Shared Key Computation:* The shared key pair is defined to be

$$\tilde{k} = \sum_{i=1}^{n} k_i \quad \text{and} \quad \tilde{\mathsf{vk}} = \prod_{i=1}^{n} \mathsf{vk}_i.$$

It is easy to see that the resulting key is well-formed by the homomorphic property of the underlying VRF, and that each party can compute the share of the the evaluation at any point x, once again by appealing to the linear homomorphism of the key. The above scheme only ensures correctness against semi-honest parties, whereas if one wants to defend against rogue-key attacks the aggregation function presented in Sect. 4.2 can be used.

5.2 Proof of Stake Blockchains

To see how aggregate VRF provide substantial advantages when applied to proof of stake (PoS) protocols. The discussion here is deliberately informal, and we refer the reader to [21] for a thorough treatment of aggregatable PoS protocols. Recall that in a PoS protocol [15] any user in the system uploads their VRF verification key to the blockchain. In each epoch, each user can evaluate the VRF on the slot number (which will play the role as a unique identifier for the epoch) to compute the corresponding image. If the image is less than (a function of) their wealth, then that user is identified as the designated party to generate the new block. Such party can then compute a proof to convince the all other parties that the VRF was evaluated correctly. This way, it can be publicly verified that the user was indeed the winner of the round. In committee-based PoS protocols, such as Algorand [27], a similar process is performed in order to elect members of a committee, that collectively determine the value of the next block.

Plugging in a KH-VRF in the above protocol, we immediately obtain the following properties:

1. In committee-based PoS, we can aggregate individual proofs

$$\tilde{\pi} \leftarrow \mathsf{AggProve}(x, \{\mathsf{vk}_i, \pi_i\}_{i=1\ldots n})$$

   so that the size of the proof becomes *independent* of the size of the committee. By the aggregate binding of the VRF, an attacker cannot later change the images in such a way that they will pass the aggregate verification test.
2. One does not need to store a proof for each block (which is needed to make the blockchain publicly verifiable) but one can instead store a *single proof* throughout the lifetime of the system, by recursively aggregating all of the existing proofs. One can still verify that the images are still valid, and an attacker cannot change images after the fact, in such a way that they will pass the aggregate verification test (assuming that the VRF satisfies aggregate binding).

5.3 Verifiable Symmetric-Key Proxy Re-Encryption

The work of [6] introduces the notion of key-homomorphic pseudorandom functions (KH-PRF), which is analogous to KH-VRF, with the crucial difference that there is no verification procedure for the output of the PRF. In [6] the authors show a number of applications for this primitive, such as distributed PRFs, symmetric-key proxy re-encryption, updatable encryption, and PRFs secure against related-key attacks. Our key-homomorphic VRF construction can be used as a drop-in replacement to augment all of these applications with an efficient verification procedure.

To exemplify this claim, we discuss the case of symmetric-key proxy re-encryption. Suppose we have a dataset where each message $M_i \in \mathbb{G}_T$ is encrypted as

$$\mathsf{Enc}(k, M_i) = \mathsf{Eval}(k, n_i) \cdot M_i$$

where n_i is a public nonce associated with the i-th location. A trivial solution to update the key k is to download the entire dataset and re-encrypt all data. A more efficient solution is for the client to provide the server with the key $\delta - k$ (where $\delta \in \mathbb{Z}_p$ is the newly sampled key). Then the server can update all ciphertext by computing

$$\mathsf{Enc}(k, M_i) \cdot \mathsf{Eval}(\delta - k, n_i) = \mathsf{Eval}(\delta, n_i) \cdot M_i = \mathsf{Enc}(\delta, M_i).$$

Provided that the server deletes $\delta - k$, it is shown that this protocol satisfies security. If we plug a key-homomorphic VRF in this template, we enable an additional *verification* property: For each ciphertext, the server can efficiently compute a proof that certifies that the update was done correctly. An external auditor, can be convinced that cipheretexts have been correctly updated, without the need to know the secret key $\delta - k$.

6 Assumptions in the GGM

We briefly discuss why the assumption that we consider are true (unconditionally) in the generic group model (GGM). The following discussion assumes familiarity with basic concepts of the GGM, and we refer the reader to [45] for a thorough introduction. Since both assumptions are static, it suffices to show that there is no way to express that "target" variable τ (i.e., the group element that is different in the two distributions) as a linear combination of other variables in $\mathbb{G}_T$. This is without loss of generality, since we can always move all variables in the target group by computing all possible pairings with the given variables. The claim then follows by Lmm. 1.

For Definition 2, the adversary's view contain the handles corresponding to the formal variables

$$\left(1, a, b, c, ab, ac, bc, \frac{c}{a}, \frac{c^2}{a}, \tau\right) \text{ where } \tau = \left\{\frac{bc}{a}, r\right\}$$

by taking all possible pairings with the element from the source groups. Since τ is linearly independent from all other variables, our claim follows.

For Definition 3, again moving all variables to the target group, we have that the adversary's view contains

$$\left(1, a, ab, ac, d, dc, ad, adc, abd, abdc, acd, adc^2, \tau_0, \tau_1, \tau_2\right)$$

where

$$\tau_0 = \left\{\frac{b}{c}, r\right\} \text{ and } \tau_1 = \left\{\frac{bd}{c}, rd\right\} \text{ and } \tau_2 = \{bd, rdc\}\,.$$

Once again, the claim follows from the fact that τ_0, τ_1, τ_2 are linearly independent from the rest of the variables.

Acknowledgements. Work supported by the European Research Council through an ERC Starting Grant (Grant agreement No. 101077455, ObfusQation).

The author wishes to thank the anonymous reviewers of TCC 2024 for their comments.

References

1. Au, M.H., Susilo, W., Mu, Y.: Practical compact E-Cash. In: Pieprzyk, J., Ghodosi, H., Dawson, E. (eds.) ACISP 2007. LNCS, vol. 4586, pp. 431–445. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-73458-1_31
2. Belenkiy, M., Chase, M., Kohlweiss, M., Lysyanskaya, A.: Compact E-Cash and simulatable VRFs revisited. In: Shacham, H., Waters, B. (eds.) Pairing 2009. LNCS, vol. 5671, pp. 114–131. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-03298-1_9
3. Bitansky, N.: Verifiable random functions from non-interactive witness-indistinguishable proofs. In: Kalai, Y., Reyzin, L. (eds.) TCC 2017. LNCS, vol. 10678, pp. 567–594. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-70503-3_19

4. Boneh, D., Drijvers, M., Neven, G.: Compact multi-signatures for smaller blockchains. In: Peyrin, T., Galbraith, S. (eds.) ASIACRYPT 2018. LNCS, vol. 11273, pp. 435–464. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-03329-3_15
5. Boneh, D., Gentry, C., Lynn, B., Shacham, H.: Aggregate and verifiably encrypted signatures from bilinear maps. In: Biham, E. (ed.) EUROCRYPT 2003. LNCS, vol. 2656, pp. 416–432. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-39200-9_26
6. Boneh, D., Lewi, K., Montgomery, H., Raghunathan, A.: Key homomorphic PRFs and their applications. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013. LNCS, vol. 8042, pp. 410–428. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40041-4_23
7. Boneh, D., Lynn, B., Shacham, H.: Short signatures from the Weil pairing. J. Cryptol. **17**(4), 297–319 (2004)
8. Brakerski, Z., Brodsky, M.F., Kalai, Y.T., Lombardi, A., Paneth, O.: Snargs for monotone policy batch NP. In: Handschuh, H., Lysyanskaya, A., (eds.) Advances in Cryptology - CRYPTO 2023 - 43rd Annual International Cryptology Conference, CRYPTO 2023, Santa Barbara, CA, USA, August 20-24, 2023, Proceedings, Part II, volume 14082 of LNCS, pp. 252–283. Springer (2023)
9. Brakerski, Z., Döttling, N., Garg, S., Malavolta, G.: Leveraging linear decryption: rate-1 fully-homomorphic encryption and time-lock puzzles. In: Hofheinz, D., Rosen, A. (eds.) TCC 2019. LNCS, vol. 11892, pp. 407–437. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-36033-7_16
10. Buser, M., et al.: Post-quantum verifiable random function from symmetric primitives in POS blockchain. In: Atluri, V., Di Pietro, R., Jensen, C.D., Meng, W., (eds.) Computer Security - ESORICS 2022 - 27th European Symposium on Research in Computer Security, Copenhagen, Denmark, September 26-30, 2022, Proceedings, Part I, volume 13554 of LNCS, pp. 25–45. Springer (2022)
11. Choudhuri, A.R., Garg, S., Jain, A., Jin, Z., Zhang, J.: Correlation intractability and snargs from sub-exponential DDH. In: Handschuh, H., Lysyanskaya, A. (eds.) Advances in Cryptology - CRYPTO 2023 - 43rd Annual International Cryptology Conference, CRYPTO 2023, Santa Barbara, CA, USA, August 20-24, 2023, Proceedings, Part IV, volume 14084 of LNCS, pp. 635–668. Springer (2023)
12. Choudhuri, A.R., Jain, A., Jin, Z.: Non-interactive batch arguments for NP from standard assumptions. In: Malkin, T., Peikert, C. (eds.) CRYPTO 2021. LNCS, vol. 12828, pp. 394–423. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-84259-8_14
13. Choudhuri, A.R., Jain, A., Jin, Z.: Snargs for $\mathcal{P}$ from LWE. In: 62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021, Denver, CO, USA, February 7-10, 2022, pp. 68–79. IEEE (2021)
14. David, B., Dowsley, R., Konring, A., Larangeira, M.: MUSEN: Aggregatable key-evolving verifiable random functions and applications. Cryptology ePrint Archive, Paper 2024/628 (2024)
15. David, B., Gaži, P., Kiayias, A., Russell, A.: Ouroboros Praos: an adaptively-secure, semi-synchronous proof-of-stake blockchain. In: Nielsen, J.B., Rijmen, V. (eds.) EUROCRYPT 2018. LNCS, vol. 10821, pp. 66–98. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-78375-8_3
16. Dodis, Y., Vaikuntanathan, V., Wichs, D.: Extracting randomness from extractor-dependent sources. In: Canteaut, A., Ishai, Y. (eds.) EUROCRYPT 2020. LNCS, vol. 12105, pp. 313–342. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-45721-1_12

17. Dodis, Y., Yampolskiy, A.: A verifiable random function with short proofs and keys. In: Vaudenay, S. (ed.) PKC 2005. LNCS, vol. 3386, pp. 416–431. Springer, Heidelberg (2005). https://doi.org/10.1007/978-3-540-30580-4_28
18. Esgin, M.F., et al.: Practical post-quantum few-time verifiable random function with applications to Algorand. In: Borisov, N., Diaz, C. (eds.) FC 2021. LNCS, vol. 12675, pp. 560–578. Springer, Heidelberg (2021). https://doi.org/10.1007/978-3-662-64331-0_29
19. Esgin, M.F., Steinfeld, R., Liu, D., Ruj, S.: Efficient hybrid exact/relaxed lattice proofs and applications to rounding and VRFs. In: Handschuh, H., Lysyanskaya, A. (eds.) Advances in Cryptology - CRYPTO 2023 - 43rd Annual International Cryptology Conference, CRYPTO 2023, Santa Barbara, CA, USA, August 20-24, 2023, Proceedings, Part V, volume 14085 of LNCS, pp. 484–517. Springer (2023)
20. Feige, U., Lapidot, D., Shamir, A.: Multiple non-interactive zero knowledge proofs based on a single random string (extended abstract). In: 31st Annual Symposium on Foundations of Computer Science, St. Louis, Missouri, USA, October 22-24, 1990, Volume I, pp. 308–317. IEEE Computer Society (1990)
21. Fleischhacker, N., Hall-Andersen, M., Simkin, M., Wagner, B.: Jackpot: Non-interactive aggregatable lotteries. Cryptology ePrint Archive, Paper 2023/1570 (2023)
22. Freitag, C., Waters, B., Wu, D.J.: How to use (plain) witness encryption: registered ABE, flexible broadcast, and more. In: Handschuh, H., Lysyanskaya, A. (eds.) Advances in Cryptology - CRYPTO 2023 - 43rd Annual International Cryptology Conference, CRYPTO 2023, Santa Barbara, CA, USA, August 20-24, 2023, Proceedings, Part IV, volume 14084 of LNCS, pp. 498–531. Springer (2023)
23. Fuchsbauer, G., Kiltz, E., Loss, J.: The algebraic group model and its applications. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018. LNCS, vol. 10992, pp. 33–62. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-96881-0_2
24. Galindo, D., Liu, J., Ordean, M., Wong, J.-M.: Fully distributed verifiable random functions and their application to decentralised random beacons. In: IEEE European Symposium on Security and Privacy, EuroS&P 2021, Vienna, Austria, September 6-10, 2021, pp. 88–102. IEEE (2021)
25. Ganesh, C., Orlandi, C., Tschudi, D.: Proof-of-stake protocols for privacy-aware blockchains. In: Ishai, Y., Rijmen, V. (eds.) EUROCRYPT 2019. LNCS, vol. 11476, pp. 690–719. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-17653-2_23
26. Gentry, C., Halevi, S.: Compressible FHE with applications to PIR. In: Hofheinz, D., Rosen, A. (eds.) TCC 2019. LNCS, vol. 11892, pp. 438–464. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-36033-7_17
27. Gilad, Y., Hemo, R., Micali, S., Vlachos, G., Zeldovich, N.: Algorand: scaling byzantine agreements for cryptocurrencies. In: Proceedings of the 26th Symposium on Operating Systems Principles, Shanghai, China, October 28-31, 2017, pp. 51–68. ACM (2017)
28. Goldberg, S., Naor, M., Papadopoulos, D., Reyzin, L., Vasant, S., Ziv, A.: NSEC5: provably preventing DNSSEC zone enumeration. In: 22nd Annual Network and Distributed System Security Symposium, NDSS 2015, San Diego, California, USA, February 8-11, 2015. The Internet Society (2015)
29. Goyal, R., Hohenberger, S., Koppula, V., Waters, B.: A generic approach to constructing and proving verifiable random functions. In: Kalai, Y., Reyzin, L. (eds.) TCC 2017. LNCS, vol. 10678, pp. 537–566. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-70503-3_18

30. Goyal, R., Koppula, V., Waters, B.: Lockable obfuscation. In: Umans, C. (ed.) 58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017, pp. 612–621. IEEE Computer Society (2017)
31. Hanke, T., Movahedi, M., Williams, D.: Dfinity technology overview series, consensus system (2018)
32. Hofheinz, D., Jager, T.: Verifiable random functions from standard assumptions. In: Kushilevitz, E., Malkin, T. (eds.) TCC 2016. LNCS, vol. 9562, pp. 336–362. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49096-9_14
33. Hohenberger, S., Waters, B.: Constructing verifiable random functions with large input spaces. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 656–672. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-13190-5_33
34. Hubácek, P., Wichs, D.: On the communication complexity of secure function evaluation with long output. In: Roughgarden, T. (ed.) Proceedings of the 2015 Conference on Innovations in Theoretical Computer Science, ITCS 2015, Rehovot, Israel, January 11-13, 2015, pp. 163–172. ACM (2015)
35. Jager, T.: Verifiable random functions from weaker assumptions. In: Dodis, Y., Nielsen, J.B. (eds.) TCC 2015. LNCS, vol. 9015, pp. 121–143. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46497-7_5
36. Jarecki, S., Shmatikov, V.: Handcuffing big brother: an abuse-resilient transaction escrow scheme. In: Cachin, C., Camenisch, J.L. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 590–608. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24676-3_35
37. Kalai, Y., Lombardi, A., Vaikuntanathan, V., Wichs, D.: Boosting batch arguments and RAM delegation. In: Saha, B., Servedio, R.A. (eds.) Proceedings of the 55th Annual ACM Symposium on Theory of Computing, STOC 2023, Orlando, FL, USA, June 20-23, 2023, pp. 1545–1552. ACM (2023)
38. Kohl, L.: Hunting and gathering - verifiable random functions from standard assumptions with short proofs. In: Lin, D., Sako, K. (eds.) Public-Key Cryptography - PKC 2019 - 22nd IACR International Conference on Practice and Theory of Public-Key Cryptography, Beijing, China, April 14-17, 2019, Proceedings, Part II, volume 11443 of LNCS, pp. 408–437. Springer (2019)
39. Kuchta, V., Manulis, M.: Unique aggregate signatures with applications to distributed verifiable random functions. In: Abdalla, M., Nita-Rotaru, C., Dahab, R. (eds.) CANS 2013. LNCS, vol. 8257, pp. 251–270. Springer, Cham (2013). https://doi.org/10.1007/978-3-319-02937-5_14
40. Liang, B., Banegas, G., Mitrokotsa, A.: Statically aggregate verifiable random functions and application to e-lottery. Cryptogr. **4**(4), 37 (2020)
41. Liskov, M.: Updatable zero-knowledge databases. In: Roy, B. (ed.) ASIACRYPT 2005. LNCS, vol. 3788, pp. 174–198. Springer, Heidelberg (2005). https://doi.org/10.1007/11593447_10
42. Lysyanskaya, A.: Unique signatures and verifiable random functions from the DH-DDH separation. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, pp. 597–612. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-45708-9_38
43. Micali, S., Rabin, M.O., Vadhan, S.P.: Verifiable random functions. In: 40th Annual Symposium on Foundations of Computer Science, FOCS '99, 17-18 October, 1999, New York, NY, USA, pp. 120–130. IEEE Computer Society (1999)
44. Scott, M.: A note on group membership tests for g_1, g_2 and g_t on bls pairing-friendly curves. Cryptology ePrint Archive, Paper 2021/1130 (2021)

45. Shoup, V.: Lower bounds for discrete logarithms and related problems. In: Fumy, W. (ed.) EUROCRYPT 1997. LNCS, vol. 1233, pp. 256–266. Springer, Heidelberg (1997). https://doi.org/10.1007/3-540-69053-0_18
46. Waters, B., Wu, D.J.: Batch arguments for SFNP and more from standard bilinear group assumptions. In: Dodis, Y., Shrimpton, T. (eds.) Advances in Cryptology - CRYPTO 2022 - 42nd Annual International Cryptology Conference, CRYPTO 2022, Santa Barbara, CA, USA, August 15-18, 2022, Proceedings, Part II, volume 13508 of LNCS, pp. 433–463. Springer (2022)
47. Wichs, D., Zirdelis, G.: Obfuscating compute-and-compare programs under LWE. In: Umans, C. (ed.) 58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017, pp. 600–611. IEEE Computer Society (2017)

More Efficient Functional Bootstrapping for General Functions in Polynomial Modulus

Han Xia[1,2], Feng-Hao Liu[3], and Han Wang[1,2(✉)]

[1] Key Laboratory of Cyberspace Security Defense, Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China
{xiahan,wanghan}@iie.ac.cn

[2] School of Cyber Security, University of Chinese Academy of Sciences, Beijing, China

[3] Washington State University, Pullman, WA, USA
feng-hao.liu@wsu.edu

Abstract. Functional bootstrapping seamlessly integrates the benefits of homomorphic computation using a look-up table and the noise reduction capabilities of bootstrapping. Its wide-ranging applications in privacy-preserving protocols underscore its broad impacts and significance. In this work, our objective is to craft more efficient and less restricted functional bootstrapping methods for general functions within a polynomial modulus. We introduce a series of novel techniques, proving that functional bootstrapping for general functions can be essentially as efficient as regular FHEW/TFHE bootstrapping. Our new algorithms operate within the realm of prime-power and odd composite cyclotomic rings, offering versatility without any additional requirements on input noise and message space beyond correct decryption.

1 Introduction

Fully homomorphic encryption (FHE) has been identified as a powerful cryptographic tool, allowing arbitrary computation over ciphertexts without first decrypting it. Gentry pioneered FHE in his seminal work [33], sparking numerous subsequent studies such as [5,12,13,15,16,21,22,30,35,44,45]. In addition to theoretical progress, practical strides have been made with the development of several useful FHE libraries along this research trajectory [3,20,24,56,57,73], contributing significantly to potential real-world applications.

Bootstrapping with Polynomial Error Growth. As a pivotal breakthrough introduced by Gentry in [33], bootstrapping plays a crucial role in achieving "fully" homomorphic encryption. In a nutshell, the bootstrapping paradigm takes as input an FHE ciphertext $\boldsymbol{c} \in \mathsf{Enc}(m)$ and some bootstrapping key, and outputs another FHE ciphertext $\boldsymbol{c}' \in \mathsf{Enc}(\mathsf{Dec}(\boldsymbol{c})) \subset \mathsf{Enc}(m)$, with significantly reduced noise. As homomorphic computations in current FHE schemes inevitably incur noise, reaching a point where decryption becomes incorrect, bootstrapping

E. Boyle and M. Mahmoody (Eds.): TCC 2024, LNCS 15367, pp. 130–163, 2025.
https://doi.org/10.1007/978-3-031-78023-3_5

becomes the critical key that enables an arbitrary number of homomorphic operations and thus "F"-HE.

Among various FHE schemes, the work [16] showed for the first time that bootstrapping would only incur a polynomial error growth, though their method requires very large polynomial runtimes due to the reliance on Barrington's Theorem [6]. Thereafter, the work (referred to as AP14) [5] showed how to bootstrap with error growth and runtime being both small polynomials by treating decryption as an arithmetic function. The AP14 method critically relies on the GSW schemes [35] (known as the third generation of FHE schemes), yet their explicit method was in the plain lattice (i.e., LWE [63]) setting, which is not expected to be concretely efficient. Subsequently, FHEW [30] and TFHE [23] refined and optimized the AP14 method in the ring setting, introducing substantial new insights. These optimizations resulted in bootstrapping achievements within subseconds in practical implementations. Their impact is evident in the inclusion of these methods in various libraries, such as OpenFHE [3] and TFHE [24,73], highlighting their tangible real-world relevance to the community.

This work focuses on the setting of (functional) bootstrapping along the line of FHEW/TFHE, i.e., methods with polynomial error growth, which implies smaller FHE parameters and thus smaller FHE keys. We notice that the FHEW/TFHE computation is suited for computation expressed by boolean circuits, e.g., comparisons and decision diagram computations [11,28], with smaller memory requirements.

Functional Bootstrapping. Following the FHEW-like framework [53] (including FHEW [30] and TFHE [23]), the works [8,10] identified that in the power-of-2's cyclotomic rings, the bootstrapping method can be slightly modified with almost no additional cost, outputting $\boldsymbol{c}' \in \mathsf{Enc}(f(m))$ for any negacyclic $f : \mathbb{Z}_p \to \mathbb{Z}_p$ where p is the plaintext domain[1]. This is called functional bootstrapping [10,27,36,42,47,48] (or programmable bootstrapping [7,25,26]), which integrates noise reduction and (small) look-up table computation at comparable efficiency as the bootstrapping. A trivial way to overcome the negacyclicity is to use only half-domain of the plaintext space [10,11,36]. However, this significantly limits the construction of applications. To support both full-domian and general functions, several recent works [7,26,42,47,72] have aimed for efficient designs, which can be broadly categorized into the following two types.

- **Single Input/Output:** In these schemes [26,42,47,72], both the input and output are single LWE ciphertexts. They typically require more than two calls to the regular FHEW/TFHE bootstrapping, and some even impose additional constraints on the noise level of the input LWE ciphertext [47,72].
- **Multiple Inputs/Outputs:** This framework was first introduced in [36] and later optimized in [7,26]. It decomposes the plaintext into multiple bits or digits[2] and encrypts each segment separately. As a result, the number of

[1] For commonly used FHEW/TFHE parameters [53], the plaintext domain is roughly 3-bit to 5-bit. Also notice that f might be more general that works on $\mathbb{Z}_p \to \mathbb{Z}_{p'}$, i.e., the output ciphertext might be associated with a different plaintext domain.

[2] Here, "digits" means decomposing by another integer base B, i.e., $B > 2$.

ciphertexts and the invocations of regular bootstrapping in these schemes increase with the precision of the input plaintext. Moreover, to support digits rather than just bits, there must be constraints on the input ciphertexts [7].

As functional bootstrapping has many applications, such as privacy-preserving machine learning [42,46,49], and it can serve as an important building block of conversion between different types of FHE schemes with high precisions [47,49], it becomes an important open problem to optimize the functionality and efficiency of functional bootstrapping designs, either theoretically or practically. This motivates the main question of this work.

(**Main Question**). Can we *simultaneously* eliminate *all* constraints of functional bootstrapping (within a polynomial modulus) for general functions and make it as efficient as the regular FHEW/TFHE bootstrapping?

1.1 Our Contributions

To tackle the main question, this work designs a series of new full-domain functional bootstrapping algorithms for general functions that are essentially as efficient as the regular FHEW/TFHE bootstrapping. We summarize as follows.

Functionality. We design three new functional bootstrapping algorithms over general cyclotomic rings with prime, prime-power, and odd composite indices[3] for single LWE input[4]. All of them satisfy all the following desirable properties:

- Arbitrary plaintext modulus/encoding for input ciphertext.
- No additional input noise requirement beyond correct decryption.
- General functions $f : \mathbb{Z}_p \to \mathbb{Z}_{p'}$ with arbitrary positive integers p and p'.

Briefly, all of them have no restrictions on the input LWE ciphertext (with a fixed modulus), which implies the following result:

Functional bootstrapping for general functions works for inputs that are any valid LWE *ciphertext (i.e., correctly decryptable).*

Efficiency. Let n be the dimension of the input LWE ciphertext. All of our functional bootstrapping algorithms come at the cost of $n + O(\log n)$ homomorphic multiplications (i.e., FHE external products). Compared to the regular FHEW/TFHE bootstrapping that requires n homomorphic multiplications (in the dominating "Blind Rotation" procedure), this implies that the ratio of efficiency achieves $1 + o(1)$. In Table 1, we present a summary of our results and a comparison with prior full-domain designs.

[3] In summary, we support cyclotomic rings for two general categories – (1) odd and (2) power-of-2 indices.

[4] Our algorithms directly follow the technical line of single input/output. However, they can also be used to remove the constraints on input noise and encoding in schemes with multiple inputs/outputs. In other words, our optimizations lie at the core part of functional bootstrapping algorithms and can be applied to enhance the functionality and efficiency of all existing functional bootstrapping designs.

Table 1. Prior and our full-domain functional bootstrapping schemes for general functions. The modulus of the input LWE ciphertext is q. The minimal cyclotomic index stands for the smallest ring required for bootstrapping the input LWE ciphertext, which directly determines the efficiency of basic ring operations. Note that Blind Rotation is the core part that dominates the efficiency of FHEW/TFHE bootstrapping.

	# of Blind Rotations	Minimal Cyclotomic Index	Type of Cyclotomic Index	Without Restrictions on Input Error
[26]	2	$2q$	Power-of-2	Yes
	3	q	Power-of-2	Yes
[42]	$1 + d_g$ †	q	Power-of-2	Yes
[47]	2	$2q$	Power-of-2	No*
	3	$4q$	Power-of-2	Yes
[7]	$O(\beta d_g)$ ♯	$O(q)$	Power-of-2	Yes/No**
Ours	$1 + o(1)$	q	Prime-Power‡	Yes
	$1 + o(1)$	q	Composite§	Yes

† d_g is the gadget decomposition dimension satisfying $d_g > 1$.
♯ β is the precision (bit-length) of the input plaintext modulus. The value $O(\beta d_g)$ could be even larger for relatively large β (e.g., $\beta > 28$ as reported in [7]).
‡ We support arbitrary prime-power index (including pure prime and power-of-2).
§ We only support odd composite index (see Challenge 2 in Sect. 6).
* It requires the input noise to be less than roughly half of the maximal allowable bound (i.e., the upper bound for correct decryption). This method was also independently discovered in [72].
** "Yes" is only for the case where each input ciphertext encrypts only a single bit. Encrypting digits in a single ciphertext would require plaintext-ciphertext multiplication and homomorphic subtraction of the input ciphertext, which leads to the constraint on the input noise level.

1.2 Technical Overview

In this section, we highlight some critical insights in our designs. First, we briefly review the FHEW/TFHE (functional) bootstrapping framework.

FHEW/TFHE Framework. As discussed before, this framework takes as input an LWE ciphertext $\boldsymbol{c} \in \mathsf{Enc}(m)$ and some bootstrapping key, and aims to output another LWE ciphertext $\boldsymbol{c}' \in \mathsf{Enc}(m)$ or $\boldsymbol{c}' \in \mathsf{Enc}(f(m))$ with reduced noise. We further denote $\boldsymbol{c} = (b, \boldsymbol{a}) \in \mathbb{Z}_q \times \mathbb{Z}_q^n$ and notice that the decryption procedure is $\mathsf{Round}\left((b - \langle \boldsymbol{a}, \boldsymbol{s} \rangle \mod q)\right)$, where $\boldsymbol{s}$ is the secret key and $\mathsf{Round}(\cdot)$ is some rounding function for decoding. To achieve (functional) bootstrapping, the framework first uses the Blind Rotation technique that produces a RLWE ciphertext that encrypts $\zeta_q^{b-\langle \boldsymbol{a}, \boldsymbol{s} \rangle} := \zeta_q^{\alpha}$ where ζ_q is a primitive q-th root of unity, and then extracts an $\mathsf{LWE.Enc}(\mathsf{Round}(\alpha))$ of more general $\mathsf{LWE.Enc}(f(\alpha))$ given $\mathsf{RLWE.Enc}(\zeta_q^{\alpha})$. As α is computed in the exponent of ζ_q's power, it naturally takes the modulo q.

For the complexity, the Blind Rotation procedure takes n homomorphic multiplications (specifically the external products), dominating the overall complexity of the bootstrapping procedure. In power-of-two cyclotomic rings, the extraction procedure can be efficiently achieved (almost free) for the cases of $\mathsf{Round}(\cdot)$ function and negacyclic functions. For general functions however, current methods use different design concepts that require more than two calls to the core bootstrapping (Blind Rotation), and some of them even require ring extension (see Table 1 for the minimal cyclotomic index).

Our Goal. To achieve a more efficient method, we aim to design a more powerful and efficient function evaluation procedure than extraction, particularly at the cost of $o(n)$ homomorphic multiplications (external products). Combining with one call to the Blind Rotation, this would imply the overall cost to be $n + o(n)$ homomorphic multiplications, meaning $1+o(1)$ times the regular FHEW/TFHE bootstrapping. Below, we present our new insights on how we achieve our goal.

Our Blueprint. We aim to resolve the problem of the most general form where the input noise is only required to be bounded by the maximal allowable value of correct decryption, and the function has no restriction. Particularly, we first observe that any discrete function can be expressed by the linear combination of the equality test function: Define the equality test as $\mathsf{EqT}(\zeta_q^\alpha, \beta)$ that takes some power of ζ_q and $\beta \in \mathbb{Z}_q$, and outputs 1 if $\alpha = \beta \bmod q$ or otherwise outputs 0. Then any function $f : \mathbb{Z}_q \to \mathbb{Z}_h$ (for any positive integer h) can be expressed as $f(\alpha) = \sum_{\beta \in \mathbb{Z}_q} f(\beta) \cdot \mathsf{EqT}(\zeta_q^\alpha, \beta)$. Based on this idea, if there exists such a homomorphic equality test, we can construct a function evaluation algorithm that takes as input $\mathsf{RLWE.Enc}(\zeta_q^\alpha)$ and outputs $\mathsf{RLWE.Enc}(f(\alpha))$. So, our remaining task is to find an equality test function that can be efficiently computed homomorphically. In this way, the desired function can be evaluated. Our idea exploits the technique of homomorphic equality test and the algebraic trace over three different types of cyclotomic rings, as we elaborate below.

The Case of Prime Cyclotomic Rings. In this setting, the algebraic trace inherently possesses properties close to what we require. Particularly, when q is prime, the algebraic trace function has exactly two branches as follows:

$$\mathsf{Tr}_{\mathbb{Q}(\zeta_q)/\mathbb{Q}}(\zeta_q^{\alpha-\beta}) = \begin{cases} q-1 & \text{if} \quad \alpha = \beta \mod q \\ -1 & \text{otherwise} \end{cases}.$$

Thus, we can use the function $\mathsf{Tr}_{\mathbb{Q}(\zeta_q)/\mathbb{Q}}(\cdot)+1$ as the equality test function (scaled by q) in the following way. Given $\mathsf{RLWE.Enc}(\zeta_q^\alpha)$, we first multiply it by $\sum_\beta f(\beta) \cdot \zeta_q^{-\beta}$ and then perform the homomorphic trace evaluation (then plus $\sum_\beta f(\beta)$), resulting in a ciphertext of $\mathsf{RLWE.Enc}\left(\mathsf{Tr}\left(\sum_\beta \zeta_q^{\alpha-\beta} \cdot f(\beta)\right) + \sum_\beta f(\beta)\right)$. By the linearity of trace, the resulting plaintext would be $\sum_\beta \left(\mathsf{Tr}(\zeta_q^{\alpha-\beta}) + 1\right) \cdot f(\beta)$, and by the equality test's property, this would be equal to $q \cdot f(\alpha)$, successfully extracting the desired term (with the scaling factor q).

The Case of Prime-Power Cyclotomic Rings. In the prime-power setting, i.e., $q = p^r$ where p is any prime number and $r > 1$, the previously discussed

equality test method is no longer applicable. To handle this, we use the equality test observed in [1] that works over arbitrary cyclotomic rings: $\sum_{i=0}^{q-1} \zeta_q^{(\alpha-\beta)i}$, which equals to q if $\alpha = \beta \bmod q$ or otherwise 0. To homomorphically evaluate this equality test however, we need to overcome the following challenges.

Challenge 1. As suggested in [1], homomorphic evaluation of this equality test requires $O(q)$ homomorphic multiplications for a general q, which means directly applying their method would not meet our pre-set goal. Moreover, we cannot utilize the linearity of trace to evaluate all the equality tests in parallel, as we did in the prime case. However, we found that in the prime-power case, this equality test can be related to the algebraic trace and evaluated with only $O(\log q)$ homomorphic multiplications. Our first key observation is a partition for $\mathbb{Z}_q = \{0, 1, \ldots, p^r - 1\}$, which is $\mathbb{Z}_q \setminus \{0\} = \bigcup_{i=1}^{r} p^{r-i} \cdot \mathbb{Z}_{p^i}^*$. Then, we can derive a new equivalent expression for the original equality test (see Lemma 5.2):

$$\sum_{i \in \mathbb{Z}_q} \zeta_q^{(\alpha-\beta)\cdot i} = 1 + \sum_{i=1}^{r} \mathsf{Tr}_{\mathbb{Q}(\zeta_{p^i})/\mathbb{Q}} \left(\zeta_{p^i}^{\alpha-\beta} \right), \tag{1.1}$$

which relates the algebraic trace of sub-extensions to the original equality test.

Challenge 2. In the new formula, we need to compute encryptions of $\zeta_{p^i}^{\alpha-\beta}$. How to efficiently obtain these encryptions from $\mathsf{RLWE.Enc}(\zeta_q^{\alpha})$ is a new challenge, e.g., using $O(1)$ homomorphic-friendly operations. To address this issue, we observe that $\zeta_{p^i}^{\alpha} = \zeta_q^{p^{r-i}\cdot\alpha} = \zeta_q^{(p^{r-i}-1)\cdot\alpha} \cdot \zeta_q^{\alpha}$. Thus, we can first perform an automorphism evaluation of $\zeta_q \mapsto \zeta_q^{p^{r-i}-1}$ to get $\mathsf{RLWE.Enc}\left(\zeta_q^{(p^{r-i}-1)\cdot\alpha}\right)$ and then use a homomorphic multiplication with $\mathsf{RLWE.Enc}(\zeta_q^{\alpha})$ to obtain $\mathsf{RLWE.Enc}(\zeta_{p^i}^{\alpha})$.

Challenge 3. If we compute all the trace of sub-extensions in the summation separately, the overall computational complexity could become somewhat large. Fortunately, we further observe that all the trace evaluations in the summation are contained in the tower of field extensions $\mathbb{Q}(\zeta_{p^r})/\mathbb{Q}(\zeta_{p^{r-1}})/\cdots/\mathbb{Q}(\zeta_p)/\mathbb{Q}$. The summation can be computed by adding the encryptions of sub-ring elements (e.g., $\mathsf{RLWE.Enc}(\zeta_{p^i}^{\alpha-\beta})$) to the intermediate result during the evaluation of the trace tower. Consequently, we only need to evaluate the trace once.

In Sect. 5, we elaborate on the new techniques we developed to overcome all the abovementioned challenges.

The Case of Composite Cyclotomic Rings. In the composite setting, i.e., $q = q_1 \cdots q_k$ where the q_i's are distinct prime-powers $p_i^{r_i}$ for $i \in \{1, \ldots, k\}$, we notice that all the prior methods based on algebraic trace cannot serve as the equality test again. To achieve our goal, our key insight is to propose the following equation with two branches for the (scaled) equality test:

$$\prod_{i=1}^{k} \left(\sum_{j \in \mathbb{Z}_{q_i}} \zeta_{q_i}^{(\alpha-\beta)\cdot j} \right) = \begin{cases} q & \text{if } \alpha = \beta \mod q \\ 0 & \text{if } \alpha \neq \beta \mod q \end{cases}.$$

Intuitively, this design captures the idea that $\alpha = \beta \bmod q$ if and only if $\alpha = \beta \bmod q_i$ for all the branches modulo q_i by the Chinese Remainder Theorem, and each parenthesis is an equality test from [1]. Since each q_i is some prime-power, we can combine it with Eq. 1.1 to get the following equivalent expression:

$$\prod_{i=1}^{k}\left(\sum_{j\in\mathbb{Z}_{q_i}}\zeta_{q_i}^{(\alpha-\beta)\cdot j}\right)=\prod_{i=1}^{k}\left(1+\sum_{j=1}^{r_i}\mathsf{Tr}_{\mathbb{Q}(\zeta_{p_i^j})/\mathbb{Q}}\left(\zeta_{p_i^j}^{\alpha-\beta}\right)\right)$$

which relates the equality test to algebraic trace. To homomorphically compute this equality test however, we need to tackle various challenges.

Challenge 1. While we can utilize our method for the prime-power case to handle each branch, the outer product form seems to require additional homomorphic multiplications on the results of several trace functions. This may incur significant computational cost and noise blowup. To address this issue, we find a new equivalent expression that is the sum of several trace functions (see Lemma 6.3):

$$\prod_{i=1}^{k}\left(1+\sum_{j=1}^{r_i}\mathsf{Tr}_{\mathbb{Q}(\zeta_{p_i^j})/\mathbb{Q}}\left(\zeta_{p_i^j}^{\alpha-\beta}\right)\right)=1+\sum_{w|q,w\neq 1}\mathsf{Tr}_{\mathbb{Q}(\zeta_w)/\mathbb{Q}}\left(\zeta_w^{\alpha-\beta}\right).$$

Challenge 2. In the new formula, we need to compute encryptions of $\zeta_w^{\alpha-\beta}$. How to efficiently obtain these encryptions from $\mathsf{RLWE.Enc}(\zeta_q^{\alpha})$ is a new challenge. Similar to our solution to Challenge 2 for the prime-power case, we can write $\zeta_w^{\alpha} = \zeta_q^{(q/w)\cdot\alpha} = \zeta_q^{(q/w-1)\cdot\alpha}\cdot\zeta_q^{\alpha}$. Unfortunately, $\zeta_q \mapsto \zeta_q^{q/w-1}$ may not be an automorphism when q is general, so we use a more general formula that $\zeta_w^{\alpha} = \zeta_q^{(q/w)\cdot\alpha} = \zeta_q^{(q/w-c)\cdot\alpha}\cdot\zeta_q^{c\cdot\alpha}$ for some $c\in\mathbb{Z}_q$. If both $q/w-c$ and c are in $\mathbb{Z}_q^*$, we can use two automorphism evaluations plus one homomorphic multiplication to obtain the encryption of ζ_w^{α}. We prove that we can always find such c for any $w \mid q, w \neq 1, q$ when q is an odd composite number (see Proposition 6.4).

Challenge 3. There exist many different trace computations in the summation, some of which may not contained in a consecutive tower. It seems that we need to perform these trace evaluations individually. To further improve efficiency, we identify a new algebraic equation (see Lemma 6.6) that allows one single computation of $\mathsf{Tr}_{\mathbb{Q}(\zeta_q)/\mathbb{Q}}$, integrating all the intermediate trace computations.

In Sect. 6, we further describe our new techniques and designs to address all the aforementioned challenges.

Computational Complexity. In our constructions, the computational complexity is dominated by the homomorphic evaluation of the algebraic trace, which would require $N-1$ homomorphic automorphism evaluations in a trivial way where N is the degree of field extension. It is currently known that there are two typical cases where the trace evaluation can be completed with much fewer (e.g., $O(\log N)$) automorphism evaluations. The first case is when the extension $\mathbb{Q}(\zeta_q)/\mathbb{Q}$ exhibits a tower structure which is widely used in [4,19,44,45]. The second case is when the extension $\mathbb{Q}(\zeta_q)/\mathbb{Q}$ is a cyclic extension (i.e., the

Galois group $\mathsf{Gal}(\mathbb{Q}(\zeta_q)/\mathbb{Q})$ is cyclic), which is first mentioned in [37] and generalized in [74]. These methods can be used to handle our prime-power case (as $\mathsf{Gal}(\mathbb{Q}(\zeta_q)/\mathbb{Q}) \cong \mathbb{Z}_q^*$ is cyclic for an odd prime-power q, and a power-of-2 q exhibits a base-2 logarithmic length tower of field extensions). It appears that there is currently no efficient solution for the composite case.

To address this, we have found that these two approaches can be combined. For example, suppose $q = q_1 q_2$ where q_1 and q_2 are two distinct odd prime-powers. Then we have the tower structure of extensions $\mathbb{Q}(\zeta_{q_1 q_2})/\mathbb{Q}(\zeta_{q_2})/\mathbb{Q}$, which fits the first case. Moreover, the Galois groups $\mathsf{Gal}(\mathbb{Q}(\zeta_{q_1 q_2})/\mathbb{Q}(\zeta_{q_2})) \cong \mathbb{Z}_{q_1}^*$ and $\mathsf{Gal}(\mathbb{Q}(\zeta_{q_2})/\mathbb{Q}) \cong \mathbb{Z}_{q_2}^*$ are both cyclic, which matches the second case. Thus, we can combine these two approaches to achieve a logarithmic complexity for the trace evaluation of the composite case. Furthermore, we found that this combination can be generalized to arbitrary cyclotomic extensions. We give an informal theorem below and refer to Sect. 3.3 for details.

Theorem 1.1 (Informal). *For a cyclotomic extension $\mathbb{Q}(\zeta_m)/\mathbb{Q}$ with $[\mathbb{Q}(\zeta_m) : \mathbb{Q}] = N$ and m being an arbitrary positive integer, $\mathsf{Tr}_{\mathbb{Q}(\zeta_m)/\mathbb{Q}}$ can be computed with $O(\log N)$ automorphisms. In the FHE context, homomorphic evaluation of $\mathsf{Tr}_{\mathbb{Q}(\zeta_m)/\mathbb{Q}}$ only requires $O(\log N)$ homomorphic automorphism evaluations.*

Why General Cyclotomic Rings?. Below, we discuss the rationale for considering general cyclotomic rings and further applications of our techniques.

- **More Modulus/Secret Choices:** In the FHEW/TFHE context, some recent works [43,70] have explored general LWE secret key distributions (as opposed to binary or ternary) to support more applications. In such cases, modulus switching may cause noise explosion when the norm of the secret key is relatively large. Hence, with the requirement that the LWE modulus q must divide the cyclotomic index m (to ensure the embedding $\mathbb{Z}_q \to \langle \zeta_m \rangle$), a flexible m allows for more options in the selection of q and secret key.
- **Compatibility with Batch Bootstrapping:** Our new algebraic insights are compatible with the Batch Bootstrapping framework of [44,45]. As the Batch paradigm crucially relies on tensor rings (including general cyclotomic rings for their tensor decompositions), our findings illuminate new paths for achieving SIMD functional bootstrapping for general functions within a polynomial modulus. Moreover, our strategy for trace computation can be employed in the framework to achieve the most flexible parameter choices.
- **General Applications:** Our new techniques are highly versatile and are applicable to schemes like BGV [13]/BFV [12,31], which inherently require general cyclotomic rings to support plaintext slots of finite fields or Galois rings [34,39,65]. Additionally, our new strategies for trace computation and new equality tests from insights on the structure of $\mathbb{Z}_q$ and $\mathbb{Z}_q^*$ may benefit other applications that rely on related computational number theory.

1.3 Other Related Work

A recent work [48] constructs new functional bootstrapping methods based on the BFV [12,31] scheme. However, their method would incur a super-polynomial

noise growth and thus require a super-polynomial modulus, which is not within the scope of the study in this work. Another recent work [52] improves the parameter selection and concrete efficiency of [26,47]. However, regarding functional bootstrapping for general functions, their algorithms do not show improvements in asymptotic complexity and functionality. Notably, the work [41] first discussed functional bootstrapping over general polynomial quotient rings, but their method fails to support both full-domain and general functions.

2 Preliminaries

2.1 Notations

In this paper, we denote the set of the rational numbers by $\mathbb{Q}$, the integers in $\mathbb{Q}$ by $\mathbb{Z}$, the real numbers by $\mathbb{R}$, and the complex numbers by $\mathbb{C}$. For an integer modulus q, $\mathbb{Z}_q = \mathbb{Z}/q\mathbb{Z}$ is the quotient ring of integers modulo q. We use the representative set $\mathbb{Z}_q = \{0, 1, \ldots, q-1\}$ for simplicity and let $[x]_q$ denote the modulo q operation into $\mathbb{Z}_q$ for an integer x. Let $[n] = \{1, \ldots, n\}$, where n is a positive integer. Notation log refers to the base-2 logarithm unless explicitly specified otherwise. We denote $[a, b]$ as the set $[a, b] \cap \mathbb{Z}$ for any integers $a \leq b$. We denote a column vector by a bold lower-case letter, e.g. $\boldsymbol{x}$, and x_i to denote the i-th entry of $\boldsymbol{x}$. The transpose of $\boldsymbol{x}$, namely the corresponding row vector, is denoted by $\boldsymbol{x}^\top$. We define the ℓ_∞-norm of $\boldsymbol{x}$ by $\|\boldsymbol{x}\|_\infty = \max_i\{|x_i|\}$.

Given a set $\mathbb{A}$ and a distribution $\mathcal{P}$ over $\mathbb{A}$, we use $a \leftarrow \mathbb{A}$ to denote that a is uniformly chosen from $\mathbb{A}$ and $a \leftarrow \mathcal{P}$ to denote that a is chosen randomly according to the distribution $\mathcal{P}$.

2.2 Subgaussian Random Variables

We call a random variable X over $\mathbb{R}$ is subgaussian with parameter $s > 0$, if for all $t \in \mathbb{R}$, the (scaled) moment-generating function satisfies: $\mathbb{E}[e^{2\pi tX}] \leq e^{\pi s^2 t^2}$. Especially, any B-bounded symmetric random variable X(i.e., $\mathbb{E}[X] = 0$ and $|X| \leq B$) is subgaussian with parameter $B\sqrt{2\pi}$. Subgaussians satisfy the following properties as discussed in [5,30]:

- *Homogeneity*: If X is a subgaussian variable with parameter s, then cX is subgaussian with parameter cs for any positive $c \in \mathbb{R}$.
- *Pythagorean additivity*: For $s_i \geq 0$, and random variables X_i for $i \in [k]$, if X_i is subgaussian with parameter s_i conditioning on any values of $X_1, ..., X_{i-1}$, then $\sum_{i\in[k]} X_i$ is subgaussian with parameter $(\sum_{i\in[k]} s_i^2)^{1/2}$.
- *Boundedness*: For any subgaussian variable X with parameter s, we have the probability bound $\Pr[|X| > t] < 2 \cdot \exp(-\pi t^2/s^2)$.

Remark 2.1. *For a subgaussian variable X with parameter δ_x, we have $\Pr[|X| > C \cdot \delta_x] < 2 \cdot \exp(-\pi \cdot C^2)$ by the boundedness. Hence, by setting C to be a proper constant, we can deduce that $|X| \leq C \cdot \delta_x$ with overwhelming probability. For another subgaussian variable Y with parameter δ_y that is independent of X, we use a subgaussian variable with parameter $O(\delta_x \cdot \delta_y)$ as an upper bound to demonstrate the asymptotic behavior of $|X \cdot Y|$ in this paper.*

2.3 Cyclotomic Rings

Let ζ_m be an m-th primitive root of unity. Then $K = \mathbb{Q}(\zeta_m)$ is an algebraic number field known as the m-th cyclotomic field, where the number m is referred to as the cyclotomic index. From algebraic number theory, the ring of the integers of the field K, which we usually denote by $\mathcal{O}_K$, is $\mathbb{Z}[\zeta_m]$. Note that we have $\mathbb{Z}[\zeta_m] \cong \mathbb{Z}[X]/(f(X))$ where $f(X)$ is the minimal polynomial of ζ_m with degree $N = \phi(m)$ (the Euler's totient of m). In the cyclotomic extension case, $f(X)$ is the m-th cyclotomic polynomial $\Phi_m(X) = \prod_{i \in \mathbb{Z}_m^*} (X - \omega_m^i) \in \mathbb{Z}[X]$, where $\mathbb{Z}_m^*$ denotes the set of integers in $\mathbb{Z}_m$ that are coprime to m, and $\omega_m \in \mathbb{C}$ is any primitive m-th complex root of unity, e.g., $\omega_m = e^{2\pi\sqrt{-1}/m}$.

Let $\mathcal{R} = \mathbb{Z}[\zeta_m]$ and $\mathcal{R}_q = \mathcal{R}/q\mathcal{R}$. Then the set $\{1, \zeta_m, \ldots, \zeta_m^{N-1}\}$ forms a $\mathbb{Z}$-basis of $\mathcal{R}$ and thus a $\mathbb{Z}_q$-basis of $\mathcal{R}_q$. This basis is often called the power basis of $\mathcal{R}$. For $a \in \mathcal{R}$, we can uniquely write it as $a = a_0 + a_1\zeta_m + \cdots + a_{N-1}\zeta_m^{N-1}$ where $a_i \in \mathbb{Z}$ for $i = 0, 1, \ldots, N-1$. We call $(a_0, \ldots, a_{N-1})$ the representation or the coefficient embedding under the power basis.

Canonical Embedding.T he m-th cyclotomic number field $K = \mathbb{Q}(\zeta_m)$ of degree $N = \phi(m)$ has exactly N ring embeddings $\sigma_i : K \to \mathbb{C}$ that fix every element of $\mathbb{Q}$. Let these embeddings be indexed by $\mathbb{Z}_m^*$. Then for $i \in \mathbb{Z}_m^*$, each embedding σ_i is defined by $\sigma_i(\zeta_m) = \omega_m^i$ where $\omega_m \in \mathbb{C}$ is some fixed complex primitive m-th root of unity (e.g., $\omega_m = e^{2\pi\sqrt{-1}/m}$). Then the canonical embedding $\sigma : K \to \mathbb{C}^N$ is defined as

$$\sigma(x) = (\sigma_i(x))_{i \in \mathbb{Z}_m^*}.$$

Note that it is a ring homomorphism from K to $\mathbb{C}^N$, where addition and multiplication in the latter are both component-wise. For $a \in \mathcal{R}$, the canonical embedding norm of a is defined as the ℓ_∞-norm of $\sigma(a)$, namely, $\|\sigma(a)\|_\infty$. It possesses the following nice property: For $a, b \in \mathcal{R}$, we have

- $\|\sigma(a+b)\|_\infty \leq \|\sigma(a)\|_\infty + \|\sigma(b)\|_\infty$,
- $\|\sigma(a \cdot b)\|_\infty \leq \|\sigma(a)\|_\infty \cdot \|\sigma(b)\|_\infty$.

Due to the independence of the representation of elements in $\mathcal{R}$ and the above property, we can easily bound the canonical embedding norm for elements in general cyclotomic rings. One can refer to [39,51] for the relation between the canonical embedding norm and the norm under some $\mathbb{Z}$-basis of $\mathcal{R}$.

Algebraic Trace. For two number fields $K' \subset K$, suppose the field extension K over K' (denoted as K/K') is a Galois extension with the Galois group $\mathsf{Gal}(K/K')$. Then for any element $a \in K$, the trace of a over K' is defined as

$$\mathsf{Tr}_{K/K'}(a) = \sum_{\tau \in \mathsf{Gal}(K/K')} \tau(a) \in K'.$$

An important fact is that $\mathsf{Tr}_{K/K'}(\mathcal{O}_K) \subseteq \mathcal{O}_{K'}$. Moreover, the trace $\mathsf{Tr}_{K/K'}(\cdot)$ has the K'-*linearity* as follows.

– For $a \in K, c \in K'$, we have $\mathsf{Tr}_{K/K'}(c \cdot a) = c \cdot \mathsf{Tr}_{K/K'}(a)$.
– For $a, b \in K$, we have $\mathsf{Tr}_{K/K'}(a + b) = \mathsf{Tr}_{K/K'}(a) + \mathsf{Tr}_{K/K'}(b)$.

For a tower of number field extensions $K_r/K_{r-1}/\cdots/K_2/K_1$, the trace has the following property, which is the so-called *transitivity*:

$$\mathsf{Tr}_{K_r/K_1}(a) = \mathsf{Tr}_{K_2/K_1}(\cdots(\mathsf{Tr}_{K_{r-1}/K_{r-2}}(\mathsf{Tr}_{K_r/K_{r-1}}(a)))\cdots)$$

for any $a \in K_r$. Moreover, we have the following useful fact for computation.

Lemma 2.2 ([51]). *Let m be a power of a prime p and $m' = m/p$, then for $i \in \mathbb{Z}$,*

$$\mathsf{Tr}_{\mathbb{Q}(\zeta_m)/\mathbb{Q}}(\zeta_m^i) = \begin{cases} \varphi(p) \cdot m' & \text{if } \ i = 0 \bmod m \\ -m' & \text{if } \ i = 0 \bmod m' \ \text{ and } \ i \neq 0 \bmod m \\ 0 & \text{otherwise.} \end{cases}$$

2.4 (Ring) Learning with Errors

The learning with errors (LWE) problem was first introduced by Regev [63]. Before the definition of LWE, we first introduce the distribution $A_{\boldsymbol{s},\chi}$. For a distribution χ over $\mathbb{Z}$ and a vector $\boldsymbol{s} \in \mathbb{Z}_q^n$, a sample from the distribution $A_{\boldsymbol{s},\chi}$ is of the form $(b, \boldsymbol{a}) \in \mathbb{Z}_q \times \mathbb{Z}_q^n$ with $b = [\langle \boldsymbol{a}, \boldsymbol{s} \rangle + e]_q$, where $\boldsymbol{a} \leftarrow \mathbb{Z}_q^n$ and $e \leftarrow \chi$.

Definition 2.3 (DLWE). *For a security parameter λ, let $n := n(\lambda)$ be an integer dimension, let $q = q(\lambda) \geq 2$ be an integer modulus, and let $\chi = \chi(\lambda)$ be an error distribution over $\mathbb{Z}$. Given some independent samples from $A_{\boldsymbol{s},\chi}$, the decision version of* LWE*, denoted by* $\mathsf{DLWE}_{n,q,\chi}$*, is to distinguish them from the same number of uniformly random and independent samples from $\mathbb{Z}_q \times \mathbb{Z}_q^n$.*

The $\mathsf{DLWE}_{n,q,\chi}$ problem defined above is known to be at least as hard as certain lattice problems [14,59,63]. To improve the efficiency of LWE-based schemes, the ring version of LWE, namely RLWE, was introduced [50,66]. We use the primal version of RLWE where the secret is defined in the ring rather than its dual. More discussions on different variants of RLWE can be found in [18,29,60,61].

For a distribution χ over $\mathcal{R}$ and a ring element $z \in \mathcal{R}_q$, a sample from the distribution $\mathcal{A}_{z,\chi}$ is of the form $(b, a) \in \mathcal{R}_q^2$ with $b = a \cdot z + e$ where $a \leftarrow \mathcal{R}_q$ and $e \leftarrow \chi$.

Definition 2.4 (RLWE). *For a security parameter λ, let $N := N(\lambda)$ be the degree of the ring, let $q = q(\lambda) \geq 2$ be an integer modulus, and let $\chi = \chi(\lambda)$ be an error distribution χ over $\mathcal{R}$. Given some independent samples from $\mathcal{A}_{z,\chi}$, the decision version of* RLWE*, denoted by* $\mathcal{R}$-$\mathsf{DLWE}_{N,q,\chi}$*, is to distinguish them from the same number of uniformly random and independent samples from $\mathcal{R}_q^2$.*

There are reductions showing that the $\mathcal{R}$-$\mathsf{DLWE}_{N,q,\chi}$ problem defined above is at least as hard as certain computational problems in ideal lattices [50,62].

2.5 (Ring) LWE-Based Symmetric Encryption

We first review a symmetric encryption scheme based on LWE, which is the base scheme to be bootstrapped. We use the most significant bits (MSBs) for the plaintext encoding for ease of description. Note that the algorithms we propose in this work are independent of the plaintext encoding methods for the input LWE scheme. The basic definitions of LWE-based encryption are as follows.

- **Encryption.** Let $\mathbb{Z}_t$ be the plaintext domain, and q be the LWE modulus. Then the set of valid LWE ciphertexts for plaintext $\mu \in \mathbb{Z}_t$ under the secret key $\boldsymbol{s}$, denoted as $\mathsf{LWE}_{\boldsymbol{s}}(\delta \cdot \mu)$[5] is defined as:

$$\mathsf{LWE}_{\boldsymbol{s}}(\delta \cdot \mu) = \{([\langle \boldsymbol{a}, \boldsymbol{s} \rangle + e + \delta \cdot \mu]_q, \boldsymbol{a}) \in \mathbb{Z}_q^{n+1}\},$$

  where $\boldsymbol{a} \in \mathbb{Z}_q^n$ and $e \leftarrow \chi$ for some typical error distribution χ over $\mathbb{Z}$ (e.g., Gaussian), and $\delta = \lfloor \frac{q}{t} \rceil$ is the scaling factor.
- **Decryption.** A ciphertext $\boldsymbol{c} = (b, \boldsymbol{a}) \in \mathbb{Z}_q^{n+1}$ can be decrypted by computing

$$\mathsf{Dec}(\boldsymbol{s}, \boldsymbol{c}) = \mathsf{Dcd}([b - \langle \boldsymbol{a}, \boldsymbol{s} \rangle]_q)$$

  where $\mathsf{Dcd} : \mathbb{Z}_q \to \mathbb{Z}_t$ is the decoding function $\mathsf{Dcd}(x) = [\lfloor \frac{t}{q} \cdot x \rceil]_t$.

The above notation can be extended to RLWE-based schemes. Let $\mathcal{R}_p$ be the plaintext space and Q be the ciphertext modulus. Then, we define the following set for valid RLWE encryptions of the plaintext $\mu \in \mathcal{R}_p$ under the secret key z:

$$\mathsf{RLWE}_z(\Delta \cdot \mu) = \left\{([a \cdot z + e + \Delta \cdot \mu]_Q, a) \in \mathcal{R}_Q^2\right\},$$

where $a \in \mathcal{R}_Q$ and $e \leftarrow \chi$ for some typical error distribution χ over $\mathcal{R}$ (e.g., Gaussian), and $\Delta = \lfloor \frac{Q}{p} \rceil$ is the scaling factor. Note that $\lfloor \cdot \rceil$ and $[\cdot]_Q$ for elements in $\mathcal{R}$ mean coordinate-wise rounding to the nearest integer and coordinate-wise modulo Q with respect to some fixed $\mathbb{Z}$-basis of $\mathcal{R}$, respectively. We use the following notation to denote the error in a ciphertext.

Definition 2.5. *For a ciphertext $\boldsymbol{c} \in \mathsf{RLWE}_z(\mu)$, the error of $\boldsymbol{c}$ is defined as*

$$\mathsf{Err}(\boldsymbol{c}) := \langle (1, -z), \boldsymbol{c} \rangle - \mu.$$

3 Basic Homomorphic Computations

In this section, we review some necessary background on homomorphic encryption, encompassing homomorphic operations, along with some new techniques.

We start with the homomorphic operations on RLWE ciphertexts used in this work. More operations, including the external product with Ring-GSW [23,30,35] ciphertexts over general cyclotomic rings can be found in [44,45]. We assume that the underlying ring for the RLWE scheme is $\mathcal{R} = \mathbb{Z}[\zeta_m]$ with $N = \phi(m)$, and the ciphertext modulus is Q. The proofs of all lemmas, propositions, and theorems in this section are provided in the full version.

[5] An equivalent notation $\mathsf{LWE}_{\boldsymbol{s}}^{t/q}(\mu)$ is also used in the literature.

3.1 BFV Homomorphic Multiplication

Let $\mathsf{BFV.Mul}(\cdot)$ be a homomorphic multiplication algorithm [12,31] that takes as input two RLWE ciphertexts $\boldsymbol{c}_i \in \mathsf{RLWE}_z(\Delta \cdot \mu_i)$ where $i = 1, 2$ and $\Delta := \lfloor Q/p \rceil$, and some relinearization key RelKey, and outputs a RLWE ciphertext $\boldsymbol{c}' \in \mathsf{RLWE}_z(\Delta \cdot \mu_1 \cdot \mu_2)$. Since we will only encounter the case where the plaintext is some root of unity, the following lemma is restricted to this case only for a simpler error bound.

Lemma 3.1. *Suppose* $\boldsymbol{c}' \leftarrow \mathsf{BFV.Mul}(\boldsymbol{c}_1, \boldsymbol{c}_2)$ *and both* $\boldsymbol{c}_1, \boldsymbol{c}_2$ *are encryptions of roots of unity. If* $\|\sigma(\mathsf{Err}(\boldsymbol{c}_1))\|_\infty$ *and* $\|\sigma(\mathsf{Err}(\boldsymbol{c}_2))\|_\infty$ *are upper bounded by subgaussian variables with parameter* δ_1, δ_2, $\|\sigma(\mathsf{Err}(\mathsf{RelKey}))\|_\infty = \|\sigma(z)\|_\infty = O(\sqrt{N})$. *Then* $\|\sigma(\mathsf{Err}(\boldsymbol{c}'))\|_\infty$ *is upper bounded by a subgaussian variable with parameter* $O(\sqrt{(p/Q)^2\delta_1^2\delta_2^2 + p^2N^2(\delta_1^2+\delta_2^2) + N^2 \log Q})$.

3.2 Homomorphic Automorphism Evaluation

We also use homomorphic automorphism evaluation over general cyclotomic rings as studied and used in [37,39,44,45]. Let $\mathsf{EvalAuto}(\cdot)$ be a homomorphic evaluation algorithm that takes as input a RLWE ciphertext $\boldsymbol{c} \in \mathsf{RLWE}_z(\mu)$, an automorphism $\tau \in \mathsf{Gal}(\mathbb{Q}(\zeta_m)/\mathbb{Q})$ and some automorphism key AutKey, and outputs a RLWE ciphertext $\boldsymbol{c}' \in \mathsf{RLWE}_z(\tau(\mu))$. The following lemma demonstrates the error growth in the homomorphic automorphism evaluation.

Lemma 3.2. *Suppose that* $\boldsymbol{c}' \leftarrow \mathsf{EvalAuto}(\boldsymbol{c}, \tau, \mathsf{AutKey})$. *If* $\|\sigma(\mathsf{Err}(\boldsymbol{c}))\|_\infty$ *and* $\|\sigma(\mathsf{Err}(\mathsf{AutKey}))\|_\infty$ *are upper bounded by subgaussian variables with parameter* δ_c *and* $\delta_{\mathtt{aut}}$ *respectively, then* $\|\sigma(\mathsf{Err}(\boldsymbol{c}'))\|_\infty$ *is upper bounded by a subgaussian variable with parameter* $O(\sqrt{\delta_c^2 + \delta_{\mathtt{aut}}^2 N \log Q})$.

3.3 Homomorphic Trace Evaluation

By definition, the trace evaluation requires performing all automorphisms in the Galois group $\mathsf{Gal}(\mathbb{Q}(\zeta_m)/\mathbb{Q})$ and summing the results, which trivially necessitates $N-1$ automorphism evaluations. However, it is currently known that there are two typical cases where the trace evaluation can be computed with much fewer (e.g., $O(\log N)$) automorphism evaluations: (1) when the extension $\mathbb{Q}(\zeta_m)/\mathbb{Q}$ exhibits a tower structure [4,19,44,45], and (2) when the extension $\mathbb{Q}(\zeta_m)/\mathbb{Q}$ is a cyclic extension, i.e., $\mathsf{Gal}(\mathbb{Q}(\zeta_m)/\mathbb{Q})$ forms a cyclic group [37,74]. We first discuss each case below in a more general style (i.e., using relative field extensions).

The Tower of Extensions Case. Suppose the tower of number field extensions $\mathbb{Q}(\zeta_m) = K_r/K_{r-1}/\cdots/K_1/K_0 = \mathbb{Q}$ where each K_i/K_{i-1} is a Galois extension for all $i \in [r]$. For $0 \le j < i \le r$, let $\mathsf{EvalTr}_{K_i/K_j}(\cdot)$ be a homomorphic evaluation algorithm that takes as input a RLWE ciphertext $\boldsymbol{c} \in \mathsf{RLWE}_z(\mu)$ and some automorphism key AutKey, and outputs a RLWE ciphertext $\boldsymbol{c}' \in \mathsf{RLWE}_z(\mathsf{Tr}_{K_i/K_j}(\mu))$.

Then, owing to the transitivity of the trace, $\mathsf{Tr}_{\mathbb{Q}(\zeta_m)/\mathbb{Q}}(\mu) = \mathsf{Tr}_{K_r/K_0}(\mu)$ can be homomorphically evaluated as follows (we omit AutKey for simplicity):

$$\mathsf{EvalTr}_{K_1/K_0}\left(\mathsf{EvalTr}_{K_2/K_1}\left(\cdots \mathsf{EvalTr}_{K_r/K_{r-1}}(\boldsymbol{c})\cdots\right)\right).$$

Denote the degree of the extensions by $[K_i : K_{i-1}] = d_i$ for $i \in [r]$ (hence $N = \prod_{i\in[r]} d_i$), thus the above sequence of homomorphic computation requires only $\sum_{i\in[r]}(d_i - 1)$ automorphism evaluations (-1 comes from the identity maps). The following lemma provides the error growth of trace evaluation in this case.

Lemma 3.3. *Suppose that* $\boldsymbol{c}' \leftarrow \mathsf{EvalTr}_{K_i/K_j}(\boldsymbol{c}, \mathsf{AutKey})$. *If* $\|\sigma(\mathsf{Err}(\boldsymbol{c}))\|_\infty$ *and* $\|\sigma(\mathsf{Err}(\mathsf{AutKey}))\|_\infty$ *are upper bounded by subgaussian variables with parameters* δ_c *and* $\delta_{\mathtt{aut}}$, *respectively. Then* $\|\sigma(\mathsf{Err}(\boldsymbol{c}'))\|_\infty$ *is upper bounded by a subgaussian variable with parameter* $O(\sqrt{d\delta_c^2 + (d-1)\delta_{\mathtt{aut}}^2 N \log Q})$ *where* $d = [K_i : K_j]$.

The Cyclic Extension Case. Suppose the Galois extensions $\mathbb{Q}(\zeta_m)/K/F/\mathbb{Q}$ with $[K : F] = M$ and $\mathsf{Gal}(K/F)$ being a cyclic group with generator τ_g, then we can write $\mathsf{Gal}(K/F) = \{\mathsf{id} = \tau_g^0, \tau_g, \tau_g^2, \ldots, \tau_g^{M-1}\}$ where id is the identity map. Let $T_k(\mu) = \sum_{i\in[0,k-1]} \tau_g^i(\mu)$ for $\mu \in F$, then we have

$$T_k(\mu) = \begin{cases} T_{k/2}(\mu) + \tau_g^{k/2}(T_{k/2}(\mu)) & \text{if } k \text{ is even} \\ T_{(k-1)/2}(\mu) + \tau_g^{(k-1)/2}\left(T_{(k-1)/2}(\mu)\right) + \tau_g^{k-1}(\mu) & \text{if } k \text{ is odd} \end{cases}.$$

Consequently, we can reduce the scale of the required automorphism summations in a recursive way for the homomorphic evaluation of $\mathsf{Tr}_{K/F}(\mu) = T_M(\mu)$. It has been proved in [74] that the homomorphic evaluation of $T_M(\mu)$ requires at most $2\log M$ (which is essentially $O(\log M)$) automorphism evaluations. The following useful fact captures the case of cyclotomic extensions.

Fact 3.4 ([69]) *For a cyclotomic field extension* K/F *with* $K = F(\zeta_t)$ *and* $[K : F] = \phi(t)$, *we have* $\mathsf{Gal}(K/F) \cong \mathbb{Z}_t^*$, *which is cyclic if and only if* $t = 1, 2, 4, p^r, 2p^r$ *where* p *is an odd prime and* r *is some positive integer.*

Let $\mathsf{EvalTr}_{K/F}(\cdot)$ be a homomorphic evaluation algorithm that takes as input a RLWE ciphertext $\boldsymbol{c} \in \mathsf{RLWE}_z(\mu)$ and some automorphism key AutKey, and outputs a RLWE ciphertext $\boldsymbol{c}' \in \mathsf{RLWE}_z(\mathsf{Tr}_{K/F}(\mu))$. The following lemma provides the error growth of trace evaluation in the cyclic extension case.

Lemma 3.5 *Suppose that* $\boldsymbol{c}' \leftarrow \mathsf{EvalTr}_{K/F}(\boldsymbol{c}, \mathsf{AutKey})$. *If* $\|\sigma(\mathsf{Err}(\boldsymbol{c}))\|_\infty$ *and* $\|\sigma(\mathsf{Err}(\mathsf{AutKey}))\|_\infty$ *are upper bounded by subgaussian variables with parameters* δ_c *and* $\delta_{\mathtt{aut}}$, *respectively. Then* $\|\sigma(\mathsf{Err}(\boldsymbol{c}'))\|_\infty$ *is upper bounded by a subgaussian variable with parameter* $O(\sqrt{M\delta_c^2 + (M-1)\delta_{\mathtt{aut}}^2 N \log Q})$.

Our Combination. By integrating the above two methods, we can construct efficient trace evaluation algorithms over arbitrary cyclotomic rings. Specifically, let the cyclotomic index m be an arbitrary positive integer. By the fundamental

theorem of arithmetic, we can write $m = \prod_{i\in[r]} p_i^{e_i}$ where p_i's are distinct primes, and e_i's are positive integers. Then denote $K_i = \mathbb{Q}\left(\zeta_{\prod_{j=1}^{i} p_j^{e_j}}\right)$ for $i \in [r]$, we have the tower of extensions $\mathbb{Q}(\zeta_m) = K_r/K_{r-1}/\cdots/K_1/K_0 = \mathbb{Q}$. To further compute $\mathsf{Tr}_{K_i/K_{i-1}}$ in the tower, we have the following discussion that covers all possible cases based on whether m is odd or even.

- **Case 1:** If $p_i \neq 2$ for all $i \in [r]$, we have $K_i = K_{i-1}\left(\zeta_{p_i^{e_i}}\right)$ which implies $\mathsf{Gal}(K_i/K_{i-1}) \cong \mathbb{Z}_{p_i^{e_i}}$ and thus K_i/K_{i-1} is a cyclic extension by Fact 3.4. Hence, each $\mathsf{Tr}_{K_i/K_{i-1}}$ can be computed with $O\left(\log \phi(p_i^{e_i})\right)$ automorphisms, yielding a total of $\sum_{i\in[r]} O\left(\log \phi(p_i^{e_i})\right) = O(\log N)$ for $\mathsf{Tr}_{\mathbb{Q}(\zeta_m)/\mathbb{Q}}$.
- **Case 2:** If $p_i = 2$ for some $i \in [r]$, suppose without loss of generality that $p_1 = 2$ and $K_1 = \mathbb{Q}(\zeta_{2^{e_1}})$. Then we have a second layer of tower extensions: $K_1 = \mathbb{Q}(\zeta_{2^{e_1}})/\mathbb{Q}(\zeta_{2^{e_1-1}})/\cdots/\mathbb{Q}(\zeta_2) = \mathbb{Q}$, which implies $\mathsf{Tr}_{K_1/\mathbb{Q}}$ requires $O(\log 2^{e_1})$ automorphisms. As discussed in Case 1, each $\mathsf{Tr}_{K_i/K_{i-1}}$ requires $O\left(\log \phi(p_i^{e_i})\right)$ automorphisms for $i > 1$, yielding a total of $O(\log N)$ for $\mathsf{Tr}_{\mathbb{Q}(\zeta_m)/\mathbb{Q}}$.

All these strategies can be naturally extended to homomorphic evaluations of $\mathsf{Tr}_{\mathbb{Q}(\zeta_m)/\mathbb{Q}}$ in the ciphertext domain for arbitrary cyclotomic index m. We use the following theorem to summarize the above discussion.

Theorem 3.6 *Suppose the underlying ring $\mathcal{R} = \mathbb{Z}[\zeta_m]$ with m being an arbitrary positive integer. For a* RLWE *ciphertext $\boldsymbol{c} \in \mathsf{RLWE}_z(\mu)$ and some automorphism key* AutKey*, there exist efficient algorithms $\mathsf{EvalTr}_{\mathbb{Q}(\zeta_m)/\mathbb{Q}}(\boldsymbol{c}, \mathsf{AutKey})$ that output $\boldsymbol{c}' \in \mathsf{RLWE}_z(\mathsf{Tr}_{\mathbb{Q}(\zeta_m)/\mathbb{Q}}(\mu))$ using only $O(\log N)$ automorphism evaluations.*

Moreover, if $\|\sigma(\mathsf{Err}(\boldsymbol{c}))\|_\infty$ and $\|\sigma(\mathsf{Err}(\mathsf{AutKey}))\|_\infty$ are upper bounded by subgaussian variables with parameters δ_c and $\delta_{\mathtt{aut}}$, respectively. Then, for both Case 1 and 2, the output error norm $\|\sigma(\mathsf{Err}(\boldsymbol{c}'))\|_\infty$ is upper bounded by a subgaussian variable with parameter $O(\sqrt{N\delta_c^2 + (N-1)\delta_{\mathtt{aut}}^2 N \log Q})$.

3.4 Computational Complexity

For the sake of comparison, we standardize the units of measurement for the computational complexity of homomorphic operations. Same as the FHEW/TFHE bootstrapping algorithms, the most time-consuming operation in our algorithms is the external product [22,23,44,45]. Thus, we propose the following proposition that demonstrates the relationship between the costs of the above homomorphic operations and the external product.

Proposition 3.7 *Suppose the same underlying ring $\mathcal{R} = \mathbb{Z}[\zeta_m]$ with the same modulus for all homomorphic operations. We have the following approximations:*

- *For the same gadget decomposition base, both* $\mathsf{BFV.Mul}(\cdot)$ *and* $\mathsf{EvalAuto}(\cdot)$ *require approximately $1/2$ times the cost of the external product.*
- *For different gadget decomposition bases, both* $\mathsf{BFV.Mul}(\cdot)$ *and* $\mathsf{EvalAuto}(\cdot)$ *require $O(1)$ times the cost of the external product.*

Consequently, $\mathsf{EvalTr}_{\mathbb{Q}(\zeta_m)/\mathbb{Q}}(\cdot)$ *requires $O(\log N)$ times the cost of the external product for arbitrary positive integer m.*

4 Functional Bootstrapping: A Warm-Up

In this section, we first review the techniques in the FHEW/TFHE bootstrapping framework, clarifying the context in which our algorithm will be operational. Then, we present the core building block of our functional bootstrapping algorithm over prime cyclotomic rings, which constitutes the simplest case.

4.1 The Functional Bootstrapping Framework

Recall the task of the functional bootstrapping – basically, the algorithm takes input an LWE ciphertext $\boldsymbol{c}$ with some bootstrapping key and returns an LWE ciphertext $\boldsymbol{c}'$ that encrypts $f(\mathsf{Dec}(\boldsymbol{c}))$ for some pre-determinded function f.

Parameters. We first describe parameters used in the FHEW/TFHE (functional) bootstrapping and this work. Note that all the parameters with magnitudes (i.e., n, q, t, m, Q, h) are polynomially bounded in the security parameter.

- n: The dimension of the input LWE scheme.
- q: The modulus of the input LWE scheme.
- t: The plaintext modulus of the input LWE scheme.
- $\boldsymbol{s}$: The secret key of the input LWE scheme.
- $\mathcal{R}$: The underlying ring $\mathbb{Z}[\zeta_m]$ of the RLWE scheme with $q \mid m$ and $N := \phi(m)$.
- Q: The modulus of the RLWE scheme.
- z: The secret key of the RLWE scheme.
- h: The plaintext modulus of the output LWE ciphertext.

The FHEW/TFHE Framework. On input an LWE ciphertext $\boldsymbol{c} = (b, \boldsymbol{a}) \in \mathsf{LWE}_{\boldsymbol{s}}^{t/q}(\mu)$, the bootstrapping algorithm first performs a Blind Rotation to obtain a RLWE ciphertext $\tilde{\boldsymbol{c}} \in \mathsf{RLWE}_z(v \cdot \zeta_q^{\varphi})$ where $\varphi = [b - \langle \boldsymbol{a}, \boldsymbol{s} \rangle]_q$ and $v \in \mathcal{R}_Q$ is some encoding of the composite of a negacyclic function $f : \mathbb{Z}_t \to \mathbb{Z}_h$ and the decoding function Dcd. The constant term (i.e., the coefficient of the basis 1) of the plaintext of $\tilde{\boldsymbol{c}}$ is actually $\Delta' \cdot f(\mathsf{Dcd}(\varphi)) = \Delta' \cdot f(\mu)$ where $\Delta' \approx Q/h$. Thus, the algorithm proceeds a Sample Extract to obtain an LWE ciphertext $\boldsymbol{c}' \in \mathsf{LWE}_{\boldsymbol{z}}^{h/Q}(f(\mu))$ where $\boldsymbol{z}$ is the coefficient vector of z. Finally, it performs a sequence of modulus switching, key switching, and modulus switching to obtain a ciphertext in $\mathsf{LWE}_{\boldsymbol{s}}^{h/q}(f(\mu))$. The framework is illustrated in Fig. 1. We state the functionality and error analysis of the Blind Rotation in the following lemma and refer to the details and proof in [23] and the full version of this work.

Lemma 4.1 (Blind Rotation [23], Adapted) *For a ciphertext $\boldsymbol{c} = (b, \boldsymbol{a}) \in \mathsf{LWE}_{\boldsymbol{s}}^{t/q}(\mu)$ where $\boldsymbol{s} \in \{0,1\}^n$, there exists an algorithm $\mathsf{BlindRotate}$ that takes input $\boldsymbol{c}$ and the bootstrapping key BK, and outputs a RLWE ciphertext $\boldsymbol{c}' \in \mathsf{RLWE}_z(v \cdot \zeta_q^{b - \langle \boldsymbol{a}, \boldsymbol{s} \rangle})$ for arbitrary $v \in \mathcal{R}_Q$, requiring n times external product. If $\|\sigma(\mathsf{Err}(\mathsf{BK}))\|_\infty$ is upper bounded by a subgaussian variable with parameter δ_{bk} for all $i \in [n]$, then $\|\sigma(\mathsf{Err}(\boldsymbol{c}'))\|_\infty$ is upper bounded by a subgaussian variable with parameter $O(\delta_{\mathsf{bk}} \sqrt{nN \log Q})$.*

Remark 4.2 *The secret $\boldsymbol{s}$ can be ternary or general distributions as well. Several recent results [9,43,53,70,71] show how to do Blind Rotation for these cases with comparable (or slightly less) efficiency as the case of binary secrets.*

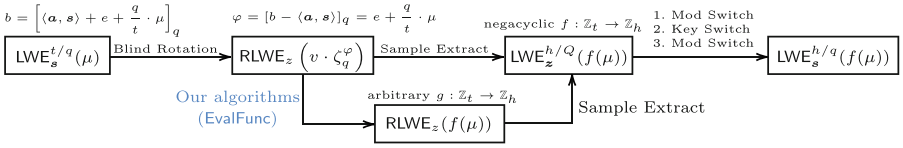


Fig. 1. The relation of FHEW/TFHE bootstrapping framework and our algorithms.

Our Blueprint. As also depicted in Fig. 1, our approach broadly adheres to the same line as the FHEW/TFHE framework. The sole distinction lies in that, subsequent to obtaining the ciphertext through Blind Rotation, we construct a series of new algorithms $\mathsf{EvalFunc}$, each of which works over different cyclotomic rings and only incurs polynomial error growth. The algorithms take as input a ciphertext $\boldsymbol{c} \in \mathsf{RLWE}_z(v \cdot \zeta_q^{\varphi})$ for some $v \in \mathcal{R}_Q$ and some auxiliary keys, and output a RLWE ciphertext $\boldsymbol{c}' \in \mathsf{RLWE}_z(f(\varphi))$ for an arbitrary function $f : \mathbb{Z}_q \to \mathbb{Z}_h$. Let $f = g \circ \mathsf{Dcd}$ for an arbitrary function $g : \mathbb{Z}_t \to \mathbb{Z}_h$, then $\boldsymbol{c}'$ actually encrypts $f(\varphi) = g(\mathsf{Dcd}(\varphi)) = g(\mu)$.

Our advantage of computational efficiency stems from the fact that the algorithm $\mathsf{EvalFunc}$ requires only $O(\log \phi(q))$ times the cost of the external product. By adopting the typical parameter setting in FHEW/TFHE bootstrapping that $q = O(n)$, the overall cost is approximately $n + O(\log n)$ times the cost of the external product. Considering that regular FHEW/TFHE bootstrapping requires n external products plus minimal overhead, our findings suggest that functional bootstrapping for arbitrary functions is essentially as efficient as regular bootstrapping. In other words, the ratio of efficiency between functional and regular FHEW/TFHE bootstrapping approaches $1 + o(1)$.

The functional superiority, such as the support for arbitrary input plaintext modulus/encoding and general functions, is derived from our adoption of new equality test techniques. Specifically, our constructions rely on the fact that any discrete function can be expressed by a linear combination of the equality test function. Define the equality test for the exponent of ζ_q as

$$\mathsf{EqT}(\zeta_q^{\alpha}, \beta) = \begin{cases} 1 & \text{if} \quad \alpha = \beta \mod q \\ 0 & \text{if} \quad \alpha \neq \beta \mod q \end{cases},$$

then for any $\alpha \in \mathbb{Z}_q$, we have $f(\alpha) = \sum_{\beta \in \mathbb{Z}_q} f(\beta) \cdot \mathsf{EqT}(\zeta_q^{\alpha}, \beta)$.

Remaining Tasks. Now, our remaining task is to find efficient solutions for the homomorphic evaluation of EqT over cyclotomic rings, based on which we can instantiate the algorithm $\mathsf{EvalFunc}$. In the following subsection, we present an instantiation of $\mathsf{EvalFunc}$ over prime cyclotomic rings, and subsequently, we will delve into other instantiations for more general cases.

Algorithm 4.1. $\mathsf{EvalFunc}(\boldsymbol{c}, \mathsf{AutKey}, f)$

Parameters:

Δ: the scaling factor $\lfloor Q/(h \cdot q) \rceil$ of the input encoding
Δ': the scaling factor $\Delta \cdot q \approx Q/h$ of the output encoding
q: a prime number satisfying $q \mid m$
h: the plaintext modulus of the output ciphertext
v: an encoding of the function $v := \Delta \cdot \sum_{\beta \in \mathbb{Z}_q} f(\beta) \cdot \zeta_q^{-\beta} \in \mathcal{R}_Q$

Input:

A RLWE ciphertext $\boldsymbol{c} \in \mathsf{RLWE}_z(v \cdot \zeta_q^{\alpha})$
The key for homomorphic automorphism evaluation AutKey
An arbitrary function $f : \mathbb{Z}_q \to \mathbb{Z}_h$

Output: A RLWE ciphertext $\boldsymbol{c}' \in \mathsf{RLWE}_z(\Delta' \cdot f(\alpha))$

1: $\boldsymbol{c}' \leftarrow \mathsf{EvalTr}_{\mathbb{Q}(\zeta_q)/\mathbb{Q}}(\boldsymbol{c}, \mathsf{AutKey})$
2: $\boldsymbol{c}' \leftarrow \boldsymbol{c}' + (\Delta \cdot \sum_{\beta \in \mathbb{Z}_q} f(\beta), 0)^{\top}$
3: **return** $\boldsymbol{c}'$

4.2 Arbitrary Function Evaluation over Prime Cyclotomic Rings

We first consider the simple case where q is prime. From Lemma 2.2, we identify the following equation that can serve as the equality test: For any $\alpha, \beta \in \mathbb{Z}_q$,

$$\mathsf{Tr}_{\mathbb{Q}(\zeta_q)/\mathbb{Q}}(\zeta_q^{\alpha-\beta}) + 1 = \begin{cases} q & \text{if} \quad \alpha = \beta \mod q \\ 0 & \text{if} \quad \alpha \neq \beta \mod q \end{cases}. \tag{4.1}$$

Then, we present the instantiation of $\mathsf{EvalFunc}$ in Algorithm 4.1 and its correctness together with analysis in Theorem 4.3.

Theorem 4.3 *Algorithm 4.1 is correct, i.e., the input-output behavior satisfies as described. Moreover, it possesses the following properties:*

- *Complexity: it requires $O(\log \phi(q))$ times the cost of the external product.*
- *Error growth: if $\|\sigma(\mathsf{Err}(\boldsymbol{c}))\|_\infty$ and $\|\sigma(\mathsf{Err}(\mathsf{AutKey}))\|_\infty$ are upper bounded by subgaussian variables with parameters δ_c and $\delta_{\mathtt{aut}}$. Then $\|\sigma(\mathsf{Err}(\boldsymbol{c}'))\|_\infty$ is upper bounded by a subgaussian variable with parameter*

$$O\left(\sqrt{(q-1)\delta_c^2 + (q-2)\delta_{\mathtt{aut}}^2 N \log Q}\right).$$

Proof. A simple calculation yields that in *line* 2 we have $\boldsymbol{c}'$ encrypts

$$\begin{aligned} & \mathsf{Tr}_{\mathbb{Q}(\zeta_q)/\mathbb{Q}}\left(\Delta \cdot \sum_{\beta \in \mathbb{Z}_q} f(\beta) \cdot \zeta_q^{\alpha-\beta}\right) + \Delta \cdot \sum_{\beta \in \mathbb{Z}_q} f(\beta) \\ &= \Delta \cdot \sum_{\beta \in \mathbb{Z}_q} f(\beta) \cdot \left(\mathsf{Tr}_{\mathbb{Q}(\zeta_q)/\mathbb{Q}}(\zeta_q^{\alpha-\beta}) + 1\right) && \text{(by } \mathbb{Q}\text{-linearity)} \\ &= \Delta \cdot f(\alpha) \cdot q && \text{(by Eq. 4.1)} \\ &= \Delta' \cdot f(\alpha) && \text{(by the definition of } \Delta' := \Delta \cdot q) \end{aligned}$$

Since the errors can be upper bounded by subgaussian variables, we summarize the corresponding subgaussian parameter in each line as follows.

- $\boldsymbol{c}'$ in *line* 1: $O\big(\sqrt{(q-1)\delta_c^2 + (q-2)\delta_{\mathtt{aut}}^2 N \log Q}\big)$ by Lemma 3.5.
- $\boldsymbol{c}'$ in *line* 2: Adding an error-free ciphertext does not affect the error.

This proves the claim of error growth. The claim of computational complexity follows directly from Theorem 3.6 and Proposition 3.7. □

5 The Case of Prime-Power Cyclotomic Rings

In this section, we focus on the case of prime-power cyclotomic rings, namely $q = p^r$ for any prime number p and integer $r > 1$. In this case, the previously discussed instantiation for the prime case based on trace plus one is no longer applicable (see Lemma 2.2). Alternatively, we consider another instantiation of equality test observed in [1] that works over arbitrary cyclotomic rings:

$$\sum_{i=0}^{q-1} \zeta_q^{(\alpha-\beta)\cdot i} = 1 + \zeta_q^{\alpha-\beta} + \cdots + \zeta_q^{(q-1)(\alpha-\beta)} = \begin{cases} q & \text{if } \alpha = \beta \bmod q \\ 0 & \text{if } \alpha \neq \beta \bmod q \end{cases}. \tag{5.1}$$

To homomorphically evaluate this equality test however, we need to confront the following challenges.

Challenge 1. As also suggested in [1], homomorphic evaluation of the above equality test given an encryption of $\zeta_q^{\alpha-\beta}$ requires $O(q)$ homomorphic multiplications for a general q (and also requires Ring-GSW encryptions), which means directly applying their method would not meet our pre-set goal (Sect. 4.1). Additionally, this formula seems to preclude computing all the equality tests in parallel as we did for the prime case based on the linearity of the trace.

Solution. We will show that this goal can be achieved with only $O(\log \phi(q))$ homomorphic multiplications for the special case of prime-power cyclotomic rings (and only requires RLWE encryption of $\zeta_q^{\alpha-\beta}$). Our approach associates the equality test with the trace to exploit its linearity and computational efficiency. As the trace $\mathsf{Tr}_{\mathbb{Q}(\zeta_q)/\mathbb{Q}}(\zeta_q^{\alpha-\beta}) = \sum_{i\in\mathbb{Z}_q^*} \zeta_q^{(\alpha-\beta)\cdot i}$ can handle the powers in $\mathbb{Z}_q^*$, but the equality test requires the powers in $\mathbb{Z}_q$ (we use the representative set $\mathbb{Z}_q = [0, q-1]$), we thus first establish the following relation between $\mathbb{Z}_q^*$ (and its subgroups) and $\mathbb{Z}_q$, which is actually a partition of $\mathbb{Z}_q$ for a prime-power q.

Lemma 5.1 *For any prime number p and positive integer r, we have*

$$\mathbb{Z}_{p^r} \setminus \{0\} = \bigcup_{i\in[r]} p^{r-i} \cdot \mathbb{Z}_{p^i}^* \quad \Big(= \mathbb{Z}_{p^r}^* \cup p \cdot \mathbb{Z}_{p^{r-1}}^* \cup \cdots \cup p^{r-1} \cdot \mathbb{Z}_p^*\Big).$$

Proof. By definition, $\mathbb{Z}_{p^i}^*$ is the set of all numbers in $\mathbb{Z}_{p^i}$ that are coprime to p^i, which is equivalent to the set of all numbers in $\mathbb{Z}_{p^i}$ that are not divisible by p.

Since the set of all numbers in $\mathbb{Z}_{p^i}$ that are divisible by p is $p \cdot \mathbb{Z}_{p^{i-1}}$, we have $\mathbb{Z}_{p^i} = \mathbb{Z}^*_{p^i} \cup p \cdot \mathbb{Z}_{p^{i-1}}$ for all $i \in [r]$. Thus, by induction, we have

$$\begin{aligned}\mathbb{Z}_{p^r} &= \mathbb{Z}^*_{p^r} \cup p \cdot \mathbb{Z}_{p^{r-1}} \\ &= \mathbb{Z}^*_{p^r} \cup p \cdot \mathbb{Z}^*_{p^{r-1}} \cup p^2 \cdot \mathbb{Z}_{p^{r-2}} \\ &= \cdots \\ &= \mathbb{Z}^*_{p^r} \cup p \cdot \mathbb{Z}^*_{p^{r-1}} \cup \cdots \cup p^{r-1} \cdot \mathbb{Z}^*_p \cup \{0\},\end{aligned}$$

which completes the proof. □

Now, we are able to establish the following lemma that relates the algebraic trace to the equality test for the prime-power case.

Lemma 5.2 *For any $\alpha, \beta \in \mathbb{Z}_q$ where $q = p^r$ and p is any prime, we have*

$$\sum_{i \in \mathbb{Z}_q} \zeta_q^{(\alpha-\beta)\cdot i} = 1 + \sum_{i \in [r]} \mathsf{Tr}_{\mathbb{Q}(\zeta_{p^i})/\mathbb{Q}}\left(\zeta_{p^i}^{\alpha-\beta}\right)$$

Proof. We can verify that

$$\begin{aligned}\sum_{i \in \mathbb{Z}_q} \zeta_q^{(\alpha-\beta)\cdot i} &= \zeta_q^{(\alpha-\beta)\cdot 0} + \sum_{i \in [r]} \sum_{j \in p^{r-i} \cdot \mathbb{Z}^*_{p^i}} \zeta_q^{(\alpha-\beta)\cdot j} && \text{(by Lemma 5.1)} \\ &= 1 + \sum_{i \in [r]} \sum_{j \in \mathbb{Z}^*_{p^i}} \zeta_q^{p^{r-i}\cdot(\alpha-\beta)\cdot j} && \text{(modify the index } j\text{)} \\ &= 1 + \sum_{i \in [r]} \mathsf{Tr}_{\mathbb{Q}(\zeta_{p^i})/\mathbb{Q}}\left(\zeta_{p^i}^{\alpha-\beta}\right) && \text{(by } \star \text{ and the definition of trace)}\end{aligned}$$

where "$\star$" is the fact that $\zeta_q^{p^{r-i}} = \zeta_q^{q/p^i} = \zeta_{p^i} \in \mathbb{Z}[\zeta_{p^i}]$. □

To homomorphically evaluate this new equation of equality test however, we encounter the following new challenges.

Challenge 2. The input in each trace is of the form $\zeta_{p^i}^{\alpha-\beta}$. We need to efficiently compute this term from ζ_q^α, e.g., using $O(1)$ homomorphic-friendly operations.

Solution: We present an efficient method that only takes two homomorphic-friendly operations. Particularly, consider the automorphism $\tau : \zeta_q \mapsto \zeta_q^{p^{r-i}-1}$ as $p^{r-i}-1 \in \mathbb{Z}^*_q$ for $i \in [1, r-1]$, and then we have $\zeta_{p^i}^\alpha = \zeta_q^{p^{r-i}\cdot\alpha} = \zeta_q^{(p^{r-i}-1)\cdot\alpha} \cdot \zeta_q^\alpha = \tau(\zeta_q^\alpha) \cdot \zeta_q^\alpha$. For the homomorphic evaluation, this requires one homomorphic automorphism and one homomorphic multiplication, which would be roughly $O(1)$ homomorphic multiplication.

Challenge 3. If we compute the trace computations in the summation separately, the overall computational complexity is $\sum_{i \in [r]} O(\log \phi(p^i)) = O(\log^2 \phi(q))$. Although this is already much better than the $O(q)$ complexity of the method in [1], it still does not meet the requirements outlined in our blueprint (Sect. 4.1).

Algorithm 5.1. $\mathsf{EvalFunc}(\boldsymbol{c}, \mathsf{AutKey}, \mathsf{RelKey}, f)$

Parameters:

Δ: the scaling factor $\lfloor Q/(h \cdot q) \rceil$ of the input encoding
Δ': the scaling factor $\Delta \cdot q \approx Q/h$ of the output encoding
q: a prime power $q = p^r$ for some small prime p and integer $r > 1$
h: the plaintext modulus of the output ciphertext

Input:

A RLWE ciphertext $\boldsymbol{c} \in \mathsf{RLWE}_z(\Delta \cdot \zeta_q^{\alpha})$
The key for homomorphic automorphism evaluation AutKey
The relinearization key for BFV multiplication RelKey
An arbitrary function $f : \mathbb{Z}_q \to \mathbb{Z}_h$

Output: A RLWE ciphertext $\boldsymbol{c}' \in \mathsf{RLWE}_z(\Delta' \cdot f(\alpha))$.

$\boldsymbol{c}' \leftarrow \boldsymbol{c} \cdot \left(\sum_{\beta \in \mathbb{Z}_q} f(\beta) \cdot \zeta_q^{-\beta}\right)$
$\boldsymbol{c}' \leftarrow \mathsf{EvalTr}_{\mathbb{Q}(\zeta_q)/\mathbb{Q}(\zeta_{p^{r-1}})}(\boldsymbol{c}', \mathsf{AutKey})$
for $i \in [1, r-1]$ **do**
  $\boldsymbol{c}_{\mathtt{tmp}} \leftarrow \mathsf{EvalAuto}(\boldsymbol{c}, \zeta_q \mapsto \zeta_q^{p^i - 1}, \mathsf{AutKey})$
  $\boldsymbol{c}_{\mathtt{tmp}} \leftarrow \mathsf{BFV.Mul}(\boldsymbol{c}, \boldsymbol{c}_{\mathtt{tmp}}, \mathsf{RelKey})$
  $\boldsymbol{c}_{\mathtt{tmp}} \leftarrow \boldsymbol{c}_{\mathtt{tmp}} \cdot \left(\sum_{\beta \in \mathbb{Z}_q} f(\beta) \cdot \zeta_{p^{r-i}}^{-\beta}\right)$
  $\boldsymbol{c}' \leftarrow \boldsymbol{c}' + \boldsymbol{c}_{\mathtt{tmp}}$
  $\boldsymbol{c}' \leftarrow \mathsf{EvalTr}_{\mathbb{Q}(\zeta_{p^{r-i}})/\mathbb{Q}(\zeta_{p^{r-i-1}})}(\boldsymbol{c}', \mathsf{AutKey})$
end for
$\boldsymbol{c}' \leftarrow \boldsymbol{c}' + (\Delta \cdot \sum_{\beta \in \mathbb{Z}_q} f(\beta), 0)^{\top}$
return $\boldsymbol{c}'$

Solution: We further observe that all the trace evaluations in the summation are contained in the tower of field extensions $\mathbb{Q}(\zeta_{p^r})/\mathbb{Q}(\zeta_{p^{r-1}})/\cdots/\mathbb{Q}(\zeta_p)/\mathbb{Q}$. Hence, we can directly evaluate the trace following the tower when p is small (e.g., $p = 2, 3$, and see Remark 5.4 and Sect. 6 for solutions for large p). The summation can be computed by adding the encryptions of $\sum f(\beta) \cdot \zeta_{p^i}^{\alpha-\beta}$ to the intermediate result during the evaluation of the trace tower. Consequently, the overall computational complexity is reduced to $O(\log \phi(q))$ since we only need to evaluate the trace once following the tower of field extensions.

We are now ready to present our construction. We provide the formal description of $\mathsf{EvalFunc}$ over prime-power cyclotomic rings in Algorithm 5.1 and its correctness along with analysis in Theorem 5.3.

Theorem 5.3 *Algorithm 5.1 is correct, i.e., the input-output behavior satisfies what is described. Moreover, it possesses the following properties:*

- *Complexity: it requires $O(\log \phi(q))$ times the cost of the external product.*
- *Error growth: if $\|\sigma(\mathsf{Err}(\boldsymbol{c}))\|_\infty$, $\|\sigma(\mathsf{Err}(\mathsf{AutKey}))\|_\infty$ and $\|\sigma(\mathsf{Err}(\mathsf{RelKey}))\|_\infty$ are upper bounded by subgaussian variables with parameters δ_c, $\delta_{\mathtt{aut}}$ and δ_r, respectively. Let $\delta_{\mathtt{aut}}, \delta_r, \delta_z = O(\sqrt{N})$ and assume that $\delta_c < \Delta$, then we have $\|\sigma(\mathsf{Err}(\boldsymbol{c}'))\|_\infty$ is upper bounded by a subgaussian variable with parameter*

$$O\left(Nh^2 q^{2.5} \sqrt{2\delta_c^2 + N^2 \log Q}\right).$$

Proof. We first analyze the ciphertext $\boldsymbol{c}_{\mathtt{tmp}}$ in the loop from *line* 3 to *line* 9.

- In *line* 4, we have $\boldsymbol{c}_{\mathtt{tmp}} \in \mathsf{RLWE}_z(\Delta \cdot \zeta_q^{(p^i-1)\cdot\alpha})$ by Lemma 3.2.
- In *line* 5, we have $\boldsymbol{c}_{\mathtt{tmp}} \in \mathsf{RLWE}_z(\Delta \cdot \zeta_q^{p^i\cdot\alpha}) \subset \mathsf{RLWE}_z(\Delta \cdot \zeta_{p^{r-i}}^{\alpha})$ by Lemma 3.1.
- In *line* 6, we have $\boldsymbol{c}_{\mathtt{tmp}} \in \mathsf{RLWE}_z\left(\Delta \cdot \sum_{\beta\in\mathbb{Z}_q}\left(f(\beta)\cdot\zeta_{p^{r-i}}^{\alpha-\beta}\right)\right)$.

After *line* 9, we have the plaintext of $\boldsymbol{c}_{\mathtt{tmp}}$ in the i-th iteration (for $i \in [1, r-1]$) goes through the evaluation of

$$\begin{aligned}
&\mathsf{Tr}_{\mathbb{Q}(\zeta_p)/\mathbb{Q}}\left(\cdots \mathsf{Tr}_{\mathbb{Q}(\zeta_{p^{r-i}})/\mathbb{Q}(\zeta_{p^{r-i-1}})}\left(\Delta\cdot\sum_{\beta\in\mathbb{Z}_q}\left(f(\beta)\cdot\zeta_{p^{r-i}}^{\alpha-\beta}\right)\right)\cdots\right)\\
&= \mathsf{Tr}_{\mathbb{Q}(\zeta_{p^{r-i}})/\mathbb{Q}}\left(\Delta\cdot\sum_{\beta\in\mathbb{Z}_q}\left(f(\beta)\cdot\zeta_{p^{r-i}}^{\alpha-\beta}\right)\right). \qquad \text{(by transitivity)}
\end{aligned}$$

Thus, after *line* 10, we have that $\boldsymbol{c}'$ encrypts

$$\begin{aligned}
&\Delta\cdot\sum_{\beta\in\mathbb{Z}_q} f(\beta) + \sum_{i\in[0,r-1]} \mathsf{Tr}_{\mathbb{Q}(\zeta_{p^{r-i}})/\mathbb{Q}}\left(\Delta\cdot\sum_{\beta\in\mathbb{Z}_q}\left(f(\beta)\cdot\zeta_{p^{r-i}}^{\alpha-\beta}\right)\right)\\
&= \Delta\cdot\sum_{\beta\in\mathbb{Z}_q} f(\beta) + \sum_{i\in[r]} \mathsf{Tr}_{\mathbb{Q}(\zeta_{p^i})/\mathbb{Q}}\left(\Delta\cdot\sum_{\beta\in\mathbb{Z}_q}\left(f(\beta)\cdot\zeta_{p^{i}}^{\alpha-\beta}\right)\right) \qquad \text{(modify index)}\\
&= \Delta\cdot\sum_{\beta\in\mathbb{Z}_q} f(\beta) + \Delta\cdot\sum_{\beta\in\mathbb{Z}_q} f(\beta)\cdot\left(\sum_{i\in[r]} \mathsf{Tr}_{\mathbb{Q}(\zeta_{p^i})/\mathbb{Q}}\left(\zeta_{p^i}^{\alpha-\beta}\right)\right) \qquad \text{(by } \mathbb{Q}\text{-linearity)}\\
&= \Delta\cdot\sum_{\beta\in\mathbb{Z}_q} f(\beta)\cdot\left(1+\sum_{i\in[r]} \mathsf{Tr}_{\mathbb{Q}(\zeta_{p^i})/\mathbb{Q}}\left(\zeta_{p^i}^{\alpha-\beta}\right)\right) \qquad \text{(combine terms)}\\
&= \Delta\cdot\sum_{\beta\in\mathbb{Z}_q} f(\beta)\cdot\begin{cases} q & \text{if } \alpha=\beta \mod q\\ 0 & \text{if } \alpha\neq\beta \mod q\end{cases} \qquad \text{(by Lemma 5.2 and Eqs. 5.1)}\\
&= \Delta\cdot f(\alpha)\cdot q \qquad \text{(only } f(\alpha) \text{ is multiplied by } q\text{, others are multiplied by 0)}\\
&= \Delta'\cdot f(\alpha), \qquad \text{(by the definition of } \Delta' := \Delta\cdot q)
\end{aligned}$$

which proves the correctness. Since the errors can be upper bounded by subgaussian variables, we summarize the corresponding subgaussian parameter in each line as follows.

- $\boldsymbol{c}'$ in *line* 1: $O(h\delta_c\sqrt{q})$ by the fact that $\|\sigma(\sum_{\beta\in\mathbb{Z}_q} f(\beta)\cdot\zeta_q^{-\beta})\|_\infty \le h\sqrt{q}$.
- $\boldsymbol{c}'$ in *line* 2: $O(\sqrt{pqh^2\delta_c^2 + (p-1)N^2\log Q})$ by Theorem 3.6.
- $\boldsymbol{c}_{\mathtt{tmp}}$ in *line* 4: $O(\sqrt{\delta_c^2 + N^2\log Q})$ by Lemma 3.2.
- $\boldsymbol{c}_{\mathtt{tmp}}$ in *line* 5: $O(hqN\sqrt{2\delta_c^2 + N^2\log Q})$ by Lemma 3.1 and $\delta_c < \Delta$.

– $\boldsymbol{c}_{\texttt{tmp}}$ in *line* 6: $O(Nh^2q^{1.5}\sqrt{2\delta_c^2 + N^2 \log Q})$ by the fact that

$$\left\| \sigma \left(\sum_{\beta \in \mathbb{Z}_q} f(\beta) \cdot \zeta_{p^{r-i}}^{-\beta} \right) \right\|_\infty = O(h\sqrt{q}),$$

which is obtained by modeling $f(\beta)$ as a uniformly random variable in $\mathbb{Z}_h$.
– $\boldsymbol{c}'$ after *line* 9: $O(Nh^2q^{2.5}\sqrt{2\delta_c^2 + N^2 \log Q})$ by Theorem 3.6.
– $\boldsymbol{c}'$ in *line* 10: Adding an error-free ciphertext does not affect the error.

This proves the claim of error growth. For the computational complexity, we have $r = \log_p q = O(\log \phi(q))$ for the loop from *line* 3 to 9, and each loop contains one automorphism evaluation and one BFV multiplication, which implies $O(\log \phi(q))$ times the cost of external product by Proposition 3.7. Moreover, all the trace evaluation totally requires $O(\log \phi(q))$ times the cost of the external product by Proposition 3.7, which implies our claim of computational complexity. □

Remark 5.4 *The strategy for trace evaluation in Algorithm 5.1 can achieve a logarithmic complexity only when p is small (e.g., $p = 2, 3$). For larger p, we can employ a new general scaled equality test (see Lemma 6.6), allowing us to perform the trace evaluation of $\mathsf{Tr}_{\mathbb{Q}(\zeta_{p^r})/\mathbb{Q}}$ only once. The complexity in this case corresponds to the trace evaluation of the cyclic extension case as discussed in Sect. 3.3 and thus has a logarithmic complexity. Since this scenario is a special case of the situations we will address in the next section, we omit its details here.*

6 The Case of Composite Cyclotomic Rings

Now we move to the most general case where q is a composite number, i.e., $q = \prod_{i=1}^{k} q_i$, and q_i's are distinct prime-powers. Below, we first describe some critical intuitions and lemmas, and then present the final algorithm.

We first use the plaintext computation for an easier explanation of our idea. Recall our high-level goal of equality test: given $\zeta_q^{\alpha-\beta}$ (for some $\alpha, \beta \in \mathbb{Z}_q$), we can compute γ where $\gamma = q$ if $\alpha = \beta \bmod q$ or otherwise $\gamma = 0$, using some homomorphic-friendly operations. In the setting of composite $q = \prod_{i=1}^{k} q_i$, we identify several challenges where the techniques from the prior section do not carry through easily. Particularly, Eq. 5.1 still holds, but we can not directly apply Lemma 5.2 to relate it with algebraic trace. To tackle this, we identify the following equation:

$$\prod_{i=1}^{k} \left(\sum_{j \in \mathbb{Z}_{q_i}} \zeta_{q_i}^{(\alpha-\beta)\cdot j} \right) = \begin{cases} q & \text{if } \alpha = \beta \mod q \\ 0 & \text{if } \alpha \neq \beta \mod q \end{cases}. \tag{6.1}$$

The intuition is clear: $\alpha = \beta \bmod q$ if and only if $\alpha = \beta \bmod q_i$ for all the branches modulo q_i by the Chinese Remainder Theorem. As each $\sum_{j \in \mathbb{Z}_{q_i}} \zeta_{q_i}^{(\alpha-\beta)\cdot j}$

can serve as the equality test for the branch modulo q_i, one can easily verify the validity of this formula. Denote $q_i = p_i^{r_i}$ where p_i is prime for $i \in [k]$, we have

$$\prod_{i=1}^{k}\left(\sum_{j\in\mathbb{Z}_{q_i}} \zeta_{q_i}^{(\alpha-\beta)\cdot j}\right) = \prod_{i=1}^{k}\left(1+\sum_{j\in[r_i]} \mathsf{Tr}_{\mathbb{Q}(\zeta_{p_i^j})/\mathbb{Q}}\left(\zeta_{p_i^j}^{\alpha-\beta}\right)\right) \tag{6.2}$$

by Lemma 5.2, which serves as our new equality test. To homomorphically evaluate this new equation, here comes the first (and main) challenge.

(Main) Challenge 1. While we can utilize the method in Sect. 5 to compute the trace summation, elegantly handling the outer product poses a challenging problem. Specifically, we need to avoid the direct consecutive use of BFV multiplication due to the rapid error growth it would introduce.

Solution. We elaborate a critical equivalent expression as Eq. 6.2. Before introducing our new equation, we need to first discuss several elegant properties we have discovered regarding trace computations in some special cases.

Lemma 6.1 *Suppose $q = q_1 q_2$ where q_1 and q_2 are coprime, then for any $i \in \mathbb{Z}$,*

$$\mathsf{Tr}_{\mathbb{Q}(\zeta_q)/\mathbb{Q}}(\zeta_q^i) = \mathsf{Tr}_{\mathbb{Q}(\zeta_{q_1})/\mathbb{Q}}(\zeta_{q_1}^i)\cdot \mathsf{Tr}_{\mathbb{Q}(\zeta_{q_2})/\mathbb{Q}}(\zeta_{q_2}^i).$$

Proof. Using the linearity and transitivity of the trace, we can verify that

$$\begin{aligned}
\mathsf{Tr}_{\mathbb{Q}(\zeta_q)/\mathbb{Q}}(\zeta_q^i) &= \mathsf{Tr}_{\mathbb{Q}(\zeta_q)/\mathbb{Q}}(\zeta_{q_1}^i\cdot\zeta_{q_2}^i) && \text{(by coprimality of } q_1, q_2)\\
&= \mathsf{Tr}_{\mathbb{Q}(\zeta_{q_2})/\mathbb{Q}}\left(\mathsf{Tr}_{\mathbb{Q}(\zeta_q)/\mathbb{Q}(\zeta_{q_2})}(\zeta_{q_1}^i\cdot\zeta_{q_2}^i)\right) && \text{(by transitivity)}\\
&= \mathsf{Tr}_{\mathbb{Q}(\zeta_{q_2})/\mathbb{Q}}\left(\zeta_{q_2}^i\cdot\mathsf{Tr}_{\mathbb{Q}(\zeta_q)/\mathbb{Q}(\zeta_{q_2})}(\zeta_{q_1}^i)\right) && \text{(by } \mathbb{Q}(\zeta_{q_2})\text{-linearity)}\\
&= \mathsf{Tr}_{\mathbb{Q}(\zeta_q)/\mathbb{Q}(\zeta_{q_2})}(\zeta_{q_1}^i)\cdot\mathsf{Tr}_{\mathbb{Q}(\zeta_{q_2})/\mathbb{Q}}(\zeta_{q_2}^i) && \text{(by } \star \text{ and } \mathbb{Q}\text{-linearity)}\\
&= \mathsf{Tr}_{\mathbb{Q}(\zeta_{q_1})/\mathbb{Q}}(\zeta_{q_1}^i)\cdot\mathsf{Tr}_{\mathbb{Q}(\zeta_{q_2})/\mathbb{Q}}(\zeta_{q_2}^i) && \text{(by } \star)
\end{aligned}$$

where "$\star$" is the fact $\mathsf{Tr}_{\mathbb{Q}(\zeta_q)/\mathbb{Q}(\zeta_{q_2})}(\zeta_{q_1}^i) = \mathsf{Tr}_{\mathbb{Q}(\zeta_{q_1})/\mathbb{Q}}(\zeta_{q_1}^i)\in\mathbb{Z}$. □

Corollary 6.2 *Suppose $q = \prod_{i=1}^{k} q_i$ where all the q_i's are distinct prime-powers, then for any $j \in \mathbb{Z}$,*

$$\prod_{i=1}^{k}\mathsf{Tr}_{\mathbb{Q}(\zeta_{q_i})/\mathbb{Q}}(\zeta_{q_i}^j) = \mathsf{Tr}_{\mathbb{Q}(\zeta_q)/\mathbb{Q}}(\zeta_q^j) \quad \left(=\mathsf{Tr}_{\mathbb{Q}(\zeta_{\prod_{i=1}^{k} q_i})/\mathbb{Q}}\left(\zeta_{\prod_{i=1}^{k} q_i}^j\right)\right).$$

Proof. By continuously using Lemma 6.1, we have

$$\begin{aligned}
\prod_{i=1}^{k}\mathsf{Tr}_{\mathbb{Q}(\zeta_{q_i})/\mathbb{Q}}(\zeta_{q_i}^j) &= \mathsf{Tr}_{\mathbb{Q}(\zeta_{q_1})/\mathbb{Q}}(\zeta_{q_1}^j)\cdot\mathsf{Tr}_{\mathbb{Q}(\zeta_{q_2})/\mathbb{Q}}(\zeta_{q_2}^j)\cdots\mathsf{Tr}_{\mathbb{Q}(\zeta_{q_k})/\mathbb{Q}}(\zeta_{q_k}^j)\\
&= \mathsf{Tr}_{\mathbb{Q}(\zeta_{q_1q_2})/\mathbb{Q}}(\zeta_{q_1q_2}^j)\cdot\mathsf{Tr}_{\mathbb{Q}(\zeta_{q_3})/\mathbb{Q}}(\zeta_{q_3}^j)\cdots\mathsf{Tr}_{\mathbb{Q}(\zeta_{q_k})/\mathbb{Q}}(\zeta_{q_k}^j)\\
&= \cdots = \mathsf{Tr}_{\mathbb{Q}(\zeta_{q_1\cdots q_k})/\mathbb{Q}}(\zeta_{q_1\cdots q_k}^j),
\end{aligned}$$

which is exactly $\mathsf{Tr}_{\mathbb{Q}(\zeta_q)/\mathbb{Q}}(\zeta_q^j)$. □

Now, we can establish the following equivalent expression for Eq. 6.2.

Lemma 6.3 *For any $\alpha, \beta \in \mathbb{Z}_q$ where q is any positive integer with prime-power factorization $q = \prod_{i=1}^{k} p_i^{r_i}$, we have*

$$\prod_{i=1}^{k} \left(1 + \sum_{j \in [r_i]} \mathsf{Tr}_{\mathbb{Q}(\zeta_{p_i^j})/\mathbb{Q}} \left(\zeta_{p_i^j}^{\alpha-\beta}\right)\right) = 1 + \sum_{w|q, w \neq 1} \mathsf{Tr}_{\mathbb{Q}(\zeta_w)/\mathbb{Q}} \left(\zeta_w^{\alpha-\beta}\right).$$

Proof. After the direct expansion of the equation on the left, we have the summation of all possible products of $\mathsf{Tr}_{\mathbb{Q}(\zeta_{p_i^j})/\mathbb{Q}} \left(\zeta_{p_i^j}^{\alpha-\beta}\right)$ for each $i \in [k]$ and $j \in [r_i]$. Since all p_i^j's are distinct prime-powers, we can apply Corollary 6.2 to make the products of trace "into" the products of the indices of primitive roots of unity. These products actually traverse all the factors of q (except 1), so the result exactly matches the equation on the right. □

To homomorphically evaluate this new equation of equality test however, we encounter the following new challenges.

Challenge 2. The input in each trace is of the form $\zeta_w^{\alpha-\beta}$. We need to efficiently compute this term from ζ_q^{α}, i.e., using $O(1)$ homomorphic-friendly operations.

Solution. Similar to our solution to Challenge 2 in Sect. 5, we can write $\zeta_w^{\alpha} = \zeta_q^{(q/w)\cdot\alpha} = \zeta_q^{(q/w-1)\cdot\alpha} \cdot \zeta_q^{\alpha}$. Unfortunately, $\zeta_q \mapsto \zeta_q^{q/w-1}$ may not be an automorphism, so we need a more general formula that $\zeta_w^{\alpha} = \zeta_q^{(q/w)\cdot\alpha} = \zeta_q^{(q/w-c)\cdot\alpha} \cdot \zeta_q^{c\cdot\alpha}$ for some $c \in \mathbb{Z}_q$. We hope that both $q/w - c$ and c are in $\mathbb{Z}_q^*$, then we can use two automorphism evaluations plus one homomorphic multiplication to obtain the encryption of ζ_w^{α}. However, we can not always find such c for all $w \mid q, w \neq 1, q$ when q is an arbitrary number. Especially we can not choose an even q, the intuition is straightforward: For any odd factor p of q, we will encounter the case of computing ζ_q^p. Since p is odd, either $p - c$ or c is an even number, which implies either $p - c$ or c is not in $\mathbb{Z}_q^*$ as $2 \mid q$. Instead, we can prove its existence for any odd composite q as below.

Proposition 6.4 *Let q be a odd number has factorization $q = \prod_{i=1}^{k} p_i^{r_i}$ where p_i's are distinct odd primes and $k > 1$. For each $v \mid q, v \neq 1, q$, there exists at least one $c \in \mathbb{Z}_q^*$ such that $[v - c]_q \in \mathbb{Z}_q^*$.*

Proof. Suppose all the prime factors contained in v are p_i for $i \in S \subsetneq [k]$ and $S \neq \emptyset$. Then, we can construct c as $c = \left[\prod_{j \in [k] \setminus S} p_j - v\right]_q$. Now, we will prove that both c and $[v - c]_q$ are in $\mathbb{Z}_q^*$.

- Write $c = \prod_{j \in [k] \setminus S} p_j - v + q \cdot I \in \mathbb{Z}_q$ for some $I \in \mathbb{Z}$ (in fact, $I \in \{0, 1\}$). Suppose $c \notin \mathbb{Z}_q^*$, which implies that c and q share at least one common prime

factor p. Namely, $p \in \{p_i\}_{i\in[k]}$ and $p \mid c$. Then

$$\text{If } p \in \{p_i\}_{i\in S} \Longrightarrow p \mid v \overset{p|q}{\Longrightarrow} p \mid (v - q \cdot I) \overset{p|c}{\Longrightarrow} p \mid \prod_{j\in[k]\setminus S} p_j, \text{ a contradiction.}$$

$$\text{If } p \in \{p_j\}_{j\in[k]\setminus S} \overset{p|q}{\Longrightarrow} p \mid \prod_{j\in[k]\setminus S} p_j + q \cdot I \overset{p|c}{\Longrightarrow} p \mid v, \text{ a contradiction.}$$

Hence, c and q share no common prime factor, which implies $c \in \mathbb{Z}_q^*$.

– Write $[v - c]_q = 2 \cdot v - \prod_{j\in[k]\setminus S} p_j + q \cdot J \in \mathbb{Z}_q$ for some $J \in \mathbb{Z}$, we can easily show $[v - c] \in \mathbb{Z}_q^*$ by a quite similar argument as the prior proof of $c \in \mathbb{Z}_q^*$ since q does not contain the factor 2.

Now, the remaining case is $S = [k]$, i.e., v contains all the prime factors of q. In this case, we can easily verify that $v - 1 \in \mathbb{Z}_q^*$, which completes the proof. □

Remark 6.5 *Our proof of Proposition 6.4 is constructive and provides a direct method to determine one desired c for any v. However, for many values of v, we observe by experiments using* `SageMath` [68] *that $c = 1$ is also usable. We prioritize a usable $v - 1$ as it saves one homomorphic automorphism evaluation.*

Challenge 3. There exist many different trace computations in the summation (Lemma 6.3), where some of them may not contained in a consecutive tower down to $\mathbb{Q}$. Thus, we can not apply the strategy in Sect. 5, and computing them separately will not meet the pre-set goal in our blueprint (Sect. 4.1).

Solution. We present a new technique that "swaps" the order of summation and trace, meaning that we can first aggregate the inputs and just compute the trace once. Particularly, we prove the following lemma as our final equality test:

Lemma 6.6 *For any positive integer $q > 1$ and any $\alpha, \beta \in \mathbb{Z}_q$, we have*

$$\mathsf{Tr}_{\mathbb{Q}(\zeta_q)/\mathbb{Q}}\left(1 + \sum_{w|q, w\neq 1} \phi(w) \cdot \zeta_w^{\alpha-\beta}\right) = \begin{cases} \phi(q) \cdot q & \text{if } \alpha = \beta \mod q \\ 0 & \text{if } \alpha \neq \beta \mod q \end{cases}.$$

Proof. We can verify that

$$\begin{aligned}
&\mathsf{Tr}_{\mathbb{Q}(\zeta_q)/\mathbb{Q}}\left(1 + \sum_{w|q, w\neq 1} \phi(w) \cdot \zeta_w^{\alpha-\beta}\right) \\
&= \phi(q) + \sum_{w|q, w\neq 1} \phi(w) \cdot \mathsf{Tr}_{\mathbb{Q}(\zeta_q)/\mathbb{Q}}\left(\zeta_w^{\alpha-\beta}\right) && \text{(by } \mathbb{Q}\text{-linearity)} \\
&= \phi(q) + \sum_{w|q, w\neq 1} \phi(w) \cdot \frac{\phi(q)}{\phi(w)} \cdot \mathsf{Tr}_{\mathbb{Q}(\zeta_w)/\mathbb{Q}}\left(\zeta_w^{\alpha-\beta}\right) && (\star) \\
&= \phi(q) \cdot \left(1 + \sum_{w|q, w\neq 1} \mathsf{Tr}_{\mathbb{Q}(\zeta_w)/\mathbb{Q}}\left(\zeta_w^{\alpha-\beta}\right)\right) && \text{(combine terms)} \\
&= \begin{cases} \phi(q) \cdot q & \text{if } \alpha = \beta \mod q \\ 0 & \text{if } \alpha \neq \beta \mod q \end{cases} && \text{(by Lemma 6.3 and Eq. 6.1, 6.2)}
\end{aligned}$$

Algorithm 6.1. $\mathsf{EvalFunc}(\boldsymbol{c}, \mathsf{AutKey}, \mathsf{RelKey}, f)$

Parameters:

Δ: the scaling factor $\lfloor Q/(h \cdot q \cdot \phi(q)) \rceil$ of the input encoding
Δ': the scaling factor $\Delta \cdot q \cdot \phi(q) \approx Q/h$ of the output encoding
q: an odd number satisfying $q \mid m$ with prime-power factorization $q = \prod_{i \in [k]} p_i^{r_i}$
$\{c_v\}$: the automorphism index set $\{c_v\}_{v|q, v \neq 1, q} \subset \mathbb{Z}_q^*$ obtained by Proposition 6.4 and Remark 6.5 such that $\{v - c_v\}_{v|q, v \neq 1, q} \subset \mathbb{Z}_q^*$.
h: the plaintext modulus of the output ciphertext

Input:

A RLWE ciphertext $\boldsymbol{c} \in \mathsf{RLWE}_z(\Delta \cdot \zeta_q^{\alpha})$
The key for homomorphic automorphism evaluation AutKey
The relinearization key for BFV multiplication RelKey
An arbitrary function $f : \mathbb{Z}_q \to \mathbb{Z}_h$

Output: A RLWE ciphertext $\boldsymbol{c}' \in \mathsf{RLWE}_z(\Delta' \cdot f(\alpha))$.

$\boldsymbol{c}' \leftarrow \boldsymbol{c} \cdot \phi(q) \cdot \sum_{\beta \in \mathbb{Z}_q} f(\beta) \cdot \zeta_q^{-\beta}$
for $w \mid q$ and $w \neq 1, q$ **do**
  $\boldsymbol{c}_{\mathtt{tmp}} \leftarrow \mathsf{EvalAuto}(\boldsymbol{c}, \zeta_q \mapsto \zeta_q^{q/w - c_{q/w}}, \mathsf{AutKey})$
  if $c_{q/w} = 1$ **then**
    $\boldsymbol{c}_{\mathtt{tmp}} \leftarrow \mathsf{BFV.Mul}(\boldsymbol{c}, \boldsymbol{c}_{\mathtt{tmp}}, \mathsf{RelKey})$
  else
    $\boldsymbol{c}'_{\mathtt{tmp}} \leftarrow \mathsf{EvalAuto}(\boldsymbol{c}, \zeta_q \mapsto \zeta_q^{c_{q/w}}, \mathsf{AutKey})$
    $\boldsymbol{c}_{\mathtt{tmp}} \leftarrow \mathsf{BFV.Mul}(\boldsymbol{c}'_{\mathtt{tmp}}, \boldsymbol{c}_{\mathtt{tmp}}, \mathsf{RelKey})$
  end if
  $\boldsymbol{c}_{\mathtt{tmp}} \leftarrow \boldsymbol{c}_{\mathtt{tmp}} \cdot \phi(w) \cdot \left(\sum_{\beta \in \mathbb{Z}_q} \left(f(\beta) \cdot \zeta_w^{-\beta} \right) \right)$
  $\boldsymbol{c}' \leftarrow \boldsymbol{c}' + \boldsymbol{c}_{\mathtt{tmp}}$
end for
$\boldsymbol{c}' \leftarrow \boldsymbol{c}' + (\Delta \cdot \sum_{\beta \in \mathbb{Z}_q} f(\beta), 0)^{\top}$
$\boldsymbol{c}' \leftarrow \mathsf{EvalTr}_{\mathbb{Q}(\zeta_q)/\mathbb{Q}}(\boldsymbol{c}', \mathsf{AutKey})$
return $\boldsymbol{c}'$

where $(\star)$ follows from the fact that

$$\begin{aligned} \mathsf{Tr}_{\mathbb{Q}(\zeta_q)/\mathbb{Q}}\left(\zeta_w^{\alpha-\beta}\right) &= \mathsf{Tr}_{\mathbb{Q}(\zeta_w)/\mathbb{Q}}\left(\mathsf{Tr}_{\mathbb{Q}(\zeta_q)/\mathbb{Q}(\zeta_w)}(\zeta_w^{\alpha-\beta})\right) && \text{(by transitivity)} \\ &= \mathsf{Tr}_{\mathbb{Q}(\zeta_w)/\mathbb{Q}}\left(\zeta_w^{\alpha-\beta} \cdot \mathsf{Tr}_{\mathbb{Q}(\zeta_q)/\mathbb{Q}(\zeta_w)}(1)\right) && \text{(by } \mathbb{Q}(\zeta_w)\text{-linearity)} \\ &= (\phi(q)/\phi(w)) \cdot \mathsf{Tr}_{\mathbb{Q}(\zeta_w)/\mathbb{Q}}(\zeta_w^{\alpha-\beta}) \end{aligned}$$

by $\mathsf{Tr}_{\mathbb{Q}(\zeta_q)/\mathbb{Q}(\zeta_w)}(1) = [\mathbb{Q}(\zeta_q) : \mathbb{Q}(\zeta_w)] = \phi(q)/\phi(w)$ and $\mathbb{Q}$-linearity. □

By using this form of the equality test, we are able to design Algorithm 6.1. The following theorem demonstrates its correctness and analysis.

Theorem 6.7 *Algorithm 6.1 is correct, i.e., the input-output behavior satisfies what is described. Let* $r = \prod_{i \in [k]}(r_i + 1)$*, it possesses the following properties:*

- *Complexity: it requires* $O(\log \phi(q) + r)$ *times the cost of the external product.*
- *Error growth: if* $\|\sigma(\mathsf{Err}(\boldsymbol{c}))\|_{\infty}$*,* $\|\sigma(\mathsf{Err}(\mathsf{AutKey}))\|_{\infty}$ *and* $\|\sigma(\mathsf{Err}(\mathsf{RelKey}))\|_{\infty}$ *are upper bounded by subgaussian variables with parameters* δ_c*,* $\delta_{\mathtt{aut}}$ *and* δ_r*,*

respectively. Let $\delta_{\mathtt{aut}}, \delta_r, \delta_z = O(\sqrt{N})$ *and assume* $\delta_c < \Delta$*, then the output error* $\|\sigma(\mathsf{Err}(\boldsymbol{c}'))\|_\infty$ *is upper bounded by a subgaussian variable with parameter* $O(rh^2Nq^4\sqrt{\delta_c^2 + N^2 \log Q})$.

Proof. We start with a step-by-step analysis of the loop from *line* 2 to *line* 12.

- In *line* 3, we have $\boldsymbol{c}_{\mathtt{tmp}} \in \mathsf{RLWE}_z\left(\Delta \cdot \zeta_q^{(q/w - c_{q/w})\cdot\alpha}\right)$ by Lemma 3.2.
- In *line* 5, if $c_{q/w} = 1$, we have $\boldsymbol{c}_{\mathtt{tmp}} \in \mathsf{RLWE}_z\left(\Delta \cdot \zeta_q^{(q/w)\cdot\alpha}\right) \subset \mathsf{RLWE}_z\left(\Delta \cdot \zeta_w^{\alpha}\right)$ by Lemma 3.1.
- In *line* 6–9, if $c_{q/w} \neq 1$, then $\boldsymbol{c}_{\mathtt{tmp}} \in \mathsf{RLWE}_z\left(\Delta \cdot \zeta_q^{(q/w)\cdot\alpha}\right) \subset \mathsf{RLWE}_z\left(\Delta \cdot \zeta_w^{\alpha}\right)$ by Lemma 3.1 and 3.2.
- In *line* 10, we have $\boldsymbol{c}_{\mathtt{tmp}} \in \mathsf{RLWE}_z\left(\Delta \cdot \phi(w) \cdot \left(\sum_{\beta\in\mathbb{Z}_q}\left(f(\beta)\cdot\zeta_w^{\alpha-\beta}\right)\right)\right)$.

Hence, after *line* 14, we have $\boldsymbol{c}'$ encrypts

$$
\begin{aligned}
&\mathsf{Tr}_{\mathbb{Q}(\zeta_q)/\mathbb{Q}}\left(\Delta \cdot \sum_{\beta\in\mathbb{Z}_q} f(\beta) + \sum_{w|q, w\neq 1} \Delta\cdot\phi(w)\cdot\left(\sum_{\beta\in\mathbb{Z}_q}\left(f(\beta)\cdot\zeta_w^{\alpha-\beta}\right)\right)\right)\\
&= \Delta \cdot \sum_{\beta\in\mathbb{Z}_q} f(\beta)\cdot \mathsf{Tr}_{\mathbb{Q}(\zeta_q)/\mathbb{Q}}\left(1 + \sum_{w|q,w\neq 1}\phi(w)\cdot\zeta_w^{\alpha-\beta}\right) && \text{(by } \mathbb{Q}\text{-linearity)}\\
&= \Delta \cdot \sum_{\beta\in\mathbb{Z}_q} f(\beta)\cdot \begin{cases}\phi(q)\cdot q & \text{if } [\alpha-\beta]_q = 0\\ 0 & \text{otherwise}\end{cases} && \text{(by Lemma 6.6)}\\
&= \Delta\cdot f(\alpha)\cdot\phi(q)\cdot q && \text{(only } f(\alpha) \text{ is multiplied by } \phi(q)\cdot q\text{, others are multiplied by 0)}\\
&= \Delta'\cdot f(\alpha), && \text{(by the definition of } \Delta' := \Delta\cdot q\cdot\phi(q))
\end{aligned}
$$

which proves the correctness. Since the errors can be upper bounded by subgaussian variables, we summarize the corresponding subgaussian parameter in each line as follows.

- $\boldsymbol{c}'$ in *line* 1: $O(h\delta_c\phi(q)\sqrt{q})$ by the fact that $\|\sigma(\sum_{\beta\in\mathbb{Z}_q} f(\beta)\cdot\zeta_q^{-\beta})\|_\infty \le h\sqrt{q}$.
- $\boldsymbol{c}_{\mathtt{tmp}}$ in *line* 3: $O(\sqrt{\delta_c^2 + N^2\log Q})$ by Lemma 3.2.
- $\boldsymbol{c}_{\mathtt{tmp}}$ in *line* 4–9: $O(hNq^2\sqrt{\delta_c^2 + N^2\log Q})$ by Lemma 3.1 and $\delta_c < \Delta$ (we use the worst case that $c_{q/w} \neq 1$).
- $\boldsymbol{c}_{\mathtt{tmp}}$ in *line* 10: $O(h^2Nq^{3.5}\sqrt{\delta_c^2 + N^2\log Q})$ by plaintext-ciphertext multiplication and the fact that $\left\|\sigma\left(\phi(w)\cdot\left(\sum_{\beta\in\mathbb{Z}_q}\left(f(\beta)\cdot\zeta_w^{-\beta}\right)\right)\right)\right\|_\infty = O(hq^{1.5})$, which is obtained by modeling $f(\beta)$ as a uniformly random variable in $\mathbb{Z}_h$.
- $\boldsymbol{c}'$ in *line* 13: $O(rh^2Nq^{3.5}\sqrt{\delta_c^2 + N^2\log Q})$ by the fact that we have $r-2$ loops from *line* 2 to *line* 12.
- $\boldsymbol{c}'$ in *line* 14: $O(rh^2Nq^4\sqrt{\delta_c^2 + N^2\log Q})$ by Theorem 3.6.

This proves the claim of error growth. For the computational complexity, we have $r-2$ for the loop from *line* 2 to 12, and each loop contains at most two automorphism evaluations and one BFV multiplication, which implies $O(r)$ times the cost of the external product by Proposition 3.7. Finally, the final trace evaluation requires $O(\log \phi(q))$ times the cost of the external product by Proposition 3.7, which completes the proof of our claim of computational complexity. □

Remark 6.8 *In the error analysis of Algorithm 6.1, we use rather loose estimates, i.e., $\phi(w) < q$ for all $w \mid q, w \neq 1$. These terms may be much smaller than q in the concrete parameter settings. Moreover, there is a term r in the upper bound of the overall error norm. As r is generally sub-linear in q [55, 67], we note that the overall term is still polynomially bounded (in the security parameter).*

Asymptotic Setting. In the asymptotic setting, we can set $r = O(\log \phi(q))$ or fix r to some small constant. Then we have that the overall computational complexity of $\mathsf{EvalFunc}$ is $O(\log \phi(q))$ as we desired in our blueprint in Sect. 4.1.

Parallel Computation. Our algorithm $\mathsf{EvalFunc}$ is friendly to the parallel computation architecture, as the computation in each for loop (from *line* 2 to *line* 12) does not have a dependency on the others. Additionally, the homomorphic trace is also parallel friendly, as pointed out by [38]. Thus, the overall throughput can be easily improved on parallel-friendly (e.g., multi-core) platforms.

7 Conclusions and Future Work

In this work, we show for the first time that functional bootstrapping for general functions within a polynomial modulus can work over a wide range of general cyclotomic rings and *simultaneously* satisfy the following two properties:

- Supporting arbitrary correctly decryptable input LWE ciphertexts.
- Essentially as efficient as regular bootstrapping.

Based on the advantages and limitations of our current techniques, we also notice two interesting directions that warrant further investigation:

- **Theory:** As mentioned in the solution to Challenge 2 of Sect. 6, our method cannot handle cyclotomic rings with even composite indices, namely when the cyclotomic index is composite and contains the prime factor 2. Overcoming this limitation to support arbitrary cyclotomic rings while maintaining functionality and efficiency presents a significant challenge.
- **Practice:** Note that our algorithms support cyclotomic rings with indices that are powers of small primes (e.g., powers-of-2), which have been shown to be concretely efficient due to their full compatibility with the standard (or slightly twisted) Fast Fourier Transform (FFT) [24,30,73] and Number Theoretic Transform (NTT) [3,20]. It is also promising for cyclotomic rings with composite indices by combining the Batch framework of [44,45] with insights from [51,56]. We leave it as an interesting follow-up work to determine the concrete efficiency of our algorithms over different cyclotomic rings.

Acknowledgements. We would like to thank the anonymous reviewers of TCC 2024 for their insightful advices that help improve the presentation. Feng-Hao Liu is supported by NSF CNS-2402031. Han Xia and Han Wang are supported by the National Key R&D Program of China under Grant 2020YFA0712303, the Strategic Priority Research Program of the Chinese Academy of Sciences under Grant XDB0690200, and State Key Laboratory of Information Security under Grant TC20221013042.

References

1. Abla, P., Liu, F.H., Wang, H., Wang, Z.: Ring-based identity based encryption - asymptotically shorter MPK and tighter security. In: Nissim, K., Waters, B. (eds.) TCC 2021, Part III. LNCS, vol. 13044, pp. 157–187. Springer, Cham (Nov 2021). https://doi.org/10.1007/978-3-030-90456-2_6
2. Agrawal, S., Lin, D. (eds.): ASIACRYPT 2022, Part II, LNCS, vol. 13792. Springer, Cham (Dec (2022)
3. Al Badawi, A., et al.: OpenFHE: Open-source fully homomorphic encryption library. In: Proceedings of the 10th Workshop on Encrypted Computing & Applied Homomorphic Cryptography, pp. 53–63. WAHC'22, Association for Computing Machinery, New York, NY, USA (2022). https://doi.org/10.1145/3560827.3563379
4. Alperin-Sheriff, J., Peikert, C.: Practical bootstrapping in quasilinear time. In: Canetti and Garay [17], pp. 1–20. https://doi.org/10.1007/978-3-642-40041-4_1
5. Alperin-Sheriff, J., Peikert, C.: Faster bootstrapping with polynomial error. In: Garay and Gennaro [32], pp. 297–314. https://doi.org/10.1007/978-3-662-44371-2_17
6. Barrington, D.A.M.: Bounded-width polynomial-size branching programs recognize exactly those languages in NC^1. In: 18th ACM STOC, pp. 1–5. ACM Press (May 1986). https://doi.org/10.1145/12130.12131
7. Bergerat, L., et al.: Parameter optimization and larger precision for (T)FHE. J. Cryptol. **36**(3), 28 (2023). https://doi.org/10.1007/s00145-023-09463-5
8. Biasse, J.F., Ruiz, L.: FHEW with efficient multibit bootstrapping. In: Lauter, K.E., Rodríguez-Henríquez, F. (eds.) LATINCRYPT 2015. LNCS, vol. 9230, pp. 119–135. Springer, Cham (Aug 2015). https://doi.org/10.1007/978-3-319-22174-8_7
9. Bonte, C., Iliashenko, I., Park, J., Pereira, H.V.L., Smart, N.P.: FINAL: Faster FHE instantiated with NTRU and LWE. In: Agrawal and Lin [2], pp. 188–215. https://doi.org/10.1007/978-3-031-22966-4_7
10. Boura, C., Gama, N., Georgieva, M., Jetchev, D.: Simulating homomorphic evaluation of deep learning predictions. In: International Symposium on Cyber Security Cryptography and Machine Learning, pp. 212–230. Springer (2019). https://doi.org/10.1007/978-3-030-20951-3_20
11. Bourse, F., Sanders, O., Traoré, J.: Improved secure integer comparison via homomorphic encryption. In: Jarecki, S. (ed.) CT-RSA 2020. LNCS, vol. 12006, pp. 391–416. Springer, Cham (Feb 2020). https://doi.org/10.1007/978-3-030-40186-3_17
12. Brakerski, Z.: Fully homomorphic encryption without modulus switching from classical GapSVP. In: Safavi-Naini and Canetti [64], pp. 868–886. https://doi.org/10.1007/978-3-642-32009-5_50
13. Brakerski, Z., Gentry, C., Vaikuntanathan, V.: (Leveled) fully homomorphic encryption without bootstrapping. In: Goldwasser, S. (ed.) ITCS 2012, pp. 309–325. ACM (Jan 2012). https://doi.org/10.1145/2090236.2090262

14. Brakerski, Z., Langlois, A., Peikert, C., Regev, O., Stehlé, D.: Classical hardness of learning with errors. In: Boneh, D., Roughgarden, T., Feigenbaum, J. (eds.) 45th ACM STOC, pp. 575–584. ACM Press (Jun 2013). https://doi.org/10.1145/2488608.2488680
15. Brakerski, Z., Vaikuntanathan, V.: Efficient fully homomorphic encryption from (standard) LWE. In: Ostrovsky, R. (ed.) 52nd FOCS, pp. 97–106. IEEE Computer Society Press (Oct 2011). https://doi.org/10.1109/FOCS.2011.12
16. Brakerski, Z., Vaikuntanathan, V.: Lattice-based FHE as secure as PKE. In: Naor, M. (ed.) ITCS 2014, pp. 1–12. ACM (Jan 2014). https://doi.org/10.1145/2554797.2554799
17. Canetti, R., Garay, J.A. (eds.): CRYPTO 2013, Part I, LNCS, vol. 8042. Springer, Berlin, Heidelberg (Aug (2013)
18. Castryck, W., Iliashenko, I., Vercauteren, F.: Provably weak instances of ring-lwe revisited. In: Fischlin, M., Coron, J.-S. (eds.) Advances in Cryptology – EUROCRYPT 2016: 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part I, pp. 147–167. Springer Berlin Heidelberg, Berlin, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49890-3_6
19. Chen, H., Dai, W., Kim, M., Song, Y.: Efficient homomorphic conversion between (Ring) LWE Ciphertexts. In: Sako, K., Tippenhauer, N.O. (eds.) Applied Cryptography and Network Security: 19th International Conference, ACNS 2021, Kamakura, Japan, June 21–24, 2021, Proceedings, Part I, pp. 460–479. Springer International Publishing, Cham (2021). https://doi.org/10.1007/978-3-030-78372-3_18
20. Chen, H., Laine, K., Player, R.: Simple encrypted arithmetic library - SEAL v2.1. In: Brenner, M., Rohloff, K., Bonneau, J., Miller, A., Ryan, P.Y.A., Teague, V., Bracciali, A., Sala, M., Pintore, F., Jakobsson, M. (eds.) FC 2017 Workshops. LNCS, vol. 10323, pp. 3–18. Springer, Cham (Apr 2017). https://doi.org/10.1007/978-3-319-70278-0_1
21. Cheon, J.H., Kim, A., Kim, M., Song, Y.S.: Homomorphic encryption for arithmetic of approximate numbers. In: Takagi, T., Peyrin, T. (eds.) ASIACRYPT 2017, Part I. LNCS, vol. 10624, pp. 409–437. Springer, Cham (Dec 2017). https://doi.org/10.1007/978-3-319-70694-8_15
22. Chillotti, I., Gama, N., Georgieva, M., Izabachène, M.: Faster Fully Homomorphic Encryption: Bootstrapping in Less Than 0.1 seconds. In: Cheon, J.H., Takagi, T. (eds.) Advances in Cryptology – ASIACRYPT 2016: 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part I, pp. 3–33. Springer Berlin Heidelberg, Berlin, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53887-6_1
23. Chillotti, I., Gama, N., Georgieva, M., Izabachène, M.: TFHE: fast fully homomorphic encryption over the torus. J. Cryptol. **33**(1), 34–91 (2019). https://doi.org/10.1007/s00145-019-09319-x
24. Chillotti, I., Gama, N., Georgieva, M., Izabachène, M.: TFHE: Fast fully homomorphic encryption library. GitHub (2023). https://github.com/tfhe/tfhe
25. Chillotti, I., Joye, M., Paillier, P.: Programmable bootstrapping enables efficient homomorphic inference of deep neural networks. In: Cyber Security Cryptography and Machine Learning: 5th International Symposium, CSCML 2021, Be'er Sheva, Israel, July 8–9, 2021, Proceedings 5, pp. 1–19. Springer (2021). https://doi.org/10.1007/978-3-030-78086-9_1

26. Chillotti, I., Ligier, D., Orfila, J.B., Tap, S.: Improved programmable bootstrapping with larger precision and efficient arithmetic circuits for TFHE. In: Tibouchi, M., Wang, H. (eds.) ASIACRYPT 2021, Part III. LNCS, vol. 13092, pp. 670–699. Springer, Cham (Dec 2021). https://doi.org/10.1007/978-3-030-92078-4_23
27. Clet, P.E., Zuber, M., Boudguiga, A., Sirdey, R., Gouy-Pailler, C.: Putting up the swiss army knife of homomorphic calculations by means of TFHE functional bootstrapping. Cryptology ePrint Archive, Report 2022/149 (2022). https://eprint.iacr.org/2022/149
28. Cong, K., Das, D., Park, J., Pereira, H.V.L.: SortingHat: Efficient private decision tree evaluation via homomorphic encryption and transciphering. In: Yin, H., Stavrou, A., Cremers, C., Shi, E. (eds.) ACM CCS 2022, pp. 563–577. ACM Press (Nov 2022). https://doi.org/10.1145/3548606.3560702
29. Crockett, E., Peikert, C.: Challenges for ring-LWE. Cryptology ePrint Archive, Report 2016/782 (2016). https://eprint.iacr.org/2016/782
30. Ducas, L., Micciancio, D.: FHEW: bootstrapping homomorphic encryption in less than a second. In: Oswald and Fischlin [58], pp. 617–640. https://doi.org/10.1007/978-3-662-46800-5_24
31. Fan, J., Vercauteren, F.: Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive, Report 2012/144 (2012). https://eprint.iacr.org/2012/144
32. Garay, J.A., Gennaro, R. (eds.): CRYPTO 2014, Part I, LNCS, vol. 8616. Springer, Berlin, Heidelberg (Aug (2014)
33. Gentry, C.: Fully homomorphic encryption using ideal lattices. In: Mitzenmacher [54], pp. 169–178. https://doi.org/10.1145/1536414.1536440
34. Gentry, C., Halevi, S., Smart, N.P.: Homomorphic evaluation of the AES circuit. In: Safavi-Naini and Canetti [64], pp. 850–867. https://doi.org/10.1007/978-3-642-32009-5_49
35. Gentry, C., Sahai, A., Waters, B.: Homomorphic encryption from learning with errors: conceptually-simpler, asymptotically-faster, attribute-based. In: Canetti and Garay [17], pp. 75–92. https://doi.org/10.1007/978-3-642-40041-4_5
36. Guimarães, A., Borin, E., Aranha, D.F.: Revisiting the functional bootstrap in TFHE. IACR TCHES **2021**(2), 229–253 (2021). https://doi.org/10.46586/tches.v2021.i2.229-253, https://tches.iacr.org/index.php/TCHES/article/view/8793
37. Halevi, S., Shoup, V.: Algorithms in HElib. In: Garay and Gennaro [32], pp. 554–571. https://doi.org/10.1007/978-3-662-44371-2_31
38. Halevi, S., Shoup, V.: Bootstrapping for HElib. In: Oswald and Fischlin [58], pp. 641–670. https://doi.org/10.1007/978-3-662-46800-5_25
39. Halevi, S., Shoup, V.: Design and implementation of HElib: a homomorphic encryption library. Cryptology ePrint Archive, Report 2020/1481 (2020). https://eprint.iacr.org/2020/1481
40. Hazay, C., Stam, M. (eds.): EUROCRYPT 2023, Part III, LNCS, vol. 14006. Springer, Cham (Apr (2023)
41. Joye, M., Walter, M.: Liberating TFHE: programmable bootstrapping with general quotient polynomials. In: Proceedings of the 10th Workshop on Encrypted Computing & Applied Homomorphic Cryptography, pp. 1–11 (2022). https://doi.org/10.1145/3560827.3563376
42. Kluczniak, K., Schild, L.: FDFB: Full domain functional bootstrapping towards practical fully homomorphic encryption. IACR TCHES **2023**(1), 501–537 (2023). https://doi.org/10.46586/tches.v2023.i1.501-537
43. Lee, Y., et al.: Efficient FHEW bootstrapping with small evaluation keys, and applications to threshold homomorphic encryption. In: Hazay and Stam [40], pp. 227–256. https://doi.org/10.1007/978-3-031-30620-4_8

44. Liu, F.H., Wang, H.: Batch bootstrapping I: a new framework for SIMD bootstrapping in polynomial modulus. In: Hazay and Stam [40], pp. 321–352. https://doi.org/10.1007/978-3-031-30620-4_11
45. Liu, F.H., Wang, H.: Batch bootstrapping II: bootstrapping in polynomial modulus only requires $\tilde{O}(1)$ FHE multiplications in amortization. In: Hazay and Stam [40], pp. 353–384. https://doi.org/10.1007/978-3-031-30620-4_12
46. Katsikas, S., Abie, H., Ranise, S., Verderame, L., Cambiaso, E., Ugarelli, R., Praça, I., Li, W., Meng, W., Furnell, S., Katt, B., Pirbhulal, S., Shukla, A., Ianni, M., Dalla Preda, M., Choo, K.-K.R., Pupo Correia, M., Abhishta, A., Sileno, G., Alishahi, M., Kalutarage, H., Yanai, N. (eds.): Computer Security. ESORICS 2023 International Workshops: CPS4CIP, ADIoT, SecAssure, WASP, TAURIN, PriST-AI, and SECAI, The Hague, The Netherlands, September 25–29, 2023, Revised Selected Papers, Part II. Springer Nature Switzerland, Cham (2024)
47. Liu, Z., Micciancio, D., Polyakov, Y.: Large-precision homomorphic sign evaluation using FHEW/TFHE bootstrapping. In: Agrawal and Lin [2], pp. 130–160. https://doi.org/10.1007/978-3-031-22966-4_5
48. Liu, Z., Wang, Y.: Amortized functional bootstrapping in less than 7 ms, with $\tilde{\mathrm{O}}(1)$ polynomial multiplications. In: Guo, J., Steinfeld, R. (eds.) ASIACRYPT 2023, Part VI. LNCS, vol. 14443, pp. 101–132. Springer, Singapore (Dec 2023). https://doi.org/10.1007/978-981-99-8736-8_4
49. Lu, W.J., Huang, Z., Hong, C., Ma, Y., Qu, H.: PEGASUS: Bridging polynomial and non-polynomial evaluations in homomorphic encryption. In: 2021 IEEE Symposium on Security and Privacy. pp. 1057–1073. IEEE Computer Society Press (May 2021). https://doi.org/10.1109/SP40001.2021.00043
50. Lyubashevsky, V., Peikert, C., Regev, O.: On ideal lattices and learning with errors over rings. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 1–23. Springer, Berlin, Heidelberg (May / Jun 2010). https://doi.org/10.1007/978-3-642-13190-5_1
51. Lyubashevsky, V., Peikert, C., Regev, O.: A toolkit for ring-LWE cryptography. In: Johansson, T., Nguyen, P.Q. (eds.) Advances in Cryptology – EUROCRYPT 2013: 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings, pp. 35–54. Springer Berlin Heidelberg, Berlin, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38348-9_3
52. Ma, S., Huang, T., Wang, A., Zhou, Q., Wang, X.: Fast and accurate: efficient full-domain functional bootstrap and digit decomposition for homomorphic computation. IACR TCHES **2024**(1), 592–616 (2024). https://doi.org/10.46586/tches.v2024.i1.592-616
53. Micciancio, D., Polyakov, Y.: Bootstrapping in FHEW-like cryptosystems. In: WAHC '21: Proceedings of the 9th on Workshop on Encrypted Computing & Applied Homomorphic Cryptography, Virtual Event, Korea, 15 November 2021, pp. 17–28. WAHC@ACM (2021). https://doi.org/10.1145/3474366.3486924
54. Mitzenmacher, M. (ed.): 41st ACM STOC. ACM Press (May / Jun 2009)
55. from MO (https://mathoverflow.net/users/11919/gh-from mo), G.: Upper bound for product of exponents of prime factorization. MathOverflow. https://mathoverflow.net/q/256452
56. Open Source: HElib. GitHub. https://github.com/shaih/HElib
57. Open Source: Palisade lattice cryptography library. GitLab. https://gitlab.com/palisade
58. Oswald, E., Fischlin, M. (eds.): EUROCRYPT 2015, Part I, LNCS, vol. 9056. Springer, Berlin, Heidelberg (Apr (2015)

59. Peikert, C.: Public-key cryptosystems from the worst-case shortest vector problem: extended abstract. In: Mitzenmacher [54], pp. 333–342. https://doi.org/10.1145/1536414.1536461
60. Peikert, C.: How (not) to instantiate ring-LWE. In: Zikas, V., De Prisco, R. (eds.) SCN 16. LNCS, vol. 9841, pp. 411–430. Springer, Cham (Aug / Sep 2016). https://doi.org/10.1007/978-3-319-44618-9_22
61. Peikert, C., Pepin, Z.: Algebraically structured LWE, revisited. In: Hofheinz, D., Rosen, A. (eds.) TCC 2019, Part I. LNCS, vol. 11891, pp. 1–23. Springer, Cham (Dec 2019). https://doi.org/10.1007/978-3-030-36030-6_1
62. Peikert, C., Regev, O., Stephens-Davidowitz, N.: Pseudorandomness of ring-LWE for any ring and modulus. In: Hatami, H., McKenzie, P., King, V. (eds.) 49th ACM STOC, pp. 461–473. ACM Press (Jun 2017). https://doi.org/10.1145/3055399.3055489
63. Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. In: Gabow, H.N., Fagin, R. (eds.) 37th ACM STOC, pp. 84–93. ACM Press (May 2005). https://doi.org/10.1145/1060590.1060603
64. Safavi-Naini, R., Canetti, R. (eds.): CRYPTO 2012, LNCS, vol. 7417. Springer, Berlin, Heidelberg (Aug (2012)
65. Smart, N.P., Vercauteren, F.: Fully homomorphic SIMD operations. Des. Codes Crypt. **71**(1), 57–81 (2012). https://doi.org/10.1007/s10623-012-9720-4
66. Stehlé, D., Steinfeld, R., Tanaka, K., Xagawa, K.: Efficient public key encryption based on ideal lattices. In: Matsui, M. (ed.) ASIACRYPT 2009. LNCS, vol. 5912, pp. 617–635. Springer, Berlin, Heidelberg (Dec 2009). https://doi.org/10.1007/978-3-642-10366-7_36
67. Tenenbaum, G.: Introduction to analytic and probabilistic number theory, vol. 163. American Mathematical Soc. (2015)
68. The Sage Developers: Sagemath, the Sage Mathematics Software System (Version 10.2) (2023). https://www.sagemath.org
69. Vinogradov, I.M.: Chapter VI: Primitive roots and indices. In: Elements of number theory. pp. 105–121. Dover Publications (2003). https://books.google.com/books?id=xlIfdGPM9t4C&pg=PA105
70. Wang, R., et al.: Circuit bootstrapping: Faster and smaller. In: Joye, M., Leander, G. (eds.) EUROCRYPT 2024, Part II. LNCS, vol. 14652, pp. 342–372. Springer, Cham (May 2024). https://doi.org/10.1007/978-3-031-58723-8_12
71. Xiang, B., Zhang, J., Deng, Y., Dai, Y., Feng, D.: Fast blind rotation for bootstrapping FHEs. In: Handschuh, H., Lysyanskaya, A. (eds.) CRYPTO 2023, Part IV. LNCS, vol. 14084, pp. 3–36. Springer, Cham (Aug 2023). https://doi.org/10.1007/978-3-031-38551-3_1
72. Yang, Z., Xie, X., Shen, H., Chen, S., Zhou, J.: TOTA: Fully homomorphic encryption with smaller parameters and stronger security. Cryptology ePrint Archive, Report 2021/1347 (2021). https://eprint.iacr.org/2021/1347
73. Zama: TFHE-rs: A Pure Rust Implementation of the TFHE Scheme for Boolean and Integer Arithmetics Over Encrypted Data (2022). https://github.com/zama-ai/tfhe-rs
74. Zheng, X., Li, H., Wang, D.: A new framework for fast homomorphic matrix multiplication. Cryptology ePrint Archive, Paper 2023/1649 (2023). https://eprint.iacr.org/2023/1649

Multi-party Computation

A Note on Low-Communication Secure Multiparty Computation via Circuit Depth-Reduction

Pierre Charbit[1(✉)], Geoffroy Couteau[1], Pierre Meyer[2], and Reza Naserasr[1]

[1] Université Paris Cité, CNRS, IRIF, Paris, France
{charbit,couteau,reza}@irif.fr
[2] Aarhus University, Aarhus, Denmark
pierre.meyer@cs.au.dk

Abstract. We consider the graph-theoretic problem of removing (few) nodes from a directed acyclic graph in order to reduce its depth. While this problem is intractable in the general case, we provide a variety of algorithms in the case where the graph is that of a circuit of fan-in (at most) two, and explore applications of these algorithms to secure multiparty computation with low communication. Over the past few years, a paradigm for low-communication secure multiparty computation has found success based on decomposing a circuit into low-depth "chunks". This approach was however previously limited to circuits with a "layered" structure. Our graph-theoretic approach extends this paradigm to *all* circuits. In particular, we obtain the following contributions:

- **Fractionally linear-communication MPC in the correlated randomness model.** We provide an N-party protocol for computing an n-input, m-output $\mathbb{F}$-arithmetic circuit with s internal gates (over any basis of binary gates) with communication complexity $(\frac{2}{3}s+n+m)\cdot N\cdot\log|\mathbb{F}|$, which can be improved to $((1+\epsilon)\cdot\frac{2}{5}s+n+m)\cdot N\cdot\log|\mathbb{F}|$ (at the cost of increasing the computational overhead from a small constant factor to a large one). Previously, comparable protocols either used more than $s\cdot N\cdot\log|\mathbb{F}|$ bits of communication, required super-polynomial computation, were restricted to layered circuits, or tolerated a sub-optimal corruption threshold.
- **Sublinear-Communication MPC.** Assuming the existence of N-party Homomorphic Secret Sharing for logarithmic depth circuits (respectively doubly logarithmic depth circuits), we show there exists sublinear-communication secure N-party computation for *all* $\log^{1+o(1)}$-depth (resp. $(\log\log)^{1+o(1)}$-depth) circuits. Previously, this result was limited to $(\mathcal{O}(\log))$-depth (resp. $(\mathcal{O}(\log\log))$-depth) circuits, or to circuits with a specific structure (*e.g.* layered).
- **The $\binom{N}{1}$-OT complexity of MPC.** We introduce the "$\binom{N}{1}$-*OT complexity of MPC*" of a function f, denoted $C_N(f)$, as the number of oracle calls required to securely compute f in the $\binom{N}{1}$-OT hybrid

P. Meyer—Most of this work was done while P. Meyer was a PhD student jointly at Reichman University, Herzliya, ISRAEL and Université Paris Cité, CNRS, IRIF, Paris, FRANCE.

E. Boyle and M. Mahmoody (Eds.): TCC 2024, LNCS 15367, pp. 167–199, 2025.
https://doi.org/10.1007/978-3-031-78023-3_6

model. We establish the following upper bound: for every $N \geq 2$, $C_N(f) \leq (1 + g(N)) \cdot \frac{2|f|}{5}$, where $g(N)$ is an explicit vanishing function.

We also obtain additional contributions to reducing the amount of bootstrapping for fully homomorphic encryption, and to other types of sublinear-communication MPC protocols such as those based on correlated symmetric private information retrieval.

1 Introduction

Secure multiparty computation (MPC) [Yao86, GMW87] enables mutually distrusting parties to jointly compute a function on their private inputs, while revealing nothing beyond the function's output. The seminal protocols of the 1980s [Yao86, GMW87, BGW88, CCD88] all require an amount of communication which scales *linearly* with the computational complexity of the function: given a fan-in two circuit representation of the function, the parties are required to communicate for every (non-linear) gate of the circuit. In this paper, we propose to achieve MPC with less communication than the circuit-size by limiting interaction to only a few key gates, which we identify using a graph-theoretic algorithm. Specifically, the goal is to find a small set of nodes in the underlying graph whose removal yields a low-depth directed acyclic graph (DAG). The idea is to reduce the secure computation of an arbitrary circuit to that of a circuit of logarithmic, doubly-logarithmic, or even constant depth for which, generally speaking, there are known protocols using circuit-independent communication.

This problem is motivated by the fact that, aside from protocols based on fully homomorphic encryption, all known ways to achieve sublinear-communication secure computation are restricted to low-depth circuits, and can only currently be extended to deeper circuit if they have a specific "layered" structure (meaning the circuits' gates are partitioned into ordered layers, and each wire only connects one layer to the next).

1.1 Background

This paper proposes to solve the cryptographic problem of securely computing *any* circuit with low communication by reducing it to a graph-theoretic problem. We recall in Sect. 1.1.1 what is known about low-communication secure "general-purpose" computation (including protocols restricted to layered circuits or low-depth circuits, but excluding special-purpose protocols such as *Private Information Retrieval*), and in Sect. 1.1.2 the literature on the graph-theoretic problem to which we reduce the secure computation task.

1.1.1 Sublinear-Communication Secure Multiparty Computation. We start by surveying sublinear-communication secure computation protocols,

but restrict ourselves to those using a polynomial amount of computation[1] and tolerating the optimal number of corruptions. Because the amount of computation typically grows exponentially (or even doubly exponentially) with the depth of the computation, most of these protocols only support computation up to a certain depth. Throughout this section, s denotes the size of the circuit, d denotes its depth, N denotes the number of parties, and $\mathbb{F}$ denotes the field over which the circuit's arithmetic is performed.

In the Correlated Randomness Model, $d = \log\log s$. Ishai, Kushilevitz, Meldgaard, Orlandi, and Paskin-Cherniavsky [IKM+13] showed how to securely compute any function with a polynomial-size look-up table using circuit-independent communication (only proportional to input and output size). Couteau [Cou19] extended this approach to any $\log\log$-depth (boolean or arithmetic) circuit. In turn, this yields a protocol for securely computing any size-s *layered* circuit using $\mathcal{O}(s/\log\log s \cdot N \cdot \log|\mathbb{F}|)$ bits of communication.

From Fully Homomorphic Encryption (FHE), any d. Gentry [Gen09] provided the first construction of *fully homomorphic encryption* from ideal lattices. This powerful primitive, which was later instantiated under a standard variant of the learning with errors assumption (LWE) [BV11,BGV12], allows for a computation to be performed on encrypted data. FHE was later shown to yield asymptotically optimal-communication secure multiparty computation (in the computational setting) [DFH12,AJL+12].

From Homomorphic Secret Sharing (HSS), $d = \mathcal{O}(\log s)$. Boyle, Gilboa, and Ishai [BGI16a] built two-party *homomorphic secret-sharing* (HSS) for branching programs from the decisional Diffie-Hellman assumption (DDH). This yields two-party protocols for securely computing $\mathcal{O}(\log s)$-depth circuits using circuit-independent communication, or arbitrarily deep layered boolean circuits with communication $\mathcal{O}(s/\log s \cdot N \cdot \log|\mathbb{F}|)$. This template for HSS for branching programs was later instantiated under the decision composite residuosity assumption (DCR) [FGJS17,OSY21,RS21], learning with errors (LWE) with a polynomial noise-to-modulus ratio [BKS19], and assumptions based on class groups of imaginary quadratic fields [ADOS22]. We note that some of the above HSS schemes [BGI16a,FGJS17,BKS19] have a noticeable error in correctness, but this has no bearing on the application to low-communication two-party com-

[1] If we lift this restriction, Beaver, Feigenbaum, Kilian, and Rogoway [BFKR91] showed that any function from $\{0,1\}^n$ to $\{0,1\}$ can be securely computed in the presence of up to t corruptions by $N = \mathcal{O}(t \cdot n/\log n)$ computationally unbounded parties (which the protocol indeed requires to use an exponential amount of computation) using $\mathsf{poly}(n+N)$ communication. In addition, the protocol of [IKM+13], which we state as a result for securely computing functions with a polynomial-size look-up table, can be applied to general functions if the parties are computationally unbounded (and have access to a doubly exponential amount of correlated randomness).

putation. The DDH-based approach only supports boolean circuits, while the LWE- and DCR-based approaches work in any field.[2]

From Homomorphic Secret Sharing (HSS), $d = \log\log s - \log\log\log s$. There exists HSS supporting circuits of depth $(\log\log s - \log\log\log s)$:[3] in the two-party setting from the quasi-polynomial learning parity with noise assumption (LPN) [CM21], in the four-party setting assuming DCR and constant-depth pseudorandom generators (PRG) [COS+22, BCM23], and in the general multiparty setting assuming sparse LPN [DIJL23]. In turn, this yields 2-, 4-, or N-party computation of $(\log\log s - \log\log\log s)$-depth circuits with circuit-independent communication, or that of arbitrarily deep layered circuits with communication $\mathcal{O}(s/\log\log s \cdot N \cdot \log|\mathbb{F}|)$. All these approaches support arbitrary fields $\mathbb{F}$. The scheme based on super-polynomial LPN [CM21] is a "single-circuit" scheme (it requires the circuit's topology to be known at input-sharing time) and the scheme based on sparse LPN [DIJL23] has a noticeable error in correctness, but here again these limitations have no bearing on the application to low-communication secure multiparty computation.

Other Approaches, $d = \log\log s - \log\log\log s$. Boyle, Couteau, and Meyer [BCM22] introduced *correlated symmetric private information retrieval* (correlated SPIR), and instantiated it under LPN plus any of the following assumptions: DDH, the quadratic residuosity assumption (QR), DCR, or poly-modulus LWE. This primitive yields secure two-party computation of doubly logarithmic-depth[4] boolean circuits using $\mathcal{O}(n+m+\sqrt{s}\cdot\mathsf{poly}(\lambda)\cdot(n+m)^{2/3})$ bits of communication, as well as secure two-party computation of *synchronous*[5] boolean circuits using communication $\mathcal{O}(s/\log\log s + d^{1/3}\cdot s^{2(1+\epsilon)/3})$ (which is $\mathcal{O}(s/\log\log s)$ provided $d = o(s^{1-\epsilon}/\mathsf{poly}(\lambda))$, which is to say the circuit is not too "tall and skinny"). Boyle, Couteau, and Meyer [BCM23] later provided a framework, based on 2- or 4-party HSS and correlated SPIR, to extended these protocols to the 3- or 5-party settings. Finally, Abram, Roy, and Scholl [ARS24] extended this approach to the general multiparty setting by instead only relying on a form of two-party HSS with *succinct share size*, and instantiated it under a DDH-like assumption in class groups, DCR, or LWE with a super-polynomial noise-to-modulus ratio: for general layered boolean circuits with communication $\mathcal{O}(s/\log\log s \cdot N)$, or even $\mathcal{O}(s/\log s \cdot N)$ if the layered boolean circuit is wide enough. All these approaches are restricted to the boolean setting.

1.1.2 DAG Depth-Reduction.

The problem of reducing a DAG's depth by removing nodes can be expressed as a hitting set problem: given a directed

[2] These HSS schemes support bounded-integer computation, which translates to low-communication secure computation over arbitrary fields using hybrid encryption.

[3] Note that this class is somewhat related to logarithmic-depth branching programs.

[4] The complexity we provide here is for when the depth is at most $(\log\log s)/4$, not $(\log\log s - \log\log\log s)$; this is done in order to simplify the expression and absorb some negligible terms.

[5] A *synchronous* circuit is a layered circuit whose input gates are all in the first layer.

graph $G = (V, E)$, we are looking for a set of vertices which "hits" (*i.e.* contains a vertex from) each k-path in G. Ultimately, we will present a family of secure computation protocols whose communication complexity scales with the size of such a hitting set in a well-chosen graph, closely related to a circuit representation of the function to be securely computed. Towards upper bounding the communication complexity of securely computing *any* circuit, we are interested in upper bounding the size of the smallest k-path hitting set for very n-vertex DAG of in-degree at most two.

For every $k \geq 2$, the optimisation version of the directed k-path hitting set problem was shown to be NP-complete for general DAGs by Paindavoine and Vialla [PV16]. For $k = 1$, the problem coincides with the vertex cover problem[6] which was shown to be NP-complete in general DAGs by Naumann [Nau09].

The analogue problem on undirected graphs (which is also NP-complete in its optimisation version) has been studied by Brešar, Kardoš, Katrenič, and Semanišin [BKKS11] and by Jianhua [Tu22]. Since every directed k-path in a DAG corresponds to a k-path in the DAG's underlying (undirected) graph, establishing an upper bound in the undirected case (*i.e.* establishing an upper bound on the smallest k-path hitting set of the worst 2-degenerate graph on n vertices) also applies to the directed case. For $k = 3$, which in the undirected setting corresponds to a bound on the *dissociation number*, applying the result of Brešar, Kardoš, Katrenič, and Semanišin [BKKS11, Corollary 10] yields a bound[7] of $2n/3$. For all $k \geq 2$, [BKKS11, Theorem 4] yields a bound[8] of $(\frac{3}{5} + \frac{2}{5k}) \cdot n$.

Finally, we mention that this problem is related to the *bootstrap(ping) problem* [LP13, PV16, BLMZ17]. Most constructions of fully homomorphic encryption are based on LWE and follow the same blueprint. Each ciphertext is associated with some noise, which grows at each homomorphic operation. Typically, the noise grows additively for linear gates and multiplicatively for non-linear gates. Once the noise reaches a certain limit, decryption is no longer possible (at least not without significant error in correctness). The key technique to manage this noise growth, due to Gentry [Gen09], is called "bootstrapping". By using an encryption of the secret key (which requires circular-security) one can homomorphically run the decryption procedure on the noisy ciphertext in order to produce a fresh ciphertext, encrypting the same value under the same key, but with a reset noise level. This bootstrapping operation is computationally heavy however, which motivates the question to try and reduce the number of times it is required for computing an arbitrary circuit. The *bootstrap problem*, formalised by Benhamouda, Lepoint, Mathieu, and Zhou [BLMZ17], asks to minimise the number of bootstraps required to compute a circuit. Consider a DAG whose vertices all have in-degree exactly 0 or 2 and are coloured in one of three colours:

[6] We note there is some inconsistency in the literature in how the vertex cover problem is extended to directed graphs.

[8] To obtain these bounds, observe that the undirected underlying graphs of fan-in two DAGs have average degree no more than 4. We do not formally state these results (which are direct corollaries of [BKKS11]) in the body however, as we provide improved bounds in Sect. 4.2.

nodes of in-degree 0 are white (corresponding to inputs to the computation), and the other nodes are either blue (corresponding to additions) or red (corresponding to multiplications). In graph theory terms, the bootstrap problem for maximum noise level L asks to find a small set of *marked* vertices such that every path containing $L+1$ red nodes must also contain at least one marked vertex. In other words, the bootstrap problem asks to reduce the *multiplicative* depth of a circuit by removing a few nodes.

1.2 Overview of Our Results

We present a framework for achieving low-communication MPC for *all* circuits, sometimes allowing for a restriction on the depth, but never relying on a specific circuit topology (such as layered circuits). The secure computation protocol is based on the following pebbling game on the circuit's underlying DAG:

1. A node u may be pebbled if it is a source node, or if every length-k directed path (the length is counted in vertices) ending in u contains a node which has already been pebbled.
2. Eventually, every node corresponding to an output gate must be pebbled (this includes every sink).

At a high level, the protocol has the parties compute some "masked version" (secret shares or ciphertexts) of the value of each pebbled gate. The parties start with the inputs, then iteratively compute shares of the value of each pebbled gate using the fact that each one can be computed as a depth-k circuit of already computed masked values. Finally, the parties "open up" the masked output values. Typically, the communication required to generate the masked inputs should scale only with the input size, the cost of computing the intermediary masked values should scale only with the number of pebbled gates, and the cost of opening the masked outputs should scale only with the output size. Instantiating this framework requires two key ingredients:

1. An algorithm to find a good pebbling of the circuit's underlying DAG.
2. A low-communication protocol to compute a masked version of the output of a depth-k circuit, given masked version of the inputs.

Solving the Pebbling Game. Our approach is to consider the circuit's underlying DAG, remove the input and output nodes, and then identify a directed k-path hitting set of the resulting DAG. Removing all these nodes yields a DAG of depth $k-1$.

Main Theorem 1 (Depth-reduction algorithms for fan-in two circuits)**.** *Let G be an in-degree two, depth-d DAG on n vertices. Then*

$$h_k(G) \leq \begin{cases} \lfloor 2n/3 \rfloor & \text{if } k = 2 \quad (1a) \\ \frac{2n}{5} \cdot \left(1 + \frac{3/2}{k}\right) & \text{for any } 1 \leq k < d \quad (1b) \\ \mathcal{O}\left(n \cdot \left(1 - \frac{\log k}{\log d}\right)\right) & \text{for any } 1 \leq k < d \quad (1c) \end{cases}$$

where $h_k(G)$ is the size of the smallest k-path hitting set of G. Furthermore this bound is constructive. Note that the last expression is particularly interesting in the $k = d^{1-o(1)}$ regime, as it yields a hitting set of size $o(n)$.

Main Corollary 1 (Circuit depth-reduction from k-path hitting set)**.** *Let $\mathcal{R}$ be a finite ring. Let C be a fan-in two, depth-d, n-input, m-output circuit with s gates*[9]*. Let (k, M) be any of the following combinations of parameters:*

$$\begin{cases} M = m + \lfloor 2(s-m)/3 \rfloor & \text{and} \quad k = 2 \\ M = m + \frac{2(s-m)}{5} \cdot \left(1 + \frac{3/2}{k}\right) & \text{and} \quad 1 \leq k < d \\ M = m + \mathcal{O}\left((s-m) \cdot \left(1 - \frac{\log k}{\log d}\right)\right) & \text{and} \quad 1 \leq k < d \end{cases}$$

There exists a sequence of M fan-in two, depth-k, single-output circuits $C_1, \ldots, C_M$ such that:

$$\forall \vec{x} \in \mathcal{R}^n,\ C(\vec{x}) = C_M\left(\vec{x}, (C_j(\vec{x}, (C_i(\vec{x}, \ldots))_{i<j}))_{j<M}\right).$$

(Or in algorithmic form, if

1. $y_1 \leftarrow C_1(\vec{x})$
2. *For* $i = 1 \ldots M$, $y_i \leftarrow C_i(\vec{x}, (y_1, \ldots, y_{i-1}))$

then $C(\vec{x}) = C_M(\vec{x}, (y_i)_{i=1}^M)$.)

Instantiating the Framework. We demonstrate the usefulness of main Theorem 1 by showing how each of the three expressions has applications to low-communication secure multiparty computation: the parameters of Eq. (1a) are useful to obtain *fractionally linear-communication MPC in the correlated randomness model*, those of Eq. (1b) are best suited for 1-*out-of*-2^{2^k} *OT-complexity of 2PC*, and finally the parameters of Eq. (1c) are best applied to *sublinear-communication secure computation low-depth circuits*.

Corollary 1 (MPC protocols through the depth-reduction lens).

1. **Information-theoretic MPC in the correlated randomness model.** *Let C be an n-input, m-output, $\mathbb{F}$-arithmetic circuit with s non-output computation gates (over any basis of binary gates). There exists an N-party protocol for perfectly securely computing C in the correlated randomness model in the presence of a semi-honest adversary corrupting up to $N-1$ parties. The protocol uses the following resources (in bits):*
   - Total communication:

$$\left(\frac{2}{3}s + n + m\right) \cdot N \cdot \log|\mathbb{F}|$$

[9] In order to simplify the expressions of M, we assume none of the input gates are also outputs.

  - Correlated randomness per party: $\mathcal{O}(s+n)\cdot\log|\mathbb{F}|$ *bits of function-dependent correlated randomness per party (or alternatively* $B^{\mathcal{O}(1)}\cdot\log|\mathbb{F}|$ *bits of function-independent correlated randomness, where* B *is some* a priori *bound on* $s+n+m$*),*
  - Local computation per party: $\mathcal{O}(s\cdot N\cdot\log^2|\mathbb{F}|)$.
2. **Sublinear-communication MPC from homomorphic secret-sharing.** *Assuming the existence of* N*-party homomorphic secret-sharing supporting logarithmic depth (respectively doubly logarithmic depth circuits)* $\mathbb{F}$*-arithmetic fan-in two circuits, there exists sublinear-communication secure* N*-party computation for* all $\log^{1+o(1)}$*-depth (resp.* $(\log\log)^{1+o(1)}$*-depth) circuits.*
3. **The 1-out-of-M-OT complexity of 2PC.** *Let* $f\colon\{0,1\}^n\to\{0,1\}$ *be a function which can be computed by a boolean circuit with* s *computation gates (over any basis of binary gates). For every* $M\geq 2$*, there exists a two-party protocol for computing* C *with passive security in the* 1*-out-of-*M*-OT hybrid model; this protocol makes* $\frac{2}{5}\left(1+\frac{3/2}{\lceil\log\log M\rceil+1}\right)\cdot s$ *calls to the oblivious transfer functionality.*

We note that the protocols of Corollary 1 can be extended to the malicious setting. The results for sublinear-communication from HSS also apply to correlated SPIR: Assuming the existence of correlated SPIR, there exists a sublinear-communication secure two-party protocol for $(\log\log)^{1+o(1)}$-depth circuits. Finally our depth-reduction algorithms (notably those in the parameter range of Eq. (1b)) provide upper bounds on the number of bootstraps required to achieve fully homomorphic encryption while tolerating a small constant maximum noise level: Given an FHE scheme tolerating a maximum noise level of L,

Table 1. Resources in field elements per party for computing a fan-in two arithmetic circuit with n inputs, m outputs, and s computation gates ($s_\times$ of which are multiplicative, and $s-s_\times$ linear), in the two-party setting. In the last column, k is not necessarily a constant. Note that the GMW protocol coincides with an extension of our protocol to $k=1$ (up to free addition), as a "1-path hitting set" is the set of all vertices.

Previous Works		
	"GMW with OT correlations" [GMW87, Bea92]	"Tiny Tables" [DNNR17]
Communication	$s_\times+n+m$	$s+n+m$
Correlated Randomness	$4s_\times+n$	$3s+n$
Function-Independent CR?	Yes	No
Computation	$s^{\mathcal{O}(1)}$	$s^{\mathcal{O}(1)}$
This Work		
	Colouring-based	FVS-based (Any $k\geq 2$)
Communication	$\frac{2}{3}s+n+m$	$\frac{2}{5}(1+\frac{3/2}{k})\cdot s+n+m$
Correlated Randomness	$10s+n^\dagger$	$2^{2^k}\cdot s+n^\dagger$
Function-Independent CR?	No†	No†
Computation	$s^{\mathcal{O}(1)}$	$\mathcal{O}(2^{2^k})\cdot s^{\mathcal{O}(1)}$

†The correlated randomness can be made *function-independent* at the cost of increasing it to B^{2^k} bits, where B is some *a priori* bound on $n+m+s$.

only $\frac{2}{5}s \cdot (1 + \frac{1.5}{L+1})$ bootstraps are required to homomorphically evaluate a size-s circuit (over an arbitrary basis of binary gates).

Concrete Efficiency. We conclude this section by comparing our fractionally linear-communication protocol to the previously best known linear-communication protocols in the correlated randomness model, in the two-party setting. The comparison is summarized in Table 1. When compared with the "Tiny Tables" protocol of Damgaard, Nielsen, Nielsen, and Ranelluci [DNNR17], we save a factor 1.5 in communication at the cost of a factor 4 in computation. The savings in communication can be asymptotically increased to a factor 2.5, but only by losing a very large factor in computation.

1.3 Technical Overview

We refer to Sect. 2 for definitions of the graph-theoretical notions discussed here. We consider three families of depth-reduction algorithms in this paper. Recall our goal is to constructively give an upper bound for $h_k(G)$ where the DAG G has n nodes and is of in-degree-2.

Colouring-Based Depth-Reduction. Observe that 2-paths are exactly (directed) edges, so the hitting set we are looking for is exactly a vertex cover. A DAG of in-degree two is 3-colourable, and furthermore a 3-colouring can be found in polynomial time by the following greedy algorithm: colour all sources with the same colour, then iteratively colour nodes in topological order with any of the three colours not already used by its parents. Since each node has at most two parents, the algorithm never gets "stuck" and only ever needs three colours.

Consider the 3-partition of the nodes obtained by this colouring. The union of the two smallest partition must have size at most $\lfloor 2n/3 \rfloor$ (or they would not be the smallest), and they must be a vertex cover of the graph (or there would be an edge whose two endpoints are of the third colour, which would violate the proper 3-colouring property).

FVS-Based Depth-Reduction. Our second family of depth-reduction algorithms is based on the following approach:

1. First, remove vertices from the in-degree-2 DAG so the resulting graph lies in some graph class $\mathcal{G}$.
2. Then, use a special-purpose depth-reduction algorithm for class $\mathcal{G}$.

Specifically, we consider the class $\mathcal{G}$ of all directed forests. This is motivated by the fact that directed trees (which in particular can be seen as layered graphs) admit very efficient depth-reduction algorithms. More specifically, we can reduce the depth of a directed forest to $k-1$ while removing only a 1-in-k fraction of its vertices. Indeed, while the forest still has depth at least k we can iteratively remove the $(k-1)^{\text{th}}$ ancestor ⓤ of the deepest leaf ⓥ. Whenever we remove a node in this fashion, we are guaranteed we will never need to remove any of the $k-1$ nodes of the path from ⓤ (excluded) to ⓥ (included). We say these $k-1$ nodes have been "saved by ⓤ". Because each node can be saved by at

most one node's removal, when the algorithm terminates we are guaranteed to have removed only a 1-in-k fraction of the nodes (if we removed ℓ nodes, then we saved $\ell \cdot (k-1)$; because $\ell + \ell \cdot (k-1) \leq n$, necessarily $\ell \leq \lfloor n/k \rfloor$).

The first step is handled by identifying a feedback vertex set (FVS) of the in-degree-2 DAG's underlying undirected graph. Fortunately, while the problem of finding a small FVS is NP-complete in general graphs, it is known that every 2-degenerate graph (that is, one whose edges can be oriented as an in-degree-2 DAG) admits an FVS of size $\lfloor 2n/5 \rfloor$ [BDBS14].

By combining these two steps, every n-vertex, in-degree-2 DAG has a k-path hitting set of size $\lfloor 2n/5 \rfloor + \lfloor \frac{n - \lfloor 2n/5 \rfloor}{k} \rfloor$ (which is at most $\frac{2n}{5} \cdot (1 + \frac{3/2}{k})$).

Valiant's [Val77] ***Edge-Partitioning-Based Depth-Reduction.*** The third algorithm is one proposed by Valiant [Val77], and which we simply apply to secure multiparty computation[10]. For completeness, we provide here a sketch of how this algorithm works. This DAG depth-reduction algorithm is based on deleting edges. In an in-degree-2 DAG however, the number of edges is within a factor 2 of the number of vertices, so the problem is essentially the same as the variant we consider. Valiant's algorithm starts by partitioning the vertices in d layers, according to their depth in the DAG. Note that because we do not assume the DAG itself has a layered structure, the edges are allowed to join any pair of layers. The edges are partitioned as $X_1 \sqcup \cdots \sqcup X_{\log d}$ as follows: an edge connecting two vertices in layers L_1 and L_2 is placed in partition X_i where i is the most significant bit in which the binary representations of L_1 and L_2 differ. We provide a visual representation of this partition in Fig. 1: any edge "crossing the blue line" is in X_1; any edge not in X_1 but "crossing a red line" is in X_2; any edge not in $X_1 \cup X_2$ but "crossing a green line" is in X_3. The key observation is that whenever we iteratively remove one of the partitions, the depth of the DAG is reduced by a factor 2. Removing the $\log(d/k)$ smallest partitions (whose

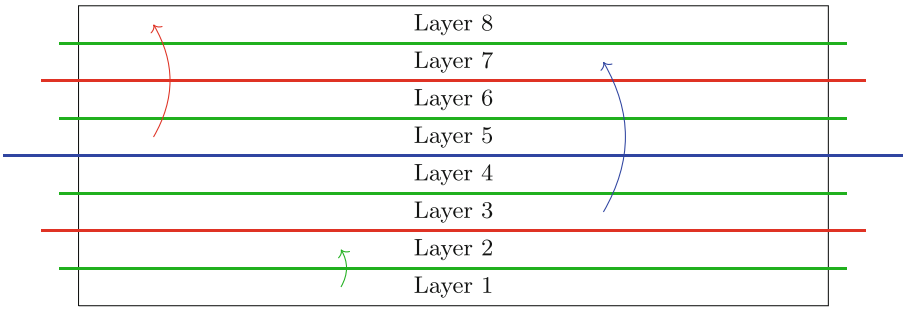


Fig. 1. Visual representation of the edge partitioning mechanism in Valiant's [Val77] depth-reduction algorithm.

[10] We note that Valiant's algorithm was previously applied in essentially the same way to other—yet incomparable—notions of depth-reduction and pebbling games, in the study of memory-hard functions [AB16, ABH17].

union has size at most $(\log(d/k)) \cdot \frac{\#\text{edges}}{\log d}$) therefore yields a depth-k DAG. It follows that every in-degree-2, n-vertex DAG of depth d admits a $(k+1)$-path hitting set of size $\mathcal{O}(n \cdot (1 - \frac{\log k}{\log d}))$.

2 Preliminaries

2.1 Cryptographic Notions

Definition 1 (Homomorphic Secret Sharing). *An N-party* Homomorphic Secret-Sharing (HSS) *scheme (with additive reconstruction) for a class $\mathcal{F}$ of functions over a finite field $\mathbb{F}$ is a pair of algorithms* $\mathsf{HSS} = (\mathsf{HSS.Share}, \mathsf{HSS.Eval})$ *with the following syntax and properties:*

- $\mathsf{Share}(1^\lambda, x)$*: On input* 1^λ *(the security parameter) and* $x \in \mathbb{F}^{n(\lambda)}$ *(the input), the* sharing algorithm Share *outputs N input shares* $(x^{(1)}, \ldots, x^{(N)})$.
- $\mathsf{Eval}(i, f, x^{(i)})$*: On input* $i \in [N]$ *(the party index),* $f \in \mathcal{F}$ *(the function to be homomorphically evaluated, implicitly assumed to specify input and output lengths n, m), and* $x^{(i)}$ *(the* i^{th} *input share), the* evaluation algorithm Eval *outputs the* i^{th} *output share* $y^{(i)} \in \mathbb{F}^m$.
- ***Correctness:*** *For any* 1^λ*, input* $x \in \mathbb{F}^{n(\lambda)}$*, and any function* $f \in \mathcal{F}$*,*

$$\Pr\left[y^{(1)} + \cdots + y^{(N)} = f(x) \ : \ \begin{array}{l}(x^{(1)}, \ldots, x^{(N)}) \xleftarrow{\$} \mathsf{HSS.Share}(1^\lambda, x) \\ y^{(i)} \xleftarrow{\$} \mathsf{HSS.Eval}(i, f, x^{(i)}),\ i = 1 \ldots N\end{array}\right] = 1.$$

- ***Security:*** *For every set of corrupted parties* $\mathcal{D} \subsetneq [N]$*, we consider the following game:*
  1. *The adversary* $\mathcal{A}$ *sends inputs* (x_0, x_1) *with* $|x_0| = |x_1|$.
  2. *The challenger picks* $b \xleftarrow{\$} \{0,1\}$ *and* $(x^{(1)}, \cdots, x^{(N)}) \xleftarrow{\$} \mathsf{HSS.Share}(1^\lambda, x_b)$.
  3. *The adversary outputs a guess* $b' \leftarrow \mathcal{A}((x^{(i)})_{i \in \mathcal{D}})$.

We let $\mathsf{Adv}(1^\lambda, \mathcal{A}, \mathcal{D})$ *denote the advantage* $|1/2 - \Pr[b = b']|$ *of* $\mathcal{A}$*. The scheme is* secure *if for any* $\mathcal{D} \subsetneq [N]$ *and any PPT adversary* $\mathcal{A}$*,* $\mathsf{Adv}(1^\lambda, \mathcal{A}, \mathcal{D})$ *is negligible.*

2.2 Graph-Theoretic Notions

In this paper, we consider only simple graphs, both directed and undirected. For a graph G we let $V(G)$ and $E(G)$ be the sets of vertices and edges of G, respectively. For a subset $U \subseteq V(G)$, we use $G[U]$ to denote the subgraph of G induced by U, *i.e.* the graph $(U, E(G) \cap U^2)$. We sometimes use the acronym *DAG* as shorthand for "directed acyclic graph" and *di-graph* or *digraph* as shorthand for "directed graph". In a DAG, the nodes of in-degree 0 are called *sources* and the nodes of out-degree 0 are called *sinks*. Borrowing from circuit terminology, we allow a subset of a DAG's nodes to be identified as *output nodes*: this can be an arbitrary subset, provided that it contain all sinks and no sources.

(Directed) Paths. A (directed path) *in* a (directed) graph is a non-repeating sequence of vertices such that each pair of consecutive vertices form an (arc) edge of the graph. The *length* of a (directed) path is the number of vertices[11] in this sequence. The *depth* of a directed acyclic graph is the length of the longest path in it.

Feedback Vertex Set. A *feedback vertex set* (FVS) of an undirected graph G with vertex set V is a set $S \subseteq V$ of its vertices such that the graph $G[V \setminus S]$ is a forest. The *FVS decision problem* asks, on input an undirected graph G and a positive integer k, whether G admits an FVS of size at most k.

Vertex Colouring. A *(proper) vertex colouring* of graph $G = (V, E)$ is a function $\pi \colon V \to \mathbb{N}$ which assigns different values (called *colours*) to neighbouring vertices. A *k-vertex-colouring* (or simply a *k-colouring*) is a vertex colouring which uses at most k colours.

Independent Set. An *independent set* is a set of vertices in a graph such that no pair of them are adjacent to each other.

Graph Degeneracy. The *degeneracy*[LW70] (*a.k.a.* the coloring number -1 [EH66]) of a graph G is the least integer $k \geq 0$ such that every induced subgraph of G contains a vertex with at most k neighbours. If a graph has degeneracy at most d, we say it is *d-degenerate*. A graph is d-degenerate if and only if it admits a d-elimination ordering [LW70], *i.e.* an ordering of the vertices in which each vertex appears after at most d of its neighbours. A graph is d-degenerate if and only if the edges of G can be oriented to form a directed acyclic graph with in-degree at most d [CE91] (Fig. 2).

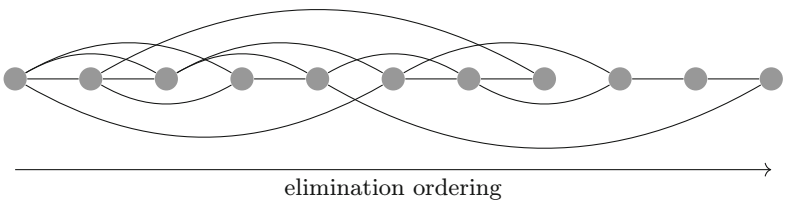


Fig. 2. The nodes of a 2-degenerate graph can be ordered in such a way that each node is preceded (in the ordering) by at most 2 of its neighbours.

3 Depth-Reduction Pebbling Game

The goal of this section is to show how solving the graph-theoretic problem of reducing an in-degree-2 DAG's depth by removing (few) nodes leads to low-communication MPC protocols.

[11] Beware that the literature is inconsistent on whether the length of a path should be counted in vertices or edges.

In Sect. 3.1 we state the graph theoretic problem of *DAG depth reduction*. In Sect. 3.2 we introduce a "pebbling game" for the gates of a (boolean or arithmetic) circuit, and show it can be solved using DAG depth-reduction. In Sect. 3.3 we explain how many existing low-communication MPC protocols implicitly rely on finding a good pebbling of the circuit to be securely computed.

3.1 The Depth-Reduction Problem for Directed Acyclic Graphs

Definition 2 (DAG Depth-Reduction).

- **DAG Depth-Reducibility.** *Let $k, \ell \in \mathbb{N}^\star$. We say a directed acyclic graph $G = (V, E)$ is* (k, ℓ)-depth-reducible *if there exists a subset $S \subseteq V$ of size at most ℓ such that $G[V \setminus S]$ has depth at most k (*i.e. *the longest directed path in $G[V \setminus S]$ contains at most k vertices).*
- **The DAG Depth-Reduction Problem.** *Given a directed acyclic graph $G = (V, E)$, the* (k, ℓ)-depth-reduction problem *(denoted (k, ℓ)-*DR*) asks to find a subset $S \subseteq V$ of size at most ℓ such that $G[V \setminus S]$ has depth at most k (*i.e. *the longest directed path in $G[V \setminus S]$ contains at most k vertices), or output $\perp$ if no such subset exists.*

Remark 1 (Equivalent Problems to DAG Depth-Reduction). The following holds:

1. The (k, ℓ)-DR problem is that of finding a hitting set of size at most ℓ for all directed $(k+1)$-paths in G.
2. The $(1, \ell)$-DR problem is that of finding a vertex cover of size at most ℓ.

We bring the reader's attention to the fact that removing a k-path hitting set from a DAG yields a DAG of depth at most $k - 1$.

3.2 A Depth-Reduction Pebbling Abstraction

We outline below a convenient intermediate abstraction to interface between the depth-reduction problem over digraphs and the execution of a secure multiparty computation protocol. We model the distributed computation of a function f, given by a boolean circuit C, as the *Depth-Reduction pebbling game*. In this pebbling game the parties can place a pebble on a node s of the underlying graph G of the circuit if the following condition is met: either the node is an input node, or all directed path of length $k+1$ in G ending at s already contain a pebbled node. Formally,

Definition 3 (DR pebbling game).

Given a digraph $G = (V, E)$ and a depth parameter k, a depth-reduction *(*DR*) pebbling game* PG *is a list of subsets $S \subset V$ of the nodes of G. A* DR *pebbling game* $\mathsf{PG} = (S_1, \cdots, S_{|\mathsf{PG}|})$ *is* valid *if after executing the commands* $\mathsf{PebbleGates}(S_i)$ *sequentially for $i = 1$ to $|\mathsf{PG}|$, defined in Fig. 3,*

- *no command returns an* invalid *flag,*

– *the termination condition is met (*i.e. *all output nodes are pebbled).*

The cost $\mathsf{cost}(\mathsf{PG})$ *of a valid* DR *pebbling game* PG *is defined as the total number of pebbles placed throughout the game,*

$$\mathsf{cost}(\mathsf{PG}) = \sum_{S \in \mathsf{PG}} |S|$$

and its length $\mathsf{length}(\mathsf{PG}) = |\mathsf{PG}|$ *is the length of the list (*i.e. *the number of subsets).*

The rules of the DR pebbling game are formally described on Fig. 3.

DR pebbling rules

Parameters. The game is parameterised by a directed acyclic graph $G = (V, E)$ with in-degree 2, and a depth parameter k.

Initialisation. At the start of the game, a pebble is placed on all input nodes.

Moves. The player can execute the following command:

▷ $\mathsf{PebbleGates}(S)$: given a set $S \subset V$ of gates,
- for every $s \in S$, check that every path in G of length $k+1$ ending at s contains a pebbled node. If not, return the flag $\mathsf{invalid}$.
- Place a pebble on all nodes in S.

Termination. The game terminates once all output nodes have been pebbled.

Fig. 3. The moves and termination condition of a depth-reduction pebbling game.

Then, we have the following straightforward lemma that relates the cost of a pebbling game to the k-depth-reducibility of G:

Lemma 1. *Let $G = (V, E)$ be a digraph with m output nodes*[12]*; we denote V_{sources} and V_{outputs} the sets of sources and outputs of G, respectively. Let $G' := G[V \setminus (V_{\mathsf{sources}} \sqcup V_{\mathsf{outputs}})]$. If G' is (k, ℓ)-depth-reducible, then there is a valid* DR *pebbling* PG *of G with depth parameter k, cost $\mathsf{cost}(\mathsf{PG}) \leq \ell + m$, and whose length $|\mathsf{PG}|$ is the largest number of pebbles on a path from a source to a sink in G.*

Proof. The proof follows from a straightforward greedy procedure. Because G' is (k, ℓ)-depth reducible, there exists a size-ℓ subset V' of the nodes of G' whose removal yields a depth-k DAG. Initially, all sources in G are pebbled. At each

[12] We refer to Sect. 2.2 for what we mean by "output nodes of a DAG".

round, identify the largest set $S \subseteq V' \cup V_{\mathsf{outputs}}$ of unpebbled nodes such that $\mathsf{PebbleGates}(S)$ does not return $\mathsf{invalid}$ (i.e., the set of all nodes $s \in V'$ such that every path in G of length $k+1$ ending at s contains a pebbled node), and pebble this set. Iterate until all nodes in V_{outputs} have been pebbled. It follows from the definition of (k, ℓ)-depth-reducibility terminates with all the sinks coloured. Indeed, assume toward contradiction that a node $s \in V_{\mathsf{outputs}}$ was not pebbled by this procedure. Then there must be an ancestor a of s (at distance at most k of s) which was not pebbled by this procedure (otherwise, s would have been pebbled). Repeating this argument from the node a, we arrive after at most $\mathsf{depth}(G)$ steps at an input node, which is pebbled by assumption, reaching a contradiction. Furthermore, all nodes pebbled in this process are nodes from $V' \cup V_{\mathsf{outputs}}$, hence $\mathsf{cost}(\mathsf{PG}) \leq |V' \cup V_{\mathsf{outputs}}| \leq \ell + m$. This concludes the proof. □

3.2.1 Example: A DR Pebbling of Layered Graphs. We say that an in-degree-2 directed acyclic graph is *layered* [GJ11] if its nodes can be divided into layers, such that any edge only connects adjacent layers. Let $G = (V, E)$ be a layered in-degree-2 directed acyclic graph with $|V| = s$ nodes, and let k be an arbitrary parameter. Let n be the number of sources and m be the number of sinks in G. Let d be the depth of G.

Claim. There exists a valid DR pebbling PG of G with $|\mathsf{PG}| \leq \lceil d/(k+1) \rceil + 1$ and $\mathsf{cost}(\mathsf{PG}) \leq m + \lceil s/k \rceil$.

Equivalently, the proof below shows that layered graphs of size s with m outputs are $(k-1, m + \lceil s/k \rceil)$-depth reducible for any k.

Proof. First, observe that d is necessarily equal to the number of layers in G. Second, divide G into $\lceil d/k \rceil$ chunks of k consecutive layers (written from top to bottom). Let $i \in [1, k]$ be an index such that the total number of nodes in the i-th layers of all chunks is at most $\lceil s/k \rceil$ (such an index i exists by the pigeonhole principle).

The pebbling strategy proceed in $\lceil d/k \rceil + 1$ rounds: in the j-th round, place pebbles on all nodes in the i-th layer of the j-th chunk, as well as on all unpebbled sinks above the i-th layer of the j-th chunk. Observe that when $j = 1$, all nodes in the i-th layer of the first chunk, and all sinks above this layer, have depth at most $i \leq k$, hence every path of length k ending at a node of the i-layer must contain an source (and length-k paths can only exist when $i = k$), which is pebbled.

When $j > 1$, all k-ancestors of the i-th layer of the j-th chunk are on the i-th layer of the $(j-1)$-th chunk (because the graph is layered), hence they have been pebbled at the round $j - 1$. Furthermore, each path of length k ending on an unpebbled sink above the j-th chunk contains a node on the i-th layer of the ℓ-th chunk, for some $\ell < j$, which was pebbled in one of the previous rounds.

Eventually, in the $(\lceil d/k \rceil + 1)$'s round (the last round), it only remains to place a pebble on the remaining unpebbled sinks (if any) situated after the i-th layer

of the last chunk, and all length-k path ending in these nodes contain a pebbled node from the i-layer of the last chunk. Therefore, all moves are valid, and all sinks are pebbled at the end of the DR pebbling game. In total, the game places pebbles on the i-th layer of each chunk (at most $\lceil s/k \rceil$ pebbles in total) plus a pebble on each sink (at most m additional pebbled), hence $\mathsf{cost}(\mathsf{PG}) \leq m + \lceil s/k \rceil$. This concludes the proof. □

3.3 Recasting MPC Protocols Through the Lens of DR Pebblings

The DR pebbling game abstraction allows to recast several existing low-communication MPC protocols in a unified way which isolates their cryptographic component from the graph algorithm that they implicitly rely on to traverse the circuit of the function. The literature contains several protocols which require less communication than the size of the circuit, for a suitable class of "well-structured" circuits. For example, [BGI16a, Def 4.6] introduces the notion of circuits over branching programs, while [Cou19, CM21] use layered circuits, and [BCM22, BCM23] use synchronous circuits.

All these protocols (and others) evaluate the circuit by distributively and iteratively computing the values carried on a subset of intermediate nodes in a hidden fashion (the values are either shared [BGI16a, Cou19], masked [BCM22], or encrypted [Gen09, BV11, BGV12]. Computing the cost of these protocols (in terms of communication, computation, and storage overhead) can be abstracted out as follows:

- the cryptographic mechanism introduced in the protocol induces a (storage, communication, and computation) cost for each intermediate node whose value is (securely) computed, and
- the final cost of running the protocol is derived by summing the costs of computing the intermediate nodes, where the nodes are selected using a suitable valid DR pebbling of the graph of the circuit.

Cast in this language, there is nothing mysterious in the restriction to circuit classes such as layered or synchronous circuit: it simply comes from the fact that these are classes of circuits whose underlying graph G is (k, ℓ)-depth-reducible with non-trivial parameters (k, ℓ), which in turns implies by Lemma 1 the existence of a good DR pebbling game on G; see also Sect. 3.2.1 for an explicit description of an efficient pebbling strategy for layered graph. It also follows immediately that all of these results can be extended to larger classes of circuits provided that we can find sufficiently non-trivial DR pebbling games for their underlying family of graphs.

For the sake of concreteness, and to facilitate the statements of the corollaries which we obtain in this paper, we work out explicitly the abstraction on a few illustrative examples below. We stress that we do not prove new results on secure computation in what follows: rather, for a list of existing protocols that were originally described over a restricted class of circuits, we observe that the protocols work identically over general circuits but that their efficiency depends

on the existence of an efficient DR pebbling. Our lemmas and corollaries simply reformulate the existing results in this setting.

3.3.1 Information-Theoretic Secure Computation in the Correlated Randomness Model. In [Cou19], Couteau introduced the first information-theoretic secure computation protocol in the correlated randomness model which achieves sublinear communication complexity $O(s/\log\log s)$ for all layered circuits of size s, using a polynomial amount of computation and correlated randomness. For simplicity, we focus here on the case of secure computation of boolean circuits with semi-honest security, but the result of [Cou19] extends to arithmetic circuit and to the malicious setting.

On Fig. 4, we recall the protocol Π_{corr} of [Cou19]. Rather than focusing on layered circuits, we describe the protocol for an arbitrary circuit C given a suitable pebbling of the underlying graph of C. Let C be a circuit, and let PG be a valid DR pebbling of the graph G of C with depth parameter $k(|C|) = \log\log|C|$.

Lemma 2. *Let C be an n-input boolean circuit with m output gates, and let* PG *be a valid* DR *pebbling of its graph $G = (V, E)$. Then the protocol Π_{corr} of Fig. 4 is an information-theoretically secure N-party protocol for computing C in the correlated randomness model. Furthermore, the protocol Π_{corr} has the following efficiency properties:*

- Correlated randomness: *each party receives at most $|V| + \mathsf{cost}(\mathsf{PG}) \cdot 2^{2^k}$ bits of correlated randomness from the trusted dealer.*
- Communication: *the total communication of the protocol is upper bounded by $N \cdot (n + \mathsf{cost}(\mathsf{PG}))$.*
- Computation: *each party performs at most $O(\mathsf{cost}(\mathsf{PG}) \cdot (2^{2^k} + N))$ boolean operations.*

Lemma 2 is a reformulation of [Cou19, Theorem 1] in the setting of general circuits with a DR pebbling game, and the proof of Lemma 2 is a direct adaptation of the analysis in [Cou19]. Due to the choice of $k = \log\log(s)$, note that the amount of correlated randomness and computation remain polynomial. Furthermore, whenever $\mathsf{cost}(\mathsf{PG}) \ll s$, the total communication of the protocol is $\ll N \cdot (n + s)$, *i.e.*, below the "circuit-size barrier". Plugging the efficient valid DR pebbling of layered graphs described in Sect. 3.2.1 (whose cost is at most $m + \lceil s/\log\log(s)\rceil$) recovers the exact statement of Theorem 1 in [Cou19] for layered boolean circuits.

3.3.2 Secure Computation from Homomorphic Secret Sharing. We recall below another approach for sublinear secure computation, which was originally introduced in [BGI16b]. This protocol established HSS as the first known alternative to FHE to obtain sublinear secure computation for an expressive class of circuits. On Fig. 5, we recall a simplified variant of the protocol of [BGI16b] for arbitrary circuits with a valid DR pebbling given a statistically correct HSS scheme for the class NC^1, and discuss extensions that rely on weaker forms of

Protocol Π_{corr}

PARAMETERS. Let C be a boolean circuit whose underlying digraph is $G = (V, E)$. Let $k(|C|) = \log\log|C|$ be a depth parameter, and let PG be a valid DR pebbling of G. The protocol involves N parties $(P_1, \cdots, P_N)$.

CORRELATED RANDOMNESS. The trusted dealer proceeds as follows:

- For each node $v \in V$ which is not an output node, the trusted dealer samples a random mask $r_v \xleftarrow{\$} \mathbb{F}_2$. For each output node $v \in V$, the trusted dealer sets $r_v \leftarrow 0$. For all nodes $v \in V$, the dealer samples N random shares $(r_v^{(i)})_{i \leq N}$ of r_v.
- For each set $S \in \mathsf{PG}$, for each $s \in S$, let $V_s \subset V$ denote a set containing one pebbled node from each length-k path ending at s (V_s exists because PG is a valid pebbling). Let $f_s : \{0,1\}^{|V_s|} \mapsto \{0,1\}$ denote the function which computes the value carried by the node s in C from the values carried on all nodes $v \in V_s$. Let M_s denote the truth-table of the function $x \mapsto f_s(x \oplus (r_v)_{v \in V_s})$. The dealer samples N random shares $(M_s^{(i)})_{i \leq N}$ of M_s.
- The dealer sends $(r_v^{(i)})_{v \in V}$ and $(M_s^{(i)})_{s \in S}$ for all sets $S \in \mathsf{PG}$ to each party P_i. The dealer also sends r_v for each output node v to all parties.

PROTOCOL. We assume w.l.o.g that the parties hold shares of all inputs.

- For each input node s and each input shares $(x^{(1)}, \cdots, x^{(N)})$ of an input value x on the node s, each party P_i sends $x^{(i)} \oplus r_s^{(i)}$. All parties reconstruct $x \oplus r_s = \bigoplus_i (x^{(i)} \oplus r_s^{(i)})$. Throughout the protocol the parties will maintain the following invariant: each time they pebble a node v, they will reconstruct $z_v = u_v \oplus r_v$, where u_v is the value carried on this node.
- For each set $S \in \mathsf{PG}$, for each $s \in S$, the parties retrieve the masked values z_v for each $v \in V_s$, which they computed previously. Each party P_i broadcasts $M_s^{(i)}[(z_v)_{v \in V_s}] \oplus r_s^{(i)}$ and all parties reconstruct $\bigoplus_i (M_s^{(i)}[(z_v)_{v \in V_s}] \oplus r_s^{(i)}) = M_s[(z_v)_{v \in V_s}] \oplus r_s = f_s((u_v)_{v \in V_s}) \oplus r_s = u_s \oplus r_s = z_s$.
- Once z_v has been computed for all output nodes v, all parties output the z_v's. Note that $z_v = u_v + r_v$ where u_v is the value carried on the node, and $r_v = 0$ for all output nodes.

Fig. 4. An information-theoretic secure computation protocol in the correlated randomness model

HSS afterwards. Let C be a circuit, and let PG be a valid DR pebbling of the graph G of C with depth parameter $k(|C|) = \log|C|$.

Protocol Π_{hss}

PARAMETERS. Let C be a boolean circuit whose underlying digraph is $G = (V, E)$. Let $k(|C|) = \log |C|$ be a depth parameter, and let PG be a valid DR pebbling of G. Let $\sigma : V \mapsto \{0,1\}$ be a function which, on input $v \in V$, returns 0 if v is an output node, and 1 otherwise. The protocol involves 2 parties (P_0, P_1). Let $\mathsf{HSS} = (\mathsf{HSS.Share}, \mathsf{HSS.Eval})$ be an HSS scheme for the class of functions NC^1, and let $\{F_K\}_{K\in\{0,1\}^\lambda}$ be a PRF family in NC^1. Let Π_{Share} denote a secure 2-party protocol which, on input $K_b \in \{0,1\}^\lambda$ from each party P_b, computes $(K_b^{(0)}, K_b^{(1)}) \xleftarrow{\$} \mathsf{HSS.Share}(1^\lambda, K_b)$ for $b = 0, 1$, and outputs $(K_0^{(0)}, K_1^{(0)})$ to P_0 and $(K_0^{(1)}, K_1^{(1)})$ to P_1.

INITIALISATION. We assume w.l.o.g. that the two parties hold additive shares of all inputs.

- Each party P_b samples $K_b \xleftarrow{\$} \{0,1\}^\lambda$.
- For $b = 0, 1$, the two parties run Π_{Share} on inputs (K_0, K_1). Each party P_b gets $(K_0^{(b)}, K_1^{(b)})$.
- For each input node $v \in V$ and each input shares $(x^{(0)}, x^{(1)})$ of an input value x on the node v, each party P_b broadcasts $v_b \leftarrow F_{K_b}(v) \oplus x^{(b)}$. All parties reconstruct $v^\star = v_0 \oplus v_1 = x \oplus (F_{K_0}(v) \oplus F_{K_1}(v))$ (it helps to view the value $v^\star$ as the pebble on v).

PROTOCOL. Let us denote $\mathsf{PG} = (S_1, \cdots, S_{|\mathsf{PG}|})$. For $i = 1$ to $|\mathsf{PG}|$,

- For each $s \in S_i$, let $V_s \subset V$ denote a set containing one pebbled node in S_{i-1} from each length-k path ending at s (V_s exists because PG is a valid pebbling). Let $f_s : \{0,1\}^{|V_s|} \mapsto \{0,1\}$ denote the function which computes the value carried during the computation (in the circuit C) by the node s from the values carried on all nodes $v \in V_s$. Let us denote $(v_1, \cdots, v_{|V_s|})$ the elements of V_s. For $j = 1$ to $|V_s|$, the parties retrieve the values $v_j^\star$ computed in the previous round. Let $v^\star \leftarrow (v_1^\star, \cdots, v_{|V_s|}^\star)$ Then, the parties define the following function g_s:

$$g_s : \begin{cases} \{0,1\}^{|V_s|} & \mapsto \{0,1\} \\ (K_0, K_1) & \to f_s(v^\star \oplus (F_{K_0}(v_j) \oplus F_{K_1}(v_j))_{j\le|V_s|}) \oplus \sigma(s) \cdot (F_{K_0}(s) \oplus F_{K_1}(s)) \end{cases}$$

  Note that $g_s \in \mathsf{NC}^1$ (because the circuit of f_s is a log-depth circuit and the PRF is in NC^1). Furthermore, by construction, $g_s(K_0, K_1) = y \oplus (F_{K_0}(s) \oplus F_{K_1}(s))$ if s is not an output node, where y is the value carried by s during the computation, and $g_s(K_0, K_1) = y$ if s is an output node.
- For each $s \in S_i$, each party P_b computes and sends $s_b \leftarrow \mathsf{HSS.Eval}(b, g_s, (K_0^{(b)}, K_1^{(b)}))$. Both parties reconstruct $s^\star \leftarrow s_0 \oplus s_1$.

OUTPUT. For each output node $v \in V$, the parties output $v^\star$.

Fig. 5. A two-party protocol for C with semi-honest security

Lemma 3. *Let C be a boolean circuit with n input gates and m output gates, and let* PG *be a valid* DR *pebbling of its graph $G = (V, E)$. Then the protocol Π_{hss} of Fig. 5 is a secure 2-party protocol in the honest-but-curious setting for computing C. Furthermore, the total communication of the protocol Π_{hss} is upper bounded by $2 \cdot (n + \mathsf{cost}(\mathsf{PG})) + \mathsf{poly}(\lambda)$.*

Lemma 3 follows directly from the same analysis as HSS-based secure computation protocols from previous works [BGI16b]. Above, the $\mathsf{poly}(\lambda)$ term refers to the fixed communication cost (independent of n, m, and the circuit size) of the secure 2-party protocol for distributively running $\mathsf{HSS.Share}$ on a pair of λ-bit inputs. Plugging the efficient valid DR pebbling of layered graphs described in Sect. 3.2.1 (whose cost is at most $m + \lceil s/\log(s) \rceil$) recovers the result of previous works from HSS for NC^1 together with PRFs in NC^1.

Extension 1: Las Vegas HSS. We note that the construction of Lemma 3 is simplified compared to the original construction of [BGI16b] since it assumes a statistically correct HSS scheme for NC^1. While such schemes were later constructed from assumptions such as DCR [OSY21] or class groups [ADOS22], they were not known (without using indistinguishability obfuscation of FHE-style primitives) at the time of [BGI16b]. Instead, [BGI16b] relied on a DDH-based construction of *Las Vegas* HSS, which satisfies a weaker correctness property. A similar, slightly more complex construction works nonetheless given Las Vegas HSS, by letting the functions g_s encode their output with a suitable low-complexity error correcting code. Combining the efficient valid DR pebbling of layered graphs from Sect. 3.2.1 with this variant recovers the result of [BGI16b].

Extension 2: Single-Function HSS. Another extension replaces the HSS by *single-function* HSS, a weaker notion where $\mathsf{HSS.Share}$ must specify in advance the circuit to be computed on the shares. One can also modify the previous construction to work with single-function HSS, by initially distributing $\mathsf{HSS} \cdot \mathsf{Share}(1^\lambda, K_b, (g_s)_{s \in S})$ for all sets $S \in \mathsf{PG}$ (viewing $(g_s)_{s \in S}$ as a single function that takes as input (K_0, K_1) and outputs $(g_s(K_0, K_1))_{s \in S}$). This increases the total communication to $2 \cdot (n + \mathsf{cost}(\mathsf{PG})) + |\mathsf{PG}| \cdot \mathsf{poly}(\lambda)$, which is still sublinear for layered circuits which are not too "tall-and-skinny" (since there, $|\mathsf{PG}| = O(d/k)$ where d is the circuit depth).

Combining the efficient valid DR pebbling of layered graphs from Sect. 3.2.1 with this variant, and setting $k \leftarrow \log \log s$, almost recovers the result of [CM21] which achieves sublinear 2-party computation from the super-polynomial hardness of LPN, by building single-function HSS for loglog-depth circuits from superpoly-LPN. A minor distinction is that our construction would require a PRF computable in depth loglog. At a high level, the PRF is used in our construction to turn a (possibly non-compact) HSS into a compact HSS (whose share size on input x is $|x| + \mathsf{poly}(\lambda)$) using a hybrid encryption technique. Instead, the work of [CM21] avoids this additional assumption by directly building a compact single-function HSS from the super-polynomial hardness of LPN. Summing up:

Corollary 2. *Let C be a boolean circuit with m output gates, and let* PG *be a valid* DR *pebbling of its graph $G = (V, E)$ of depth k. Then:*

- *if* $k = \log |C|$ *and assuming the hardness of either DCR or DDH, there exists a secure* 2*-party protocol for* C *with communication* $2 \cdot (n + \mathsf{cost}(\mathsf{PG})) + \mathsf{poly}(\lambda)$*, and*
- *i* $k = \log \log |C|$ *and assuming the super-polynomial hardness of LPN (semi-honest setting) or additionally the existence of collision-resistant hash functions (malicious setting), there exists a secure* 2*-party protocol for* C *with communication* $2 \cdot (n + \mathsf{cost}(\mathsf{PG})) + |\mathsf{PG}| \cdot \mathsf{poly}(\lambda)$

In the corollary above, the statement about security in the malicious setting comes from the existence of a communication-preserving semi-honest to malicious compiler given succinct zero-knowledge arguments (via the GMW compiler [GMW86]), which exists assuming collision-resistant hash functions [Kil92]. The latter is implied by either DDH or DCR, but not by the flavour of LPN used in [CM21] (though strong forms of LPN imply CRHF [YZW+19]).

3.3.3 Further Protocols that Fit the Abstraction. We now list other secure computation protocols whose recasting as protocols computing a circuit through a DR pebbling of its graph captures adequately some of their efficiency properties.

2-Party Computation from Correlated Symmetric PIR. The work of [BCM22] achieves sublinear secure computation using a strong form of private information retrieval, called correlated symmetric PIR. In a similar fashion as for the protocols of the previous sections, the protocol of [BCM22] can be described for all circuits given a suitable DR pebbling of their graph with depth parameter $k = c \cdot \log \log |C|$ for an appropriate constant c. The dependency in the parameters of the DR pebbling is slightly more complex, but still translates to a protocol with sublinear communication whenever $\mathsf{cost}(\mathsf{PG}) = o(|C|)$ and the circuit is not too "small and skinny" (which translates to $|\mathsf{PG}|$ being significantly smaller than $\mathsf{cost}(\mathsf{PG})$). The lemma below is the generalization of Corollary 18 in [BCM22] to arbitrary circuits C:

Lemma 4. *Let* C *be a circuit with* n *inputs,* m *outputs, and let* PG *be a valid* DR *pebbling of the graph* G *of* C *with depth parameter* k*. Assuming the existence of correlated SPIR, there exists a secure 2-party protocol for* C *with communication complexity*

$$O\left(n + m + |\mathsf{PG}|^{1/3} \cdot \left(2^{k+2^k} \cdot \mathsf{cost}(\mathsf{PG})\right)^{2/3} \cdot \mathsf{poly}(\lambda) + \mathsf{cost}(\mathsf{PG})\right).$$

Correlated SPIR can be constructed assuming the LPN assumption (with polynomial hardness) and either of QR, DDH, DCR, or LWE [BCM22, BCM23]. This implies secure 2-party computation with the communication complexity outlined above under these combinations of assumptions.

Sublinear Multiparty Computation. The recent work of [BCM23] introduced an approach for sublinear secure computation which, at a high level, combines

correlated SPIR with N-party homomorphic secret sharing to achieve sublinear MPC for $N + 1$ parties. Combining this with existing constructions of 2-party and 4-party HSS, they obtain constructions of sublinear 3-party and 5-party secure computation protocols for layered circuits. Another recent protocol that fits our abstraction is given in the work of [DIJL23], which introduced an N-party homomorphic secret sharing scheme (with imperfect correctness) for any polynomial N and all $\log \log$-depth circuits, and derived a sublinear N-party secure computation protocol for all layered graphs. We briefly note that all these approaches also fit our framework, and their protocols can be seen to work identically over any circuit C with a suitable DR pebbling.

FHE-Based Secure Computation with Reduced Bootstrapping. Until now, we outlined protocols whose *communication complexity* depends on finding a suitable DR pebbling of the graph of the circuit. We now show that this abstraction is also useful beyond this setting. It is well known that fully homomorphic encryption (FHE) [Gen09] implies secure computation of arbitrary function with communication independent of the circuit size. However, evaluating arbitrary circuits involves bootstrapping, which is typically quite expensive.

The work of [BLMZ17] initiated the study of the number of bootstrapping operations required to homomorphically evaluate a circuit. In their abstraction, each FHE ciphertext is associated to a *noise level*, represented as an integer. Evaluating any non-linear gate increase the noise level by 1. When some pre-specified maximum noise level is reach, an expensive bootstrapping must be performed. While one could set the maximum noise level to be above the circuit size to avoid bootstrapping altogether, allowing higher noise levels typically results in much larger ciphertexts and much less efficient homomorphic operations. This suggests the following question studied in [BLMZ17]:

Given a maximum noise level k and a circuit C, how many bootstrappings are required to homomorphically compute C?

Concretely, the model is as follows: fix a maximum noise level k. All inputs are encrypted with respect to a noise level 0. At each gate, the output of the gate becomes encrypted with respect to a noise level equal to the maximum noise levels of its inputs (for linear gates) or the maximum noise level of its inputs plus 1 (non-linear gates). If the noise level of an input to a gate is equal to k, a bootstrapping operation must be performed, which resets the noise level to 0.

Let C be a circuit with n inputs, m outputs, and let PG be a valid DR pebbling of the graph G of C with depth parameter k. The following follows almost immediately from the definition of DR pebblings:

Lemma 5. *The circuit C can be homomorphically evaluated with FHE ciphertexts of maximum noise level k using at most $\mathsf{cost}(\mathsf{PG})$ bootstrapping operations.*

Proof. The proof is straightforward: the homomorphic evaluation runs a bootstrapping evaluation at each gate where a pebble is placed during the game. Because all path of length k ending in a pebbled gate are guaranteed to contain

a pebbled node already (which implies that the ciphertext encrypting the output of the node has noise level 0 because a bootstrapping was performed), the noise level of the ciphertexts encrypting the inputs to the gate is at most k. □

Remark 2. The size $\mathsf{cost}(\mathsf{PG})$ of a DR pebbling of the underlying digraph of the circuit C yields an upper bound on the number of bootstrapping operations required to homomorphically evaluate a circuit. We note that, since this measure depends solely of the graph of the circuit, it is agnostic of the type of gates. On the downside, this means that it does not take advantage of the fact that addition gates are typically "for free" in FHE schemes. On the positive sides, it yields an upper bound that holds for homomorphic evaluation of boolean circuits over an arbitrary boolean basis.

We note that our result is not directly comparable to the result of [BLMZ17]: their work showed that closely approximating the minimal number of bootstrapping is NP-hard, and provided a polytime k-approximation of the best solution. However, their result does not provide any bound on the size of the best possible solution. In contrast, we provide an upper bound on the number of bootstrapping required for *any* boolean circuit, as a function of its DR pebbling complexity. Looking ahead, combined with the non-trivial DR pebbling algorithms which we introduce in the next section, this will yield algorithms to homomorphically evaluate low-depth circuits with a sublinear number of bootstrapping operations, and algorithms to homomorphically evaluate any circuit using bootstrapping for a constant fraction of all gates.

The Complexity of OT-Based Secure Computation. An 1-out-of-n oblivious transfer (OT) is a protocol that allows a sender holding inputs $(s_1, \cdots, s_n)$ to reveal s_i to a receiver with input $i \in \{1, \cdots, n\}$ without learning i, and without revealing any s_j for $j \neq i$ to the receiver. The seminal GMW protocol [GMW87] showed that any circuit can be securely evaluated (in the 2-party setting) using a 1-out-of-4 oblivious transfer protocol. Informally, the 1-out-of-4 OT is used to let one party obliviously retrieve its share from the truth table (of size 4) of a binary gate (scrambled with the masks held by the other party). This approach generalizes immediately to securely computing circuits with k-ary gates using 1-out-of-2^k oblivious transfer. This suggests the following natural question: given a circuit C and a 1-out-of-n oblivious transfer protocol, how many invocations of the OT are necessary to securely evaluate C?

For $n = 2$, the OT complexity of secure computation was previously studied in [BIKK14]. In this section, we observe that for general values of n, the protocols of [GMW87, DNNR17] immediately yield an information-theoretic protocol given access to 1-out-of-n OT functionality for circuits with n-ary gates. Now, given an efficient DR pebbling with depth parameter k of the graph of a circuit C, observe that the value of each pebbled node v can be computed as a function of the value of at most 2^k pebbled ancestors of v (since all path of length k ending in v must contain a pebbled node, and there are at most 2^k such paths). In turn, this implies that the computation of v from its pebbled ancestors can be viewed as a 2^k-ary gate which, using [GMW87, DNNR17], can be evaluated with

one call to a 1-out-of-2^{2^k} OT functionality. Summarising, we have the following lemma:

Lemma 6. *Let C be a circuit with n inputs, m outputs, and let PG be a valid DR pebbling of the graph G of C with depth parameter k. There exists an information-theoretic secure 2-party protocol for C in the $\binom{2^{2^k}}{1}$-OT-hybrid model which makes at most $m + \mathsf{cost}(\mathsf{PG})$ to the OT functionality, and requires no further communication.*

4 Depth-Reduction Algorithms for Fan-In Two Circuits

In this section, we present depth-reduction algorithms for (the underlying DAG of) fan-in 2 circuits.

In Sect. 4.1, we provide a conservative depth-reduction algorithm for *any* fan-in two circuit which removes only a *sublinear* number of nodes. However, because the reduction in depth is only sub-polynomial, we can only reach (doubly) logarithmic depth if the starting circuit is already shallow.

In Sect. 4.2 we provide extreme depth-reduction algorithms, reducing *any* in-degree-2 circuit's depth to a constant, while removing a constant fraction of the vertices.

In Sect. 4.3 we exclude the existence of a "best of both worlds" result, by establishing there are high-depth circuits whose depth cannot be reduced polynomially without removing a linear number of nodes.

In Sect. 4.4 we list the implications for secure multiparty computation.

4.1 Depth-Reduction of Low-Depth Circuits

Valiant [Val77, Theorem 5.1] (recalled in this section as Theorem 1) established that the depth of *any* circuit can be reduced sub-polynomially (*i.e.* the depth goes from d to $d^{1-o(1)}$, thereby saving the sub-polynomial factor $d^{o(1)}$) by the removal of only a sublinear number of vertices.

Theorem 1 (Subpolynomial depth-reduction for all circuits, Immediate Corollary of [Val77, Theorem 5.1]). *Let G be an in-degree-2, depth-d, n-vertex DAG. For every $k \leq d$, there exists a subset of $\mathcal{O}(n \cdot (1 - \frac{\log k}{\log d}))$ vertices whose removal yields a depth-k DAG.*

Proof. [Val77, Theorem 5.1] states that the smallest in-degree-2, depth-d DAG whose depth cannot be reduced to k by removing ℓ *edges* has order at least $(\ell \cdot \log d)/(\log(d/k))$. It follows that every in-degree-2, depth-d DAG on n vertices can have its depth reduced to k by removing ℓ *edges* if the following inequality holds: $n \leq (\ell \cdot \log d)/(\log(d/k))$. By setting $\ell \leftarrow n \cdot (1 - \frac{\log k}{\log d})$ and noting that removing a giving set of k edges can also be done by removing k nodes, we get the desired result. □

To better understand the trade-off between depth-reduction and number of nodes removed in Theorem 1, it may be instructive to introduce the variable change $\kappa \leftarrow \log(d/k)$ and observe that the theorem can be restated as:

> Let G be an in-degree-2, depth-d, n-vertex DAG. For every $\kappa \leq \log d$, there exists a subset of $\mathcal{O}(n \cdot \kappa / \log d)$ vertices whose removal yields a depth-$(d/2^{\kappa})$ DAG.

It should now be apparent that if we are only willing to remove $o(n)$ nodes, then we need to set $\kappa = o(\log d)$, which means that the depth of the DAG will only be reduced to $d^{1-o(1)}$. If this quantity is to be logarithmic or even doubly logarithmic in n (as is required for some applications of Sect. 3.3), the result can only be applied to circuits which are already low-depth. We state the result for low-depth circuits in Corollary 3.

Corollary 3 (Depth-reduction of low-depth circuits, Adapted from [Val77, Corollary 5.3]). *Let G be an in-degree-2, n-vertex DAG of depth $\log^{1+o(1)} n$ (resp. $(\log \log n)^{1+o(1)}$). There exists a subset of $o(n)$ vertices whose removal yields a DAG of depth $\mathcal{O}(\log n)$ (resp. $\mathcal{O}(\log \log n)$).*

4.2 Depth-Reduction of General Circuits

In this section we show how removing a constant fraction of the vertices can reduce the depth of an in-degree-2 DAG all the way down to a constant.

4.2.1 Reduction to Depth $k = 1$ Based on 3-Colouring. Our first solution removes a fraction 2/3 of the vertices in order to reduce the depth to 1.

Theorem 2 (Colouring-based depth reduction). *Any in-degree-in 2, n-vertex DAG admits a subset of $\lfloor \frac{2n}{3} \rfloor$ vertices whose removal yields a depth-1 DAG (*i.e. *an independent set).*

Proof. An independent set in a DAG is the same thing as an independent set of the underlying (undirected) graph. Recall that the underlying graph of an in-degree-in 2 DAG is 2-degenerate. A 2-degenerate graph is 3-colourable [Mat68, LW70] and furthermore a 3-colouring can be found greedily in polynomial time: colour vertices following a 2-elimination ordering, always assigning the smallest available colour (by definition of a 2-elimination ordering, whenever we colour a vertex, at most two of its neighbours have already been assigned a colour, so the greedy algorithm will never be stuck and will never need to use more than three colours). The vertices are now partitioned into three colours, and removing the two smallest partitions (this union has size at most $\lfloor 2n/3 \rfloor$) yields an independent set.

Note that Theorem 2 is tight in the sense that there exist n-vertex in-degree-2 DAGs whose independence number (*i.e.* the size of its maximum independent set) is $n - \lfloor 2n/3 \rfloor$.

4.2.2 Reduction to Depth $k \geq 1$ Based on Feedback Vertex Set. We now present an alternative solution, which removes a smaller fraction than 2/3 of the vertices, but at the cost of reducing the depth to "only" a constant, not one. The algorithm first removes a feedback vertex set, and then proceeds to remove vertices from the resulting forest.

Depth-Reduction of Forests. The first observation is that forests can be reduced to depth k by removing a fraction $1/(k+1)$ of its vertices.

Algorithm Depth-Reduction for Forests

On input a directed forest $G = (V, E)$ and an integer k, $\mathsf{DR}_{\mathsf{forest}}(\cdot, \cdot)$ does the following:

1. Let D be the depth of G.
2. If $D \leq k$ return $\varnothing$.
3. Else:
   (a) Let u_0 be a depth-D vertex, and for $i \in [k]$ let u_i be its ancestor at depth $D - i$.
   (b) Return $\{u_k\} \cup \mathsf{DR}_{\mathsf{forest}}(k, V \setminus \{u_i\}_{i \in [0,k]})$.

Fig. 6. Algorithm which, on input an n-vertex directed forest and an integer k, produces a set of at most $\lfloor \frac{n}{k+1} \rfloor$ vertices whose removal yields a DAG of depth at most k.

Lemma 7 (Depth-reduction algorithm for directed forests). *Let $k \in \mathbb{N}^\star$. Every n-vertex directed forest admits a set of $\lfloor \frac{n}{k+1} \rfloor$ vertices whose removal yields a depth-k DAG. Furthermore the (deterministic) algorithm of Fig. 6 finds such a set in polynomial time.*

Proof. The fact that $\mathsf{DR}_{\mathsf{forest}}$ runs in polynomial-time follows from inspection. Let us show by induction on $n \in \mathbb{N}^\star$ that for every n-vertex directed forest $G = (V, E)$ and every integer $k \geq 1$, $\mathsf{DR}_{\mathsf{forest}}(G, k)$ outputs a set $S \subseteq V$ of size at most $\lfloor \frac{n}{k+1} \rfloor$ such that $G[V \setminus S]$ has depth at most k.

- *Initialisation:* Let $k \geq 1$. Any graph G with a single vertex has depth 1, therefore $\mathsf{DR}_{\mathsf{forest}}(G, k)$ therefore returns $\emptyset$, and the claim is true.
- *Induction Step:* Let $n \in \mathbb{N}^\star$, and assume the induction hypothesis is true from ranks 1 to n. Let $G = (V, E)$ be an $(n+1)$-vertex forest and let $k \in \mathbb{N}^\star$. If G has depth at most k, then the claim is trivially true. If G has depth more than k, then in particular $n \geq k + 1$. Furthermore, by induction hypothesis $\mathsf{DR}_{\mathsf{forest}}(G, k)$ outputs a set of size at most $1 + \lfloor \frac{n-(k+1)}{k+1} \rfloor = \lfloor \frac{n}{k+1} \rfloor$. □

Note that the algorithm of Fig. 6 is optimal in the sense that for n-vertex directed paths, the smallest set whose removal yields a graph of depth at most k has size $\lfloor \frac{n}{k+1} \rfloor$.

From Forest Depth-Reduction to Circuit Depth-Reduction. The second observation is that finding an FVS reduces the depth-reduction problem from in-degree-2 DAGs to directed forests.

Lemma 8 (FVS-based depth reduction). *Let G be an n-vertex DAG, and denote G' the underlying (undirected) graph of G. If G' has a feedback vertex set of size f, then G admits a set of $\lfloor \frac{n-f}{k+1} \rfloor$ vertices whose removal yields a depth-k DAG.*

Proof. By definition of a feedback vertex set, removing a size-f FVS from G' yields an $(n-f)$-vertex forest. Evidently, removing the same size-f vertex set from G yields an $(n-f)$-vertex directed forest. The desired result follows from applying Lemma 7: removing an additional $\lfloor \frac{n-f}{k+1} \rfloor$ vertices from G yields a graph of depth at most k. □

Lemma 9 (A feedback vertex set for all 2-degenerate graphs, [BDBS14, Theorems 2, 3]). *Every 2-degenerate (undirected) graph on n vertices admits a feedback vertex set of size at most $\lfloor 2n/5 \rfloor$, and furthermore such an FVS can be found in polynomial time.*

Note that Lemma 9 is tight in the sense that there exist n-vertex graphs whose smallest feedback vertex set has size $\lfloor 2n/5 \rfloor$ [BDBS14, Theorem 4].

Wrapping-up. We are now ready to conclude by combining Lemma 8 and 9.

Theorem 3 (FVS-based depth reduction). *Let $k \geq 1$. Any in-degree-2, n-vertex DAG admits a subset of $\frac{2n}{5} \cdot (1 + \frac{3/2}{k+1})$ vertices whose removal yields a depth-k di-graph.*

Proof. Let G be an in-degree-2, n-vertex DAG. The underlying (undirected) graph of G is 2-degenerate, therefore by combining Lemma 8 and 9, G admits a subset of $\lfloor \frac{2n}{5} \rfloor + \lfloor \frac{n-\lfloor 2n/5 \rfloor}{k+1} \rfloor$ vertices whose removal yields a depth-k graph. Since $(\frac{2n}{5} \cdot (1 + \frac{3/2}{k+1})) - (\lfloor \frac{2n}{5} \rfloor + \lfloor \frac{n-\lfloor 2n/5 \rfloor}{k+1} \rfloor) \geq (\frac{2n}{5} \cdot (1 + \frac{3/2}{k+1})) - (\lfloor \frac{2n}{5} \rfloor + \frac{n-\lfloor 2n/5 \rfloor}{k+1}) = (\frac{2n}{5} - \lfloor \frac{2n}{5} \rfloor) \cdot \frac{k}{k+1} \geq 0$, the simplified expression stated in the theorem is also correct. □

4.3 Lower Bounds on Depth-Reduction

The depth-reduction algorithm of Sect. 4.1 is better in the sense it removes only a sublinear number of nodes, while those of Sect. 4.2 are better in the sense they drastically reduce the depth of the circuit. One may wonder if it is possible to improve on this result, and reduce the depth of any circuit polynomially (*i.e.* from d to d^ϵ for some constant $0 \leq \epsilon < 1$) while still only removing a sublinear number of vertices. Unfortunately, Schnitger [Sch83, Theorem A] showed that the sub-polynomial limitation on depth-reduction was inherent by producing family of *constant fan-in* circuits whose depth cannot be reduced polynomially without removing a linear fraction of vertices. Alwen, Blocki, and Pietrzak [ABP17] later

introduced a technique to reduce a circuit family's fan-in from δ to 2 while preserving its depth-robustness up to a factor δ. We state the combined result in Theorem 4.

Theorem 4 (Polynomial depth-reduction requires removing a linear number of vertices in some graphs, Combination of [Sch83, Theorem A] and [ABP17, Lemma 1]). *For each ϵ ($0 \leq \epsilon < 1$), there is a family of in-degree-2 DAGs $(G_{n,\epsilon})_{n\in\mathbb{N}^\star}$ such that $G_{n,\epsilon}$ has $\mathcal{O}(n)$ vertices, but $\Omega(n)$ vertices have to be removed to reduce its depth to $(n/\log n)^\epsilon$.*

Proof. Let $\epsilon \in [0, 1)$. By [Sch83, Theorem A], there is a family of *constant* in-degree DAGs $(\tilde{G}_{n,\epsilon})_{n\in\mathbb{N}^\star}$ such that $\tilde{G}_{n,\epsilon}$ has n vertices, but $\Omega(n)$ vertices have to be removed to reduce its depth to $(n/\log n)^\epsilon$. Let δ be the in-degree of $(\tilde{G}_{n,\epsilon})_{n\in\mathbb{N}^\star}$. Applying [ABP17, Lemma 1] (with $\gamma = 1$) yields the desired result. □

4.4 Applications to Cryptography

We conclude by combining the results of Sect. 3.3, which re-casts cryptographic results in the lens of our pebbling game, and the depth-reduction algorithms of Sects. 4.1 and 4.2.

By combining Lemma 2 and Theorem 2 we obtain Corollary 4.

Corollary 4 (Fractionally linear-communication MPC in the correlated randomness model, Concrete Result). *Let C be an n-input, m-output, depth-d, $\mathbb{F}$-arithmetic circuit with s non-output computation gates. There exists a passive, perfectly secure N-party protocol for securely computing C in the correlated randomness model, using the following resources (in bits):*

- Correlated randomness per party: $((11 + 2/3) \cdot s + n + 16 \cdot m) \cdot N \cdot \log|\mathbb{F}|$
- Total communication: $(\frac{2}{3}s + n + m) \cdot N \cdot \log|\mathbb{F}|$
- Local computation per party: $\mathcal{O}((\frac{2s}{3} + m) \cdot (N + \log|\mathbb{F}|))$

If instead we combine Lemma 2 and Theorem 3 we obtain Corollary 5.

Corollary 5 (Fractionally linear-communication MPC in the correlated randomness model, Asymptotic Result). *Let C be an n-input, m-output, depth-d, $\mathbb{F}$-arithmetic circuit with s non-output computation gates. For every $\epsilon \geq \frac{3}{2\log\log s}$, there exists a passive, perfectly secure N-party protocol for securely computing C in the correlated randomness model, using the following resources (in bits):*

- Correlated randomness per party: $n + 2^{\sqrt{8}^{1/\epsilon}}(s + m)N \cdot \log|\mathbb{F}|$
- Total communication: $((1 + \epsilon)\frac{2}{5}s + n + m) \cdot N \cdot \log|\mathbb{F}|$
- Local computation per party: $\mathcal{O}(2^{\sqrt{8}^{1/\epsilon}}((1 + \epsilon)\frac{2}{5}s + m) \cdot (N + \log^2|\mathbb{F}|))$

By combining Lemma 3 and Theorem 1 we get Corollary 6. To clarify, the quantification in Corollary 6 is as follows: for every infinite family of circuits $(C_\lambda)_{\lambda\in\mathbb{N}}$ of size $s = s(\lambda)$ such that there exists a vanishing function $\alpha(\cdot) \in o(1)$ such that the depth of C_λ is at most $\log^{1+\alpha(s(\lambda))}(s(\lambda))$ (resp. $[\log\log(s(\lambda))]^{1+\alpha(s(\lambda))}$), there exists a protocol for securely computing C_λ assuming HSS support logarithmic (resp. doubly logarithmic) depth homomorphic evaluations.

Corollary 6 (Sub-polynomially deeper HSS-based sublinear-communication secure computation). *Assuming the existence of N-party homomorphic secret-sharing supporting logarithmic depth (respectively doubly logarithmic depth circuits) $\mathbb{F}$-arithmetic fan-in two circuits, there exists sublinear-communication secure N-party computation for* all $\log^{1+o(1)}$*-depth (resp.* $(\log\log)^{1+o(1)}$*-depth) circuits.*

By combining Lemma 5 and Theorem 1 we obtain Corollary 7, establishing an upper bound on the number of bootstraps required in FHE, given a maximum noise level.

Corollary 7 (FHE-based secure computation with reduced bootstrapping, informal). *Given an FHE scheme tolerating a maximum noise level of L, only $\frac{2}{5}s \cdot (1 + \frac{1.5}{L+1})$ bootstraps are required to homomorphically evaluate a size-s circuit (over an arbitrary basis of binary gates).*

Corollary 8 (Upper bounds on the 1-*out-of*-M *OT*-complexity of secure multiparty computation). *Let $f\colon \{0,1\}^n \to \{0,1\}$ be a boolean function. For every $M \geq 2$, there is a two-party protocol for passively securely computing f (we assume the parties initially hold shares of the inputs, but this captures the case where each input is held by one of the parties) with perfect security in the $\binom{M}{1}$-OT hybrid model while making*

$$1 + \frac{2}{5}|f| \cdot \left(1 + \frac{3/2}{1 + \lceil \log\log M \rceil}\right)$$

calls to the OT functionality, where $|f|$ is the computational complexity of f with respect to the basis of all binary boolean gates.

Acknowledgments. We thank Mikaël Rabie and Elette Boyle for helpful discussions and pointers to respectively degenerate graphs and memory-hard functions.

Geoffroy Couteau was supported by the French Agence Nationale de la Recherche (ANR), under grant ANR-20-CE39-0001 (project SCENE), and by the France 2030 ANR Project ANR22-PECY-003 SecureCompute. Pierre Meyer was supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme under grant agreements numbers 852952 (HSS) and 803096 (SPEC).

References

AB16. Alwen, J., Blocki, J.: Efficiently computing data-independent memory-hard functions. In: Robshaw, M., Katz, J. (eds.) CRYPTO 2016. LNCS, Part II, vol. 9815, pp. 241–271. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53008-5_9

ABH17. Alwen, J., Blocki, J., Harsha, B: Practical graphs for optimal side-channel resistant memory-hard functions. In: Thuraisingham, B.M., Evans, D., Malkin, T., Xu, D. (eds.) ACM CCS 2017, pp. 1001–1017. ACM Press, October/November 2017

ABP17. Alwen, J., Blocki, J., Pietrzak, K.: Depth-robust graphs and their cumulative memory complexity. In: Coron, J.-S., Nielsen, J.B. (eds.) EUROCRYPT 2017. LNCS, Part III, vol. 10212, pp. 3–32. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-56617-7_1

ADOS22. Abram, D., Damgård, I., Orlandi, C., Scholl, P.: An algebraic framework for silent preprocessing with trustless setup and active security. In: Dodis, Y., Shrimpton, T. (eds.) CRYPTO 2022. LNCS, Part IV, vol. 13510, pp. 421–452 (2022). Springer, Cham. https://doi.org/10.1007/978-3-031-15985-5_15

AJL+12. Asharov, G., Jain, A., López-Alt, A., Tromer, E., Vaikuntanathan, V., Wichs, D.: Multiparty computation with low communication, computation and interaction via threshold FHE. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 483–501. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-29011-4_29

ARS24. Abram, D., Roy, L., Scholl, P.: Succinct homomorphic secret sharing. In: Joye, M., Leander, G. (eds.) EUROCRYPT 2024. LNCS, vol. 14656, pp. 301–330. Springer, Cham (2024). https://doi.org/10.1007/978-3-031-58751-1_11

BCM22. Boyle, E., Couteau, G., Meyer, P.: Sublinear secure computation from new assumptions. In: Kiltz, E., Vaikuntanathan, V. (eds.) TCC 2022. LNCS, Part II, vol. 13748, pp. 121–150. Springer, Cham. (2022). https://doi.org/10.1007/978-3-031-22365-5_5

BCM23. Boyle, E., Couteau, G., Meyer, P.: Sublinear-communication secure multiparty computation does not require FHE. In: Hazay, C., Stam, M. (eds.) EUROCRYPT 2023. LNCS, Part II, vol. 14005, pp. 159–189. Springer, Cham (2023). https://doi.org/10.1007/978-3-031-30617-4_6

BDBS14. Borowiecki, M., Drgas-Burchardt, E., Sidorowicz, E.: A feedback vertex set of 2-degenerate graphs. Theor. Comput. Sci. **557**, 50–58 (2014)

Bea92. Beaver, D.: Efficient multiparty protocols using circuit randomization. In: Feigenbaum, J. (ed.) CRYPTO 1991. LNCS, vol. 576, pp. 420–432. Springer, Heidelberg (1992). https://doi.org/10.1007/3-540-46766-1_34

BFKR91. Beaver, D., Feigenbaum, J., Kilian, J., Rogaway, P.: Security with low communication overhead. In: Menezes, A.J., Vanstone, S.A. (eds.) CRYPTO 1990. LNCS, vol. 537, pp. 62–76. Springer, Heidelberg (1991). https://doi.org/10.1007/3-540-38424-3_5

BGI16a. Boyle, E., Gilboa, N., Ishai, Y.: Breaking the circuit size barrier for secure computation under DDH. In: Robshaw, M., Katz, J. (eds.) CRYPTO 2016. LNCS, Part I, vol. 9814, pp. 509–539. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53018-4_19

BGI16b. Boyle, E., Gilboa, N., Ishai, Y.: Function secret sharing: improvements and extensions. In: Weippl, E.R., Katzenbeisser, S., Kruegel, C., Myers, A.C., Halevi, S. (eds.) ACM CCS 2016, pp. 1292–1303. ACM Press, October 2016

BGV12. Brakerski, Z., Gentry, C., Vaikuntanathan, V.: (Leveled) fully homomorphic encryption without bootstrapping. In: Goldwasser, S. (ed.) ITCS 2012, pp. 309–325. ACM, January 2012

BGW88. Ben-Or, M., Goldwasser, S., Wigderson, A.: Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In: 20th ACM STOC, pp. 1–10. ACM Press, May 1988

BIKK14. Beimel, A., Ishai, Y., Kumaresan, R., Kushilevitz, E.: On the cryptographic complexity of the worst functions. In: Lindell, Y. (ed.) TCC 2014. LNCS, vol. 8349, pp. 317–342. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-642-54242-8_14

BKKS11. Brešar, B., Kardoš, F., Katrenič, J., Semanišin, G.: Minimum k-path vertex cover. Discrete Appl. Math. **159**(12), 1189–1195 (2011)

BKS19. Boyle, E., Kohl, L., Scholl, P.: Homomorphic secret sharing from lattices without FHE. In: Ishai, Y., Rijmen, V. (eds.) EUROCRYPT 2019. LNCS, Part II, vol. 11477, pp. 3–33. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-17656-3_1

BLMZ17. Benhamouda, F., Lepoint, T., Mathieu, C., Zhou, H.: Optimization of bootstrapping in circuits. In: Klein, P.N. (ed.) 28th SODA, pp. 2423–2433. ACM-SIAM, January 2017

BV11. Brakerski, Z., Vaikuntanathan, V.: Efficient fully homomorphic encryption from (standard) LWE. In: Ostrovsky, R. (ed.) 52nd FOCS, pp. 97–106. IEEE Computer Society Press, October 2011

CCD88. Chaum, D., Crépeau, C., Damgård, I.: Multiparty unconditionally secure protocols (extended abstract). In: 20th ACM STOC, pp. 11–19. ACM Press, May 1988

CE91. Chrobak, M., Eppstein, D.: Planar orientations with low out-degree and compaction of adjacency matrices. Theor. Comput. Sci. **86**(2), 243–266 (1991)

CM21. Couteau, G., Meyer, P.: Breaking the circuit size barrier for secure computation under quasi-polynomial LPN. In: Canteaut, A., Standaert, F.-X. (eds.) EUROCRYPT 2021. LNCS, Part II, vol. 12697, pp. 842–870. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-77886-6_29

COS+22. Chillotti, I., Orsini, E., Scholl, P., Smart, N.P., Van Leeuwen, B.: Scooby: improved multi-party homomorphic secret sharing based on FHE. In: SCN 2022 (2022). https://eprint.iacr.org/2022/862

Cou19. Couteau, G.: A note on the communication complexity of multiparty computation in the correlated randomness model. In: Ishai, Y., Rijmen, V. (eds.) EUROCRYPT 2019. LNCS, Part II, vol. 11477, pp. 473–503. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-17656-3_17

DFH12. Damgård, I., Faust, S., Hazay, C.: Secure two-party computation with low communication. In: Cramer, R. (ed.) TCC 2012. LNCS, vol. 7194, pp. 54–74. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-28914-9_4

DIJL23. Dao, Q., Ishai, Y., Jain, A., Lin, H.: Multi-party homomorphic secret sharing and sublinear MPC from sparse LPN. In: Handschuh, H., Lysyanskaya, A. (eds.) CRYPTO 2023. LNCS, vol. 14082, pp. 315–348. Springer, Cham (2023). https://doi.org/10.1007/978-3-031-38545-2_11

DNNR17. Damgård, I., Nielsen, J.B., Nielsen, M., Ranellucci, S.: The TinyTable protocol for 2-party secure computation, or: gate-scrambling revisited. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017. LNCS, Part I, vol. 10401, pp. 167–187. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63688-7_6

EH66. Erdös, P., Hajnal, A.: On chromatic number of graphs and set-systems. Acta Mathematica Academiae Scientiarum Hungarica **17**, 61–99 (1966)

FGJS17. Fazio, N., Gennaro, R., Jafarikhah, T., Skeith, W.E.: Homomorphic secret sharing from Paillier encryption. In: Okamoto, T., Yu, Y., Au, M.H., Li, Y. (eds.) ProvSec 2017. LNCS, vol. 10592, pp. 381–399. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-68637-0_23

Gen09. Gentry, C.: Fully homomorphic encryption using ideal lattices. In: Mitzenmacher, M. (ed.) 41st ACM STOC, pp. 169–178. ACM Press, May/June 2009

GJ11. Gál, A., Jang, J.-T.: The size and depth of layered Boolean circuits. Inf. Process. Lett. **111**(5), 213–217 (2011)

GMW86. Goldreich, O., Micali, S., Wigderson, A.: Proofs that yield nothing but their validity and a methodology of cryptographic protocol design (extended abstract). In: 27th FOCS, pp. 174–187. IEEE Computer Society Press, October 1986

GMW87. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game or a completeness theorem for protocols with honest majority. In: Aho, A. (ed.) 19th ACM STOC, pp. 218–229. ACM Press, May 1987

IKM+13. Ishai, Y., Kushilevitz, E., Meldgaard, S., Orlandi, C., Paskin-Cherniavsky, A.: On the power of correlated randomness in secure computation. In: Sahai, A. (ed.) TCC 2013. LNCS, vol. 7785, pp. 600–620. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-36594-2_34

Kil92. Kilian, J.: A note on efficient zero-knowledge proofs and arguments (extended abstract). In: 24th ACM STOC, pp. 723–732. ACM Press, May 1992

LP13. Lepoint, T., Paillier, P.: On the minimal number of bootstrappings in homomorphic circuits. In: Adams, A.A., Brenner, M., Smith, M. (eds.) FC 2013. LNCS, vol. 7862, pp. 189–200. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-41320-9_13

LW70. Lick, D.R., White, A.T.: k-degenerate graphs. Can. J. Math. **22**(5), 1082–1096 (1970)

Mat68. Matula, D.W.: A min-max theorem for graphs with application to graph coloring. In: SIAM 1968 National Meeting, SIAM Review, vol. 10, no. 4, pp. 481–482 (1968)

Nau09. Naumann, U.: Dag reversal is NP-complete. J. Discrete Algorithms **7**(4), 402–410 (2009)

OSY21. Orlandi, C., Scholl, P., Yakoubov, S.: The rise of Paillier: homomorphic secret sharing and public-key silent OT. In: Canteaut, A., Standaert, F.-X. (eds.) EUROCRYPT 2021. LNCS, Part I, vol. 12696, pp. 678–708. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-77870-5_24

PV16. Paindavoine, M., Vialla, B.: Minimizing the number of bootstrappings in fully homomorphic encryption. In: Dunkelman, O., Keliher, L. (eds.) SAC 2015. LNCS, vol. 9566, pp. 25–43. Springer, Heidelberg (2016)

RS21. Roy, L., Singh, J.: Large message homomorphic secret sharing from DCR and applications. In: Malkin, T., Peikert, C. (eds.) CRYPTO 2021. LNCS,

Part III, vol. 12827, pp. 687–717. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-84252-9_23

Sch83. Schnitger, G.: On depth-reduction and grates. In: 24th Annual Symposium on Foundations of Computer Science (SFCS 1983), pp. 323–328 (1983)

Tu22. Tu, J.: A survey on the k-path vertex cover problem. Axioms **11**(5) (2022)

Val77. Valiant, L.G.: Graph-theoretic arguments in low-level complexity. In: Gruska, J. (ed.) MFCS 1977. LNCS, vol. 53, pp. 162–176. Springer, Heidelberg (1977). https://doi.org/10.1007/3-540-08353-7_135

Yao86. Yao, A.C.-C.: How to generate and exchange secrets (extended abstract). In:27th FOCS, pp. 162–167. IEEE Computer Society Press, October 1986

YZW+19. Yu, Yu., Zhang, J., Weng, J., Guo, C., Li, X.: Collision resistant hashing from sub-exponential learning parity with noise. In: Galbraith, S.D., Moriai, S. (eds.) ASIACRYPT 2019. LNCS, Part II, vol. 11922, pp. 3–24. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-34621-8_1

General Adversary Structures in Byzantine Agreement and Multi-party Computation with Active and Omission Corruption

Konstantinos Brazitikos[1(✉)] and Vassilis Zikas[2]

[1] University of Edinburgh, Edinburgh, U.K.
K.Brazitikos@sms.ed.ac.uk
[2] Purdue University, West Lafayette, USA
vzikas@cs.purdue.edu

Abstract. Typical results in multi-party computation (in short, MPC) capture faulty parties by assuming a threshold adversary corrupting parties actively and/or fail-corrupting. These corruption types are, however, inadequate for capturing correct parties that might suffer temporary network failures and/or localized faults—these are particularly relevant for MPC over large, global scale networks. Omission faults and general adversary structures have been proposed as more suitable alternatives. However, to date, there is no characterization of the feasibility landscape combining the above ramifications of fault types and patterns.

In this work we provide a tight characterization of feasibility of MPC in the presence of general adversaries—characterized by an adversary structure—that combine omission and active corruption. To this front we first provide a tight characterization of feasibility for Byzantine agreement (BA), a key tool in MPC protocols—this BA result can be of its own separate significance. Subsequently, we demonstrate that the common techniques employed in the threshold MPC literature to deal with omission corruptions do not work in the general adversary setting, not even for proving bounds that would appear straightforward, e.g., sufficiency of the well known Q^3 condition on omission-only general adversaries. Nevertheless we provide a new protocol that implements general adversary MPC under a surprisingly complex, yet tight as we prove, bound. All our results are for the classical synchronous model of computation.

As a contribution of independent interest, our work puts forth, for the first time, a formal treatment of general-adversary MPC with (active and) omission corruptions in Canetti's universal composition framework.

1 Introduction

Multi-party computation (MPC) enables n parties to securely compute a function on their joint input. To capture parties' misbehavior one typically considers

The full version of this paper can be found at the IACR Cryptology ePrint Archive, report 2024/209.

E. Boyle and M. Mahmoody (Eds.): TCC 2024, LNCS 15367, pp. 200–233, 2025.
https://doi.org/10.1007/978-3-031-78023-3_7

a central adversary corrupting parties and using them to attack the protocol. The most common corruption type for such an adversary is *active* corruption—the adversary takes full control of a corrupted party. Security against such an active adversary offers strong guarantees, but allowing the adversary to take full control of corrupted parties is an overkill to capture more benign types of misbehavior or just faults. In fact, this typically yields restrictions both in the feasibility—e.g., tolerable number of corruptions—and in terms of efficiency. As a result, different types of corruption have been investigated to capture such benign faults scenarios.

In the opposite extreme of active corruption, *fail-corruption* (aka fail-crash corruption or fail-stop corruption) allows the adversary to make a party crash (irrevocably) at any point of the protocol he chooses—without having knowledge of the party's internal state.[1] Naturally, adversaries with fail-corruption allow for better feasibility and efficiency bounds than active adversaries, but this corruption type is often criticized as too benign. As an example, fail-corruption is too weak for capturing faults caused by temporary issues on the network of otherwise *correct*—i.e., protocol abiding—parties. This gave raise to the study of the so called *omission-corruption*, which allows the adversary to selectively drop incoming and/or outgoing messages of the corrupted party, but obliviously of the message contents or the party's internal state.

On a different dimension, the two standard ways to capture the adversary's corruption patterns are via *threshold* and *general* adversaries. A threshold adversary is specified by the maximum number (threshold) of possible corruptions. This model can, again, be considered overly pessimistic, and therefore restrictive, when one considers situations in which certain combinations of faulty parties are unlikely. The concept of a *general adversary (structure)* is the alternative, fine-grained way which better captures such a situation: Rather than the maximum number of corruptions, a general adversary structure $\mathcal{Z}$ enumerates all possible combinations of corrupted parties, therefore giving more flexibility in describing the adversary's capabilities.

Tight feasibility bounds have been established for both threshold and general adversaries in the context of active corruptions and fail-corruptions, and even their combination (see Sect. 1.1 below for an overview). However, to our knowledge, omission corruption has not been considered for general adversaries, neither in isolation nor in conjunction with active corruption. Furthermore, all work on omission corruptions or general adversaries uses the property-based security definition of MPC, as opposed to simulation based security which is not only more general but, as we discuss is needed for completing the proofs of these works that rely on composing smartly designed sub-protocols. In a nutshell, our work provides the first characterization of the feasibility landscape

[1] In the distributed computing literature, omission-corrupted parties are often considered also semi-honest. This is suitable for classical distributed computing tasks, e.g., Byzantine agreement (see below), where input privacy is a lesser issue. However, since here we are interested in MPC, we will follow the cryptographic convention of considering it separate.

of Byzantine agreement (BA)—the core primitive in fault-tolerant distributed computation and standard building block of MPC—and of secure multi-party computation (MPC) for general adversaries that might corrupt some parties actively (i.e., force byzantine faults) and, simultaneously, omission corrupt other parties. Concretely, we prove a tight feasibility bound for both synchronous consensus and broadcast in the perfect (security) setting, i.e., information theoretic security with zero error probability. We then turn to the study of MPC in this model. As we show (see discussion of MPC results in Sect. 2), translating threshold bounds to this setting is far from trivial—this reaffirms what the complex bounds of Beerliova *et al.* [5] demonstrate for the active/passive/fail case. Furthermore, existing arguments and techniques from the cryptographic literature are inadequate for proving even what one would consider a simple and intuitive feasibility result. Notwithstanding, we provide a tight feasibility bound for MPC in this setting by developing a new protocol for (publicly) detectable point-to-point secure communication and proving a tight bound for this task. In fact, a look at the complexity in the associated (*tight*) bound (see Eq. 11) serves as a perfect demonstration of the technical challenges associated with devising such a bound, protocol, and associated tightness proof.

Finally, our results are proven secure in a simulation-based composable framework. Although we do not consider this to be our key technical contribution, it is, to our knowledge, a first both for general (mixed) adversary MPC and for MPC with omission corruptions. Our treatment demonstrates the challenges of a composable treatment of omission-faults. Therefore we believe it to be a milestone in the literature which can be of independent interest.

1.1 Related Literature

In this section we discuss the related literature, where we focus on synchronous[2] protocols with perfect security, i.e., with zero error probability, which is also the type of protocols we develop here.

Byzantine Agreement (BA). BA comes in two flavors: consensus and broadcast. In consensus, n parties, each with its own input, wish to agree on a joint output, so that pre-agreement is preserved. In broadcast, only one party, the sender, has input, and the goal is to distribute it in a consistent manner to all parties, so that consistency is achieved even if some of the parties are actively corrupted (cf. Sects.3.6, 3.7). The seminal results by Lamport, Shostak, and Pease [27,32], showed that Consensus and Broadcast are feasible if and only if at most t parties are byzantine, where $t < n/3$. Follow up work has extended the above results to various models capturing different types of synchrony, alternative networks, and setup assumptions such as a public key infrastructure.

Multi-party computation (MPC). In MPC we have n parties from a set $\mathcal{P} = \{p_1, \ldots, p_n\}$, each with a private input x_i who wish to securely compute

[2] We note that the feasibility questions discussed here have not been considered in any other model, e.g., asynchronous or partially synchronous; we consider this an interesting future direction.

a function on their joint input, even in the presence of faulty parties. Faulty parties are captured by assuming a central adversary that corrupts parties and uses them to orchestrate a coordinated attack to break the protocol's security, where the two main security goals are *privacy*—corrupted parties should learn nothing beyond their prescribed inputs and output, and *correctness*—the adversary should not be able to affect the output of the computation in any other way than choosing his own inputs independently of that of uncorrupted parties. The typical type of corruption is active. Actively corrupted parties are often referred to as *malicious* or *byzantine* and the set containing them is denoted as A. MPC was introduced by Yao [35] where feasibility of two-party computation was shown. The seminal works of Ben-Or, Goldwasser, and Wigderson [7] gave the first feasibility results for perfect security (that is, information-theoretic with zero error probability) for a threshold adversary. In particular it was shown that $t < n/3$ is both necessary and sufficient for perfectly secure MPC in the synchronous malicious adversary model.

General Adversary Structures. General adversaries have also been studied for both BA and MPC. Here, for the case of perfect security, Hirt and Maurer [21,22] proved that a necessary and sufficient condition, if no setup[3] is assumed, for a general adversary structure—with active corruptions—to be tolerable is that the union of no three sets in the adversary structure $\mathcal{Z}$ covers the whole player set, a condition which is often referred to as the Q^3 condition:[4]

$$CP^{(A)}_{CONS}(\mathcal{P},\mathcal{Z}) \iff Q^3_A(\mathcal{P},\mathcal{Z}) \iff \forall A_i, A_j, A_k \in \mathcal{Z} : A_i \cup A_j \cup A_k \neq \mathcal{P}. \quad (1)$$

The above tight condition holds for perfectly secure BA (both consensus and broadcast) and MPC. This was later extended to the mixed setting adding fail-corruption faults in [2] and a combination of fail-corruption and passive corruption by Beerliova *et al.* [5] (We refer to [36] for a comprehensive survey of the relevant literature).

The results from [5] offer a first demonstration of the unstranslatability of threshold feasibility results to the general adversary setting. Indeed, in the threshold setting, the active/passive/fail (tight) bound, i.e., $3t_a + 2t_p + t_f < n$ [18], is a simple combination of the corresponding active-only ($3t_a < n$), passive-only ($2t_p < n$), and fail-crash-only ($t_f < n$) bounds. On the other hand, in the general adversary setting, the tight (necessary and sufficient) bound is the combination of the following two conditions (each of them is necessary) [5, Theorem 1]:

$$\forall (A_i,E_i,F_i),(A_j,E_j,F_j),(A_k,E_k,F_k) \in \mathcal{Z} : E_i \cup E_j \cup A_k \cup (F_i \cap F_j \cap F_k) \neq \mathcal{P} \quad (2)$$

and

$$\forall (A_i,E_i,F_i),(A_j,E_j,F_j),(A_k,E_k,F_k) \in \mathcal{Z} : E_i \cup A_j \cup A_k \cup (F_j \cap F_k) \neq \mathcal{P}. \quad (3)$$

[3] Note that "no setup" implies that we cannot use cryptographic tools such as digital signatures.

[4] Here we denote the classical Q^3 condition as Q^3_A to explicitly state that it only applies to active corruptions.

Each of the above triples (A, E, F), so-called adversary *classes*, describes the choice of the adversary specified by this class—namely the parties in A, E, and F, are actively, passively, and fail-corrupted, respectively.

In fact, the inability to translate threshold bounds to general adversaries is further demonstrated by the fact that if one is interested in non-reactive (one-shot) computation of a function, a problem often referred to as *Secure Function Evaluation (SFE)*, then the following *strictly weaker* (and substantially more complex) bound is necessary and sufficient [5, Theorem 2]: The bound from Eq. 2 together with the following condition

$$\exists \text{ an ordering } (A_1, E_1, F_1), ..., (A_m, E_m, F_m) \text{ of the maximal classes in } \mathcal{Z} \text{ s.t.} \\ \forall i, j, k \in \{1, ..., m\}, i \leq k : E_k \cup A_i \cup A_j \cup (F_i \cap F_j) \neq \mathcal{P}. \quad (4)$$

The above results demonstrate the untranslatability of threshold to general adversary results in the active/passive/fail setting. As we show in this work, a similar untranslatability—with even more counter-intuitive phenomena (see Sect. 2 for a discussion)—is evident also in our active/omission corruptions setting.

Omission Faults. The first variant of omissions was introduced to the distributed literature by Hadzilacos [20], where the notion of send-only omissions was introduced. There, and it was proven that $t < n$ (send-)omission faults are necessary and sufficient for BA. Full (send and receive) omission faults were proposed by Perry and Toueg [33], who affirmed the $t < n$ bound for that more general model. A long line of follow-ups investigated the problem. As it can be seen in [1], the recovery of crashed components is often considered a built-in feature of the distributed replication systems, meaning that crash failures are treated in essence as omissions, making omissions appear frequently in the literature.

Importantly, in [20,33], a weaker variant of BA with omissions was considered, where the consistency guarantee was limited to the output of the non-faulty (i.e., uncorrupted/honest) players—meaning that omission-corrupted players were treated as malicious, and were not given any output guarantees. The case where the output of both honest and omission-corrupted players should be guaranteed (whenever possible) was treated by Raynal and Parvedy [31,34] where it was proved that the tight bound on omissions with this requirement becomes $t < n/2$. We note in passing that this latter, more natural and challenging way is also how we treat omissions in this work.

In the cryptographic literature omission faults (also referred to as *omission corruption* and denoted by Ω) were first studied by Koo [26] who proved that for a (mixed-corruption) adversary who can corrupt up to t_a parties actively and omission corrupt up to t_ω parties, $3t_a + 2t_\omega < n$ is sufficient for Consensus and $4t_a + 3t_\omega < n$ is sufficient for MPC. Follow-up work by Hauser, Maurer, and Zikas [37] provided the first tight bounds proving that $3t_a + 2t_\omega < n$ is both necessary and sufficient for BA and MPC in the perfect security (synchronous) setting. The results were extended in [36] by adding fail corruption.

More recently, Eldefrawy, Loss, and Terner [16] investigated computational security for the case where send and receive omission faults have different thresholds t_s and t_r respectively. This case was also treated in [37] but for perfect security only. As demonstrated in [16] the shift to computational security carries unexpected complications, which is yet another indication of the challenges associated with omission-corruption. More concretely, [16] proved that in this setting $t_s + t_r + 2t_b < n$ is sufficient for MPC—where t_b is the threshold on byzantine parties. They also proved this bound tight, albeit for a weaker adversary that performs what they termed "spotty" send-corruptions: messages from a send-(omission-)corrupted player in any round are either all delivered or none of them is.

This lower bound was recently improved by Loss and Stern [29] to cover a worst-case adversary, i.e., without spotty send-omission corruptions. In fact, this seemingly simple generalization required developing novel techniques to deal with omissions, an additional indication of the challenges related to feasibility in the presence of active and omission corruptions.

We note in passing that, although in the threshold setting separating (full) omissions to send-omissions and receive-omissions helps to find tight feasibility bounds [16,29,37], this does not appear to be the case in the general adversary setting. Indeed, splitting omissions this way would complicate the description of the adversary structure—one would need two sets in each class to describe just omissions—and we conjecture this would also yield more complex and less intuitive bounds.

1.2 The Model

We consider n parties from a party set $\mathcal{P} = \{p_1, \ldots, p_n\}$. The parties can communicate via a complete network of bilateral point-to-point secure (i.e., authenticated and private) channels [7]. (We note in passing that our BA protocol does not need privacy and can just rely on standard authenticated channels; however, privacy is necessary for perfectly secure MPC results). We assume synchronous communication as in [7,12,27], i.e., all our protocols advance in rounds; every party is aware of the current round and can send messages to all other parties, where messages sent in any round are delivered to their intended recipients by the beginning of the following round.

For simplicity in the exposition, for protocols that build on top of broadcast we assume that each of their round is a broadcast round (i.e., a round where all parties can broadcast a message). This does not affect composition of the total counting of rounds as our broadcast protocol is deterministic and therefore we do not run into the known issues of probabilistic termination [13]. Furthermore, to make the protocols description simpler we will assume that each sub-protocol has a dedicated *output round* where the parties do not send any messages to each other, but use messages they have received to compute their (sub-)protocol's output(s). This does add a constant overhead on sequentially composing protocols, but makes for a much cleaner abstraction and does not affect the nature of our results which is targeted to feasibility. In fact, one can easily get rid of this

overhead by starting a next sub-protocol already during that output round of the previous one.

Simulation-Based (composable) Security. We prove our protocols secure using the synchronous adaption of Canetti's UC framework [10] put forth by Katz *et al.* [25]. We assume the reader has some familiarity with UC, but we make our best effort to keep the technicalities of the framework insulated from the protocol design and functionality description. In the following we discuss the above synchrony framework and how it is utilized here.

In a nutshell, [25] proposed a methodology for the design/embedding of synchronous protocols within the (by-default asynchronous) UC framework. In this adaptation, protocols can be designed in a synchronous manner, and [25] defines how they can be executed assuming access to a clock functionality, which ensures that (1) all parties get a chance to speak in each round, (2) parties can become aware when the clock round switches. Proving security in such a framework means that the functionalities need to also become round aware; this is taken care of in [25] by adding to the functionality dummy rounds which advance once every party has had a chance to ping the functionality in that round. This allows the environment to advance the ideal experiment if it wishes to, similar to what it can do in the real world. To keep the description cleaner, we abstract away this pinging of functionalities as dummy ("do-nothing") rounds in the functionalities we define, and explicitly make the functionality aware of the underlying (broadcast) round.

To make the two-fold contribution of our work (protocol/proofs level vs. model/UC-treatment level) clearer and isolate the techniques used in each of the two contribution types, we use the following methodology in proving our feasibility results: First we state and prove in separate claims key properties that our (sub-)protocols achieve; this is useful for understanding the protocol ideas that go into the construction and how these are used in the security proof. Then, we use these properties in the simulation proof to obtain our end result. Due to the page constraint, we refer the reader to the full version [9] for proofs and other details.

Adversary. We consider a mix of active corruption and omission-corruption characterized by general adversary structures. Concretely, the possible combinations of corruptions are described by a mixed (active/omission) general adversary structure. Such a structure is a collection $\mathcal{Z}$ of tuples of the type $(A_i, \Omega_i) \in \mathcal{P}^2$, often referred to as *classes*. Intuitively, $\mathcal{Z}$ is intended to capture all possible scenarios of corrupted parties. In particular, a tuple/class $(A_i, \Omega_i) \in \mathcal{Z}$, displays the scenario where all parties in A_i are actively corrupted and all parties in Ω_i are omission-corrupted. We will be using the terminology: *"the adversary corrupts (class)* $Z_i = (A_i, \Omega_i) \in \mathcal{Z}$*"* to refer to the above scenario and we denote it by using a * symbol at the exponent. This means for example that the sets $A^\star$ and $\Omega^\star$ denote the sets of actively corrupted and omission-corrupted players, respectively. Similarly, we refer to an adversary who might corrupt any of the sets in $\mathcal{Z}$ as a (general) $\mathcal{Z}$-adversary. The set of uncorrupted/honest players will be denoted by $\mathcal{H}$. Note that the class $Z^\star$ is not known to the players

and appears only in our security analysis. Furthermore an omission or actively corrupted party might be allowed to send or receive all its messages, in which case he is indistinguishable from an uncorrupted party. We refer to such a party as *correct* at a certain point in time if it was allowed to behave this way (correctly) up until this certain point in time. Essentially, an omission-corrupted party stops being correct the moment its first message is blocked. Finally, some of our protocol executions allow omission-corrupted parties to realize that they are corrupted; when this detection occurs, the party understands that it is in the discretion of the adversary whether or not it will be allowed to contributed inputs or receive outputs in the protocol. Therefore, in such cases the parties step out of the computation and inform all their peers about this decision; borrowing the terminology of [37] we will then say that this party becomes a *zombie*, in contrast to the rest of non-actively-corrupted parties that are considered *alive*.

We will make the following standard conventions on the adversary structure $\mathcal{Z}$: (1) For any $Z_i = (A_i, \Omega_i) \in \mathcal{Z}$, for every $A' \subseteq A_i$ and $\Omega' \subseteq \Omega_i$: $Z' = (A', \Omega') \in \mathcal{Z}$. This captures the intuitive fact that if a set of parties might jointly fail in a certain way, then any subset of them failing is also a possible corruption scenario. This convention allows us to describe $\mathcal{Z}$ by enumerating only its maximal elements. (2) For any $Z_i = (A_i, \Omega_i) \in \mathcal{Z}$ we will assume that $A_i \subseteq \Omega_i$; this is simply capturing the fact that active corruption is strictly more severe (as a misbehavior strategy) than omission and can, behave as such.

Finally, we prove our statements here with respect to *static* adversaries, i.e., the set of corrupted parties (and hence the set of possible corruptions) is decided at the beginning of the protocol and cannot depend on the exchanged messages. We note that all properties we prove here will directly hold to the adaptive security setting without changing the respective bounds [3,11]. However, the simulation-based treatment of adaptive security under parallel composition of, e.g., BA primitives is known to have several thorny issues which are beyond the scope of this submission [14,23].

1.3 Organization of the Paper

The remainder of the paper is organized as follows: Sect. 2 includes an exposition of our results and an overview of the techniques and related challenges. Section 3 includes the details on our tight feasibility results for BA and Sect. 4 our tight feasibility for MPC.

2 Technical Overview

Before diving into the technical part, it is useful to give an overview of our results and the associated techniques and challenges.

Byzantine Agreement. As our first contribution towards our MPC feasibility we prove that the following condition on the adversary structure is necessary and sufficient for perfect synchronous BA, both broadcast and consensus:

$$C_{BA}^{(A,\Omega)}(\mathcal{P},\mathcal{Z}) \iff \forall Z_i, Z_j, Z_k \in \mathcal{Z} : A_i \cup A_j \cup A_k \cup (\Omega_i \cap \Omega_j) \neq \mathcal{P}. \quad (5)$$

Without loss of generality we provide protocols for binary consensus and broadcast, i.e., the inputs and outputs of the protocol are from the field $\mathbb{F} = \{0, 1\}$. This is sufficient for arbitrary valued BA, as we can represent the inputs as bit-strings of appropriate (fixed) length and then we can invoke the bit-Consensus protocol for each of those bits.

Our feasibility result is proven in two stages. First, in Sect. 3.2 we show how to tackle one of the core challenges of omission-corruption, namely detection of dropped messages. In particular, the biggest thorn with omissions is that a party p_j who does not receive a message it expects does not know whether this happened because the sender or itself (p_j) is omission-corrupted. To tackle the above issue, we devise a simple protocol, called *FixReceive*, which allows the receiver to take this decision. We prove (see Lemma 1) that the decision will always be correct as long as the following condition[5] is satisfied

$$C_{FIXR}^{(A,\Omega)}(\mathcal{P},\mathcal{Z}) \iff \forall Z_i, Z_j \in \mathcal{Z} : \Omega_i \cup \Omega_j \neq \mathcal{P}, \quad (6)$$

which is also proven necessary for the above task in Lemma 3.2. One can view *FixReceive* as a way to lift the underlying communication network from a plain one to one with detection. When this detection is successful and a player discovers that he suffers from omissions, he steps down—becomes a zombie—for the rest of the protocol and sends a special message to let others know.

The underlying idea of *FixReceive* is simple, and similar to the corresponding protocol from [37]: the sender sends to all parties, who relay to the receiver; then the receiver tries to "fit" the received messages into the corruption pattern. However, in the threshold case, this "fitting" is rather straightforward. This is in contrast to the general-adversary case, where the right condition (and proof) is more involved. Yet, the above simplicity of *FixReceive* stems from the fact that it makes the transmitted message public to the adversary. This makes it suitable for BA but insufficient for MPC (see below) where we need detection on top of private communication. Looking ahead, this combination turns out to be particularly challenging and the private version of *FixReceive* (which we will call detectable secure message transmission) will be one of the core contributions of our paper.

Let us return to our overview of our BA feasibility result: Having added *FixReceive* to our arsenal, we can now use this to improve the communication properties that are disrupted by omission corruptions. (This can be seen as "lifting" the underlying communication network by adding (partial) corruption awareness/detection.) In particular, having improved the detection ability of communicating parties as above, we proceed to our BA construction. For this, we use the phase-king approach of Berman, Garay, and Perry [8]–which was previously adapted to general adversary structures (with fail-corruption instead

[5] Due to our assumption from earlier, the condition can also be written as $A_i \cup \Omega_j \neq \mathcal{P}$.

of omissions) by Altmann, Fitzi, and Maurer [2]. Concretely, we gradually build protocols with stronger guarantees, from *Weak Consensus* (Sect. 3.3), to *Graded Consensus* (Sect. 3.4), to *King Consensus* (Sect. 3.5), and then iterate through different parties as kings to achieve the consistency and validity conditions of consensus (see Theorem 2).

The above similarity in the structure of our protocol to that from [2] might mislead the reader to believe that the search for the tight BA bound is straightforward given the above result. This is, however, far from true. To demonstrate this, it is useful to discuss the main challenge in shifting from a combination of active corruptions and fail-corruptions (for which we know tight bound both for BA [2] and for MPC [5]) to active and omission corruptions for which nothing is known in the general-adversary setting: The main issue lies in the ability of an omission-corrupting adversary to create confusion by selectively dropping messages to some and not other parties and in some specific rounds. For instance, a standard method in the fail-corruption literature to limit the effect of fail-crashes is to embed a heartbeat after each step (or in selective protocol rounds) that allows parties to detect whether or not some party has already crashed. This approach does not work with omission corruptions, as a party might drop messages during the protocol round, but send all messages in the heartbeat procedure as if nothing happened. Thus one needs to come up with ways to counter the ability of the adversary to create such confusions. The challenge of our above protocol design is to come up with protocols that either allow for public detection of an omission-corrupted party not sending messages, or make the party aware that it is omission-corrupted—in the latter case, this party can put itself in a crashed position (a possibility which the adversary would anyway have by blocking all communication to/from that party) to allow the other parties to complete the protocol.

Having derived a consensus protocol as above, we then turn to broadcast. Interestingly, the standard reduction of broadcast to consensus—i.e., have the sender send his input to everyone and run consensus on the received values—does not work here. The reason is that a send-omission corrupted sender p_s might fail to send his input to some but not all non-actively corrupted parties, in which case consensus might end up flipping his input, which violates our requirement on the output with a non-actively corrupted sender.

We fix this by using an additional round of consensus: To guarantee that an omission-corrupted p_s never broadcasts a wrong value (but he may broadcast $\perp$ in case he is incorrect) we extend the above generic protocol as follows: after running consensus on the received bit, we have p_s send a confirmation bit to every player, i.e., a bit $b = 1$ with the meaning that p_s agrees with his output of the consensus or $b = 0$ otherwise. The players then invoke consensus on the received bit to make sure that they have a consistent view on the confirmation-bit and based on that they accept the output of the generic broadcast protocol only if $b = 1$. In the opposite case, they output $\perp$. This ensures that if they output anything, it will be the correct bit.

Finally, we prove the tightness of $C_{BA}^{(A,\Omega)}(\mathcal{P}, \mathcal{Z})$ for BA by means of a delicate player simulation argument (see Lemma 5).

Multi-party Computation. Having proven a tight characterization of BA in our model, we turn to multi-party computation (MPC). Here we first observe that translating bounds from the threshold literature, or even the existing general adversary literature (i.e., without omission-corruptions) simply does not work. In fact, there is a number of ways that we demonstrate such a translation fails. For example, it is known that in the case of active-only general adversary structures, MPC is feasible if and only if the $Q^3_A(\mathcal{P}, \mathcal{Z})$ condition (Eq. 1) holds [21]. Hence, in search of a feasibility result, one might be tempted to assume that since active corruption is more severe than omission-corruption, the natural adaptation of the above condition to the omission-only setting, i.e., the condition Q^3_Ω

$$CP^{(\Omega)}_{CONS}(\mathcal{P}, \mathcal{Z}) \iff \forall \Omega_i, \Omega_j, \Omega_k \in \mathcal{Z} : \Omega_i \cup \Omega_j \cup \Omega_k \neq \mathcal{P}, \tag{7}$$

would be sufficient for MPC. This however is not necessarily the case, as MPC protocols for active corruptions entirely give up the inputs and outputs of actively corrupted parties, something which we cannot do for omission corruption.

Similarly, drawing intuition from existing impossibility results can derail the search for lower bounds. Indeed, the general rule is that general-adversary impossibility results translate to threshold (though, not always in a trivial manner) but not the other way. Intuitively, the underlying reason is that the asymmetry of general adversary structures allows solutions which could never exist in a threshold setting. This untranslatability becomes ever more prominent when considering omission-corruptions (combined with active), and makes finding the tight condition on general structures for this case a far more challenging task than in the threshold case (in fact, it is challenging even given a tight threshold condition).

As an example, for active/passive adversaries the tight condition $3t_a + 2t_p < n$ [18] was "translated" in [19] to the general adversary setting as:

$$\forall (A_1, E_1), (A_2, E_2), (A_3, E_3) \in \mathcal{Z} : E_1 \cup E_2 \cup A_1 \cup A_2 \cup A_3 \neq \mathcal{P}, \tag{8}$$

(where sets A and E in the above bound correspond to actively and passively corrupted parties, respectively). But an analogous translation for active/omission adversaries of the tight threshold $3t_a + 2t_\omega < n$ [37] as

$$\forall (A_1, \Omega_1), (A_2, \Omega_2), (A_3, \Omega_3) \in \mathcal{Z} : \Omega_1 \cup \Omega_2 \cup A_1 \cup A_2 \cup A_3 \neq \mathcal{P}, \tag{9}$$

does *not* yield a bound necessary for MPC. In fact, in the appendix of the full version of the paper [9] we describe a structure that violates (an even more restrictive version of) the above condition but still allows for an MPC protocol.

In the same spirit, as we show in Sect. 4.1 of the full paper, common techniques used in the threshold MPC literature to recover from corruptions, such as *player elimination* [6], cannot be applied here either. A standard example of player elimination is used in the case of MPC with (threshold) byzantine corruptions with $t < n/3$. The idea is that if some p_i blames another p_j, then, as long as both p_i and p_j have had the chance to share their inputs, we can simply eliminate both of them and continue the computation with the remaining

parties—and send p_i and p_j their outputs at the end; the $t < n/3$ condition will then ensure that in the $n' = n - 2$ remaining parties set, the number t' of maximum active corruptions will still satisfy $t' < n'/3$. In fact, this technique was used in [37] to prove the first, and only to date, tight condition on MPC with active and omission corruption. However, as we show, player elimination is inapplicable in our general-adversary active/omission setting. In particular, we show that natural candidates for feasibility bounds conditions are not preserved by player elimination. This situation calls for new protocols/techniques beyond what is used in the threshold or previous general adversary literature.

To overcome this, we devise a novel protocol that aims at facilitating detectable (i.e., which might abort with the identity of a corrupted party) perfectly secure (private and authenticated) message transmission introduced in [15] (in short, DetSMT) between any two parties. Looking ahead, this will allow to neutralize the effect of omissions in MPC. The challenge in devising and proving security of the new detectable SMT primitive is evident by the new associated condition $C_{SMT}^{(A,\Omega)}(\mathcal{P}, \mathcal{Z}, p_s, p_r)$, which in combination with $C_{BA}^{(A,\Omega)}(\mathcal{P}, \mathcal{Z})$ gives us the condition that is proven to be tight for DetSMT with sender p_s and receiver p_r. The $C_{SMT}^{(A,\Omega)}(\mathcal{P}, \mathcal{Z}, p_s, p_r)$ states that for any three $Z_i, Z_j, Z_k \in \mathcal{Z}$:

$$\begin{aligned} &\text{if } (p_s \in \Omega_i \cap \Omega_j \wedge p_r \in \Omega_k) \text{ OR } (p_r \in \Omega_i \cap \Omega_j \wedge p_s \in \Omega_k) \\ &\text{then } A_i \cup A_j \cup \Omega_k \cup (\Omega_i \cap \Omega_j) \neq \mathcal{P}. \end{aligned} \tag{10}$$

The sufficiency of the above condition for detectable SMT and its necessity for (detectable) SMT (hence also for MPC) are proven in Sect. 4.1 (Lemmas 6 and 7, respectively).

Finally, we put everything together to prove our last main theorem (Theorem 4) of MPC feasibility under the same combination of conditions (where the $C_{SMT}^{(A,\Omega)}(\mathcal{P}, \mathcal{Z}, p_s, p_r)$ needs to hold true for all pairs $p_s, p_r \in \mathcal{P}$). More concretely, we prove that perfectly secure MPC against a general adversary with mixed active and omission corruptions is feasible if and only if the condition $C_{MPC}^{(A,\Omega)}(\mathcal{P}, \mathcal{Z})$ holds, where

$$C_{MPC}^{(A,\Omega)}(\mathcal{P}, \mathcal{Z}) \Leftrightarrow C_{BA}^{(A,\Omega)}(\mathcal{P}, \mathcal{Z}) \wedge \forall p_s, p_r \in \mathcal{P} : \; C_{SMT}^{(A,\Omega)}(\mathcal{P}, \mathcal{Z}, p_s, p_r). \tag{11}$$

The necessity of the above condition follows from the fact that both SMT and broadcast are special cases of MPC. In fact, since both the above are non-reactive functionalities, our impossibilities (along with the feasibility discussed below) imply that $C_{MPC}^{(A,\Omega)}(\mathcal{P}, \mathcal{Z})$ is tight for both (reactive) MPC and for SFE (i.e., non-reactive MPC.) Thus our results prove that unlike the active/passive/fail general adversary model where [5] proved a separation between the (tight feasibility bounds) for MPC and SFE, such a separation does not exist in the active/omission setting.

For the sufficiency we use the following idea: We modify the general adversary protocol from [5] by first projecting it to the active-corruption-only case (recall that [5] works for a mixed active/passive/fail adversary) and then doing the following:

- All point-to-point communication between any two parties p_i and p_j is done by the above detectable SMT.
- All broadcasts are implemented by our detectable broadcast.
- All sub-protocols in [5] for computing individual circuit gates (input, addition, multiplication, and output gates) are turned to detectable counterparts, i.e., they might abort and make the identity of a corrupted party public.
- Importantly, instead of computing the actual circuit, our MPC computes a verifiable secret sharing of the circuit's output—we prove that such a robustly reconstructible sharing is feasible under our conditions. The reason for this is that before its last reconstruction round, the MPC from [5] leaks no information to the adversary. By switching the computation's output to a secret sharing instead of actual circuit value, we ensure that no matter if or when the protocol aborts, it will leak no information on any of the non-actively corrupted player's inputs.

The above construction gives a detectable MPC protocol which either computes a verifiable secret sharing of the output of the intended circuit, or it aborts without leaking any information to the adversary while exposing a corrupted party. Such a protocol can be bootstrapped to a fully secure MPC (with guaranteed output delivery) by standard techniques: Whenever it aborts, remove the detected (corrupted) party from the player set and re-start the computation—this can be repeated at most n times as each abort exposes a new corruption. Once the protocol succeeds, use the reconstruction protocol of the verifiable secret sharing to publicly reconstruct the outputs. We note in passing that the above only computes MPC with a public output, but it can be tuned to allow for private outputs using standard techniques: every party inputs in addition to its actual output a random key which is used to blind—by one-time-pad encryption—the announced public output so that only this party can recover the plaintext [28].

UC Treatment. Last but not least, as discussed above, all our proofs are in the (synchronous) UC framework, which we view as a contribution in its own sense. Although we do not consider this to be our core technical contribution, to our knowledge, this is the first time that a general adversary protocol is proven secure in such a composable manner. In particular, existing MPC protocols for general adversaries [5,21,37] also follow a modular design approach—i.e., they design sub-protocols for each type of MPC gate (input/sharing, addition, multiplication, output/reconstruction)—and prove the security of each underlying sub-protocols separately in a property-based manner, i.e., prove the correctness and privacy of each of these sub-protocols. They then argue that these sub-protocols can be combined in the main MPC protocol. Although we believe this last statement to be true, an actual proof would require a composition proof (which is generally problematic with property-based definitions), or, alternatively a composable treatment of the whole construction, an approach which we take for the first time in this work. In fact, to our knowledge even without considering general adversaries, no work has considered (active and) omission

corruptions in UC. As it is evident by our functionalities, embedding omission corruptions in UC requires new design choices for the relevant functionalities.

Because the core novelty of our results is in the protocol constructions and proofs, to eliminate the technical burden put upon the reader in extracting the ideas from the simulation, we have employed a special proof structure: First we prove properties that our protocols have, akin to the traditional property-based approach used in the general adversary literature; subsequently, we describe our simulator and use the above properties, along with additional arguments wherever necessary, to argue perfect indistinguishability of real and ideal world.

3 Byzantine Agreement with Active and Omission Corruption

3.1 Security Conditions

In this section we present our first major result, a tight BA condition. Our results cover the case of mixed active and omission-corruption under perfect security (i.e., zero error probability).

Theorem 1. *In the model with both active and omission-corruption if no setup is assumed a set $\mathcal{P}$ of players can perfectly $\mathcal{Z}$-securely realize Consensus or Broadcast if and only if the condition $C_{BA}^{(A,\Omega)}(\mathcal{P},\mathcal{Z})$ holds where,*

$$C_{BA}^{(A,\Omega)}(\mathcal{P},\mathcal{Z}) \iff \forall Z_i, Z_j, Z_k \in \mathcal{Z} : A_i \cup A_j \cup A_k \cup (\Omega_i \cap \Omega_j) \neq \mathcal{P}. \quad (12)$$

The proof of the theorem is in 3.7 and the condition is proven to be both sufficient and necessary for both flavors of BA, i.e., Consensus and Broadcast. The theorem follows after all the necessary tools are created (namely the primitives FixReceive, Weak, Graded and King Consensus).

3.2 Detection of Omission-Failures on Public Point-to-Point Communication

We begin by investigating the main problem of omission-corruption, namely the fact that detecting such a corruption is not trivial. Indeed, when a player p_j does not receive a message she was expecting (because the channels are synchronous), and receives the default value $\perp$ she cannot be certain if the sender p_i is *actively corrupted* and did not send a message, if the sender is *omission-corrupted* and his message was blocked or if p_j *herself is omission-corrupted* and was not able to receive the message (or a combination of the above).

Our first goal is to implement a functionality $\mathcal{F}_{FR}$ in order to reinforce our communication network and render it able to detect omission-failures. Effectively, this functionality guarantees that either the adversary lets the message of the sender p_i reach its recipient p_j or it becomes known to everyone (first to p_j herself and then she will make it public) that p_j is omission-corrupted, forcing her to become a zombie.

If p_j becomes a zombie via the *FixReceive* protocol, she stops participating in any upcoming computations. Also, she notifies all other players about that by sending a special message (she can send it at every round to make sure that everyone receives it) saying that she "is out". Those properties are captured by the functionality $\mathcal{F}_{FR}$ fully described in the full version of the paper [9].

For our functionalities we follow the template of [13] for canonical synchronous functionalities. The functionality proceeds in this way. Initially $\mathcal{F}_{FR}$ sets the output value equal to $\perp$ and then waits for input from p_i. This input is made known to the adversary through the leakage function $l(x)$. On the second round the adversary has the following choices. i) If $p_i \in \Omega^\star$ (the sender is omission-corrupted), the adversary can drop the input message and turn it to $\perp$ or let it be recorded as normal. ii) If $p_j \in \Omega^\star$ the adversary can either inform p_j of his omission status or let her receive the recorded message m_{out}.
As such, we can see that if p_j remained alive then she outputs a value m_{out}, which will be either p_i's input or $\perp$. Additionally, if p_i is correct we are granted that it is the former case.

Our protocol which realizes this functionality does the following in more detail. When p_i wants to send a message x to p_j, he sends x to all $p_k \in \mathcal{P}$ to leverage all parties. Then, every p_k who received the message forwards it to p_j. If p_k did not receive a message (he denotes that by the symbol $\perp$) he sends a special message "$\mathsf{n/v}$" $\notin \mathbb{F}$ to p_j, to let her know that *no value* was received.
After that, p_j should have received a message from all $p_k \in \mathcal{P}$. If from some player she did not, she denotes that by the default character $\perp$. At that point, if there is no way according to the adversary structure $\mathcal{Z}$ that the $\perp$ symbols she received were sent by players that could be omission-corrupted *or* actively corrupted she becomes a zombie. In other words, if there is some player who sent $\perp$ but could never be corrupted, then it is clear for p_j that she has a problem in receiving messages.
In the opposite case, if there exists a value x' which was sent to p_j by people that could not be actively corrupted, it would mean that this value cannot be an erroneous one being pushed by the adversary. As such, p_j can be certain that this is the message that p_i sent, and she outputs this value x'.
Otherwise, in the case where no such value exists, meaning that p_i was not consistent with the messages he sent or he was blocked, p_j outputs $\perp$ to indicate that p_i is not correct. The protocol *FixReceive* can be found in the full version of the paper [9].

Lemma 1. *If the condition* $C^{(A,\Omega)}_{FIXR}(\mathcal{P}, \mathcal{Z})$ *(see Eq. 6) holds, the protocol* FixReceive *perfectly* $\mathcal{Z}$*-securely realizes the functionality* $\mathcal{F}_{FR}$*.*

To prove the above lemma, we will use the following properties of our protocol. i) If p_j is alive at the end of the protocol then p_j outputs a value x', where $x' \in \{x, \perp\}$, unless $p_i \in A^\star$, and $x' = x$ if p_i is correct until the end of the protocol. ii) Moreover, p_j might become a zombie only if p_j is omission-corrupted.
From there, since we prove static security and these are public state protocols where all inputs are revealed by the functionality, the simulator needs to simply

run a simulated copy of the protocol with these input. A formal simulation proof and proof of the properties can be found in the full version.

Proof. (sketch) According to the protocol, p_j becomes a zombie only if there exists no adversary class Z that could explain the $\perp$ messages p_j received. This guarantees that p_j is omission-corrupted because the $\perp$ messages he received cannot be explained in another way. Now, if p_j is alive and p_i is correct until the end of the protocol, p_j will output the correct value $x' = x$ because all the correct players will have received the x value from p_i and additionally the condition $C^{(A,\Omega)}_{FIXR}(\mathcal{P}, \mathcal{Z})$ ensures that there are enough of them to carry the correct value to p_j. On the opposite case where p_i was not correct, p_j is not guaranteed to reach a correct result. If there are conflicting values due to the malicious behavior of p_i, p_j will output $\perp$. □

We also prove in the full paper that this bound is actually tight, meaning that the $C^{(A,\Omega)}_{FIXR}(\mathcal{P}, \mathcal{Z})$ condition is also necessary for FixReceive.

Claim. If the condition $C^{(A,\Omega)}_{FIXR}(\mathcal{P}, \mathcal{Z})$ (see Eq. 6) does not hold, then no protocol can satisfy the properties of the *FixReceive* protocol stated above.

3.3 Weak Consensus

By use of Fix Receive, we will now establish an initial, basic form of consensus, called Weak Consensus. For this, we require the following properties. *Persistency:* If all alive, not actively-corrupted parties start with the same input x then all of them should output $y = x$. *Consistency:* Additionally, there cannot be disagreement between them. To do this, we allow them to output a special character "n/v" (no value) if they are unsure. So, all of them can output either the common value y or "n/v". However, no two correct players should have contradicting values. Our functionality $\mathcal{F}_{WC}$ in the full paper [9] captures those requirements.

Initially, it sets everything to $\perp$ and then receives input from the players. Again, the adversary learns those values, as in FixReceive. Afterwards, once FixReceive is concluded, the adversary is allowed to affect the output. However, he is bound by the consistency and persistency properties we mentioned. As such, he can only set the outputs to either v or "n/v", if there is no pre-agreement on the inputs. The only other action he can perform is to make players in $\Omega^\star$ become zombies, by informing them of their omission status.

Now, our *WeakConsensus* is realized as follows, using (as do all follow-up protocols) *FixReceive* for party-to-party communication. First, we have every player send his input x_i to all players (using *FixReceive*). Then, each player looks at all the possible classes of the adversary structure for the following: 1) If there exists one $Z = (A, \Omega)$ which gives him a value $x \in \mathbb{F}$ such that this value was sent to p_j by some players who are not corrupted *and* 2) the values he received which are different from both x and $\perp$ can be justified by the set of actively corrupted players, meaning that a malicious player sent the disagreeing value. Additionally, all $\perp$ values should be coming from $p_k \in \Omega$,

because FixReceive gives us the guarantee that an alive player only outputs $\perp$ if the sender is not correct. If that is the case, the player adopts this value x as his output. Otherwise, he cannot be certain and outputs the message "$\mathsf{n/v}$".

Lemma 2. *If the condition* $C_{BA}^{(A,\Omega)}(\mathcal{P}, \mathcal{Z})$ *(see Eq. 12) holds, the protocol* Weak-Consensus *perfectly* $\mathcal{Z}$*-securely realizes the functionality* $\mathcal{F}_{WC}$*.*

To prove this lemma we will make use of the following properties of our protocol. *(weak consistency)* There exists some $y \in \mathbb{F}$ such that every (alive) $p_j \in \mathcal{P} \setminus A^\star$ outputs $y_j \in \{y, \text{"}\mathsf{n/v}\text{"}\}$. *(persistency)* If every $p_i \in \mathcal{P} \setminus A^\star$ who is alive at the beginning of *WeakConsensus* has the same input x, then all alive players at the end of the protocol output $y = x$. The full proof can be found in the full version of the paper [9].

Proof. (sketch) First of, we can prove that the selection of the output value y_j is unique, assuming that there can be two that satisfy the conditions and reaching a contradiction. After that, we can prove the weak consistency and persistency properties, by using the sets $P_j^{(\perp,0,1)}$ which split the whole player set according to the values that where received. After that, we can use the condition $C_{BA}^{(A,\Omega)}(\mathcal{P}, \mathcal{Z})$ together with the requirements of the protocol for choosing y_j in order to show that the desired properties hold true. This is done after a lengthy separation of the sets of players and a case study of what values they can send. □

3.4 Graded Consensus

The next step towards our goal is *GradedConsensus*. Leveraging Weak Consensus to get a stronger type of agreement, the players here also output a bit grade g_i reflecting their certainty in their output. This is similar to the Gradecast primitive in [17]. This is a graded version of persistence—i.e., if the players who are not actively corrupted have pre-agreed on a value x, then we get that they all output x with grade $g = 1$ (*graded persistency*). Additionally, it ensures that if any non-actively corrupt party outputs $y_i = y$ with $g_i = 1$, then every non-actively corrupt alive party p_j outputs $y_j = y$ (*graded consistency*). In the opposite case, where the player is not certain about his output value, his grade of confidence is $g_i = 0$.

Our functionality $\mathcal{F}_{GC}$ presented in detail in the full paper [9] captures those properties. At its core it works in a similar manner to $\mathcal{F}_{WC}$. The difference here is that if there exists pre-agreement then all grades are set to 1 and outputs to the same value and the adversary is not allowed to change them. Else, if some grade is $g_i = 1$, then all outputs have to be the same, but the adversary can select the other grades. Otherwise, with all grades 0, the outputs are allowed to be selected by the adversary.

In more detail, the protocol first calls the *WeakConsensus* protocol in order to reach an initial step of agreement. Then all the players exchange the outputs they received, by invocation of *FixReceive*. After that, each player collects that information and decides whether to output $y_j = 0$ or $y_j = 1$. If it can be seen

from the adversary structure that non-actively corrupted players have sent the value 1, then $y_j = 1$ (as in this case every non-actively corrupted player would have output $x'_i \in \{y_j, \text{“}\mathsf{n/v}\text{”}\}$). Otherwise, if she only received 0 and “$\mathsf{n/v}$” from non-actively corrupted, she sets her output (by default) to $y_j = 0$.

Next, we have to determine the grade. If at least all non-actively corrupted players have sent to p_j either the same message as her output y_j or $\perp$ (from the players in Ω) *and* at least all uncorrupted players have definitely sent y_j, then p_j sets her grade of confidence in the fact that all uncorrupted players have the same output as $g_j = 1$. Otherwise, she sets $g_j = 0$, showing that there is no agreement from her point of view, yet.

Lemma 3. *If the condition* $C_{BA}^{(A,\Omega)}(\mathcal{P}, \mathcal{Z})$ *(see Eq. 12) holds, the protocol* GradedConsensus *perfectly* $\mathcal{Z}$*-securely realizes the functionality* $\mathcal{F}_{GC}$*.*

To prove this lemma we will make use of the following properties of our protocol. *(graded consistency)* If some $p_i \in \mathcal{P} \setminus A^\star$ outputs $(y_i, g_i) = (y, 1)$ for some $y \in \mathbb{F}$, then every (alive) $p_j \in \mathcal{P} \setminus A^\star$ outputs $y_j = y$ with some $g_j \in \{0, 1\}$. *(graded persistency)* If every $p_i \in \mathcal{P} \setminus A^\star$ who is alive at the beginning of *GradedConsensus* has input $x_i = x$, then every (alive) $p_j \in \mathcal{P} \setminus A^\star$ outputs $(y_j, g_j) = (x, 1)$.

Proof. (sketch) First of all we can prove that for any player $p_j \in \mathcal{P} \setminus A^\star$, $y_j = 1$ only when a condition respective to the case of $y_j = 0$ holds. After that, in a similar manner as in Weak Consensus we can prove the properties of *graded consistency* and *graded persistency*, leveraging the properties of Weak Consensus. All details can be found in the formal proof in the full paper [9]. □

3.5 King Consensus

The last step towards Consensus is *KingConsensus*. Here, we select one player and give him the special role of *(phase) king*. As before, if the players had pre-agreement on their inputs x, they all must output the same value x (persistency). On top of that, if the king is correct, the players will reach agreement, no matter what (king consistency).

The functionality $\mathcal{F}_{KC}$ that captures this is described in detail in the full paper [9]. The functionality guarantees that persistency is kept if it is already established. Otherwise, it could allow the adversary to change the output to a specific value v for all, subject to the king being correct. Else, the adversary is allowed to select the outputs for all players.

The protocol realizing it first uses *GradedConsensus* and then has the king send his output to all players. All players that are certain for their output keep their value, whereas all those who are uncertain adopt the value of the king. This way, using graded consistency for the grades and persistency we make sure that if the king is correct until the end of the protocol then all non-actively corrupted players will agree on the same output.

Lemma 4. *If the condition* $C_{BA}^{(A,\Omega)}(\mathcal{P}, \mathcal{Z})$ *(see Eq. 12) holds, the protocol* KingConsensus *perfectly* $\mathcal{Z}$*-securely realizes the functionality* $\mathcal{F}_{KC}$*.*

To prove this lemma we make use of the following properties of our protocol. *(king consistency)* If the king p_k is correct, then every $p_j \in \mathcal{P} \setminus A^\star$ outputs $y_j = y$. *(persistency)* If every $p_i \in \mathcal{P} \setminus A^\star$ who is alive at the beginning of *KingConsensus* has input $x_i = x$ then every (alive) p_j outputs $y_j = x$. The formal proof can be found in the full paper [9].

3.6 Consensus

Finally, we are now ready to present our Consensus primitive. The end goal of the parties is to terminate with the same output y. On top of that, if there was pre-agreement on input x, the common output should be $y = x$. Our functionality $\mathcal{F}_{CS}$, described in detail in the full paper, allows the adversary to change the common output value only if there was no pre-agreement. Also, he is able to make a player in $\Omega^\star$ zombie. All other messages are ignored.

The way that the $\mathcal{F}_{CS}$ is realized by the protocol *Consensus* is by repeatedly calling the *KingConsensus* protocol. We use as inputs the outputs of the previous iteration and each time the king is a different player, in turn for all players until we reach an honest one. Since every player becomes a king, we can be certain, as long as not all players are corrupted (which is not allowed by our security condition) that at least one king will be correct and, hence, we will achieve consistency on the output value. This point will be reached even sooner if the non-actively corrupted players have pre-agreed on a value x. What is more, once this agreement is achieved, by the persistency property of *KingConsensus*, we can be certain that it will not change, no matter what the king sends (in case he is not correct) thus the agreement will be maintained.

To be more formal, below is the property-based definition of Consensus for our relevant corruption types. A protocol perfectly $\mathcal{Z}$-securely realizes Consensus among the players in $\mathcal{P}$ if it satisfies the following properties in the presence of a $\mathcal{Z}$-adversary:

- *(consistency)* Every non-actively corrupted $p_i \in \mathcal{P}$ who is alive at the end of the protocol outputs the same value y.
- *(persistency)* Assuming that every non-actively corrupted $p_i \in \mathcal{P}$ who is alive at the beginning of the protocol has input x, the output is $y = x$.
- *(termination)* For every non-actively corrupted $p_i \in \mathcal{P}$ the protocol terminates after a finite number of rounds.

Theorem 2. *If the condition* $C_{BA}^{(A,\Omega)}(\mathcal{P}, \mathcal{Z})$ *(see Eq. 12) holds, the protocol* Consensus *perfectly* $\mathcal{Z}$*-securely realizes the Consensus functionality* $\mathcal{F}_{CS}$*.*

Proof. The proof of this theorem follows from the established protocols above. If $C_{BA}^{(A,\Omega)}(\mathcal{P}, \mathcal{Z})$ holds true we are granted that there exists an uncorrupted player in $\mathcal{P}$, as $\mathcal{P} \setminus (A_i \cup A_j \cup A_k \cup (\Omega_j \cap \Omega_k)) \neq \emptyset$, for all i, j, k selections of the three sets. Then, applying both properties of *KingConsensus* in succession creates and then maintains the agreement on the output for all iterations. This post-agreement can be achieved earlier still, if there is pre-agreement between the non-actively

corrupted players on their values. Finally, for the termination property, we are certain that the protocol repeats a finite number of times a terminating protocol, so it is guaranteed to terminate. □

3.7 Broadcast

In this section we describe our Broadcast functionality $\mathcal{F}_{BC}$ above. The idea is similar to the Consensus functionality, with the difference that only the sender p has an input x. Additionally, if he remains alive, all alive players will get the same output value $y = x$. The adversary can only make players $p_j \in \Omega^\star$ become zombie and nothing more to alter the outputs. An honest p always broadcasts the correct value and the output of broadcast is $\perp$ only if $p \in \Omega^\star$.

The formal property-based definition of *Broadcast* is as follows. A protocol perfectly $\mathcal{Z}$-securely realizes Broadcast with sender a player p whose input is x among the players in $\mathcal{P}$ if it satisfies the following properties in the presence of a $\mathcal{Z}$-adversary:

- *(consistency)* Every non-actively corrupted $p_i \in \mathcal{P}$ who is alive at the end of the protocol outputs the same value y.
- *(validity)* Assuming that the sender p is not actively corrupted, the common output y satisfies $y \in \{x, \perp\}$. Specifically, $y = x$ if p is alive and correct until the end of the protocol and $y = \perp$ if p has become a zombie.
- *(termination)* For every non-actively corrupted $p_i \in \mathcal{P}$ the protocol terminates after a finite number of rounds.

Note that, as *Broadcast* invokes *Consensus*, the condition $C_{BA}^{(A,\Omega)}(\mathcal{P}, \mathcal{Z})$ is needed for this protocol, as well. We describe below our protocol for broadcast that realizes the functionality $\mathcal{F}_{BC}$. The proof can be found in the full paper and is mainly derived from the properties of *Consensus*.

Protocol *Broadcast*$(\mathcal{P}, \mathcal{Z}, p, x)$

1. In round $\rho = 1$: The sender p sends x to every $p_j \in \mathcal{P}$ using *FixReceive*, who denotes the received value by x_j. (If p_j received $\perp$ he sets $x_j = 0$).
2. In round $\rho = 4$: The players invoke *Consensus*$(\mathcal{P}, \mathcal{Z}, (x_1, \ldots, x_n))$ on the received values. We denote p_j's output as y_j.
3. In round $\rho = 12n + 4$: The sender p sends a confirmation bit b to every $p_i \in \mathcal{P}$ using *FixReceive*, where $b = 1$ if the output of p after *Consensus* equals x and $b = 0$ otherwise; p_i denotes the received bit by b_i. (If p_i received $\perp$ he sets $b_i = 0$).
4. In round $\rho = 12n + 7$: Invoke *Consensus*$(\mathcal{P}, \mathcal{Z}, (b_1, \ldots, b_n))$.
5. In round $\rho = 24n + 7$: For each $p_i \in \mathcal{P}$, if p_i's output after *Consensus* is 1, he outputs y_i, otherwise he outputs $\perp$.
   If some $p_z \in \Omega^\star$ became zombie he outputs (omission, p_z).

Theorem 3. *If the condition* $C_{BA}^{(A,\Omega)}(\mathcal{P}, \mathcal{Z})$ *(see Eq. 12) holds, the protocol* Broadcast *perfectly* $\mathcal{Z}$*-securely realizes the functionality* $\mathcal{F}_{BC}$.

Table 1. The classes Z_1, Z_2, Z_3 with $A_1 \cup A_2 \cup A_3 \cup (\Omega_2 \cap \Omega_3) = \mathcal{P}$.

	p_1	p_2	p_3	p_4
Z_1	α			
Z_2		α		ω
Z_3			α	ω

Necessity of Conditions for Byzantine Agreement. Next, we show that the $C_{BA}^{(A,\Omega)}(\mathcal{P}, \mathcal{Z})$ condition is also necessary for Broadcast.

The following lemma shows the impossibility that arises when $C_{BA}^{(A,\Omega)}(\mathcal{P}, \mathcal{Z})$ is violated. The proof exploits adversarial strategies which create an ambiguity in the view of the players, which prevents them from deciding which corruptible class the adversary has actually chosen, contradicting correctness. For a more detailed proof with further discussion about why older techniques wouldn't work see the full version [9].

Lemma 5. *If the condition $C_{BA}^{(A,\Omega)}(\mathcal{P}, \mathcal{Z})$ does not hold, then the properties of the* Broadcast *protocol stated in Theorem 3 cannot hold, as well.*

Proof. (sketch) Assuming that the condition $C_{BA}^{(A,\Omega)}(\mathcal{P}, \mathcal{Z})(12)$ does not hold and that we have secure broadcast we will reach a contradiction. This means that $\exists\ Z_1, Z_2, Z_3$ such that $A_1 \cup A_2 \cup A_3 \cup (\Omega_2 \cap \Omega_3) = \mathcal{P}$. Using a player-simulation argument, we get three scenarios that are indistinguishable for the players, using the adversary structure shown in Table 1. We assume that p_1 is the designated sender and we want all other players to output the same value, according to the broadcast property. We consider the following scenarios. In all cases, communication from p_1 to p_4 is cut entirely.

In the first scenario (see Fig. 1), p_1 is actively corrupted and all other players are honest. He sends different values to p_2 and p_3 and nothing to p_4. This makes p_2 believe that the sender has input 0, p_3 believes that the sender has input 1 and p_4 does not have any direct information from the sender.

In the second scenario (see Fig. 2), p_2 is actively corrupted and p_4 is omission-corrupted. The sender sends his input 1 to p_3 but is blocked from reaching p_4. At the same time the adversary is using p_2 to claim that the sender sent him the value 0. This makes p_2 believe that the sender has input 0, p_3 believes that the sender has input 1 and p_4 does not have any direct information from the sender. Because the sender is not actively corrupted and is correct until the end of the protocol, due to validity, all players should output 1 with overwhelming probability.

In the third scenario (see Fig. 3), p_3 is actively corrupted and p_4 is omission-corrupted. The sender sends his input 0 to p_2 but is blocked from reaching p_4. At the same time the adversary is using p_3 to claim that the sender sent him the value 1. This makes p_2 believe that the sender has input 0, p_3 believes that the sender has input 1 and p_4 does not have any direct information from the

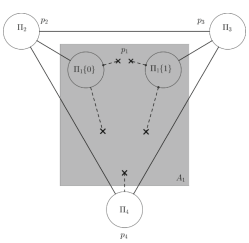


Fig. 1. p_1 is actively corrupted.

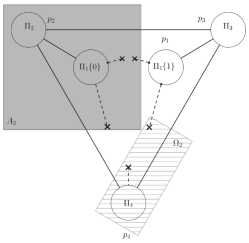


Fig. 2. p_2 is actively, p_4 is omission-corrupted.

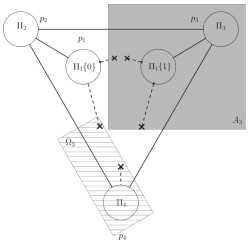


Fig. 3. p_3 is actively, p_4 is omission-corrupted.

sender. Because the sender is not actively corrupted and is correct until the end of the protocol, due to validity, all players should output 0 with overwhelming probability.

For p_4 the three scenarios are indistinguishable. Hence, the output should be the same value in all cases, since all scenarios have the same set up, meaning that the distribution of the outputs should be identical. As the result is overwhelmingly different in all cases, this leads us to a contradiction.

This proves that the condition $C_{BA}^{(A,\Omega)}(\mathcal{P}, \mathcal{Z})$ consists a tight feasibility bound for Broadcast, meaning that the desired properties hold true if and only if our condition holds. In the full paper [9] we show that this condition is tight for Consensus as well. □

4 Multi-party Computation

In this section we extend our study to multi-party computation. A discussion about the challenges of such an extension, and a showcase that existing techniques from the threshold literature either do not work, or yield counter-intuitive results can be found in the full version of the paper [9]. There we discuss and prove the ineffectiveness of player elimination, a technique frequently used in the general adversary literature.

This motivates us to introduce a new condition $C_{SMT}^{(A,\Omega)}(\mathcal{P}, \mathcal{Z})$, which together with $C_{BA}^{(A,\Omega)}(\mathcal{P}, \mathcal{Z})$ enables us to create a Secure Message Transmission primitive in Sect. 4.1. This allows any two given parties to exchange securely a message s and, specifically, the protocol either aborts while detecting a corrupted party or it provides an alive receiver with the correct message of a sender, effectively creating a publicly detectable private message functionality. In other words, either the message is delivered (keeping its privacy) or it can be publicly detected which player failed/is corrupted.

With that idea we practically overcome the problem of omission corruptions and, thus, we could use any MPC protocol for active corruption in general

adversaries to accomplish the rest of our task. We present the necessary tools and building blocks to do that in Sect. 4.2.

Next, we tackle one of the final problems, namely securely computing the gates in Sect. 4.3, with multiplication being the main difficulty while addition is pretty straight forward. There we present their functionalities and how we can implement them in our setting.

Finally, after establishing that, we will be able to provide a tight characterization of the perfectly secure MPC landscape (in terms of both feasibility and impossibility) in the remainder of the section. We compose all of our blocks and tools together in Sect. 4.4 to present our full MPC protocol and in Sect. 4.4 we prove that our conditions are also necessary, i.e., tight.

As a side note, we remind here that our MPC assumes that the parties can broadcast messages (elements from an appropriate arithmetic field $\mathbb{F}$). As we can easily see, our MPC condition $C_{MPC}^{(A,\Omega)}(\mathcal{P},\mathcal{Z})$ implies $C_{BA}^{(A,\Omega)}(\mathcal{P},\mathcal{Z})$ which means that we can use *Broadcast* for this purpose[6].

4.1 Detectable Secure Message Transmission

The first step towards our MPC protocol is to enable any pair of parties with a sender P_s and a receiver P_r to exchange a message s securely, i.e., with the privacy and correctness of the message preserved. Furthermore, we want to accomplish that in a publicly detectable way, meaning that the protocol either succeeds or it aborts having detected a corrupted party.

To achieve this, the condition $C_{BA}^{(A,\Omega)}(\mathcal{P},\mathcal{Z})$ of broadcast is no longer sufficient. On top of it, we need the condition $C_{SMT}^{(A,\Omega)}(\mathcal{P},\mathcal{Z},P_s,P_r)$ for this pair of parties, as explained in 2.

In more detail, we have the sender P_s who wants to send a message $s \in \mathbb{F}$ to the receiver P_r. The functionality dictates that this is accomplished without leaking any information to the adversary. If no input is received by the functionality from P_s due to his fault (i.e., $P_s \in \Omega^\star$ or $A^\star$) then $\mathcal{F}_{DetSMT}$ sends a default message "$\mathsf{n/v}$" to P_r. If P_r is omission-corrupted and is unable to receive the intended value from $\mathcal{F}_{DetSMT}$, he outputs a special symbol $\perp$ to denote this.

The functionality that captures our goal is presented in detail in the full version of the paper [9]. It starts by taking some input s from the designated sender. The adversary can input any value his chooses if the sender is actively corrupted. Then this value is forwarded to P_r. Meanwhile, it allows the adversary to affect the output if the sender or receiver are corrupted by in a detectable way. Also, he could cause an abort, but at the cost of making publicly known the identity of a corrupted party.

Our protocol that realizes this functionality works in a way that resembles *FixReceive*, as discussed in Sect. 3.2; The sender sends to all players and at the end all players forward the message to the receiver. However, the difference is that now we have a reliable broadcast primitive and the players can use it complain if

[6] As discussed in the introduction, we can trivially turn binary *Broadcast* to a string *Broadcast* by invoking it for each bit of the string.

they do not receive a message they were expecting. Also, we also ensure that the privacy of the message is maintained, in contrast to *FixReceive*, which was done in public communication. This is done by the use of a secret sharing scheme. The sharing is characterized by the *sharing specification* $\mathcal{S}$, according to which the shares of the message to be kept secret are distributed to the players, effectively stopping the adversary from holding all shares. In our case we will be using a sum sharing, i.e., the secret value s is split in summands $s_1, \ldots, s_m$ with $\sum_{i=1}^{m} s_i = s$, where m is the size of the sharing specification $|\mathcal{S}|$.
For each player p_j we call the vector of summands in his possession $\langle s \rangle_j = (s_{j_1}, \ldots, s_{j_k})$ as p_j's *share* of s. The complete vector of all shares is denoted as $\langle s \rangle = (\langle s \rangle_1, \ldots, \langle s \rangle_n)$ and is called a *sharing* of s. The vector of summands of s is denoted as $[s] = (s_1, \ldots, s_m)$ and their sum is equal to s. We, also, say that such a sharing $\langle s \rangle$ is a *consistent sharing* of s according to $(\mathcal{P}, \mathcal{S})$, if for each $S_k \in \mathcal{S}$ all (correct) players in S_k have the same view on s_k and $s = \sum_{k=1}^{m} s_k$.
Our selection for $\mathcal{S}$ will be the natural sharing specification $\mathcal{S}_{\mathcal{Z}}$ associated with $\mathcal{Z}$, i.e., $(S_1, \ldots, S_m) = (\mathcal{P} \setminus A_1, \ldots, \mathcal{P} \setminus A_m)$, where $m = |\mathcal{Z}|$, so that for each corruptible class Z_i all the players not included in A_i for that class will receive the share s_i. This way the adversary never obtains all summands.

Using that secret sharing scheme, P_s creates a sharing of his message s and sends each part s_q to the complement S_q of A_q. This process is done for each $q = 1, \ldots, m$. Then the players who did not receive it can complain through broadcast. Additionally, an extra round of cross checking and relay is added, during which all parties in S_q send to one another the values they received from P_s. Again, complaints are raised and the players try to see which classes of the adversary structure fit their view. Finally, once the complaints are over, all players send their vector of received values to the designated receiver of the message, P_r.

Lemma 6. *If the condition* $C_{MPC}^{(A,\Omega)}(\mathcal{P}, \mathcal{Z})$ *(see Eq. 11) holds, the protocol* DetSMT *perfectly* $\mathcal{Z}$*-securely realizes the functionality* $\mathcal{F}_{detSMT}$.

To prove this lemma we will make use of the following properties of our protocol. Either the protocol aborts with some set B of corrupted parties or it terminates and we have the following properties: If the receiver P_r is alive at the end of the protocol then he outputs a value $s_p \in \mathbb{F}$ where $s_p = s$ unless $P_s \in A^\star$. Also, P_r might become a zombie only if he is omission-corrupted. Furthermore, no information on s is leaked to the adversary. The complete proof can be found in the full paper [9]. One thing that we need to point out is the following caveat. Extra care needs to be taken in order to properly describe the simulation for the functionality, because the simulator has to act differently based on whether the adversary has access to some summand of the secret value or not. This subtlety is expanded upon in the full proof.

Necessity of SMT Condition. Additionally, we prove that the condition $C_{SMT}^{(A,\Omega)}(\mathcal{P}, \mathcal{Z}, p_1, p_2)$ is actually necessary for the (plain, not detectable) $\mathcal{F}_{SMT}$

functionality that enables p_1 to securely send a message to a receiver p_2, making our result tight (both sufficient and necessary).

Lemma 7. *If the condition $C_{SMT}^{(A,\Omega)}(\mathcal{P}, \mathcal{Z}, p_1, p_2)$ does not hold, then the functionality $\mathcal{F}_{SMT}(\mathcal{P}, \mathcal{Z}, p_1, p_2, m)$ cannot be securely realized.*

As before, further details and the complete proof can be found in the full version of the paper [9].

4.2 Building Blocks and Tools for MPC

Having established the *DetSMT* primitive to replace the network of point-to-point channels for the communication, we will now carry on with the construction of an MPC protocol by following the classic idea of creating a secure MPC protocol in the presence of a general adversary using only active corruption. Given any arithmetic circuit C—recall that this is a complete model of computation—the protocol evaluates the circuit in a gate-by-gate fashion, where the invariant is that the inputs and outputs of each gate of C are kept secret shared, see below, so that no information leaks to the adversary. Importantly, the protocols that process each gate, which we construct, might abort; however, when this happens: (1) no information leaks to the adversary, and (2) a corrupted party p is identified. This means that we can exclude p, and reset the computation without it. As we prove, the relevant sufficient condition, $C_{MPC}^{(A,\Omega)}(\mathcal{P}, \mathcal{Z})$, is preserved when eliminating such a corrupted party, which will ensure security in the reduced setting. As soon as an iteration of the above processing of the gates of C terminates without an abort–which is bound to happen after at most n resets—we invoke a reconstruction protocol to have every party (still alive) receive the output. We note that without loss of generality, we assume that the function which is computed by C has one public output. Using standard techniques, we can use a protocol for any such function to compute functions with multiple and/or private outputs [28].

In the following, we start by describing and proving the security of subprotocols that are used as building blocks and then describe how these can be stitched together in an MPC protocol.

Heartbeat. A very important part of our results is based on the fact that if the adversary blocks enough messages addressed to a player to make him reach a wrong conclusion, the player could be able to perceive this loss of messages. Then, he could step down from the calculation by becoming a zombie, as he is (omission) corrupted. The functionality $\mathcal{F}_{Hb}$ is taking as input by the player a bit $b = 1$ indicating that he is alive. The adversary is able make a player in $\Omega^\star$ aware of his omission status, effectively setting $b = 0$. Then this value is communicated to all parties. The functionality is provided in detail in the full version of the paper [9]. We can implement this through the broadcast of the bit b by the player in question. If $b = 1$ then all agree that the player is alive.

Otherwise, if a player fails to broadcast this bit to other players or broadcasts $b = 0$ it becomes apparent to all that he is a zombie, as he is corrupted. According to the properties of broadcast everyone agrees whether p is alive or not.

It should be noted that omission-corrupted players who have not yet detected their problem can learn that they are zombies from the output of the broadcast protocol.

Lemma 8. *If the condition* $C_{BA}^{(A,\Omega)}(\mathcal{P}, \mathcal{Z})$ *(see Eq. 12) holds, the protocol* Heartbeat *perfectly* $\mathcal{Z}$*-securely realizes the functionality* $\mathcal{F}_{Hb}$*.*

The proof is derived from the *Broadcast* properties. The complete version can be found in the full paper [9].

Verifiable Secret Sharing. A very important primitive that is essential in keeping the privacy of the players' input is called Verifiable Secret Sharing (VSS). On top of guaranteeing that the input of the player is kept secret, we also get that all players agree on the way that the initial value is shared among them. By splitting the message s in random summands s_k using a sum share and then giving each one of those to the corresponding set S_k we achieve the privacy property Furthermore, by having the players in each S_k cross check their values we get the verifiability property. This idea was first developed in [24] and has since been used in many MPC protocols.

The functionality that we want to instantiate is presented in detail in the full version of the paper [9]. To give a brief description, it takes as input a value s that needs to be secret shared. Then, uniformly random shares $s_1, s_2, \ldots, s_m$ where $m = |\mathcal{S}|$, are created such that $s = \sum_{k=1}^{m} s_k$. Each one of those s_k is given to the respective set S_k (which is the complement of A_k). This way, no matter which class $A^\star$ the adversary corrupts, there exists a share $s^\star$ of the set $S^\star = \mathcal{P} \setminus A^\star$ that the adversary does not obtain. Hence the privacy of s is preserved.

In the case where the dealer is actively corrupted, the adversary is allowed to select the shares of s. However, all players in each S_k still get the same value s_k. If the dealer p_d is omission-corrupted, the adversary selects if the Sharing will succeed as normal or if it will abort and p_d will be identified as omission-corrupted.

What makes our implementation simple at this point is the existence of the SMT channel. Instead of sending the messages using the existing network of point-to-point channels, our protocol sends them by invocation of the Protocol *DetSMT* we built earlier. This grants us the detectability and privacy properties directly. Next, the players cross check their shares to detect inconsistencies. If that is the case for some s_k, a special message (CONTRAST, k) is broadcast and the dealer p_d broadcasts the correct summand in the open. Finally, the players invoke a Heartbeat to communicate to all if someone became a zombie.

Lemma 9. *If the condition* $C_{MPC}^{(A,\Omega)}(\mathcal{P}, \mathcal{Z})$ *(see Eq. 11) holds and additionally we have that for all* $Z_i, Z_j \in \mathcal{Z}$ *and for all* $S_k \in \mathcal{S} : A_i \cup A_j \cup (\Omega_i \cap \Omega_j) \not\supseteq S_k$*, the protocol* VSS *perfectly* $\mathcal{Z}$*-securely realizes the functionality* $\mathcal{F}_{VSS}$*.*

To prove this lemma we will make use of the following properties of our protocol. *(correctness)* VSS either outputs a consistent sharing $\langle \hat{s} \rangle$ of some $\hat{s}$, where $\hat{s} = s$ unless the dealer p_d is actively corrupted, or it aborts with a set B of corrupted parties. *(secrecy)* No information on s leaks to the adversary. The complete proof can be found in the full version of the paper [9].

Announce and Reconstruct. The functionalities $\mathcal{F}_{ann}$ and $\mathcal{F}_{recn}$ for the Announce and Reconstruct primitives are given in the full version of the paper [9]. The protocols *Announce* and *Reconstruct* are closely related as the latter is essentially built on the former. The first one is used to publicly announce the value of a specific summand (using *Broadcast*) and the second one to publicly reconstruct a sharing of a value (using *PublicAnnounce* for all summands), respectively. Both of those protocols are robust, meaning that if our condition holds true and the sharing of the values was successful, those protocols cannot abort and they always succeed.

Lemma 10. *If the condition $C_{MPC}^{(A,\Omega)}(\mathcal{P}, \mathcal{Z})$ (see Eq. 11) holds, assuming that s_k is a summand of a consistent sharing of a value s, the protocol* PublicAnnounce *perfectly $\mathcal{Z}$-securely realizes the functionality $\mathcal{F}_{ann}$.*

Since *PublicAnnounce* is robust and does not abort, it becomes apparent that *Reconstruct* is also robust and if the protocols called up to that point have succeeded, it correctly reconstructs the desired value s from its summands that are announced one by one.

Lemma 11. *If the condition $C_{MPC}^{(A,\Omega)}(\mathcal{P}, \mathcal{Z})$ (see Eq. 11) holds, assuming that s_k is a summand of the correct sharing of a value s, the protocol* Reconstruct *perfectly $\mathcal{Z}$-securely realizes the functionality $\mathcal{F}_{recn}$.*

The complete proofs can be found in the full version of the paper [9].

4.3 Computing the Gates

Addition. The first type of gate that we need to implement is the Addition gate. At each such gate, given the sharing $\langle s \rangle$ and $\langle t \rangle$ of s, t the players need to compute a sharing of their sum $s+t$. The simplest way to create this new sharing is to have each party p_j locally compute the sum of his shares of the two values and set $\langle s + t \rangle_j = \langle s \rangle_j + \langle t \rangle_j$. This way we create a sharing that is random (as the sum of two random summands), hides the value of $s + t$ as it did for s, t and is consistent, as long as the initial sharings are consistent. As the parties can locally compute addition gates without any communication involved, we omit the rest from the main body and point to the full version of the paper [9] for more details.

Multiplication. Our next goal is to to securely compute a sharing of the product of two shared values. Its properties are that, as long as our conditions hold, given two consistent sharings $\langle s \rangle, \langle t \rangle$ it securely creates a consistent sharing of $\langle s \cdot t \rangle$, or it aborts after detecting a set B of incorrect/corrupted players. Those properties are captured by the functionality $\mathcal{F}_{mult}$ provided in the full version of the paper [9].

Initially, the functionality receives input in the form of sharings, where each player p_i inputs his shares $\langle s \rangle_i$ and $\langle t \rangle_i$ for s and t, respectively. The adversary can select the shares for the player he controls. After that, the functionality checks whether the input of all non-actively corrupted parties for every summand s_k is the same, i.e., checks whether the sharing is consistent. If it is, s_k is fixed to this value (similarly for every t_ℓ). Otherwise, the values of the first honest player are adopted. Then, the product $x_{k,\ell}$ of any two summands s_k, t_ℓ is calculated. Next each such product needs to be shared to all parties according to $\mathcal{S}$. This is performed by all players holding $x_{k,\ell}$. Once the sharing of all those products is completed, all parties can locally add their shares of $x_{k,\ell}$ over all combinations of k, ℓ to obtain a share of the final product $y = st$.

We note that if the adversary controls a party that can compute $x_{k,\ell}$, he is able to select how this product is shared, i.e., how it is split into summands $[x_{k,\ell}] = (z_1, \ldots, z_m)$ and importantly, he can impose this choice to the honest players, subject to the summands adding up to $x_{k,\ell}$. This was observed and dealt with in detail in the work of Asharov, Lindell and Rabin [4]. Alternatively, the adversary is able to completely deviate from creating a sharing of the correct value and select summands that do not add up to $x_{k,\ell}$, but in this scenario the functionality detects that and adopts the values for s_k and t_ℓ from a correct player with a default sharing.

Finally, we show that there always exists a non-actively corrupted player having both s_k and t_ℓ, from the condition $C_{MPC}^{(A,\Omega)}(\mathcal{P}, \mathcal{Z})$. Due to that, the adversary cannot tamper with the value of $x_{k,\ell}$ and in this case both s_k and t_ℓ are publicly announced to all players, so that all adopt the correct values. If at some point the adversary decides to make a player aware of his omission status, the player is informed and publicly steps down, while the functionality aborts having detected a corrupted party.

Our implementation of that functionality is the protocol *Mult* and it is based on the respective protocols of [5,30]. The idea of the protocol is the following: As s and t are shared according to $\mathcal{S}$, we can use the summands $s_1, \ldots, s_{|\mathcal{S}|}$ and $t_1, \ldots, t_{|\mathcal{S}|}$ to compute the product st as the sum of the products of all those s_i, t_j, i.e.,

$$st := \sum_{k=1}^{|\mathcal{S}|} \sum_{\ell=1}^{|\mathcal{S}|} s_k t_\ell = \sum_{k,\ell=1}^{|\mathcal{S}|} s_k t_\ell. \tag{13}$$

Each term $x_{k,\ell} = s_k t_\ell$ is shared by every player in $S_k \cap S_\ell$. After that the players try to see if they agree on the shared summands, by computing and reconstructing all the differences of the $x_{k,\ell}$ shared. If they do not agree, either the sharing was not consistent (due to the adversary inputting wrong values

earlier on) or the adversary controls some party in $S_k \cap S_\ell$. In either case, it is safe to publicly announce both s_k and t_ℓ so that everyone agrees on the value of those summands and adopt a default sharing for their product $x_{k,\ell}$.

After doing this for all combinations of k, ℓ, the players compute the sum of the shared terms $x_{k,\ell}$, which results in a sharing of st, as desired.

Lemma 12. *If the condition* $C_{MPC}^{(A,\Omega)}(\mathcal{P}, \mathcal{Z})$ *(see Eq. 11) holds,* $\langle s \rangle$ *and* $\langle t \rangle$ *are consistent sharings according to* $\mathcal{S}$ *and the following properties hold: for all* $Z_i = (A_i, \Omega_i), Z_j = (A_j, \Omega_j) \in \mathcal{Z}$ *and* $S_k \in \mathcal{S} : A_i \cup A_j \cup (\Omega_i \cap \Omega_j) \not\supseteq S_k$*, as well as for all* $S_k, S_\ell \in \mathcal{S}$ *and for all* $Z_i = (A_i, \Omega_i) \in \mathcal{Z} : S_k \cap S_\ell \not\subseteq A_i$*, the protocol* Mult *perfectly* $\mathcal{Z}$*-securely realizes the functionality* $\mathcal{F}_{mult}$*.*

To prove this lemma we will make use of the following properties of our protocol. *(correctness)* It either outputs a sharing of st according to $\mathcal{S}$ or it aborts with a non-empty set B of incorrect players. *(secrecy)* No information leaks to the adversary. The complete proof can be found in the full version of the paper [9].

4.4 The MPC Protocol

We next proceed to the construction of our *MPC* protocol, which securely realizes the functionality $\mathcal{F}_{MPC}$, detailed in the full version [9]. The function to be computed will be represented by a circuit $\mathcal{C}$.

Our protocol will compute the desired circuit on the inputs of the players. If none of the sub-protocols aborts, the protocol will succeed and give the correct output. In the opposite case, where the adversary has misbehaved and caused a protocol to abort we will identify a set B of corrupted parties. Then we will restart the computation of the protocol from the beginning with a smaller structure, setting $\mathcal{P} := \mathcal{P} \setminus B$, as the players in B are all problematic. Importantly, this action preserves the monotonicity of the condition, namely that the MPC condition is also true in the new updated adversary structure. We should also point out that even in the case of such an abortion no information about the players' input is leaked to the adversary. This is because all calculations are done with sharings of the inputs, hence the actual values are hidden. The only time where a value is actually revealed is after the *Reconstruct* protocol. However, our *Reconstruct* protocol is robust, meaning that it cannot abort and if the protocol has reached this point, it is guaranteed to succeed.

Moving on to the description of the protocol, it involves three stages, the input, the computation and the output stage. For the input stage, we have all players share their inputs according to the sharing specification $\mathcal{S}$. This is done to make sure that the inputs remain private, while still being able to perform computations with them. In the case that a player fails to share her input, e.g., if she is corrupted and the adversary blocks her messages, all players adopt a default pre-agreed sharing for her input value.

For the evaluation stage, the procedure is the following. Depending on the gate of $\mathcal{C}$ that needs to be evaluated, the players do the following. If they need

to evaluate an addition gate, each player locally computes the sum of his shares of the two values, so the output is a sharing of the sum. If they need to evaluate a multiplication gate for two values s, t, the players invoke the protocol $Mult(\mathcal{P}, \mathcal{Z}, \mathcal{S}, \langle s \rangle, \langle t \rangle)$ for the sharings $\langle s \rangle, \langle t \rangle$ and the output is a sharing $\langle st \rangle$ of the product. If they need to evaluate a random gate, each player sends a random value as input and the output is a sharing of the sum of those values.

Lastly, for the output stage where the players want to eventually get the actual value of the output v, they invoke the protocol $Reconstruct(\mathcal{P}, \mathcal{Z}, \mathcal{S}, \langle v \rangle)$ in order to publicly robustly reconstruct one by one the summands of v and after that each one gets the desired value by summing all summands.

Theorem 4. *If the condition $C^{(A,\Omega)}_{MPC}(\mathcal{P}, \mathcal{Z})$ (see Eq. 11) holds, the protocol* MPC *perfectly $\mathcal{Z}$-securely realizes the functionality $\mathcal{F}_{MPC}$.*

Necessity of Condition for MPC. Finally, we can also show that the condition $C^{(A,\Omega)}_{MPC}(\mathcal{P}, \mathcal{Z})$ is necessary to securely achieve MPC.

Lemma 13. *If $C^{(A,\Omega)}_{MPC}(\mathcal{P}, \mathcal{Z})$ is violated, then there exist n-party functions which cannot be securely evaluated while tolerating (corruptions caused by) a $\mathcal{Z}$-adversary.*

Proof. (sketch) As we have already discussed in 3.7 the condition $C^{(A,\Omega)}_{BA}(\mathcal{P}, \mathcal{Z})$ is necessary for broadcast. Also, in Sect. 4.1 we stated that the condition $C^{(A,\Omega)}_{SMT}(\mathcal{P}, \mathcal{Z}, p_i, p_j)$ is necessary to securely exchange a message between a sender p_i and p_j. As our condition $C^{(A,\Omega)}_{MPC}(\mathcal{P}, \mathcal{Z})$ considers all such pairs of players, and due to the fact that MPC implies both Broadcast and SMT, we get that our condition $C^{(A,\Omega)}_{MPC}(\mathcal{P}, \mathcal{Z})$ is necessary for MPC. □

5 Conclusion and Open Problems

We put forth the study of Byzantine agreement (BA) and multi-party computation (MPC) in the presence of a mixed general adversary with active and omission corruptions. We provided a tight characterization—necessary and sufficient conditions—for feasibility of synchronous BA—both for broadcast and consensus—tolerating such an adversary. We also provide a tight characterization of feasibility of MPC in this model, where we show that existing techniques fall short in providing feasibility results. Along the way, we also provide the first tight feasibility result for (detectable) Secure Message Transmission (SMT) in this model; by repeating upon failure while excluding the detected party, this yields the first tight feasibility result for SMT in this model. The above results make an important step forward in understanding the relevant landscape and open the floor to follow-up questions that have been resolved in the threshold adversary setting, but are wide open in the general adversary setting, e.g., allowing setup, error probability, and computationally bounded adversaries.

Acknowledgments. The authors would like to thank the anonymous reviewers for their helpful comments on the manuscript, as well as Giorgos Panagiotakos for discussions in the early stages of this work. The research was done in part while Konstantinos Brazitikos was visiting Purdue University and while Vassilis Zikas was at the University of Edinburgh. Konstantinos Brazitikos was supported in part by Input Output (iohk.io) through their funding of the Edinburgh Blockchain Technology Lab and Sunday Group. Vassilis Zikas was supported in part by Sunday Group, IOHK, NSF grants no. 2055599 & 2001096, and BSF grant no. 41000926.

References

1. Abdallah, M., Pucheral, P.: A low-cost non-blocking atomic commitment protocol for asynchronous systems. In: International Conference on Parallel and Distributed Systems, 1999, Proceedings. IEEE (1999)
2. Altmann, B., Fitzi, M., Maurer, U.M.: Byzantine agreement secure against general adversaries in the dual failure model. In: Jayanti, P. (ed.) Distributed Computing, 13th International Symposium, Bratislava, Slovak Republic, September 27-29, 1999, Proceedings. Lecture Notes in Computer Science, vol. 1693, pp. 123–137. Springer (1999). https://doi.org/10.1007/3-540-48169-9_9
3. Asharov, G., Cohen, R., Shochat, O.: Static vs. adaptive security in perfect MPC: A separation and the adaptive security of BGW. In: Dachman-Soled, D. (ed.) 3rd Conference on Information-Theoretic Cryptography, ITC 2022, July 5-7, 2022, Cambridge, MA, USA. LIPIcs, vol. 230, pp. 15:1–15:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2022). https://doi.org/10.4230/LIPIcs.ITC.2022.15
4. Asharov, G., Lindell, Y., Rabin, T.: Perfectly-secure multiplication for any t ¡ $n/3$. In: Rogaway, P. (ed.) Advances in Cryptology - CRYPTO 2011 - 31st Annual Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2011. Proceedings. Lecture Notes in Computer Science, vol. 6841, pp. 240–258. Springer (2011).https://doi.org/10.1007/978-3-642-22792-9_14
5. Beerliová-Trubíniová, Z., Fitzi, M., Hirt, M., Maurer, U.M., Zikas, V.: MPC vs. SFE: perfect security in a unified corruption model. In: Canetti, R. (ed.) Theory of Cryptography, Fifth Theory of Cryptography Conference, TCC 2008, New York, USA, March 19-21, 2008. Lecture Notes in Computer Science, vol. 4948, pp. 231–250. Springer (2008).https://doi.org/10.1007/978-3-540-78524-8_14
6. Beerliová-Trubíniová, Z., Hirt, M., Riser, M.: Efficient byzantine agreement with faulty minority. In: Kurosawa, K. (ed.) Advances in Cryptology - ASIACRYPT 2007, 13th International Conference on the Theory and Application of Cryptology and Information Security, Kuching, Malaysia, December 2-6, 2007, Proceedings. Lecture Notes in Computer Science, vol. 4833, pp. 393–409. Springer (2007).https://doi.org/10.1007/978-3-540-76900-2_24
7. Ben-Or, M., Goldwasser, S., Wigderson, A.: Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In: Simon, J. (ed.) Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA, pp. 1–10. ACM (1988). https://doi.org/10.1145/62212.62213
8. Berman, P., Garay, J.A., Perry, K.J.: Towards optimal distributed consensus (extended abstract). In: 30th Annual Symposium on Foundations of Computer Science, Research Triangle Park, North Carolina, USA, 30 October - 1 November 1989, pp. 410–415. IEEE Computer Society (1989).https://doi.org/10.1109/SFCS.1989.63511

9. Brazitikos, K., Zikas, V.: General adversary structures in byzantine agreement and multi-party computation with active and omission corruption. Cryptology ePrint Archive, Paper 2024/209 (2024). https://eprint.iacr.org/2024/209
10. Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. In: 42nd Annual Symposium on Foundations of Computer Science, FOCS 2001, 14-17 October 2001, Las Vegas, Nevada, USA, pp. 136–145. IEEE Computer Society (2001). https://doi.org/10.1109/SFCS.2001.959888
11. Canetti, R., Damgaard, I., Dziembowski, S., Ishai, Y., Malkin, T.: On adaptive vs. non-adaptive security of multiparty protocols. In: Pfitzmann, B. (ed.) EUROCRYPT 2001. LNCS, vol. 2045, pp. 262–279. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-44987-6_17
12. Chaum, D., Crépeau, C., Damgård, I.: Multiparty unconditionally secure protocols (extended abstract). In: Simon, J. (ed.) Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA, pp. 11–19. ACM (1988). https://doi.org/10.1145/62212.62214
13. Cohen, R., Coretti, S., Garay, J., Zikas, V.: Probabilistic termination and composability of cryptographic protocols. In: Robshaw, M., Katz, J. (eds.) CRYPTO 2016. LNCS, vol. 9816, pp. 240–269. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53015-3_9
14. Cohen, R., Garay, J.A., Zikas, V.: Completeness theorems for adaptively secure broadcast. In: Handschuh, H., Lysyanskaya, A. (eds.) Advances in Cryptology - CRYPTO 2023 - 43rd Annual International Cryptology Conference, CRYPTO 2023, Santa Barbara, CA, USA, August 20-24, 2023, Proceedings, Part I. Lecture Notes in Computer Science, vol. 14081, pp. 3–38. Springer (2023).https://doi.org/10.1007/978-3-031-38557-5_1
15. Dolev, D., Dwork, C., Waarts, O., Yung, M.: Perfectly secure message transmission. J. ACM **40**(1), 17–47 (1993). https://doi.org/10.1145/138027.138036
16. Eldefrawy, K., Loss, J., Terner, B.: How byzantine is a send corruption? In: Ateniese, G., Venturi, D. (eds.) Applied Cryptography and Network Security - 20th International Conference, ACNS 2022, Rome, Italy, June 20-23, 2022, Proceedings. Lecture Notes in Computer Science, vol. 13269, pp. 684–704. Springer (2022). https://doi.org/10.1007/978-3-031-09234-3_34
17. Feldman, P., Micali, S.: Optimal algorithms for byzantine agreement. In: Simon, J. (ed.) Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA, pp. 148–161. ACM (1988). https://doi.org/10.1145/62212.62225
18. Fitzi, M., Hirt, M., Maurer, U.: Trading correctness for privacy in unconditional multi-party computation. In: Krawczyk, H. (ed.) CRYPTO 1998. LNCS, vol. 1462, pp. 121–136. Springer, Heidelberg (1998). https://doi.org/10.1007/BFb0055724
19. Fitzi, M., Hirt, M., Maurer, U.: General adversaries in unconditional multi-party computation. In: Lam, K.-Y., Okamoto, E., Xing, C. (eds.) ASIACRYPT 1999. LNCS, vol. 1716, pp. 232–246. Springer, Heidelberg (1999). https://doi.org/10.1007/978-3-540-48000-6_19
20. Hadzilacos, V.: Issues of Fault Tolerance in Concurrent Computations (Databases, Reliability, Transactions, Agreement Protocols, Distributed Computing). Ph.D. thesis, Harvard University, USA (1985), aAI8520209
21. Hirt, M., Maurer, U.M.: Complete characterization of adversaries tolerable in secure multi-party computation (extended abstract). In: Burns, J.E., Attiya, H. (eds.) Proceedings of the Sixteenth Annual ACM Symposium on Principles of Distributed Computing, Santa Barbara, California, USA, August 21-24, 1997, pp. 25–34. ACM (1997). https://doi.org/10.1145/259380.259412

22. Hirt, M., Maurer, U.M.: Player simulation and general adversary structures in perfect multiparty computation. J. Cryptol. **13**(1), 31–60 (2000). https://doi.org/10.1007/s001459910003
23. Hirt, M., Zikas, V.: Adaptively secure broadcast. In: Gilbert, H. (ed.) Advances in Cryptology - EUROCRYPT 2010, 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Monaco / French Riviera, May 30 - June 3, 2010. Proceedings. Lecture Notes in Computer Science, vol. 6110, pp. 466–485. Springer (2010). https://doi.org/10.1007/978-3-642-13190-5_24
24. Ito, M., Saito, A., Nishizeki, T.: Secret sharing scheme realizing general access structure. Electron. Commun. Japan (Part III: Fundamental Electronic Science) **72**(9), 56–64 (1989)
25. Katz, J., Maurer, U., Tackmann, B., Zikas, V.: Universally composable synchronous computation. In: Sahai, A. (ed.) TCC 2013. LNCS, vol. 7785, pp. 477–498. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-36594-2_27
26. Koo, C.: Secure computation with partial message loss. In: Halevi, S., Rabin, T. (eds.) Theory of Cryptography, Third Theory of Cryptography Conference, TCC 2006, New York, NY, USA, March 4-7, 2006, Proceedings. Lecture Notes in Computer Science, vol. 3876, pp. 502–521. Springer (2006). https://doi.org/10.1007/11681878_26
27. Lamport, L., Shostak, R.E., Pease, M.C.: The byzantine generals problem. ACM Trans. Program. Lang. Syst. **4**(3), 382–401 (1982). https://doi.org/10.1145/357172.357176
28. Lindell, Y., Pinkas, B.: A proof of security of yao's protocol for two-party computation. J. Cryptol. **22**(2), 161–188 (2009). https://doi.org/10.1007/s00145-008-9036-8
29. Loss, J., Stern, G.: Zombies and ghosts: Optimal byzantine agreement in the presence of omission faults. In: Rothblum, G.N., Wee, H. (eds.) Theory of Cryptography - 21st International Conference, TCC 2023, Taipei, Taiwan, November 29 - December 2, 2023, Proceedings, Part IV. Lecture Notes in Computer Science, vol. 14372, pp. 395–421. Springer (2023).https://doi.org/10.1007/978-3-031-48624-1_15
30. Maurer, U.M.: Secure multi-party computation made simple. In: Cimato, S., Galdi, C., Persiano, G. (eds.) Security in Communication Networks, Third International Conference, SCN 2002, Amalfi, Italy, September 11-13, 2002. Revised Papers. Lecture Notes in Computer Science, vol. 2576, pp. 14–28. Springer (2002). https://doi.org/10.1007/3-540-36413-7_2
31. Parvédy, P.R., Raynal, M.: Uniform agreement despite process omission failures. In: 17th International Parallel and Distributed Processing Symposium (IPDPS 2003), 22-26 April 2003, Nice, France, CD-ROM/Abstracts Proceedings, p. 212. IEEE Computer Society (2003). https://doi.org/10.1109/IPDPS.2003.1213388
32. Pease, M.C., Shostak, R.E., Lamport, L.: Reaching agreement in the presence of faults. J. ACM **27**(2), 228–234 (1980). https://doi.org/10.1145/322186.322188
33. Perry, K.J., Toueg, S.: Distributed agreement in the presence of processor and communication faults. IEEE Trans. Software Eng. **12**(3), 477–482 (1986). https://doi.org/10.1109/TSE.1986.6312888
34. Raynal, M.: Consensus in synchronous systems: A concise guided tour. In: 9th Pacific Rim International Symposium on Dependable Computing (PRDC 2002), 16-18 December 2002, Tsukuba-City, Ibarski, Japan, pp. 221–228. IEEE Computer Society (2002). https://doi.org/10.1109/PRDC.2002.1185641

35. Yao, A.C.: Protocols for secure computations (extended abstract). In: 23rd Annual Symposium on Foundations of Computer Science, Chicago, Illinois, USA, 3-5 November 1982, pp. 160–164. IEEE Computer Society (1982). https://doi.org/10.1109/SFCS.1982.38
36. Zikas, V.: Generalized corruption models in secure multi-party computation. Ph.D. thesis, ETH Zurich (2010). https://d-nb.info/1005005729
37. Zikas, V., Hauser, S., Maurer, U.: Realistic failures in secure multi-party computation. In: Reingold, O. (ed.) TCC 2009. LNCS, vol. 5444, pp. 274–293. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-00457-5_17

Secure Computation with Parallel Calls to 2-Ary Functions

Varun Narayanan[1(✉)], Shubham Vivek Pawar[2], and Akshayaram Srinivasan[3]

[1] University of California, Los Angeles, USA
varunnkv@gmail.com
[2] Royal Holloway, University of London, Egham, UK
shubham.pawar.2022@live.rhul.ac.uk
[3] University of Toronto, Toronto, Canada
akshayaram@cs.toronto.edu

Abstract. Reductions are the workhorses of cryptography. They allow constructions of complex cryptographic primitives from simple building blocks. A prominent example is the non-interactive reduction from securely computing a "complex" function f to securely computing a "simple" function g via randomized encodings.

Prior work equated simplicity with functions of small degree. In this work, we consider a different notion of simplicity where we require g to only take inputs from a small number of parties. In other words, we want the arity of g to be as small as possible.

In more detail, we consider the problem of reducing secure computation of arbitrary functions to secure computation of functions with arity two (two is the minimal arity required to compute non-trivial functions). Specifically, we want to compute a function f via a protocol that makes parallel calls to 2-ary functions. We want this protocol to be secure against malicious adversaries that could corrupt an arbitrary number of parties. We obtain the following results:

- **Negative Result:** We show that there exists a degree-2 polynomial p such that no protocol that makes parallel calls to 2-ary functions can compute p with statistical security with abort.
- **Positive Results:** We give two ways to bypass the above impossibility result.
  1. **Weakening the Security Notion.** We show that every degree-2 polynomial can be computed with statistical privacy with knowledge of outputs (PwKO) by making parallel calls to 2-ary functions. Privacy with knowledge of outputs is weaker than security with abort.
  2. **Computational Security.** We prove that for every function f, there exists a protocol for computing f that makes parallel calls to 2-ary functions and achieves security with abort

V. Narayanan was supported by NSF Grants CNS-2246355, CCF-2220450, and CNS-2001096; S. V. Pawar was supported by EPSRC through grant number EP/S021817/1; A. Srinivasan was supported in part by a NSERC Discovery grant RGPIN-2024-03928.

E. Boyle and M. Mahmoody (Eds.): TCC 2024, LNCS 15367, pp. 234–265, 2025.
https://doi.org/10.1007/978-3-031-78023-3_8

against computationally-bounded adversaries. The security of this protocol relies on the existence of semi-honest secure oblivious transfer.

- **Applications:** We give connections between this problem and the task of reducing the encoding complexity of Multiparty Randomized Encodings (MPRE) (Applebaum, Brakerski, and Tsabary, TCC 2018). Specifically, we show that under standard computational assumptions, there exists an MPRE where the encoder can be implemented by an NC^0 circuit with constant fan-out.
- **Extensions:** We explore this problem in the honest majority setting and give similar results assuming one-way functions. We also show that if the parties have access to 3-ary functions then we can construct a computationally secure protocol in the dishonest majority setting assuming one-way functions.

1 Introduction

A key research direction in theoretical cryptography is to construct complex cryptographic primitives from simple building blocks. This direction has achieved remarkable success and has led to several fundamental results such as constructing pseudorandom generators [HILL99] and zero-knowledge proofs [GMW86] from one-way functions, secure multiparty computation protocols from oblivious transfer [GMW87, Kil88, IPS08], and CCA-secure encryption from injective trapdoor functions [HKW20]. One such foundational result is a non-interactive reduction from securely computing a "complex" function f to securely computing a "simple" function g. This is achieved using randomized encodings [Yao86, IK00, AIK04] and this approach has been instrumental in constructing round-optimal secure computation protocols [Yao86, GS18, BL18].

Prior work aimed to minimize the degree of g as much as possible and they equated simplicity with functions computable by constant degree polynomials. [IK00, AIK04] showed that under standard cryptographic assumptions, securely computing any function can be reduced to securely computing a degree-3 function. [BL18, GS18, GIS18, ACGJ18, ABT18, ABT19, ACGJ19] (using the notion of multiparty randomized encoding) showed that one can further reduce the (effective) degree to 2 if we allow local pre-processing of the inputs by the parties.

Our Work. We consider a different notion of simplicity. Specifically, we want to reduce the task of securely computing some complex n-party function f to securely computing, in parallel, a set of functions where each function g_S takes inputs only from a proper subset S of the parties (in other words, the arity $|S|$ of g_S is less than n) and delivers the output to *all parties*. It is trivial to see that most multiparty party functions—for instance, n-party AND—cannot be computed, using a single function of arity less than n. So, we need to necessarily allow parallel calls to several such functions. Our goal is to minimize their arity as much as possible.

Our Model. To be more precise, let f be an n-party function and, for a collection $\mathcal{S}$ of k-sized subsets of $[n]$, let $\{g_S\}_{S\in\mathcal{S}}$ be a set of functions, where each g_S is an arity-k function taking inputs from parties in the set S and delivers output to all parties. We say that f securely reduces to $\{g_S\}$ if there is a tuple of randomized algorithms $(\mathsf{Enc}, \mathsf{Dec})$ with the following syntax and satisfying the following two properties.

- **Syntax:** Enc is a randomized function that takes in the index i of a party, its private input x_i and a subset $S \in \mathcal{S}$ such that $i \in S$ and outputs $x_{i,S}$. Dec takes in $\left\{g_S(\{x_{i,S}\}_{i\in S})\right\}_{S\in\mathcal{S}}$ and outputs y.[1]
- **Correctness:** We want the output y of Dec to be $f(x_1, \ldots, x_n)$.
- **Security:** Even if an arbitrary subset of the parties get corrupted by an adversary, we want $\left\{g_S(\{x_{i,S}\}_{i\in S})\right\}_{S\in\mathcal{S}}$ to only reveal the output $f(x_1, \ldots, x_n)$ and nothing else about the private inputs of the uncorrupted parties.

Minimal Arity. If we consider reduction to arity-1 functions, then it is easy to see that the only functions that can be securely computed are of the form $(h_1(x_1), h_2(x_2), \ldots, h_n(x_n))$. Hence, to be useful, we need functions with arity 2 or higher.

Why is arity important? Consider the task of securely computing some complex multiparty function f involving a large number of parties. In such a case, it is unreasonable to expect all the parties to be online throughout the entire protocol execution. However, if we can break this complex computation to simple components of constant arity, then we only need a constant number of parties to be online for computing every component. Further, the computation of each component is not dependent on the outputs from other components (since we only make parallel calls to the functions). Once all the components are computed, the parties can apply the local decoding procedure to learn the output of the function f.

Case of Semi-honest Adversaries. In the semi-honest model, there is an easy transformation from securely computing a degree-d function (that has an one-to-one correspondence with degree-d polynomials) to making parallel calls to d-ary functions. Indeed, we can compute each monomial of the degree-d polynomial using an d-ary function and add additive secret shares of 0 to each monomial to ensure that only the output of the polynomial is revealed. Hence, the existing results about the completeness of degree-2 functions [GS18, GIS18, ACGJ18, ABT18, ABT19, ACGJ19] (with local pre-processing) can be extended to our

[1] We model Dec in this way so that a party that does not contribute any private input could still learn the output. This is analogous to the notion of output client in the client-server MPC protocol literature [DI05] and is equivalent to considering publicly decodable transcripts [ABG+20].

model of making parallel calls to 2-ary functions. Somewhat surprisingly, these results do not extend to the malicious setting and this is the focus of this work.

1.1 Our Results

We explore the problem of securely computing n-party functions against malicious adversaries by making parallel calls to 2-ary functions. As argued earlier, two is the minimum arity needed to compute non-trivial functions. We provide both positive and negative results and connect this problem to reducing the encoding complexity of multiparty randomized encodings [ABT18]. We also provide a couple of extensions to these results by (i) considering the honest majority setting, and (ii) allowing the arity to be larger than two.

Impossibility of Statistical Security with Abort. Our first result shows that if we want statistical security, then it is impossible to compute even degree-2 polynomials. Specifically, we give a 3-party function f (that is computable by a degree-2 polynomial) such that no protocol that makes parallel calls to 2-ary functions achieves statistical security with abort. Formally,

Theorem 1.1. *There exists a 3-party function f that can be computed using a degree-2 polynomial such that no 3-party protocol making parallel calls to 2-ary functions can compute f with statistical security with abort against a malicious adversary corrupting two parties.*

In the case of three parties with two corruptions, security with abort is equivalent to security with selective abort. Therefore, our impossibility result also rules out constructions with selective abort.

Positive Results. We give two ways to overcome the above impossibility.

1. We show that if we relax the security requirement to privacy with knowledge of outputs (PwKO) [IKP10], then every degree-2 polynomial can be securely computed with statistical PwKO by making parallel calls to 2-ary functions. PwKO relaxation guarantees that the adversary does not learn anything about the private inputs of the honest parties except the output of the function but after learning the output, it can force the honest parties to output an arbitrary value of its choice. Formally, this is modelled by having the ideal functionality first give the output of the function to the simulator and the simulator sends some arbitrary value as the output to the honest parties. All the honest parties will output the value provided by the simulator.

Theorem 1.2. *For every n-party function f computable using degree-2 polynomials, there exists a protocol for computing f that makes parallel calls to 2-ary functions and achieves statistical PwKO against a malicious adversary that could corrupt an arbitrary number of parties.*

2. A second approach to bypass the impossibility result is to relax the security to be computational. We show how to extend Theorem 1.2 to securely computing arbitrary circuits and achieve security with unanimous abort.

Theorem 1.3. *Assume the existence of a semi-honest secure oblivious transfer protocol. For every n-party function f, there exists a protocol for computing f that makes parallel calls to 2-ary functions and achieves security with unanimous abort against a computationally-bounded malicious adversary that could corrupt an arbitrary number of parties.*

Application: Reducing the Complexity of MPRE. Theorem 1.3 has an interesting application to reducing the complexity of the encoding function of Multiparty Randomized Encoding (MPRE) [GS17, ABT18, ABT19]. MPRE is the analog of randomized encodings for distributed computation protocols. In a bit more detail, MPRE for computing a multiparty function f comprises of n pre-processing functions $h_1, \ldots, h_n$ along with an encoder Enc' and a decoder Dec'. The party P_i first applies the local pre-processing function h_i on its private input and randomness. After this, we apply the encoding function Enc' on the pre-processed values. The decoding function Dec' takes in the output of the encoder and computes the output of f applied on the private inputs of all the parties. For security, we require that even if an adversary corrupts an arbitrary number of parties, the output of the encoder only reveals the output of f and nothing else about the inputs of the honest parties.

An important research direction in the study of randomized encodings is to minimize the encoding complexity as much as possible [AIK04, AIK07]. Previous results [GS18, ABT18, GIS18] gave constructions of MPRE where the encoder could be implemented in NC^0. However, the fan-out of the encoder circuit grew linearly with the number of parties. But using Theorem 1.3, we can construct an MPRE where the encoding can be done in NC^0 with constant fan-out. In other words, the encoder has constant input and output locality. This encoder is optimal in the sense that its has constant fan-in, constant fan-out, and constant depth.

This result is obtained directly from the above theorem by replacing the 2-ary functions with an existing MPRE secure against malicious adversaries [GS18]. This MPRE construction is based on the existence of a two-round, malicious-secure oblivious transfer protocol. Since this MPRE computes a two-party functionality, its fan-out is constant and hence, the overall fan-out of the encoder is constant. This is illustrated in Fig. 1. As a result, we get the following corollary.

Corollary 1.4. *Assume the existence of a two-round oblivious transfer protocol secure against malicious adversaries. Then, there is a reduction from securely computing any multiparty function f against malicious adversaries to securely computing a NC^0 function with constant fan-out against malicious adversaries.*

Extensions. We provide the following extensions.

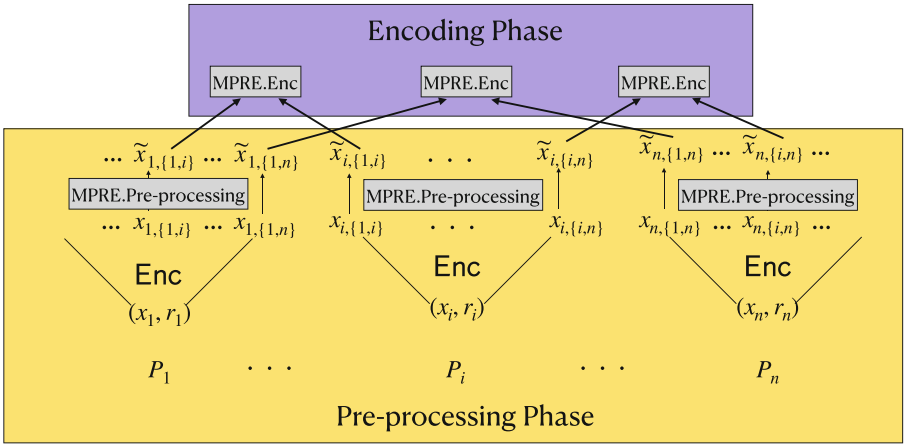


Fig. 1. Construction of MPRE where the encoder has constant input and output locality. Here, Enc denotes the encoder function from Theorem 1.3 and MPRE is for computing the 2-ary function from Theorem 1.3.

– **Honest Majority:** When security is required in the presence of an honest majority, we can replace the assumption of semi-honest secure oblivious transfer with the weaker assumption of one-way function.

Theorem 1.5. *Assume the existence of one-way functions. For every n-party function f, there exists a protocol for computing f that makes parallel calls to 2-ary functions and achieves security with unanimous abort against a computationally-bounded malicious adversary that could corrupt a strict minority of parties.*

– **3-ary functions.** We further pursue the objective of realizing computational security with unanimous abort against an arbitrary number of malicious corruptions based only on the existence of one-way functions. We show that this is possible if the parties have access to 3-ary instead of 2-ary functions.

Theorem 1.6. *Assume the existence of one-way functions. For every n-party function f, there exists a protocol for computing f that makes parallel calls to 3-ary functions and achieves security with unanimous abort against a computationally-bounded malicious adversary that could corrupt an arbitrary number of parties.*

1.2 Related Work

Fitzi et al. [FGMO01] considered the problem of constructing secure computation protocols in the dishonest majority setting with full security (i.e., guaranteed output delivery) with help of functions that take inputs from less than n parties. Without the help of additional functions, achieving fairness (which is weaker

than full security) is impossible [Cle86]. Fitzi et al. showed a negative result that it is impossible to achieve full security with access to functions of arity less than n. See [IPP+22] for a detailed discussion on the works constructing fully-secure protocols with calls to functions with arity n.

Applebaum and Goel [AG21] based on Baum et al.'s result [BOSS20] gave a black-box protocol for computing any multiparty function by making parallel calls to constant degree functions and satisfying security with identifiable abort. This constant degree function takes inputs from all the parties. Their results do not extend to our setting where we restrict the function to take inputs only from two parties.

Applebaum et al. [ABG+20] showed that if we restrict our problem to only include 2-ary functions that give outputs to the two parties that provide inputs, then it is impossible to achieve statistical security even against semi-honest adversaries. Specifically, they gave a 3-party function that cannot be computed by any protocol that makes parallel calls to such 2-ary functions with statistical security against semi-honest adversaries.

2 Technical Overview

In Sect. 2.1, we give the main intuition behind our impossibility result (see Theorem 1.1). In Sect. 2.2, we show that if one relaxes the security requirement to privacy with knowledge of outputs [IKP10], then every degree-2 polynomial can be computed with parallel calls to 2-ary functions (see Theorem 1.2). In Sect. 2.3, we explain how to securely compute arbitrary circuits with (unanimous) abort by relaxing to computational security. This assumes the existence of semi-honest secure oblivious transfer (see Theorem 1.3).

2.1 Impossibility of Achieving Statistical Security

The key intuition behind the impossibility result is that a corrupt party can send inconsistent inputs in its interaction (via the 2-ary functions) with two different honest parties. This would allow the adversary to learn additional information about the private inputs of the honest parties that is not learnable in the ideal world. However, formalizing this intuition requires great care as we need to choose an appropriate function for which the impossibility holds and also provide a formal attack that works against any protocol. We elaborate on this below.

Consider the 3-party function f that takes x_1, x_2, and x_3 belonging to $\{0, 1, 2\}$ from P_1, P_2 and P_3, respectively and outputs 1 iff $x_1 + x_2 = x_3 \bmod 3$. We will first argue that f cannot be computed by making parallel calls to 2-ary functions. We will then show how to extend this impossibility to a function that is computable by a degree-2 polynomial.

Towards a contradiction, assume that there exists a non-interactive protocol $\Pi = (\mathsf{Enc}, \{\mathcal{O}_{\{1,2\}}, \mathcal{O}_{\{2,3\}}, \mathcal{O}_{\{1,3\}}\}, \mathsf{Dec})$ that securely computes f against malicious adversaries with abort. Here, $\mathcal{O}_{\{i,j\}}$ denotes the 2-ary function that is invoked between parties P_i and P_j for each distinct $i, j \in [3]$.

Consider an adversary $\mathcal{A}_{1,2}$ that corrupts P_1 and P_2, and makes the calls to $\mathcal{O}_{\{1,3\}}$ and $\mathcal{O}_{\{2,3\}}$ by honestly emulating P_1 and P_2 after setting their inputs to x_1 and x_2 respectively. Let $y_{\{1,3\}}$ and $y_{\{2,3\}}$ be the outputs of $\mathcal{O}_{\{1,3\}}$ and $\mathcal{O}_{\{2,3\}}$ respectively. At this point, the adversary can learn $f(x_1, x_2, x_3)$ without even invoking $\mathcal{O}_{\{1,2\}}$. This can be obtained by honestly computing the output of $\mathcal{O}_{\{1,2\}}$ in its head and applying the decoder on the outputs of the functions. The output is guaranteed to be $f(x_1, x_2, x_3)$ by correctness of Π.

The crucial lemma in the impossibility is that for any $y'_{\{1,2\}}$ in the range of $\mathcal{O}_{\{1,2\}}$, the output of $\mathsf{Dec}(y'_{\{1,2\}}, y_{\{2,3\}}, y_{\{1,3\}})$ is either $\perp$ or $f(x_1, x_2, x_3)$. Otherwise, the adversary can simultaneously learn $f(x_1, x_2, x_3)$ (by computing this in its head) and force honest P_3 to output a value other than $\perp$ or $f(x_1, x_2, x_3)$ (by sending $y'_{\{1,2\}}$ as the output of $\mathcal{O}_{\{1,2\}}$). This is impossible in the ideal world, thereby, contradicting security of Π.

Appealing to another adversary $\mathcal{A}_{1,3}$ that corrupts P_1 and P_3, and behaves analogously to $\mathcal{A}_{1,2}$, we can argue that, for any $y'_{\{1,3\}}$ in the range of $\mathcal{O}_{\{1,3\}}$, the output of $\mathsf{Dec}(y_{\{1,2\}}, y_{\{2,3\}}, y'_{\{1,3\}})$ is either $\perp$ or $f(x_1, x_2, x_3)$, when $y_{\{1,2\}}$ and $y_{\{2,3\}}$ are, respectively, the outputs of $\mathcal{O}_{\{1,2\}}$ and $\mathcal{O}_{\{2,3\}}$, when corrupt P_1 and P_3 behave honestly.

Finally, consider an adversary $\mathcal{A}_1$ that only corrupts P_1 and behaves as follows. $\mathcal{A}_1$ samples x_1 uniformly and invokes $\mathcal{O}_{\{1,2\}}$ by emulating P_1 with input x_1, whereas, it invokes $\mathcal{O}_{\{1,3\}}$ by emulating P_1 honestly but with its input set to $x_1 + 1 \bmod 3$. In essence, $\mathcal{A}_1$ provides inconsistent inputs while emulating $\mathcal{O}_{\{1,2\}}$ and $\mathcal{O}_{\{1,3\}}$.

Let $y_{\{1,2\}}$, $y_{\{2,3\}}$, and $y_{\{1,3\}}$ be the outputs of $\mathcal{O}_{\{1,2\}}$, $\mathcal{O}_{\{2,3\}}$ and $\mathcal{O}_{\{1,3\}}$, respectively in the real execution of the protocol with $\mathcal{A}_1$. $\mathcal{A}_1$ then chooses $y'_{\{1,3\}}$ and $y'_{\{1,2\}}$ in the range of $\mathcal{O}_{\{1,3\}}$ and $\mathcal{O}_{\{1,2\}}$ respectively such that $z' = \mathsf{Dec}(y'_{\{1,2\}}, y_{\{2,3\}}, y_{\{1,3\}}) \neq \perp$ and $z = \mathsf{Dec}(y_{\{1,2\}}, y_{\{2,3\}}, y'_{\{1,3\}}) \neq \perp$, and outputs (z, z'). Note that such $y'_{\{1,2\}}$ and $y'_{\{1,3\}}$ will always exist as we can set them to be the outputs of honest emulations of $\mathcal{O}_{\{1,2\}}$ and $\mathcal{O}_{\{1,3\}}$ with inputs $x_1 + 1$ and x_1 respectively. The adversary simply chooses the first string in the range of these two functions that produces a non-bot output. This is where we use the fact that the adversary is computationally unbounded as such strings might not be efficiently computable.

We claim that Π is not secure against $\mathcal{A}_1$. By our previous two lemmas, since z and z' are not $\perp$, it has to be the case that $z = f(x_1, x_2, x_3)$ and $z' = f(x_1 + 1, x_2, x_3)$. Thus, $\mathcal{A}_1$ simultaneously learns the output of the functions for two possible inputs of P_1. This allows $\mathcal{A}$ to learn the value of $x_3 - x_2$ with absolute certainty. However, when x_2 and x_3 are chosen uniformly at random, an ideal adversary will fail to guess $x_2 - x_3$ with constant probability and thus, we obtain a contradiction to the security of Π.

Making f to be degree-2. We observe that the same impossibility proof extends if we consider the function f that checks if $x_1 + x_2 + x_3 = 0$ where the $+$ operation is over $\mathsf{GF}(2^2)$. This can be expressed as a degree-2 polynomial over $\{0, 1\}$.

2.2 Securely Computing Degree-2 Polynomials with PwKO

The previous impossibility crucially relied on the honest party either outputting the correct function output or $\perp$. However, this impossibility result does not extend to the case where the adversary could force the honest party to output an arbitrary value. Therefore, there is hope of obtaining a protocol that satisfies statistical privacy with knowledge of outputs (PwKO). However, such a protocol is highly non-trivial to construct and it needs new techniques. We now explain our approach to construct such a protocol for computing arbitrary degree-2 polynomials.

For simplicity, let's assume that each party P_i gets a finite field element x_i as its private input. Let $p(\cdot)$ be a degree-2 polynomial and the parties want to compute $p(x_1, \ldots, x_n) = \sum_{i,j \in [n]} c_{i,j} \cdot x_i \cdot x_j$. Our goal is to design a protocol for computing p that makes parallel calls to 2-ary functions and achieves PwKO against a malicious adversary that corrupts an arbitrary subset of the parties. In the rest of the overview, we will use $\mathcal{O}_{\{i,j\}}$ to denote the function that is computed using P_i and P_j's inputs.[2]

The starting point of our construction is the semi-honest secure protocol for computing p with parallel calls to 2-ary functions. In this semi-honest secure protocol, we define $\mathcal{O}_{\{i,j\}}$ to take $(x_i, s_i[j])$ from P_i and $(x_j, s_j[i])$ from party P_j and to output $c_{i,j} \cdot x_i \cdot x_j + s_i[j] + s_j[i]$ to every party. For every $i \in [n]$, if P_i chooses $\{s_i[j]\}_{j \in [n]}$ as a random secret sharing of 0, then the parties can add the outputs of all the 2-ary functions to obtain $p(x_1, \ldots, x_n)$. The security follows since $\{s_i[j]\}_{j \in [n]}$ are all random subject to their sum being 0 and thus, only $p(x_1, \ldots, x_n)$ is revealed. The key challenge is to extend this protocol to be secure against malicious adversaries.

If we analyze the protocol a bit more carefully, we realize that a malicious adversary can only mount two kinds of attacks:

- **Sending inconsistent inputs.** An adversary corrupting P_i could send $(x_i, \cdot)$ to $\mathcal{O}_{\{i,j\}}$ and $(x'_i, \cdot)$ (where $x_i \neq x'_i$) to $\mathcal{O}_{\{i,j'\}}$ for two different parties j, j'.
- **Generating bad secret shares.** An adversary that is corrupting party P_i could generate $\{s_i[j]\}_{j \in [n]}$ as secret shares of a value other than 0.

Let's assume for now that the adversary is restricted to sending consistent private inputs to each 2-ary function. In other words, it is disallowed from mounting the first attack strategy explained above. However, note that this adversary is still allowed to generate $\{s_i[j]\}_{j \in [n]}$ as shares of a value other than 0. If that is the case, we argue that the above protocol already satisfies PwKO. At a high-level, if the real-world adversary generates $\{s_i[j]\}_{j \in [n]}$ to be shares of a value other than 0, then it can be shown that this attack corresponds to adding an offset to the actual output. Thus, we can have the simulator (in the ideal world) to add the same offset to the output obtained from the ideal functionality and make all the honest parties obtain this modified output.

[2] For simplicity, we allow each pair of parties to potentially invoke a different function $\mathcal{O}_{\{i,j\}}$. However, this can be easily modified to compute a single function g that takes (i, j) as additional input.

The next step to consider is what happens if the adversary is allowed to send inconsistent inputs. Can the same protocol be proved to satisfy PwKO? We explain that this is not the case. First, observe that if the adversary sends x_i to $\mathcal{O}_{\{i,j\}}$ and x_i' to $\mathcal{O}_{\{i,j'\}}$, then the offset that is added to the real output is given by $c_{i,j} \cdot (x_i' - x_i) \cdot x_{j'}$. However, this offset depends on the private input of $P_{j'}$ and this is disallowed as per the PwKO definition. Specifically, the adversary is only allowed to set the output received by the honest parties based on the output of the function and it cannot depend on the private inputs of the honest parties in any other way. Therefore, the above protocol as such does not satisfy PwKO. Hence, we need a mechanism to force the adversary to give consistent inputs to every 2-ary function.

Key Idea. Suppose we construct a protocol that satisfies the following two properties.

1. If every corrupted party P_i uses consistent inputs with every honest party, i.e., P_i sends the same input x_i to every $\mathcal{O}_{\{i,j\}}$ where P_j is honest, then we want all the honest parties to compute the output of the polynomial subject to the adversary adding some offset.
2. If a corrupted party sends different x_i and $x_{i'}$ to $\mathcal{O}_{\{i,j\}}$ and $\mathcal{O}_{\{i,j'\}}$ where P_j and $P_{j'}$ are honest, then we want the adversary not to learn any information about the private inputs of the honest parties. Specifically, we want the outputs of all the 2-ary functions involving an honest party to be random.

We argue that the above two properties are sufficient to show PwKO.

- If adversary sends consistent inputs in each execution of a 2-ary function with an honest party, then the adversary can only send shares that do not add up to 0 or cheat in the executions of 2-ary functions that involve only corrupt parties. Both these attacks can be translated to the adversary adding some offset to the output of the polynomial evaluation and hence, our protocol satisfies PwKO.
- If the adversary sends inconsistent inputs to two different honest parties, we need to show that the adversary learns no information about the private inputs of the honest parties and the output obtained by the honest parties is independent of their private inputs. The first property follows directly as the outputs of each 2-ary function involving an honest party is random. To argue the second property, note that the outputs obtained by the honest parties depend only on the random values output by the 2-ary functions involving at least one honest party and the values provided by the 2-ary functions involving two corrupt parties. These are independent of the private inputs of the honest parties as required by PwKO.

However, on close observation, we realize that it is highly non-trivial to satisfy the second condition. In particular, even if the adversary cheats by sending inconsistent inputs to any pair of honest parties, then we want this cheating to be detected in all other 2-ary functions involving at least one honest party and

the adversary should get no information about these outputs. This is especially challenging since the 2-ary functions do not have any shared randomness and all the functions are invoked in parallel.

Construction. To construct such a protocol, we tightly couple the outputs of each 2-ary function. This coupling ensures that even if the output of a single 2-ary function call gets erased (or equivalently, switched to random), this creates a "domino effect" and the outputs of all the 2-ary function calls involving at least one honest party get erased (i.e., switched to random). This allows us to argue the second condition. Let us explain how this domino effect is created.

We first design a special conditional disclosure of secrets (CDS) protocol that switches the output of $\mathcal{O}_{\{j,j'\}}$ to random if a corrupt party P_i sends inconsistent inputs to $\mathcal{O}_{\{i,j\}}$ and $\mathcal{O}_{\{i,j'\}}$ where P_j and $P_{j'}$ are honest. If this happens, observe that $s_j[j']$ and $s_{j'}[j]$ are erased from the adversary's view. This means that each $\{s_j[k]\}_{k\neq j'}$ and $\{s_{j'}[k]\}_{k\neq j}$ is randomly distributed and hence, the outputs of each 2-ary function $\mathcal{O}_{\{j,k\}}$ for every $k \neq j'$ and $\mathcal{O}_{\{j',k\}}$ for $k \neq j$ is randomly distributed. Thus, for every honest party P_h, $s_h[j]$ and $s_h[j']$ get erased from the adversary's view and we can continue the above argument to switch the output of each 2-ary function's output that involves at least one honest party to random.

CDS Protocol. A key technical contribution of this work is a construction of a CDS protocol that switches the output of $\mathcal{O}_{\{j,j'\}}$ if P_i sends inconsistent inputs to $\mathcal{O}_{\{i,j\}}$ and $\mathcal{O}_{\{i,j'\}}$.

We build this CDS protocol in two steps. We first build a protocol that has constant soundness error. That is, even if the corrupt P_i is sending inconsistent inputs to $\mathcal{O}_{\{i,j\}}$ and $\mathcal{O}_{\{i,j'\}}$, there is a constant probability that the output of $\mathcal{O}_{\{j,j'\}}$ is not switched to random. In the second step, we show how to bring down this soundness error to negligible.

CDS Functionality. For simplicity of notation, let us fix $j = 1$, $j' = 2$ and $i = 3$. Assume for simplicity that P_3's private input is a single bit and P_1 and P_2's private inputs used in the CDS protocol are two random bits y_1 and y_2 respectively. The CDS protocol ensures that if P_3's input to $\mathcal{O}_{\{1,3\}}$ and $\mathcal{O}_{\{2,3\}}$ are the same then, all the parties can recover $y_1 \oplus y_2$. Else, y_1, y_2 are hidden from the view of the adversary. We observe that this is sufficient to erase the output of $\mathcal{O}_{\{1,2\}}$ as we can add $y_1 \oplus y_2$ to the original output and send this modified output to each party. If the adversary sends consistent outputs, then every party can recover $y_1 \oplus y_2$ and use this to unmask the output of $\mathcal{O}_{\{1,2\}}$. To protect the privacy of honest parties, we additionally need that CDS protocol not to leak any information about x_3 if P_3 was honest.

Overview of Conditional Disclosure Protocol. We slightly abuse the notation and we describe our CDS protocol as making parallel calls to 2-ary functions $(\mathcal{O}_{\{1,2\}}, \mathcal{O}_{\{2,3\}}, \mathcal{O}_{\{1,3\}})$. In the final protocol, these CDS computations are baked into the 2-ary functions.

The following protocol will serve as the starting point for our construction. P_3 sends to both $\mathcal{O}_{\{1,3\}}$ and $\mathcal{O}_{\{2,3\}}$ the same degree-2 polynomial $q(x)$ over $\mathbb{F}_4$ sampled uniformly at random subject to $q(0) = x_3$. Inputs of P_1 and P_2 to $\mathcal{O}_{\{1,3\}}$ and $\mathcal{O}_{\{2,3\}}$ are, respectively, α_1 and α_2, which are uniform *non-zero* elements of $\mathbb{F}_4$. The output of $\mathcal{O}_{\{1,3\}}$ (resp. $\mathcal{O}_{\{2,3\}}$) is $(\alpha_1, q(\alpha_1))$ (resp. $(\alpha_2, q(\alpha_2))$). The function $\mathcal{O}_{\{1,2\}}$ is not used in this construction.

This is not a conditional disclosure protocol since it does not reveal $y_1 \oplus y_2$ in an honest execution. But, it hides (honest) P_3's input from colluding P_1, P_2. P_1 and P_2 learn the evaluation of $q(x)$ on at most two non-zero values. This perfectly hides x_3 from the collusion for the same reason a 2-private Shamir secret sharing is perfectly secure. Furthermore, if P_1 and P_2 are honest, with at least 1/9 probability, all honest parties detect if a corrupt P_3 uses distinct values of x_3 for computing inputs to $\mathcal{O}_{\{1,3\}}$ and $\mathcal{O}_{\{2,3\}}$. Because, in this case, P_3's inputs to $\mathcal{O}_{\{1,3\}}$ and $\mathcal{O}_{\{2,3\}}$, say $q_1(x)$ and $q_2(x)$, respectively, are distinct degree-2 polynomials (if degree is more than 2, the functions output $\perp$). Hence, there exists a non-zero $\alpha \in \mathbb{F}_4$ on which both these polynomials evaluate to distinct values.[3] With probability 1/9, (honest) P_1 and P_2 would choose $\alpha_1 = \alpha$ and $\alpha_2 = \alpha$, respectively, in which case, the function outputs reveal that $q_1(x) \neq q_2(x)$, resulting in all honest parties aborting the protocol.

Next, we tweak this protocol so that $y_1 \oplus y_2$ is revealed if the evaluation of the polynomial(s) on α_1 and α_2 are not in conflict, and is perfectly hidden if $\alpha_1 = \alpha_2$ and $q_1(\alpha_1) \neq q_2(\alpha_2)$. We achieve this as follows. We use $\mathcal{O}_{\{1,2\}}$ to output a list $\{y_1 \oplus y_2 \oplus \theta_p \oplus \phi_p\}_p$, where p ranges over all polynomials of degree at most 2 over $\mathbb{F}_4$, and θ_p and ϕ_p are one-time pads chosen by P_1 and P_2 to mask the secret. The behavior of $\mathcal{O}_{\{1,3\}}$ is modified such that, for each p, it gives up the value of θ_p only if $p(\alpha_1) = q_1(\alpha_1)$, where q_1 is the polynomial input by P_3 to $\mathcal{O}_{\{1,3\}}$. Similarly, for each p, $\mathcal{O}_{\{2,3\}}$ reveals ϕ_p for each p such that $p(\alpha_1) = q_2(\alpha_1)$, where q_2 is the polynomial input by P_3 to $\mathcal{O}_{\{2,3\}}$. If there exists a p such that $p(\alpha_1) = q_1(\alpha_1)$ and $p(\alpha_2) = q_2(\alpha_2)$, then by recovering θ_p and ϕ_p, the decoder can unmask $y_1 \oplus y_2$. Clearly such a p exists when P_3 is honest and, hence, $q_1 = q_2$. However, if a corrupt P_3 provides distinct polynomials q_1 and q_2, and P_1 and P_2 chose α_1 and α_2 that coincide with α such that $q_1(\alpha) \neq q_2(\alpha)$, then there is no p for which θ_p and ϕ_p are simultaneously revealed by $\mathcal{O}_{\{1,3\}}$ and $\mathcal{O}_{\{2,3\}}$, respectively. This ensures that, with a constant probability (specifically, 1/9), $y_1 \oplus y_2$ remains completely hidden from corrupt P_3, when P_1 and P_2 are honest. Note that, privacy of an honest P_3 is still preserved: revealing the set of polynomials that agree with a randomly chosen polynomial q that evaluates to x_3 on 0 on any non-zero α_1 and α_2 of P_1 and P_2's choosing does not reveal x_3.

This protocol has a clear flaw that allows a rushing P_3 to choose $q_1 \neq q_2$ without getting caught by honest parties: P_3 can learn α_1 chosen by P_1 from the output of $\mathcal{O}_{\{1,3\}}$. And, by looking at the polynomials that agree with q_1 on α_1, it adaptively chooses $q_2(x)$ such that $q_1(0) \neq q_2(0)$. Then, the parties will never detect that $q_1 \neq q_2$ irrespective of the value of α_2. This attack (in fact, all

[3] Note that any two distinct degree-2 polynomials can have the same evaluation on at most two points on the finite field.

attacks in general from P_3) can be circumvented by ensuring that α_1 and α_2 are hidden from the adversary until the outputs of all 2-ary functions are revealed. We achieve this by appropriately masking the variables in the output of the functions that are sensitive to the value of α_1 and α_2 and ensuring that these masks can be removed only when the outputs of all function calls are known.

Boosting Soundness. The above protocol ensures that if P_3 sends inconsistent input, then with 1/9 probability, $y_1 \oplus y_2$ remains hidden. To boost this probability to be very close to 1, we use the parallel repetition. In particular, we run the above protocol k times in parallel and wherein each repetition, P_3 chooses random polynomials q_1 and q_2 with the same constant term.[4]. We additionally set the secrets of P_1 and P_2 to be the XOR of the secrets used in each repetition. This construction guarantees that for an adversary to learn information about the secrets of P_1 and P_2, it should not get caught in each of the k repetitions. The probability of this event is upper bounded by $(\frac{8}{9})^k$ (which is negligible in k).

2.3 Relaxing to Computational Security

We now explain how to bootstrap the protocol for computing degree-2 polynomials with PwKO to computing arbitrary functions with stronger security guarantees. We do this by relaxing the security to be computational. Recall that the impossibility result only holds against computationally unbounded adversaries.

Let f be an arbitrary multiparty function that is computable by a poly-sized circuit. Consider an augmented functionality g that takes in (x_i, sk_i) from party P_i where sk_i is a signing key for a digital signature scheme. For each $i \in [n]$, g first computes $y = f(x_1, \ldots, x_n)$. It then signs y using sk_i for each $i \in [n]$ to obtain the signature σ_i. Finally, g outputs $(y, \sigma_1, \sigma_2, \ldots, \sigma_n)$.[5]

The parties use an effective degree-2 MPRE with *semi-malicious* security against arbitrary corruptions (see Definition 4.2) for computing the function g.[6] The parties use the previous protocol with PwKO to compute the pre-processing phase and the encoding function for the MPRE.[7] The parties also use the 2-ary functions to broadcast the verification key vk_i corresponding to sk_i. The parties then use the MPRE decoder to learn the output of g. The parties finally check

[4] We can modify the 2-ary function to output $\perp$ if the constant terms are not the same.

[5] A similar transformation was used in [IKSS22] to obtain security with unanimous abort from PwKO.

[6] Such an MPRE was constructed in [ABT18, GS18] assuming the existence of a semi-honest secure oblivious transfer.

[7] Note that we can extend the previous protocol to first apply a local function on the parties private inputs and then compute a degree-2 polynomial on the outputs of these local functions. This allows us to compute the pre-processing phase inside the 2-ary functions and thus, allowing us to rely on an MPRE that is secure against semi-malicious adversaries.

if each σ_i is a valid signature on y under the verification key vk_i. If any of the checks do not pass, the party aborts.

Note that from the security of the PwKO protocol, the adversary only learns the output of degree-2 polynomial computation on the pre-processed values and this corresponds to the output of the encoding function of the MPRE. From the MPRE security, this encoding can be generated only given the output of g, and the inputs and randomness of the corrupted parties. If the adversary sends some other value as the output to the honest parties, then it corresponds to sending a tampered MPRE encoding. Note that all the honest parties will get the same output $(y', \sigma'_1, \ldots, \sigma'_n)$ as the decoding for MPRE does not require any secret state. If $y \neq y'$, it follows from the security of the digital signature scheme that each honest party will abort. If $y = y'$, but some signature check does not pass, then once again all the honest parties will abort as the verification check is publicly computable. The only case where all the honest parties will output y is when $y = y'$ and all the verification checks pass.

Extensions. We observe that the same construction can be instantiated with a degree-2 MPRE with *semi-malicious* security in the honest majority setting to obtain a protocol secure against malicious adversaries that corrupt a minority of the parties. [ABT18][Theorem 1.2] constructed such an MPRE assuming one-way functions. The complexity of this construction is polynomial in the number of parties and the size of the circuit representing the function. Using this MPRE construction, we obtain a protocol in our model that computes any function with unanimous abort against a computationally-bounded malicious adversary corrupting a strict minority of the parties. The security of this construction relies on one-way functions. We also show that if the parties have access to 3-ary functions (instead of 2-ary), then we can rely on the MPRE construction from Beaver et al. [BMR90] to give an analogous result in the dishonest majority. The security of this protocol again relies on the existence of one-way functions.

3 Open Problems

We highlight some interesting problems left open by our work.

- **Minimizing Assumption in Dishonest Majority.** Assuming the existence of one-way functions, 3-ary functions can be used to achieve computational security with unanimous abort against an arbitrary number of corruptions. Similarly, 2-ary functions can be used to achieve the same against a strict minority of corruptions. We leave open the possibility of achieving security with abort with parallel calls to 2-ary functions in the dishonest majority setting based on one-way functions. This work also leaves open the (im)possibility of unconditional security with unanimous abort even with parallel calls to $(n-1)$-ary functions.
- **Using Private Decoders.** Our impossibility result relied on the decoder to be public, i.e., not have any secret state. One way to get around our

impossibility in the statistical setting is to resort to using private decoders (which take the secret state of a party as an additional input) and settling for security with selective abort, where the adversary may select the set of honest parties who are denied the output of the computation. [ABT18][Theorem 1.1] showed the existence of a perfectly secure degree-2 MPRE (hence, perfectly secure semi-malicious degree-2 MPRE) in the honest majority setting for any function that is computed by a branching program. Hence, using our CDS construction (which is secure against dishonest majority), we can securely compute any branching program with statistical PwKO when there is an honest majority. We can then use a standard transformation from PwKO to selective abort described in [IKP10]. This transformation uses a Message Authentication Code (MACs) and the private decoding is needed so that the parties can check the validity of the tags.
We leave open the possibility of achieving statistical security with selective abort for arbitrary corruptions using private decoding. An astute reader may have noticed that the construction sketched above gives rise to such a protocol if there exists a statistical degree-2 MPRE for arbitrary corruptions or a statistical degree-2 RE. Hence, the existence of such MPRE or RE–a long standing open problem–acts as a barrier to proving the impossibility of constructing such protocols with private decoding.

- **Achieving Identifiable Abort.** Our protocols do not satisfy identifiable abort security which guarantees that at least one of the corrupt parties can be detected by the honest parties whenever the protocol aborts. We leave open the problem of achieving identifiable abort in our model.
- **Guaranteed Output Delivery.** We also leave open the possibility of achieving guaranteed output delivery in our model. The existing results [FGMO01] imply this is impossible when there is a dishonest majority. However, the problem remains open when there is an honest majority.

4 Definitions

Notations. The set $\{1, 2, \ldots, m\}$ is denoted by $[m]$. A sequence $(x_{i_1}, \ldots, x_{i_m})$, where $(i_1, \ldots, i_m) = S$ will be denoted by $\{x_i\}_{i \in S}$; when S is clear from the context, simply write $\{x_i\}$. When $S \subseteq [m]$, we will sometimes abuse the notation to denote $f(x_1, \ldots, x_m)$ as $f(\{x_i\}_{i \in S}, \{x_i\}_{i \in [m] \setminus S})$.

We consider a secure computation model in which n parties securely compute a function by making parallel calls to 2-ary functions with no further interaction. Universally composable security (UC-security) against malicious adversaries, as conceived in [Can01], will be the focus of this work.

Let $\lambda \in \mathbb{N}$ be the security parameter. Let $n \in \mathbb{N}$ and let $P_1, \ldots, P_n$ be a set of distinct parties. Let f be an n-party functionality $f : \mathcal{X}_1 \times \ldots \times \mathcal{X}_n \to \mathcal{Y}$.

Definition 4.1. *An n-party protocol $\Pi = (\{\mathcal{O}_{\{i,j\}}\}, \mathsf{Enc}, \mathsf{Dec})$ for securely realizing a functionality f using parallel calls to 2-ary functions $\{\mathcal{O}_{\{i,j\}}\}$, where $\mathcal{O}_{\{i,j\}} : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$ for each $\{i, j\} \in \binom{[n]}{2}$, is described by an encoding function Enc and a decoding function Dec. Π proceeds as follows.*

- *Each party P_i has private input $x_i \in \mathcal{X}_i$. It samples private randomness r_i uniformly from $\{0,1\}^*$, and computes $y_{(i,j)} = \mathsf{Enc}(1^\lambda, (i,j), x_i, r_i)$ for each $j \neq i$.*
- *For each $\{i,j\} \in \binom{[n]}{2}$, P_i and P_j invoke the 2-ary function $\mathcal{O}_{\{i,j\}}$ with inputs $y_{(i,j)}$ and $y_{(j,i)}$, respectively, which delivers $z_{\{i,j\}} = \mathcal{O}_{\{i,j\}}(y_{(i,j)}, y_{(j,i)})$ to all parties.*
- *Finally, each P_k outputs $\mathsf{Dec}(1^\lambda, \{z_{\{i,j\}}\})$ and terminates. If output of P_k is $\perp$, we say that P_k has aborted the protocol.*

We say that Π securely computes f if, for any non-uniform polynomial time adversary $\mathcal{A}$ that corrupts an arbitrary subset $M \subset [n]$ of the parties, there exists an ideal world PPT simulator Sim such that, for every choice of inputs $\{x_i\}_{i\in[n]}$:

$$(\mathsf{View}_{\mathcal{A}}, \mathsf{out}) \approx_c \mathsf{Ideal}(1^\lambda, \mathsf{Sim}^{\mathcal{F}(\{x_i\}_{i\in[n]\setminus M}, \cdot)}, \{x_i\}_{i\in M}) \tag{1}$$

In the above, $\mathsf{View}_{\mathcal{A}}$ refers to the view of the adversary $\mathcal{A}$ (which includes the input, randomness, and the outputs of the function calls) and out refers to the output of all (honest) parties in the real execution of the protocol, which coincide as they use the same decoder. Ideal refers to the ideal execution of the protocol. This starts with the Sim run on $1^\lambda, \{x_i\}_{i\in M}$. $\mathcal{F}(\{x_i\}_{i\in[n]\setminus M}, \cdot)$ takes inputs $\{\widetilde{x}_i\}_{i\in M}$ from Sim and computes the output of f with the inputs of the honest parties being fixed to $\{x_i\}_{i\in[n]\setminus M}$ and the inputs of the corrupt parties being fixed to $\{\widetilde{x}_i\}_{i\in M}$. It delivers this output to Sim. If Sim sends an instruction to deliver the outputs to the honest parties, the functionality delivers the above computed output. Else, it asks the honest parties to abort. The output of Ideal corresponds to the output of Sim and the output of all the honest parties.

We can extend the above definition to consider computationally unbounded adversaries $\mathcal{A}$. In this case, we allow the simulator Sim and the distinguisher in Equation (1) to be unbounded. In this case, we say that the protocol satisfies statistical security.

Definition 4.1 can be generalized to n-party non-interactive protocols using k-ary functions ($k < n$) which take inputs from sets of k parties.

Definition 4.1 considers a strong security guarantee where all the honest parties either compute the output or all of them abort. This is known as *security with unanimous abort*. Some of the protocols we construct enjoy a weaker notion of security, which we define below:

Privacy with Knowledge of Output. A protocol is said to guarantee privacy with knowledge of outputs (PwKO) if the malicious adversary learns only the output of the function computation, and each honest party outputs a value that is chosen by the adversary for that party based (only) on the output of the function computation. This corresponds to realizing the functionality $\mathcal{F}^{\mathrm{PwKO}}$ which works exactly like $\mathcal{F}$ until it sends the output to Sim. However, after delivering the output to Sim, the functionality accepts an input $o \in \mathrm{codomain}(f) \cup \{\perp\}$ from Sim, and delivers o as output to all honest parties.

4.1 Multiparty Randomized Encoding

Ishai and Kushilevitz introduced the notion of randomizing polynomials in [IK00] which was later generalized to *Randomized Encoding (RE)* by Applebaum, Ishai and Kushilevitz in [AIK04]. A function f is encoded by a randomized function $\hat{f}$ if the output of $\hat{f}$ allows reconstruction of the output of f and nothing more. The function f is trivially a randomized encoding of itself. The utility of this notion is rooted in the observation that it is possible to obtain randomized encodings which are *simpler* than the function itself. A well studied notion of simplicity is the degree of the randomized encoding when the output of $\hat{f}$ is viewed as a multi-variate polynomial of $x_1, \ldots, x_n, r$, where $x_1, \ldots, x_n$ are the inputs to f and r is the randomness used in the encoding.

Multiparty randomized encoding (MPRE) was introduced by Applebaum, Brakerski and Tsabary in [ABT18] as a generalization of randomized encoding. We are intereseted in MPRE that remains secure against an arbitrary number of corruption, which we define below:

Definition 4.2 (Degree-d MPRE [ABT18]). *Consider an n-party function $f : \mathcal{X}_1 \times \ldots \times \mathcal{X}_n \to \mathcal{Y}$. $\hat{f}$ is a degree d* multiparty randomized encoding *of f if $\hat{f}$ can be described as a polynomial of degree-d over $\mathsf{GF}[2]$, and the following conditions hold for a set of preprocessing functions $h_1, \ldots, h_n$.*

- ***Correctness.*** *There is a decoder Dec such that, for every input $x_1, \ldots, x_n$, and every choice of private randomness $r_1, \ldots, r_n$, we have:*

$$\mathsf{Dec}\left(\hat{f}\left(h_1(1^\lambda, x_1; r_1), \ldots, h_n(1^\lambda, x_n; r_n)\right)\right) = f(x_1, \ldots, x_n)$$

- ***Privacy.*** *For every subset $M \subset [n]$ and for any non-uniform polynomial time adversary $\mathcal{A}$ corrupting parties in M, we have a simulator Sim such that, when $\{r_i\}_{i \in [n] \setminus M}$ are uniformly sampled, for all $x_1, \ldots, x_n$ and $\{r_i\}_{i \in M}$,*

$$\hat{f}\left(h_1(1^\lambda, x_1; r_1), \ldots, h_n(1^\lambda, x_n; r_n)\right) \approx \mathsf{Sim}(f(x_1, \ldots, x_n), \{x_i, r_i\}_{i \in M}). \quad (2)$$

Here $\approx$ denotes computational indistinguishability.

In the general definition of MPRE, each party uses a private decoder which, in addition to the encoding, uses the party's input and private randomness to compute the output of the function. However, in the above definition, the decoder is public. Such an MPRE was constructed in [GS17, GS18, ABT18] by transforming a secure computation protocol that computes the function with semi-malicious security–security is guaranteed even when the adversary chooses arbitrary private randomness for corrupt parties. To arrange for public decoding of this MPRE, it suffices to append an additional round of communication to the corresponding protocol, in which the output of the function computation is broadcasted. We note that such a multi-round semi-malicious secure protocol can be constructed from any multi-round semi-honest secure OT protocol via the GMW transformation [GMW87] or making black-box use of semi-honest OT [HIK+11].

4.2 One-Time Digital Signature

Definition 4.3 (One-time digital signature scheme). *Let λ be a security parameter. A triple of algorithms* $(\mathrm{Gen}, \mathrm{Sig}, \mathrm{Ver})$ *is a one-time digital signature scheme if the following properties are met:*

Authentication. *When r is chosen uniformly, for all $x \in \{0,1\}^m$,*

$$\Pr\left[\mathrm{Ver}(x, \mathrm{Sig}(x, \mathsf{sk}), \mathsf{vk}) = 1 | (sk, vk) \leftarrow \mathrm{Gen}(1^\lambda, r)\right] = 1$$

Unforgeability. *For any non-uniform polynomial time adversary $\mathcal{A}$, for any x,*

$$\Pr\left[(x \neq x') \wedge (\mathrm{Ver}(x', t', \mathsf{vk}) = 1) \;\middle|\; \begin{array}{l} (\mathsf{sk}, \mathsf{vk}) \leftarrow \mathrm{Gen}(1^\lambda, r) \\ t = \mathrm{Sig}(x, \mathsf{sk}) \\ (x', t') \leftarrow \mathcal{A}(1^\lambda, x, \mathsf{vk}, t) \end{array}\right] = \mathrm{negl}(\lambda).$$

One-time digital signatures exist if one-way functions exist [Lam16].

5 Impossibility of Statistical Security with Abort

We begin by exploring the limitations of this model owing to non-interactivity. Indeed, secure computation with abort is impossible for certain simple 3-party functions in this model against a computationally unbounded adversary. We will also show that this impossibility can be further extended to n-party setting when the adversary may corrupt more than half of the participants.

Theorem 5.1. *There exists a 3-party function f that can be computed using a degree 2-polynomial such that no protocol making parallel uses to 2-ary functions achieves statistically secure computation of f with abort against a malicious adversary corrupting arbitrarily many parties.*

Proof. We will show that the theorem holds for the 3-party function f that takes $x_i \in \mathbb{F}_4$ from each P_i and outputs 0 if $x_1 + x_2 + x_3 = 0$ and outputs 1 otherwise. When an element x_i in $\mathbb{F}_4$ is represented as a 2-dimensional vector $(x_{i,0}, x_{i,1})$, f is computed by the following degree-2 polynomial over $\mathbb{F}_2$:

$$f((x_{1,0}, x_{1,1}), (x_{2,0}, x_{2,1}), (x_{3,0}, x_{3,1})) = (x_{1,0} \oplus x_{2,0} \oplus x_{3,0}) \vee (x_{1,2} \oplus x_{2,2} \oplus x_{3,1}).$$

Suppose $\Pi = (\{\mathcal{O}_{\{i,j\}}\}, \mathsf{Enc}, \mathsf{Dec})$ is a secure protocol computing f with abort. Let $\epsilon \geq 0$ be some constant such that the advantage of any computationally unbounded adversary in breaking the protocol is at most ϵ. That is, when $\approx_\epsilon$ denotes statistical distance of at most ϵ, for any adversary $\mathcal{A}$ corrupting $M \subset [3]$,

$$(\mathsf{View}_{\mathcal{A}}, \mathsf{out}) \approx_\epsilon \mathsf{Ideal}(1^\lambda, \mathsf{Sim}^{\mathcal{F}^{\mathrm{SA}}(\{x_i\}_{i \in [n] \setminus M}, \cdot)}, \{x_i\}_{i \in M}).$$

Consider an honest execution of Π with each x_i is distributed uniformly and independently over $\mathbb{F}_4$. Let $z_{\{1,2\}}, z_{\{2,3\}}$ and $z_{\{1,3\}}$ be the output of 2-ary functions $\mathcal{O}_{\{1,2\}}, \mathcal{O}_{\{2,3\}}$ and $\mathcal{O}_{\{3,1\}}$, respectively. That is, when r_1, r_2, r_3 are sampled uniformly, and $y_{(i,j)} = \mathsf{Enc}((i,j), x_i, r_i)$ for distinct $i, j \in [3]$,

$$\begin{aligned} z_{\{1,2\}} &= \mathcal{O}_{\{1,2\}}(y_{(1,2)}, y_{(2,1)}) \qquad z_{\{2,3\}} = \mathcal{O}_{\{2,3\}}(y_{(2,3)}, y_{(3,2)}) \\ z_{\{1,3\}} &= \mathcal{O}_{\{3,1\}}(y_{(1,3)}, y_{(3,1)}), \end{aligned} \tag{3}$$

Claim 5.1.1. *Over the randomness of* $x_1, x_2, x_3, r_1, r_2, r_3$,

$$\Pr\Big[\exists(x_1', r_1', x_2', r_2') \text{ s.t. } y'_{(1,2)} = \mathsf{Enc}((1,2), x_1', r_1'), y'_{(2,1)} = \mathsf{Enc}((2,1), x_2', r_2'), \text{ and } \mathsf{Dec}(\mathcal{O}_{\{1,2\}}(y'_{(1,2)}, y'_{(2,1)}), z_{\{1,3\}}, z_{\{2,3\}}) \notin \{\perp, f(x_1, x_2, x_3)\}\Big] \leq 9\epsilon, \quad (4)$$

and

$$\Pr\Big[\exists(x_1', r_1', x_3', r_3') \text{ s.t. } y'_{(1,3)} = \mathsf{Enc}((1,3), x_1', r_1'), y'_{(3,1)} = \mathsf{Enc}((3,1), x_3', r_3'), \text{ and } \mathsf{Dec}(z_{\{1,2\}}, \mathcal{O}_{\{1,3\}}(y'_{(1,3)}, y'_{(3,1)}), z_{\{2,3\}}) \notin \{\perp, f(x_1, x_2, x_3)\}\Big] \leq 9\epsilon. \quad (5)$$

Before proving the claim, we will use this claim to prove the theorem. Consider an adversary $\mathcal{A}_1$ that corrupts P_1 and behaves as described in Fig. 2

$\mathcal{A}_1$

1. Corrupt P_1. Sample x_1, r_1 uniformly and independently. Invoke $\mathcal{O}_{\{1,2\}}$ and $\mathcal{O}_{\{1,3\}}$ with $y_{(1,2)} = \mathsf{Enc}((1,2), x_1, r_1)$ and $\hat{y}_{(1,3)} = \mathsf{Enc}((1,3), x_1 + 1, r_1)$, respectively, as the inputs of P_1, and receive outputs $z_{\{1,2\}}$ and $\hat{z}_{\{1,3\}}$, respectively. Let $z_{\{2,3\}} = \hat{z}_{\{2,3\}}$ be the output of invocation of $\mathcal{O}_{\{2,3\}}$ by the honest parties P_2 and P_3.
2. If there exist $\hat{z}_{\{1,2\}} \in \mathcal{O}_{\{1,2\}}(\cdot,\cdot)$ and $z_{\{1,3\}} \in \mathcal{O}_{\{1,3\}}(\cdot,\cdot)$ such that $\mathrm{Dec}(\{z_{\{i,j\}}\}) = a_0, \mathrm{Dec}(\{\hat{z}_{\{i,j\}}\}) = a_1$ and $a_0, a_1 \in \{0,1\}$, output (a_0, a_1); otherwise, output $\perp$.

Fig. 2. Description of $\mathcal{A}_1$.

We now argue that Π is not secure against $\mathcal{A}_1$. Let r_2 and r_3 be the private randomness used by P_2 and P_3, respectively. Let $y_{(i,j)} = \mathsf{Enc}((i,j), x_i, r_i)$ for each $i \in [3], j \neq i$. Then, $z_{\{i,j\}} = \mathcal{O}_{\{i,j\}}(y_{(i,j)}, y_{(j,i)})$ for each $\{i,j\} \neq \{1,3\}$. Defining $\tilde{z}_{\{1,3\}} = \mathcal{O}_{\{1,2\}}(y_{(1,3)}, y_{(3,1)})$, $z_{\{1,2\}}, z_{\{2,3\}}$ and $\tilde{z}_{\{1,3\}}$ are the outputs of the function calls in an honest execution of Π with x_i as input and r_i as randomness of each P_i. Hence,

$$\Pr\left[\mathsf{Dec}(z_{\{1,2\}}, z_{\{2,3\}}, z_{\{1,3\}}) = f(x_1, x_2, x_3)\right] \geq 1 - \epsilon.$$

This along with eq. (5) implies $\Pr[a_1 = f(x_1, x_2, x_3)] \geq 1 - 10\epsilon$. Using the same line of argument, $\Pr[a_0 = f(1 + x_1, x_2, x_3)] \geq 1 - 10\epsilon$. Thus, in the real execution, $\mathcal{A}_1$ successfully computes $f(x_1, x_2, x_3)$ and $f(1+x_1, x_2, x_3)$ with probability at least $1 - 20\epsilon$.

$\Pr[f(x_1 + 1, x_2, x_3) = 1 | f(x_1, x_2, x_3) = 1] = 2/3$, and $f(x_1, x_2, x_3) = 1$ with probability $3/4$ for any x_1, when of x_2 and x_3 are chosen uniformly. Hence, the simulator will fail to simultaneously guess the values of $f(x_1, x_2, x_3)$ and $f(x_1 + 1, x_2, x_3)$ with probability at least $1/4$. Thus, for sufficiently small ϵ, this contradicts security with at most ϵ distinguishing advantage against an adversary that corrupts at most 2 parties. We conclude the proof by proving the claim.

Proof (of Claim 5.1.1). We will prove Equation (4); Equation (5) can be proved similarly. Consider an adversary $\mathcal{A}_{1,2}$ that corrupts P_1 and P_2, and behaves as described in Fig. 3.

$\mathcal{A}_{1,2}$

1. Corrupt P_1, P_2. Sample $r_1 \leftarrow \{0,1\}^*$, and invoke $\mathcal{O}_{\{1,3\}}$ with $y_{(1,3)} = \mathsf{Enc}((1,3), x_1, r_1)$ as P_1's input. Sample $r_2 \leftarrow \{0,1\}^*$ and invoke $\mathcal{O}_{\{2,3\}}$ with $y_{(2,3)} = \mathsf{Enc}((2,3), x_2, r_2)$ as P_2's input. Let $z_{\{1,3\}}$ and $z_{\{2,3\}}$ be the outputs of $\mathcal{O}_{\{1,3\}}$ and $\mathcal{O}_{\{2,3\}}$, respectively.
2. Define $y_{(1,2)} = \mathsf{Enc}((1,2), x_1, r_1)$ and $y_{(2,1)} = \mathsf{Enc}((2,1), x_2, r_2)$. Without invoking $\mathcal{O}_{\{1,2\}}$, compute $z_{\{1,2\}} = \mathcal{O}_{\{1,2\}}(y_{(1,2)}, y_{(2,1)})$. If there exist $y'_{(1,2)} = \mathsf{Enc}((1,2), x'_1, r'_1)$ and $y'_{(2,1)} = \mathsf{Enc}((2,1), x'_2, r'_2)$, for some x'_1, r'_1, x'_2, r'_2, such that
$$\mathsf{Dec}(\mathcal{O}_{\{1,2\}}(y'_{(1,2)}, y'_{(2,1)}), z_{\{1,3\}}, z_{\{2,3\}}) \notin \{\perp, \mathsf{Dec}(z_{\{1,2\}}, z_{\{1,3\}}, z_{\{2,3\}})\},$$
then call $\mathcal{O}_{\{1,2\}}$ with $y'_{(1,2)}$ and $y'_{(2,1)}$ as inputs of P_1 and P_2; else use $y_{(1,2)}$ and $y_{(2,1)}$ as inputs of P_1 and P_2.
3. Output $\mathsf{Dec}(z_{\{1,2\}}, z_{\{1,3\}}, z_{\{2,3\}})$ and terminate.

Fig. 3. Description of $\mathcal{A}_{1,2}$.

We define a couple of distinguishers D_1 and D_2 that take the view of $\mathcal{A}_{1,2}$ and inputs/outputs of P_3 and does the following:

1. D_1 extracts $z_{\{1,3\}}$ and $z_{\{2,3\}}$ from the view of $\mathcal{A}_{1,2}$. It then extracts (x_1, r_1, x_2, r_2) from $\mathcal{A}_{1,2}$'s view and computes $z_{\{1,2\}}$. If $\mathsf{Dec}(z_{\{1,2\}}, z_{\{1,3\}}, z_{\{2,3\}}) = f(x_1, x_2, x_3)$, D_1 outputs 1, and outputs 0 otherwise.
2. If the output of P_3 is not equal to $f(x_1, x_2, x_3)$ or $\perp$, D_2 outputs 1, and outputs 0 otherwise.

We prove that, when ϵ is sufficiently small, there exists no simulator that guarantees at most ϵ distinguishability advantage against both D_1 and D_2 if

$$\Pr\Big[\exists (x'_1, r'_1, x'_2, r'_2) \text{ s.t. } y'_{(1,2)} = \mathsf{Enc}((1,2), x'_1, r'_1), y'_{(2,1)} = \mathsf{Enc}((2,1), x'_2, r'_2), \\ \text{and } \mathsf{Dec}(\mathcal{O}_{\{1,2\}}(y'_{(1,2)}, y'_{(2,1)}), z_{\{1,3\}}, z_{\{2,3\}}) \notin \{\perp, f(x_1, x_2, x_3)\}\Big] > 9\epsilon. \tag{6}$$

Suppose such a simulator Sim exists. Let $\hat{x}_1$ and $\hat{x}_2$ be the inputs of P_1 and P_2 used by Sim while invoking the functionality. We use security against D_2 to argue that $\hat{x}_2 + \hat{x}_1 \neq x_2 + x_1$ with substantial probability. If $\hat{x}_2 + \hat{x}_1 = x_2 + x_1$, then $f(x_1, x_2, x_3) = f(\hat{x}_1, \hat{x}_2, x_3)$ for any x_3. Hence,

$$\begin{aligned} \Pr[D_2 \text{ outputs 1 in the ideal execution}] &= \Pr[f(x_1, x_2, x_3) \neq f(\hat{x}_1, \hat{x}_2, x_3)] \\ &\leq \Pr[\hat{x}_2 + \hat{x}_1 \neq x_2 + x_1]. \end{aligned}$$

Whereas, by our assumption in eq. (6), and the property of $\mathcal{A}_{1,2}$, D_2 outputs 1 in the real execution with probability more than 9ϵ. Since the probability with which D_2 outputs 1 in real and ideal executions are at most ϵ apart, the above observations imply that

$$\begin{aligned} 8\epsilon &< \Pr\left[D_2 \text{ outputs 1 in the real execution}\right] - \epsilon \\ &\leq \Pr\left[D_2 \text{ outputs 1 in the ideal execution}\right] \leq \Pr[\hat{x}_2 + \hat{x}_1 \neq x_2 + x_1]. \end{aligned} \tag{7}$$

Next, we use security against D_1 to argue that $\hat{x}_2 + \hat{x}_1 \neq x_2 + x_1$ can only occur with very low probability, reaching a contradiction. If $\hat{x}_2 + \hat{x}_1 \neq x_2 + x_1$, over the randomness of x_3,

$$\Pr\left[f(x_1, x_2, x_3) = 0 \mid \hat{x}_2 + \hat{x}_1 \neq x_2 + x_1, f(\hat{x}_1, \hat{x}_2, x_3) = 1\right] = 1/3.$$

Note that $\mathrm{Dec}(\mathcal{O}_{\{1,2\}}(y_{(1,2)}, y_{(2,1)}), \hat{z}_{\{1,3\}}, \hat{z}_{\{2,3\}})$ is a value that the simulator computes using $x_1, x_2, \hat{x}_1, \hat{x}_2$ and $f(\hat{x}_1, \hat{x}_2, x_3)$ and its own private randomness. Since an 2/3-biased coin cannot be guessed with non-zero advantage, the above condition implies that, when E is the event $\hat{x}_2 + \hat{x}_1 \neq x_2 + x_1, f(\hat{x}_1, \hat{x}_2, x_3) = 1$,

$$\Pr\left[\mathrm{Dec}(\mathcal{O}_{\{1,2\}}(y_{(1,2)}, y_{(2,1)}), \hat{z}_{\{1,3\}}, \hat{z}_{\{2,3\}}) \neq f(x_1, x_2, x_3) \mid E\right] \geq 1/3.$$

The above equality implies

$$\begin{aligned} &\Pr\left[D_1 \text{ outputs 1 in the ideal execution}\right] \\ &\quad = \Pr\left[\mathrm{Dec}(\mathcal{O}_{\{1,2\}}(y_{(1,2)}, y_{(2,1)}), \hat{z}_{\{1,3\}}, \hat{z}_{\{2,3\}}) = f(x_1, x_2, x_3)\right] \\ &\quad \leq 1 - \frac{1}{3}\Pr[\hat{x}_2 + \hat{x}_1 \neq x_2 + x_1, f(\hat{x}_1, \hat{x}_2, x_3) = 1] \\ &\quad \leq 1 - \frac{1}{4}\Pr[\hat{x}_2 + \hat{x}_1 \neq x_2 + x_1]. \end{aligned} \tag{8}$$

The last inequality used $f(\hat{x}_1, \hat{x}_2, x_3) = 1$ with probability 3/4 over the randomness of x_3.

Next, we bound the probability with which D_1 outputs 1 in the real execution. Since r_1 and r_2 are sampled uniformly from $\{0,1\}^*$ and P_3 is honest, random variables $(z_{\{1,2\}}, z_{\{2,3\}}, z_{\{1,3\}})$ are identically distributed as in an honest execution of Π (as described in Equation (3)). By correctness of the protocol when all parties are honest, the output of the decoder is $f(x_1, x_2, x_3)$ with probability at least $1 - \epsilon$. In other words,

$$\begin{aligned} &\Pr\left[D_1 \text{ outputs 1 in the real execution}\right] \\ =&\Pr\left[\mathsf{Dec}(z_{\{1,2\}}, z_{\{1,3\}}, z_{\{2,3\}}) = f(x_1, x_2, x_3)\right] \geq 1 - \epsilon. \end{aligned} \tag{9}$$

The probability with which D_1 outputs 1 in real and ideal executions are at most ϵ apart. Hence, by eq. (8) and eq. (9),

$$\begin{aligned} 1 - 2\epsilon &\leq \Pr\left[D_1 \text{ outputs 1 in the real execution}\right] - \epsilon \\ &\leq \Pr\left[D_1 \text{ outputs 1 in the ideal execution}\right] \leq 1 - \frac{1}{4}\Pr[\hat{x}_2 + \hat{x}_1 \neq x_2 + x_1]. \end{aligned}$$

Hence, $\Pr[\hat{x}_2 + \hat{x}_1 \neq x_2 + x_1] \leq 8\epsilon$. This contradicts eq. (7). We conclude that Sim does not exist; proof is complete. □

This completes the proof of the theorem. □

Using a similar analysis we can extend the above result to n-party functions against an adversary that corrupts a majority of the parties. The following theorem is proved in the full version [NPS24].

Theorem 5.2. *There exists an n-party function f such that no protocol making parallel uses of 2-ary functions that securely computes f with statistical security with abort against a malicious adversary corrupting at most $\lceil n/2 \rceil$ parties.*

6 Positive Results

In this section, we give our two positive results, namely, a statistical PwKO protocol for computing degree-2 functions and computationally secure protocol for computing arbitrary functions. A key building block used in these constructions is conditional disclosure of secrets protocol that is described next.

6.1 Conditional Disclosure Protocol

A crucial building block in our later constructions is the so-called conditional disclosure protocol which ensures that the inputs used by any (corrupt) party P_i during their oracle calls to the 2-ary functions in the larger non-interactive protocol are consistent.

A conditional disclosure protocol in which *parties P_i and P_j verify the consistency of P_k's inputs to $\mathcal{O}_{\{i,k\}}$ and $\mathcal{O}_{\{j,k\}}$* is a n-party non-interactive protocol with parallel use of 2-ary functions that effectively delivers a secret that is additively secret shared between P_i and P_j to all parties if P_k's input to $\mathcal{O}_{\{i,k\}}$ and $\mathcal{O}_{\{j,k\}}$ are consistent. If P_k uses inconsistent inputs, then all parties detect malpractice and abort; furthermore, the secret inputs of P_i and P_k are kept hidden from the adversary. On the other hand, when P_i and P_j are corrupt, the protocol ensures perfect privacy of P_k's input. The n-party conditional disclosure protocol in which P_i and P_j verify P_k is denoted by $\mathrm{CD}_{\{i,j\},k}$.

Definition 6.1. *Let P_i, P_j, P_k be distinct parties. Let $\mathrm{CD}_{\{i,j\},k}$ be an n-party protocol using parallel use of 2-ary functions with $x_i, x_j \in \{0,1\}^m$, respectively, as inputs of P_i, P_j and $x_k \in \{0,1\}^\ell$ as the input of P_k, and no inputs from the remaining parties, described by encoder Enc, 2-ary functions $\{\mathcal{O}_{\{i,j\}}, \mathcal{O}_{\{j,k\}}, \mathcal{O}_{\{i,k\}}\}$ and a decoder Dec. $\mathrm{CD}_{\{i,j\},k}$ is said to be a conditional disclosure protocol with P_i and P_j verifying P_k with $\epsilon(\lambda)$ soundness if the following properties are met.*

1. *If P_i, P_j, P_k are honest, all honest parties output $x_i \oplus x_j$ at the end of the protocol.*
2. *If P_k is honest, a malicious adversary corrupting P_i and P_j does not learn x_k. That is, the view of the adversary corrupting P_i, P_j is identically distributed irrespective of the value of x_k.*

3. *Π offers $\epsilon(\lambda)$ soundness if for any computationally unbounded adversary $\mathcal{A}$ that corrupts a set of parties $\{P_a\}_{a\in C}$ such that $\{i,j\}\cap C=\emptyset$, there exists an ideal world PPT simulator* Sim *such that the following conditions are met:*
   (a) *For every choice of inputs $\{x_a\}_{a\in\{i,j,k\}}$*

$$(\mathsf{View}_{\mathcal{A}}, \mathsf{out}) \equiv \mathsf{Ideal}(1^{\lambda}, \mathsf{Sim}^{\mathcal{F}_{\mathrm{CD}_{\{i,j\},k}}(x_i,x_j,\cdot)}, x_k). \tag{10}$$

   Here, $\mathsf{View}_{\mathcal{A}}$ *refers to the view of the adversary $\mathcal{A}$ (which includes the input, randomness, and the outputs of the function calls) and* out *refers to the output of all honest parties which coincide.* Ideal *refers to the ideal execution of the protocol. This starts with the* Sim *run on $1^{\lambda}, x_k$. If $\mathcal{F}(x_i, x_j, \cdot)$ receives a $b=1$ from* Sim*, it delivers $x_i \oplus x_j$ to* Sim*, and all the honest parties; if $b=0$, it delivers $\perp$ to* Sim *and all honest parties. The output of* Ideal *corresponds to the output of* Sim *and the output of all the honest parties (which coincide).*
   (b) *Conditioned on the event that the input of* Sim *to $\mathcal{O}_{\{i,k\}}$ and $\mathcal{O}_{\{j,k\}}$ are, respectively, $y_{\{i,k\}}$ and $y_{\{j,k\}}$ such that $y_{\{i,k\}}$ belongs to the domain of $\mathsf{Enc}((i,k), x'_k, \cdot)$ and $y_{\{j,k\}}$ belongs to the domain of $\mathsf{Enc}((j,k), x''_k, \cdot)$, and $x'_k \neq x''_k$, the input of* Sim *to $\mathcal{F}(x_i, x_j, \cdot)$ is 0 with probability $1-\epsilon(\lambda)$.*

Remark 6.2. Consider an execution of $\mathrm{CD}_{\{i,j\},k}$ in which an adversary corrupts P_k, while both P_i and P_j are honest. In our definition, conditioned on Sim sending $b=0$ to $\mathcal{F}_{\mathrm{CD}}$, Sim perfectly simulated the view of the adversary without knowing the secret, and the output of all honest parties is $\perp$. This models the soundness requirement that the secret that is shared between P_i and P_j remains hidden from the adversary, and that all honest parties output $\perp$. This justifyies defining soundness as the probability with which Sim sends $b=1$ conditioned on the event that the adversary sends inconsistent values of x_k to $\mathcal{O}_{\{j,k\}}$ and $\mathcal{O}_{\{i,k\}}$ on behalf of P_k.

Figure 4 provides a formal description of the protocol which clarifies how the aforementioned masking is carried out. The protocol also hides the secret in the degree 2 term instead of the constant term allowing the construction to be based on $\mathbb{F}_3$ rather than $\mathbb{F}_4$. Lemma 6.3 formally proves that the construction is a 1/9 sound conditional disclosure protocol where P_1 and P_2 verify P_3's inputs.

Lemma 6.3. *The protocol $\mathrm{CD}_{\{12\},3}$ in Fig. 4 is a conditional disclosure protocol where P_1 and P_2 with single bit secret verify the single-bit input of P_3 with 8/9 soundness.*

Proof. When P_3 is honest, for any α_1 and α_2 chosen by P_1 and P_2, there exists $p\in\mathbb{F}_3^2$ (specifically $p(u)=x_3u^2+c_1u+c_0$) such that $p(\alpha_i)=x_3(\alpha_i)^2+c_1\alpha_i+c_0$ for $i=1,2$. Hence, γ_p computed by $\mathcal{O}_{\{1,3\}}$ and γ'_p computed by $\mathcal{O}_{\{2,3\}}$ are both 1. Hence, the protocol satisfies the first condition in the definition of conditional disclosure protocol in Definition 6.1.

Next, we prove that it satisfies the second condition in Definition 6.1. The view of the adversary consists of x_1, x_2, the set of masks, α_1 and α_2 chosen by

$\mathrm{CD}_{\{12\},3}$

Let the inputs of P_1, P_2 and P_3 be $x_1, x_2, x_3 \in \{0,1\}$, respectively. Let $\mathbb{F}_3^2[u]$ be the set of all univariate polynomials over the variable u with coefficients in $\mathbb{F}_3$ and of degree at most 2 (there are 27 of them).

Execution of Enc

1. For $i \in \{2,3\}, \mathsf{Enc}((1,i), x_1, r_1) = (x_1, r_1)$.
2. For $i \in \{1,3\}, \mathsf{Enc}((2,i), x_2, r_2) = (x_2, r_2)$.
3. For $i \in \{1,2\}, \mathsf{Enc}((3,i), x_3, r_3) = (x_3, r_3)$.

Execution of $\mathcal{O}_{\{i,j\}}$

Interpret r_1 as $(\alpha_1, \{(\theta_{p,0}, \theta_{p,1}, a_p, a'_p)\}_p)$, where $\alpha_1 \leftarrow \mathbb{F}_3$, and for each $p \in \mathbb{F}_3^2[u]$, $\theta_{p,0}, \theta_{p,1}$, a_p and a'_p are uniform independent bits. Interpret r_2 as $(\alpha_2, \{(\phi_{p,0}, \phi_{p,1}, b_p, b'_p)\}_p)$, where $\alpha_2 \leftarrow \mathbb{F}_3$, and for each $p \in \mathbb{F}_3^2[u]$, $\phi_{p,0}, \phi_{p,1}$, b_p and b'_p are uniform independent bits. Interpret r_3 as (c_0, c_1), where $c_0, c_1 \leftarrow \mathbb{F}_3$, and let $q(u) = x_3u^2 + c_1u + c_0$; here we typecast x_3 in $\mathcal{F}_3$.

1. $\mathcal{O}_{\{1,2\}}((x_1, r_1), (x_2, r_2))$: Output $\{x_1 \oplus x_2 \oplus \theta_{p,1} \oplus \phi_{p,1}, a_p \oplus b'_p, b_p \oplus a'_p\}_p$.
2. $\mathcal{O}_{\{1,3\}}((x_1, r_1), (x_3, r_3))$: For each $p \in \mathbb{F}_3^2[u]$, if $p(\alpha_1) = q(\alpha_1)$, set $\gamma_p = 1$; else set $\gamma_p = 0$. Output $\{(\gamma_p \oplus a_p, \theta_{p,\gamma_p}, a'_p)\}_p$.
3. $\mathcal{O}_{\{2,3\}}((x_2, r_2), (x_3, r_3))$: For each $p \in \mathbb{F}_3^2[u]$, if $p(\alpha_2) = q(\alpha_2)$, set $\gamma'_p = 1$; else set $\gamma'_p = 0$. Output $\{(\gamma'_p \oplus b_p, \phi_{p,\gamma'_p}, b'_p)\}_p$.

Execution of Dec

1. For each $p \in \mathbb{F}_3^2[u]$, recover γ_p from $\gamma_p \oplus a_p, b'_p$ and $a_p \oplus b'_p$, which are available in the output of $\mathcal{O}_{\{1,3\}}, \mathcal{O}_{\{2,3\}}$ and $\mathcal{O}_{\{1,2\}}$, respectively.
2. Similarly, for each $p \in \mathbb{F}_3^2[u]$, recover γ'_p from $\gamma'_p \oplus b_p, a'_p$ and $b_p \oplus a'_p$.
3. If there exists p such that $\gamma_p = \gamma'_p = 1$, the recover $x_1 \oplus x_2$ from $x_1 \oplus x_2 \oplus \theta_{p,1} \oplus \phi_{p,1}, \theta_{p,1}$ and $\phi_{p,1}$ which are available in the outputs of $\mathcal{O}_{\{1,2\}}, \mathcal{O}_{\{1,3\}}$ and $\mathcal{O}_{\{2,3\}}$, respectively. If no such p exists, output $\perp$.

Fig. 4. Conditional disclosure protocol with 8/9 soundness in which P_1 and P_2 sharing a 1-bit secret verify P_3's 1-bit input to $\mathcal{O}_{\{1,3\}}$ and $\mathcal{O}_{\{2,3\}}$.

P_1 and P_2, and the class of polynomials $\mathcal{P}_1$ and $\mathcal{P}_1$, where, for each $i \in \{1,2\}$, $\mathcal{P}_i$ contains the set of all degree-2 polynomials p such that $p(\alpha_i) = x_3(\alpha_i)^2 + c_1\alpha_i + c_0$. It is easy to see that this view only reveals $x_3\alpha_i^2 + c_1\alpha_i + c_0$ for $i = 1, 2$. Since c_0 and c_1 are uniformly and independently sampled by P_3, irrespective of the values of α_1 and α_2, these values are identically distributed for $x_3 = 0, 1$. In other words, the view of the adversary can be perfectly simulated irrespective of the value of x_3, satisfying the second condition.

Next, we show that the protocol is 8/9-sound by showing that the simulator Sim in Fig. 5 satisfies the conditions in Definition 6.1.

Sim

1. On behalf of P_1, choose uniformly random r_1 and interpret it as $(\alpha_1, \{(\theta_{p,0}, \theta_{p,1}, a_p, a'_p)\}_p)$. On behalf of P_2, choose uniformly random r_2 and interpret it as $(\alpha_2, \{(\phi_{p,0}, \phi_{p,1}, b_p, b'_p)\}_p)$.
2. Send $\mathcal{O}_{\{1,2\}}((0,r_1),(0,r_2)) = \{(\theta_{p,1} \oplus \phi_{p,1}, a_p \oplus b'_p, b_p \oplus a'_p)\}_p$, when $\mathcal{A}$ demands the output of $\mathcal{O}_{\{1,2\}}$,
3. When $\mathcal{A}$ queries $\mathcal{O}_{\{1,3\}}$ with input $(x_{3,\{1,3\}}, r_{3,\{1,3\}})$, check if $\mathcal{O}_{\{2,3\}}$ is already queried. If false, send $\mathcal{O}_{\{13\}}((0,r_1),(x_{3,\{1,3\}}, r_{3,\{1,3\}}))$. If true, let $(x_{3,\{2,3\}}, r_{3,\{2,3\}})$ be P_3's input to $\mathcal{O}_{\{2,3\}}$ and $\mathcal{O}_{\{1,3\}}$, respectively. Interpret $r_{3,\{1,3\}}$ as $(c_{0,\{1,3\}}, c_{1,\{1,3\}})$ and $r_{3,\{2,3\}}$ as $(c_{0,\{2,3\}}, c_{1,\{2,3\}})$. Check if $x_{2,\{1,3\}} \neq x_{3,\{1,3\}}$, $\alpha_1 = \alpha_2$ and

$$x_{3,\{1,3\}}\alpha_1^3 + c_{1,\{1,3\}}\alpha_1 + c_{0,\{1,3\}} \neq x_{3,\{2,3\}}\alpha_2^3 + c_{1,\{2,3\}}\alpha_2 + c_{0,\{2,3\}}. \quad (11)$$

   (a) If true, send $\mathcal{O}_{\{13\}}((0,r_1),(x_{3,\{1,3\}}, r_{3,\{1,3\}}))$, and send $b = 0$ as input to $\mathcal{F}_{\mathrm{CD}_{\{1,2\},3}}(x_1, x_2, \cdot)$.
   (b) If false, invoke $\mathcal{F}_{\mathrm{CD}_{\{1,2\},3}}(x_1, x_2, \cdot)$ with $b = 1$ to receive $x_1 \oplus x_2$. Send $\mathcal{O}_{\{13\}}((0,r'_1),(x_{3,\{1,3\}}, r_{3,\{1,3\}}))$ where r'_1 is obtained from r_1 by replacing $\theta_{p,1}$ with $\theta_{p,1} \oplus x_1 \oplus x_2$ for each p.
4. Behave analogously when $\mathcal{A}$ queries $\mathcal{O}_{\{2,3\}}$.

Fig. 5. Conditional disclosure protocol with 8/9 soundness in which P_1 and P_2 sharing a 1-bit secret verify P_3's 1-bit input to $\mathcal{O}_{\{1,3\}}$ and $\mathcal{O}_{\{2,3\}}$.

We show that w.r.t. any adversary $\mathcal{A}$, the real world and the ideal world executions are computationally indistinguishable via a hybrid argument.

- Hyb_0 : This corresponds to the real world execution of the protocol.
- Hyb_1 : Let $(x_{3,\{1,3\}}, r_{3,\{1,3\}})$ and $(x_{3,\{2,3\}}, r_{3,\{2,3\}})$ be P_3's input to $\mathcal{O}_{\{2,3\}}$ and $\mathcal{O}_{\{1,3\}}$, respectively. Interpret $r_{3,\{1,3\}}$ as $(c_{0,\{1,3\}}, c_{1,\{1,3\}})$ and $r_{3,\{2,3\}}$ as $(c_{0,\{2,3\}}, c_{1,\{2,3\}})$. In this hybrid, check if $x_{2,\{1,3\}} \neq x_{3,\{1,3\}}$, $\alpha_1 = \alpha_2$ and eq. (11) are together satisfied. If not, change the output of $\mathcal{O}_{\{1,2\}}$ to $\{(\theta_{p,1} + \phi_{p,1}, a_p \oplus b'_p, b_p \oplus a'_p)\}$. If $\mathcal{O}_{\{1,3\}}$ is invoked after $\mathcal{O}_{\{2,3\}}$, replace $\theta_{p,1}$ with $x_1 \oplus x_2 \oplus \theta_{p,1}$ before computing the output of $\mathcal{O}_{\{1,3\}}$. Likewise, if $\mathcal{O}_{\{2,3\}}$ is invoked after $\mathcal{O}_{\{1,3\}}$, replace $\phi_{p,1}$ with $x_1 \oplus x_2 \oplus \theta_{p,1}$ before computing the output of $\mathcal{O}_{\{2,3\}}$. If the check succeeds, make no changes.
- Hyb_2 : In this hybrid, make the following change if the same check succeeds: change the output of $\mathcal{O}_{\{1,2\}}$ to $(\theta_{p,1} \oplus \phi_{p,1}, a_p \oplus b'_p, b_p \oplus a'_p)$.

Hyb_0 is perfectly indistinguishable from Hyb_1 because

$$\begin{aligned}
&\{x_1 \oplus x_2 \oplus (\theta_{p,1} + \phi_{p,1}, a_p \oplus b'_p, b_p \oplus a'_p), \theta_{p,1}, \phi_{p,1}, a_p, b_p, a'_p, b'_p\}_p \\
\equiv\ &\{(\theta_{p,1} + \phi_{p,1}, a_p \oplus b'_p, b_p \oplus a'_p), x_1 \oplus x_2 \oplus \theta_{p,1}, \phi_{p,1}, a_p, b_p, a'_p, b'_p\}_p \\
\equiv\ &\{(\theta_{p,1} + \phi_{p,1}, a_p \oplus b'_p, b_p \oplus a'_p), \theta_{p,1}, x_1 \oplus x_2 \oplus \phi_{p,1}, a_p, b_p, a'_p, b'_p\}_p.
\end{aligned}$$

When the check succeeds, for each p, $\gamma_p = 1 - \gamma'_p$. Since $\theta_{p,1}, \phi_{p,1}$ are uniformly and independently chosen,

$$\{x_1 \oplus x_2 \oplus (\theta_{p,1} + \phi_{p,1}, a_p \oplus b'_p, b_p \oplus a'_p), a_p, b_p, a'_p, b'_p, \theta_{p,\gamma_p}, \phi_{p,\gamma'_p}\}_p$$
$$\equiv \{(\theta_{p,1} + \phi_{p,1}, a_p \oplus b'_p, b_p \oplus a'_p), a_p, b_p, a'_p, b'_p, \theta_{p,\gamma_p}, \phi_{p,\gamma'_p}\}_p.$$

Hence $\mathsf{Hyb}_1 \equiv \mathsf{Hyb}_2$. It can be verified that Hyb_2 corresponds to the ideal execution using Sim. To show that the soundness error of the scheme is at most 8/9, we need to show that Sim sends $b = 0$ to the functionality with probability at least 1/9 when $x_{2,\{1,3\}} \neq x_{3,\{1,3\}}$. This event occurs if and only if $\alpha_1 = \alpha_2$ and eq. (11) are simultaneously satisfied, when $x_{2,\{1,3\}} \neq x_{3,\{1,3\}}$.

Owing to the symmetry, we assume, without loss of generality, that adversary rushes to obtain the output of $\mathcal{O}_{\{1,2\}}$; then adaptively chooses the input $(x_{3,\{1,3\}}, c_{0,\{1,3\}}, c_{1,\{1,3\}})$ to $\mathcal{O}_{\{1,3\}}$, obtains the output; then adaptively chooses the input $(1 \oplus x_{3,\{1,3\}}, c_{0,\{2,3\}}, c_{1,\{2,3\}})$ to $\mathcal{O}_{\{2,3\}}$, obtains its output. For any choice of $c_{0,\{1,3\}}, c_{1,\{1,3\}}$, the adversary only learns $(x_1 \oplus x_2 \oplus \theta_{p,1} \oplus \phi_{p,1}, a_p \oplus b'_p, b_p \oplus a'_p)$ for each p as the output of $\mathcal{O}_{\{1,2\}}$, and $(\gamma_p \oplus a_p, \theta_{p,\gamma_p}, a'_p)$ for each p from the output of $\mathcal{O}_{\{1,3\}}$. For any $p \in \mathbb{F}_3[u]$, a_p and b'_p are uniform independent bits. Hence, the value of γ_p is completely hidden from the adversary who knows $\gamma_p \oplus a_p$ and $a_p \oplus b'_p$. Thus, the above set of values are identically distributed irrespective of the values of α_1 (and, trivially, irrespective of α_2 since no function of α_2 has been revealed by $\mathcal{O}_{\{1,2\}}$ and $\mathcal{O}_{\{1,3\}}$). We conclude that, α_1 and α_2 are uniformly and independently distributed in $\mathbb{F}_3$, conditioned on adversary's view after the evaluation of $\mathcal{O}_{\{1,2\}}$ and $\mathcal{O}_{\{2,3\}}$. In other words, the distribution of $c_{0,\{1,3\}}, c_{1,\{1,3\}}, c_{0,\{2,3\}}$ and $c_{1,\{2,3\}}$ chosen by the adversary is independent of the distribution of α_1, α_2. Thus, when $x_{2,\{1,3\}} \neq x_{3,\{1,3\}}$, the event where $\alpha_1 = \alpha_2$ and eq. (11) are simultaneously satisfied occurs with probability at least 1/9. This concludes the proof of the lemma. □

Boosting Soundness of Conditional Disclosure. The protocol presented in the previous section only guarantees that a corrupt P_3 which provides inconsistent inputs to $\mathcal{O}_{\{1,3\}}$ and $\mathcal{O}_{\{2,3\}}$ would be detected with a constant probability, in which event the decoder reports an abort and the secret is hidden from the adversary. The following simple construction boosts the probability of this event such that it occurs with overwhelming probability.

To achieve soundness of $\mathrm{negl}(\lambda)$, the construction repeats the simple protocol λ times. If every execution succeeds, we require the parties to correctly recover the secrets. However, if the decoder reports an abort in any of the execution, which occurs independently with probability at least 1/9 in each execution if the bit being verified is inconsistent, we require the secret to be completely hidden from the adversary, and the parties to report an abort. This is easy to arrange by having each repetition reveal an additive secret share of $x_1 \oplus x_2$; for this, P_1 and P_2 provide an additive secret share of their respective inputs as inputs to each execution. Crucially, all λ repetitions of the protocol takes the same input x_3 from P_3; Formally, the input of P_3 to both $\mathcal{O}_{\{1,3\}}$ and $\mathcal{O}_{\{2,3\}}$ is

$x_3, \{c_{0,i}, c_{1,i}\}_{i\in[\lambda]}$, where $c_{b,j}$ is uniformly and independently chosen from $\mathbb{F}_3$ for each $i \in [\lambda]$ and $b \in \{0,1\}$. The polynomial used in repetition i is the polynomial $q(u) = x_3 \cdot u^2 + c_{1,\ell} \cdot u + c_{0,\ell}$. This ensures that each execution (independently) has at least 1/9 probability of failing if (corrupt) P_3 uses inconsistent values of x_3 in $\mathcal{O}_{\{1,3\}}$ and $\mathcal{O}_{\{2,3\}}$.

The above protocol can be further modified to handle string inputs from all parties. Let $x_1, x_2 \in \{0,1\}^m$ be the inputs of P_1, P_2, and $x_3 \in \{0,1\}^\ell$ be the input of P_3. Modify the previous construction to verify each of the ℓ-bits in x_3 with $\mathrm{negl}(\lambda)$ soundness error, and reveal $x_1 \oplus x_2$ only if P_3 provides consistent values for every bit in x_3. As in the previous construction, this is arranged by having the verification of each bit reveal an additive secret share of $x_1 \oplus x_2$.

By modifying the $\mathrm{CD}_{\{12\},3}$ described in Fig. 4 as discussed above, we obtain the following result as a consequence of Lemma 6.3.

Theorem 6.4. *For any m, ℓ, λ, there exists a conditional disclosure protocol $\mathrm{CD}_{\{12\},3}$ with P_1 and P_2 verifying P_3 with $\ell(8/9)^\lambda$ soundness when P_1 and P_2 have m bit inputs and P_3 has ℓ bit input.*

A proof of this theorem is provided in the full version [NPS24]. We stress that, this protocol can be modified to obtain a conditional disclosure protocol in which inputs of parties belong to arbitrary sets: it suffices to *flatten* the input being verified and the secret into strings.

6.2 Computing Degree-2 Functions with PwKO

A deterministic n-party function f that takes input $x_i \in \{0,1\}^m$ from party $P_i, i \in [n]$ has effective degree 2 if it can be decomposed into functions $\{h_{\{i,j\}}\}$, where, for each $\{i,j\} \in \binom{[n]}{2}$, $h_{\{i,j\}} : \{0,1\}^m \times \{0,1\}^m \to \mathbb{G}$ for an additive finite group $\mathbb{G}$, such that, for all $x_1, \ldots, x_n$,

$$f(x_1, \ldots, x_n) = \sum_{\{i,j\}} h_{\{i,j\}}(x_i, x_j). \tag{12}$$

In this section, we construct a protocol that securely computes any function with effective degree 2. The construction follows the intuition sketched in Sect. 2.2.

Description of the Protocol. Let f be a deterministic n-party function of effective degree 2, and let $\{h_{\{i,j\}}\}$ be as described in eq. (12). Let $\lambda \in \mathbb{N}$ be a security parameter. The Fig. 6 provides a formal description of a protocol that securely computes f with PwKO when the input of each party P_i is x_i.

The construction uses the following resources and notations: Let $\mathrm{CD}_{\{i,j\},k} = (\{\mathrm{CD}_{\{i,j\},k}.\mathcal{O}_{\{i,j\}}\}, \mathrm{CD}_{\{i,j\},k}.\mathsf{Enc}, \mathrm{CD}_{\{i,j\},k}.\mathsf{Dec})$ be a conditional disclosure protocol with $\mathrm{negl}(\lambda)$ soundness, conditionally reveals the secret that is shared between P_i and P_j after verifying the consistency of P_k's input. For conciseness, we will drop 1^λ in the argument, and denote $\mathrm{CD}_{\{i,j\},k}.\mathsf{Enc}(1^\lambda, (i,\ell), \cdot, \cdot)$ by $\mathrm{CD}_{\{i,j\},k}.\mathsf{Enc}_{(i,\ell)}(\cdot,\cdot)$, for all $\ell \in \{j,k\}$, and so on.

For legibility, we suppress the private randomness used in the encoder of Π_f as well as the conditional disclosure protocols invoked as sub-protocols. But, we stress that, an honest party uses the same private randomness while invoking the encoder to obtain their input to different 2-ary functions.

Π_f

(i). Execution of Enc

For any $j \neq i$, $(1^\lambda, x_i, \gamma_i[j], \{s_i[j,k], s_i[k,j]\}_{k \notin \{i,j\}}, R_i) \leftarrow \mathsf{Enc}(1^\lambda, (i,j), x_i, \cdot)$.
Here, $\gamma_i[j] \in \{0,1\}^l$ is such that $\{\gamma_i[k]\}_{k \neq i}$ is an additive secret sharing of $0 \in \mathbb{G}$; for each k, $s_i[j,k]$ and $s_i[k,j]$ are uniform in $\mathbb{G}$; R_i is a uniformly random string used to derive private randomness for the encoders of various parallel invocations of conditional disclosure protocols; see (ii).2.(a-c).
Recall, an honest P_i uses the same randomness (and input x_i) while invoking $\mathsf{Enc}(1^\lambda, (i,j), \cdot, \cdot)$ for all $j \neq i$. This is crucial in ensuring the required correlation among P_i's inputs to $\{\mathcal{O}_{\{i,j\}}\}_{j \neq i}$.

(ii). Execution of $\mathcal{O}_{\{i,j\}}$

When P_i's input is $(\lambda_i, x_i, \gamma_i[j], \{s_i[j,k], s_i[k,j]\}_{k \notin \{i,j\}}, R_i)$, and P_j's input is $(\lambda_j, x_j, \gamma_j[i], \{s_j[i,k], s_j[k,i]\}_{k \notin \{i,j\}}, R_j)$, $\mathcal{O}_{\{i,j\}}$ behaves as follows:

1. Compute $z_{\{i,j\}} = h_{\{i,j\}}(x_i, x_j)$. Output $\hat{z}_{\{i,j\}}$ where $\hat{z}_{\{i,j\}} = z_{\{i,j\}} + \gamma_i[j] + \gamma_j[i] + \sum_{k \notin \{i,j\}} (s_i[j,k] + s_j[i,k])$.
2. If $\lambda_j \neq \lambda_i$, output $\perp$ and terminate; otherwise, let $\lambda = \lambda_i$; for each $k \notin \{i,j\}$:
   (a) Output $\mathrm{CD}_{\{i,j\},k}.\mathcal{O}_{\{i,j\}}(y_{(i,j)}, y_{(j,i)})$, where $y_{(i,j)} \leftarrow \mathrm{CD}_{\{i,j\},k}.\mathsf{Enc}_{(i,j)}(s_i[j,k], \cdot)$ and $y_{(j,i)} \leftarrow \mathrm{CD}_{\{i,j\},k}.\mathsf{Enc}_{(j,i)}(s_j[i,k], \cdot)$. Note, fresh randomness from R_i, say $r_{i,(\{i,j\},k)}$, is used to compute P_i's encodings $\mathrm{CD}_{\{i,j\},k}.\mathsf{Enc}_{(i,j)}$ and $\mathrm{CD}_{\{i,j\},k}.\mathsf{Enc}_{(i,k)}$ (in $\mathcal{O}_{\{i,k\}}$); same for P_j.
   (b) Output $\mathrm{CD}_{\{i,k\},j}.\mathcal{O}_{\{i,j\}}(y_{(i,j)}, y_{(j,i)})$, where $y_{(i,j)} \leftarrow \mathrm{CD}_{\{i,k\},j}.\mathsf{Enc}_{(i,j)}(s_i[k,j], \cdot)$ and $y_{(j,i)} \leftarrow \mathrm{CD}_{\{i,k\},j}.\mathsf{Enc}_{(j,i)}(x_j, \cdot)$.
   (c) Output $\mathrm{CD}_{\{j,k\},i}.\mathcal{O}_{\{i,j\}}(y_{(i,j)}, y_{(j,i)})$, where $y_{(i,j)} \leftarrow \mathrm{CD}_{\{j,k\},i}.\mathsf{Enc}_{(i,j)}(x_i, \cdot)$, and $y_{(j,i)} \leftarrow \mathrm{CD}_{\{j,k\},i}.\mathsf{Enc}_{(j,i)}(s_j[k,i], \cdot)$.

(iii). Execution of Dec.

For each $\{i,j,k\} \in \binom{[n]}{3}$, let $s_{\{i,j\},k}$ be the output of $\mathrm{CD}_{\{i,j\},k}$. If $\hat{z}_{\{i,j\}} = \perp$ for some $\{i,j\}$ or $s_{\{i,j\},k} = \perp$ for some $\{i,j,k\}$, output $\perp$ and terminate; otherwise set $\hat{w}_{\{i,j\}} = \hat{z}_{\{i,j\}} - \sum_{k \notin \{i,j\}} s_{\{i,j\},k}$. Output $\sum_{\{i,j\}} \hat{w}_{\{i,j\}}$.

Fig. 6. Π_f computes f of effective degree 2 with privacy with knowledge of output.

Correctness of the protocol. When all parties behave honestly, the output of $\mathrm{CD}_{\{i,j\},k}$ is $s_{\{i,j\},k} = s_i[j,k] + s_j[i,k]$ for each $\{i,j,k\} \in \binom{[n]}{3}$. This follows from the correctness of conditional disclosure protocol. Furthermore, $\{\gamma_i[j]\}_{j \neq i}$ forms

an additive secret sharing of 0. Hence, the decoder outputs the following ensuring correctness:

$$\hat{z}_{\{i,j\}} - \sum_{k \notin \{i,j\}} s_i[j,k] + s_j[i,k] = \sum_{\{i,j\}} h_{\{i,j\}}(x_i, x_j) + \sum_i \sum_{j \neq i} \gamma_i[j] = f(x_1, \ldots, x_n).$$

The security with PwKO of the protocol follows the outline presented in the technical overview. A formal proof is provided in the full version [NPS24].

Theorem 6.5. *The protocol Π_f in Fig. 6 computes any n-party function f of effective degree 2 with statistical security while guaranteeing privacy with knowledge output against a malicious adversary that corrupts any set of parties.*

6.3 Achieving General Secure Computation with Abort

We use the protocol developed in the previous section to realize general secure computation with abort against computationally bounded adversaries. For this, we will rely on a computational semi-malicious MPRE, as defined in Definition 4.2. In [ABT18, Theorem 7.3], Applebaum, Brakerski and Tsabary showed that every n-party functionality f can be encoded by a computational degree-2 MPRE $\hat{f}$ with complexity polynomial in n and size of the circuit, by making non-black box use of a (possibly multi-round) oblivious transfer.

Since degree-2 MPRE has effective degree-2, we can compute f by first computing MPRE using the protocol in Fig. 6, and then decode the MPRE to compute the function. Owing to computational security of the MPRE, the resulting protocol securely computes f with PwKO against a computationally bounded adversary corrupting any subset of parties.

We use a standard approach to bootstrap this protocol and achieve security with abort. Instead of computing f in the aforementioned manner, compute a signed version of f in which the output of f is appended with one-time signatures (See Definition 4.3) of this value with respect to a signing key provided by each party. All parties can verify the validity of the signatures using the respective verifying keys. For this, every party broadcasts their verifying key in parallel with the function computation. If at least one of the signatures is found to be inconsistent, the parties unanimously declare abort instead of outputting the candidate output. The resulting protocol securely computes f with abort against a computationally unbounded adversary.

When all parties behave honestly, authenticity of the signature scheme (Definition 4.3) guarantees that the verification succeeds and the parties output the function output, ensuring correctness. Whereas, an adversary that attempts to modify the function output will fail to provide consistent signatures with overwhelming probability by the unforgeability property (Definition 4.3) of the one-time signature scheme, resulting in all honest parties issuing abort. Here, we have crucially relied on the PwKO guarantee of the underlying protocol to ensure that the adversary does not learn the signing key of any of the honest parties. We defer the formal proof of the following result to the full version [NPS24].

Theorem 6.6. *Assuming the existence of (possibly multi-round) oblivious transfer that is secure against semi-honest adversaries, for every n-party functionality f, there exists a non-interactive protocol using parallel calls to 2-ary functions that achieves security with unanimous abort against a computationally bounded malicious adversary that corrupts an arbitrary number of parties.*

6.4 Extensions

Honest Majority Setting. The following result is obtained by instantiating the same construction with semi-maliciously secure 2-MPRE against a dishonest minority, which exists if one-way functions exist [ABT18]:

Theorem 6.7. *Assuming the existence of one-way functions, for every n-party functionality f, there exists a non-interactive protocol using parallel calls to 2-ary functions that achieves security with unanimous abort against a computationally bounded malicious adversary that corrupts a strict minority of parties.*

3-ary functions. We consider the same problem in the setting where the parties have access to 3-ary functions instead of 2-ary. We obtain the following result.

Theorem 6.8. *Assuming the existence of one-way functions, for every n-party functionality f, there exists a non-interactive protocol using parallel calls to 3-ary functions that achieves security with unanimous abort against a computationally bounded malicious adversary that corrupts an arbitrary number of parties.*

This theorem is obtained by first constructing a protocol that securely computes degree-3 polynomials with PwKO. Then, it uses the MPRE construction from Beaver, Micali, and Rogaway [BMR90] in the bootstrapping step. We defer the construction and its analysis to the full version [NPS24].

References

ABG+20. Applebaum, B., Brakerski, Z., Garg, S., Ishai, Y., Srinivasan, A.: Separating two-round secure computation from oblivious transfer. In: Thomas Vidick, editor, ITCS 2020, vol. 151, pp. 1–18, LIPIcs (2020)

ABT18. Applebaum, B., Brakerski, Z., Tsabary, R.: Perfect secure computation in two rounds. In: Beimel and Dziembowski, pp. 152–174 (2021)

ABT19. Applebaum, B., Brakerski, Z., Tsabary, R.: Degree 2 is complete for the round-complexity of malicious MPC. In: Ishai and Rijmen, pp. 504–531 (2019)

ACGJ18. Ananth, P., Choudhuri, A.R., Goel, A., Jain, A.: Round-optimal secure multiparty computation with honest majority. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018. Part II, volume 10992 of LNCS, pp. 395–424. Springer, Heidelberg (2018)

ACGJ19. Ananth, P., Choudhuri, A.R., Goel, A., Jain, A.: Two round information-theoretic MPC with malicious security. In: Ishai and Rijmen, pp. 532–561 (2019)

AG21. Applebaum, B., Goel, A.: On actively-secure elementary MPC reductions. In: Nissim, K., Waters, B. (eds.) TCC 2021. Part I, volume 13042 of LNCS, pp. 717–749. Springer, Heidelberg (2021)
AIK04. Applebaum, B., Ishai, Y., Kushilevitz, E.: Cryptography in NC^0. In: 45th FOCS, pp. 166–175. IEEE Computer Society Press (2004)
AIK07. Applebaum, B., Ishai, Y., Kushilevitz, E.: Cryptography with constant input locality. In: Menezes, A. (ed.) CRYPTO 2007. LNCS, vol. 4622, pp. 92–110. Springer, Heidelberg (2007)
BD18. Beimel, A., Dziembowski, S. (eds.): TCC 2018, Part I. LNCS, vol. 11239. Springer, Heidelberg (2018)
BL18. Benhamouda, F., Lin, H.: k-round multiparty computation from k-round oblivious transfer via garbled interactive circuits. In: Nielsen and Rijmen, pp. 500–532 (2018)
BMR90. Beaver, D., Micali, S., Rogaway, P.: The round complexity of secure protocols (extended abstract). In: 22nd ACM STOC, pp. 503–513. ACM Press (1990)
BOSS20. Baum, C., Orsini, E., Scholl, P., Soria-Vazquez, E.: Efficient constant-round MPC with identifiable abort and public verifiability. In: Micciancio, D., Ristenpart, T. (eds.) CRYPTO 2020. Part II, volume 12171 of LNCS, pp. 562–592. Springer, Heidelberg (2020)
Can01. Canetti, R.: Universally composable security: a new paradigm for cryptographic protocols. In: 42nd FOCS, pp. 136–145. IEEE Computer Society Press (2001)
Cle86. Cleve, R.: Limits on the security of coin flips when half the processors are faulty (extended abstract). In: 18th ACM STOC, pp. 364–369. ACM Press (1986)
DI05. Damgård, I., Ishai, Y.: Constant-round multiparty computation using a black-box pseudorandom generator. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 378–394. Springer, Heidelberg (2005)
FGMO01. Fitzi, M., Garay, J.A., Maurer, U.M., Ostrovsky, R.: Minimal complete primitives for secure multi-party computation. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 80–100. Springer, Heidelberg (2001)
FOC86. 27th FOCS.: IEEE Computer Society Press (1986)
GIS18. Garg, S., Ishai, Y., Srinivasan, A.: Two-round MPC: information-theoretic and black-box. In: Beimel and Dziembowski, pp. 123–151 (2018)
GMW86. Goldreich, O., Micali, S., Wigderson, A.: Proofs that yield nothing but their validity and a methodology of cryptographic protocol design (extended abstract). In: FOCS 1986, pp. 174–187 (2019)
GMW87. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game or a completeness theorem for protocols with honest majority. In: Alfred Aho, editor, 19th ACM STOC, pp. 218–229. ACM Press (1987)
GS17. Garg, S., Srinivasan, A.: Garbled protocols and two-round MPC from bilinear maps. In: Chris Umans, editor, 58th FOCS, pp. 588–599. IEEE Computer Society Press (2017)
GS18. Garg, S., Srinivasan, A.: Two-round multiparty secure computation from minimal assumptions. In: Nielsen and Rijmen, pp. 468–499 (2022)
HIK+11. Haitner, I., Ishai, Y., Kushilevitz, E., Lindell, Y., Petrank, E.: Black-box constructions of protocols for secure computation. SIAM J. Comput. **40**(2), 225–266 (2011)
HILL99. Håstad, J., Impagliazzo, R., Levin, L.A., Luby, M.: A pseudorandom generator from any one-way function. SIAM J. Comput. **28**(4), 1364–1396 (1999)

HKW20. Hohenberger, S., Koppula, V., Waters, B.: Chosen ciphertext security from injective trapdoor functions. In: Micciancio, D., Ristenpart, T. (eds.) CRYPTO 2020. Part I, volume 12170 of LNCS, pp. 836–866. Springer, Heidelberg (2020)

IK00. Ishai, Y., Kushilevitz, E.: Randomizing polynomials: a new representation with applications to round-efficient secure computation. In: 41st FOCS, pp. 294–304. IEEE Computer Society Press (2000)

IKP10. Ishai, Y., Kushilevitz, E., Paskin, A.: Secure multiparty computation with minimal interaction. In: Rabin, T. (ed.) CRYPTO 2010. LNCS, vol. 6223, pp. 577–594. Springer, Heidelberg (2010)

IKSS22. Ishai, Y., Khurana, D., Sahai, A., Srinivasan, A.: Round-optimal black-box secure computation from two-round malicious OT. In: Kiltz, E., Vaikuntanathan, V. (eds.) Theory of Cryptography - 20th International Conference, TCC 2022, Chicago, IL, USA, 7-10 November, 2022, Proceedings, Part II, LNCS, vol. 13748, pp. 441–469. Springer, cham (2022). https://doi.org/10.1007/978-3-031-22365-5_16

IPP+22. Ishai, Y., Patra, A., Patranabis, S., Ravi, D., Srinivasan, A.: Fully-secure MPC with minimal trust. In: Kiltz, E., Vaikuntanathan, V. (eds.) Theory of Cryptography - 20th International Conference, TCC 2022, Chicago, IL, USA, 7-10 November 2022, Proceedings, Part II, LNCS, vol. 13748, pp. 470–501. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-22365-5_17

IPS08. Ishai, Y., Prabhakaran, M., Sahai, A.: Founding cryptography on oblivious transfer - efficiently. In: Wagner, D. (ed.) CRYPTO 2008. LNCS, vol. 5157, pp. 572–591. Springer, Heidelberg (2008)

IR19. Ishai, Y., Rijmen, V. (eds.): EUROCRYPT 2019, Part II. LNCS, vol. 11477. Springer, Heidelberg (2019)

Kil88. Kilian, J.: Founding cryptography on oblivious transfer. In: 20th ACM STOC, pp. 20–31. ACM Press (1988)

Lam16. Lamport, L.: Constructing digital signatures from a one way function (2016)

NPS24. Narayanan, V., Pawar, S.V., Srinivasan, A.: Secure computation with parallel calls to 2-ary functions. Cryptology ePrint Archive (2024)

NR18. Nielsen, J.B., Rijmen, V. (eds.) EUROCRYPT 2018, Part II, LNCS, vol. 10821. Springer, Heidelberg (2018)

Yao86. Yao, A.C.C.: How to generate and exchange secrets (extended abstract). In: FOCS, pp. 162–167 (1986)

Efficient Secure Communication over Dynamic Incomplete Networks with Minimal Connectivity

Ivan Damgård[1], Divya Ravi[2], Lawrence Roy[1], Daniel Tschudi[3,4(✉)], and Sophia Yakoubov[1]

[1] Aarhus University, Aarhus, Denmark
{ivan, lance.roy, sophia.yakoubov}@cs.au.dk
[2] University of Amsterdam, Amsterdam, Netherlands
tschudid@gmail.com
[3] Concordium, Zurich, Switzerland
[4] Institute for Network and Security, Eastern Switzerland University of Applied Sciences (OST), Rapperswil, Switzerland

Abstract. We study the problem of implementing unconditionally secure reliable and private communication (and hence secure computation) in dynamic incomplete networks. Our model assumes that the network is always k-connected, for some k, but the concrete connection graph is adversarially chosen in each round of interaction. We show that, with n players and t malicious corruptions, perfectly secure communication is possible if and only if $k > 2t$. This disproves a conjecture from earlier work, that $k > 3t$ is necessary. Our new protocols are much more efficient than previous work; in particular, we improve the round and communication complexity by an exponential factor (in n) in both the semi-honest and the malicious corruption setting, leading to protocols with polynomial complexity.

1 Introduction

This paper studies unconditionally secure communication (and by extension multiparty computation) when parties communicate over an incomplete and dynamic network. More specifically, we assume synchronous communication with secure point-to-point channels; however, only some of the point-to-point connections actually exist, so the network is incomplete. Moreover, the set of active connections can change from one round to the next, and the graph describing

Funded in part by the European Research Council (ERC) under the European Unions's Horizon 2020 research and innovation program under grant agreement No 669255 (MPCPRO) and No 803096 (SPEC), the Danish Independent Research Council (grant DFF-0165-00107B "C3PO", grant DFF-2064-00016B / DFF-2032-00122B "YOSO"), and the DARPA SIEVE program (contract HR001120C0085 "FROMAGER"). Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of DARPA. Distribution Statement "A" (Approved for Public Release, Distribution Unlimited). We thank the authors of [DRTY23] for the source code of the tables and protocol boxes.

E. Boyle and M. Mahmoody (Eds.): TCC 2024, LNCS 15367, pp. 266–292, 2025.
https://doi.org/10.1007/978-3-031-78023-3_9

the active connections is adversarially chosen in each round. This is called a *dynamic* incomplete network, in contrast to a *static* incomplete network, where the active connections stay the same throughout the protocol.

Static incomplete networks were first studied by Dolev [Dol82]. Here, it was shown that when t of the n parties are malicious, one can do secure broadcast if and only if the network is at least $2t+1$-connected[1], and $3t < n$. Later, Dolev et al. [DDWY93] showed that, in a static incomplete network, one can communicate privately and reliably if and only if the network is $2t+1$-connected. Using the protocols from that work, one can emulate a complete network with secure point-to-point channels. This can be combined with any MPC protocol that is based on a complete network, and one can then conclude that, in a static incomplete network, $2t+1$-connectivity is necessary and sufficient for unconditionally secure MPC to be possible.

The case of dynamic incomplete networks was first considered in Maurer *et al.* [MTD15], who studied reliable (non-private) communication. They defined a notion called *dynamic min-cut* which is a number that can be derived from the entire sequence of networks graphs. They then showed that reliable communication is possible between any pair of parties if and only if the dynamic min-cut is larger than $2t$. Although this is the weakest possible condition that allows for reliable communication, it makes a rather complicated statement on the entire sequence of network graphs, and in a given practical setting it would be hard to assess whether it is going to be satisfied or not. Also, from a theoretical standpoint, it is natural to ask if there is a condition on each *individual* network graph that would enable both reliable and private communication.

These questions were considered in [DRTY23]. They initiated the study of (unconditionally) private communication in dynamic networks, and introduced a model that is a natural extension of the static case, where it is required that in each round, the network graph for that round is at least k-connected (and this is the only condition assumed). The model further assumes that honest parties do not know the network topology. This is reasonable, as connections may be down because mobile devices move, or equipment crashes, and such events cannot be predicted locally. Finally, it is assumed that at most t parties are corrupted by an adversary, either passively (semi-honest) or actively (malicious).

The main results from [DRTY23] are as follows: reliable communication is possible in a dynamic network if and only if $k > 0$ in the passive case and $k > 2t$ in the active case. Private communication can be done for a passive adversary, if and only if $k > t$. For an active adversary, $k > 3t$ is sufficient for perfectly secure private communication, whereas $k > 2t$ is necessary (which follows from known results in the static case). It was conjectured that in fact $k > 3t$ is necessary for perfect security (whereas they showed that $k > 2t$ is sufficient for statistical security).

[1] A graph is k-connected if any pair of distinct nodes are connected by at least k disjoint paths, or equivalently, if it remains connected when one removes any set of less than k vertices.

The protocols from [DRTY23] (and [MTD15]) introduce quite a large performance penalty compared to the static case. They work by trying many different paths from sender to receiver in the hope that the message will eventually arrive along enough disjoint paths. For private communication, key material is sent along many different paths in the hope that some of it will make it to the sender via paths consisting of only honest players, which allows extraction of a key that is completely unknown to the adversary. Unfortunately, the upper bounds shown for both round and communication complexity of these approaches were exponential in n in the worst case.

1.1 Our Contributions

In this paper, we disprove the conjecture from [DRTY23] and show that in fact $k > 2t$ is sufficient for perfect private communication. Since this condition was already known to be necessary, this completes the characterization of dynamic incomplete networks allowing for private and reliable communication. We also give new protocols for private communication in the passive case, and for reliable and private communication in the active case. The protocols are based on several new techniques, allowing us to remove the exponential dependency on n in the protocols from [DRTY23]. Indeed, all our protocols have complexity polynomial in n for all values of n, k and t for which secure communication is possible.

We stick to the network model from [DRTY23] for most of the paper, as it allows for simple descriptions of protocols and clean statements about their properties. But we emphasize that our protocols do not need to assume that the network graph is k-connected in every round, and in fact they work under much weaker assumptions on connectivity. We discuss this in more detail in Sect. 6.

The complexities of our constructions are summarized in Tables 1 and 2.

1.2 Technical Overview

Reliable Communication. Reliable communication with a passive adversary can be done as long as $k > 0$. A simple flooding approach will work in n rounds, as was already noted in [DRTY23]. For reliable communication in the active adversary case we introduce a new protocol and proof. In a nutshell, we also use a flooding approach here, but each copy of the message carries metadata, namely a graph describing the paths the message traveled along to reach the current party. The party assembles new metadata based on what it received, and passes it along in the next round. After some number of rounds, the final receiver R has received possibly several different messages and metadata (as the adversary is free to fabricate incorrect messages and metadata). We show that after $\mathcal{O}(n)$ rounds, R has enough data to decide with probability 1 what the correct message is.

An overview of how we arrive at this result: Earlier work for dynamic networks tracked all paths on which a message has traveled, and the receiver would believe a message if it arrives via more than t disjoint paths, as this implies it must have traveled on a path with only honest players. Unfortunately, the

Table 1. Reliable Communication Protocols for a message m, over k-connected networks. The communication complexity is the total communication of honest parties in the protocol. M denotes the total bits of all messages sent by corrupt parties (which assuming PPT adversaries remains polynomial).

Scheme	Corruption		Graph	Complexity	
	Type	Threshold	k	Rounds	Communication
[DRTY23], Protocol 1	passive	$t < n$	$k \geq 1$	n	$\mathcal{O}(n^3\|m\|)$
[DRTY23], Protocol 2	active	$t < \frac{n}{2}$	$k > 2t$	$\mathcal{O}(n2^n)$	$\mathcal{O}(n^4 2^{2n}\|m\|)$
This work, Protocol 3	active	$t < \frac{n}{2}$	$k > 2t$	$\leq n$	$\mathcal{O}(n^3\|m\| + (M+1)n^6 \log_2(n))$

number of potential paths can be exponential, leading to inefficient protocols. An obvious idea for improvement is to collect the paths on which the message traveled into a graph that can hopefully be described more compactly than the set of paths. In case of a static network, it is quite straightforward to see that this will work: the receiver can decide whether to believe a certain message m based on the max-flow or equivalently min-cut of the graph that is sent along with m (considering cuts separating sender from receiver). If this number is larger that t, the sender believes the message.

However, things get more complicated in the dynamic case. Min-cut is not defined for a dynamic network, so we construct a different type of graph that captures the history of a message traveling through the network. There is a node for each pair (P_i, j), where P_i is one of the parties and j indicates a round in the protocol. We put an edge from $(\mathsf{P}_i, j-1)$ to $(\mathsf{P}_{i'}, j)$ if P_i sent the message to $\mathsf{P}_{i'}$ in round j (and by default a party is connected to itself from each round to the next)[2]. We then introduce a notion called relaxed labelled min-cut in this type of graph, where we allow fractions as values, in contrast to the standard notion of min-cut which is always an integer. This allows us to formulate an efficiently computable predicate which we show is satisfied after n rounds by the graph traveling with the correct message, and cannot be satisfied for any incorrect message.

Private Communication. For private communication, our main idea is as follows: instead of trying to send the message via many paths, as in [DRTY23], we ask each party to choose a secret key for each other party and try to send,

[2] For technical reasons, we even need to have several nodes for one party in each round. As we explain later, this has to do with the fact that a corrupt party can claim they heard a fake message from an honest party P_h, but at the same time P_h might report the same fake message because he heard from another corrupt party. These two "stories" must be treated separately and we do this by having two different nodes for P_h in the relevant round.

Table 2. Private Communication Protocols for a message m, over k-connected networks with corruption threshold t. The complexity of Protocol 4 is in terms of the complexities of its building blocks, namely reliable communication and secure message transmission; where $\rho_{\mathtt{Rel}}$, $\rho_{\mathtt{SMT}}$ denote the round complexities and $c_{\mathtt{Rel}}(x)$, $c_{\mathtt{SMT}}(x)$ denote the communication complexities for transmitting x bits. There exist instantiations of these building blocks that maintain these complexities to be polynomial (elaborated in the relevant technical section). Note that the protocols in this paper are the first to offer round and communication complexity which is sub-exponential in the number of parties. All protocols in the table have perfect security.

Scheme	Corruption		Graph		Complexity	
	Type	t	k	directed	Rounds	Communication
[DRTY23], Protocol 3	passive	$t < n$	$k > t$	✓	$\mathcal{O}(n2^n)$	$\mathcal{O}\big(2^{2n}n^3(n+\lvert m\rvert)\big)$
[DRTY23], Protocol 5	active	$t < \frac{n}{3}$	$k > 3t$	✓	$\mathcal{O}(n2^n)$	$\mathcal{O}\big(2^{n^2+n}n^3(n+\lvert m\rvert)\big)$
[DRTY23], Protocol 6	active	$t < \frac{n}{4}$	$k > 4t$	✓	$\mathcal{O}(n2^n)$	$\mathcal{O}(2^{3n}n^5(n+\lvert m\rvert))$
This work, Protocol 1 , Corollary 3	passive	$t < n$	$k > t$	✓	$2\lfloor\frac{n}{k}\rfloor + 3$	$\mathcal{O}\left(\frac{n^4}{k}(n\log(n)+\lvert m\rvert)\right)$
This work, Protocol 2 , Corollary 6	passive	$t < n$	$k > t$	✗	$\lfloor\frac{n}{k}\rfloor + 2$	$\mathcal{O}\left(\frac{n^4\lvert m\rvert}{k}\right)$
This work, Protocol 4	active	$t < \frac{n}{3}$	$k > 2t$	✓	$1 + 2\rho_{\mathtt{Rel}} + \rho_{\mathtt{SMT}}\rho_{\mathtt{Rel}}$	$\mathcal{O}\big(nc_{\mathtt{Rel}}(n^2) + nc_{\mathtt{SMT}}(\lvert m\rvert)c_{\mathtt{Rel}}(n\log(n))\big)$

in a single round, each key to the party it was chosen for. Then we determine via public discussion which connections worked, using the fact that we already know how to do reliable (non-private) communication efficiently. Intuitively, this "freezes" the network graph G that existed in the round where the keys were sent.

Skipping a few details, the keys exchanged can be used to send[3] data privately along a path in G, such that the adversary will have no information on what is sent if the path contains only honest players. This means that in the passive case, the sender can secret-share its message additively and send shares privately to the receiver along a set of disjoint paths in G. This will work because G is sufficiently connected so that at least one share will remain unknown to the adversary.

In the active case, we assume the G is at least $2t + 1$-connected. We can think of the (at least) $2t + 1$ disjoint paths from sender to receiver as channels connecting sender to receiver, of which at most t can be corrupted. We can then use known efficient protocols for maliciously secure perfect message transmission to send a private message.

This disproves the conjecture from [DRTY23], that connectivity must be at least $3t + 1$. In [DRTY23] evidence was given for the conjecture by arguing that any protocol in the class of solutions they considered would fail for connectivity less than $3t + 1$. As also noted there, this is of course not a proof, and indeed

[3] The actual communication is done using multicast which can be achieved with one of the reliable communication protocols from above.

our protocol falls outside the class because it crucially uses public discussion to decide on the paths to use later. This option was not considered in [DRTY23].

2 Preliminaries

In this work we consider the setting as defined in [DRTY23]. Let $\mathbb{F}$ be a finite field. Let $\mathcal{P} = \{\mathsf{P}_1, \ldots, \mathsf{P}_n\}$ denote the set of n parties. The *sender* $\mathsf{S} \in \mathcal{P}$ wants to send a message $m \in \mathbb{F}$ to *receiver* $\mathsf{R} \in \mathcal{P}$.

2.1 Adversary Model

A computationally-unbounded, central adversary corrupts at most t parties. The party corruption is *static*, i.e., the adversary is required to select the set of corrupted parties before the protocol execution. We distinguish between *passive* corruption where the adversary can access the internal state of corrupted parties and *active* corruption where the adversary additionally has full control over the behavior of corrupted parties.

2.2 Communication Network

Parties communicate over a *dynamic incomplete network* of secure (private and authentic) synchronous channels in the presence of a *rushing adaptive* network adversary. In more details, the protocol proceeds in rounds, also known as time steps. Let $\mathcal{G}$ be a publicly-known family of graphs over $\mathcal{P}$. Intuitively, the graph family $\mathcal{G}$ models the network guarantees for honest parties. For example, $\mathcal{G}$ could be the family of connected graphs. In each round a party *may attempt* to use any channel in their neighborhood of $\bar{\mathsf{G}} = \bigcup_{\mathsf{G} \in \mathcal{G}} \mathsf{G}$[4]. The adversary decides on the actual communication graph $\mathsf{G}_r \in \mathcal{G}$ for round r after having observed the communication attempts. Any message input on a channel within G_r will then be delivered by the end of the round. We note that honest parties are *oblivious* of the actual communication graph G_r. In particular, honest parties do not learn which of their outgoing transmissions were successful. If not specified otherwise, the graphs in $\mathcal{G}$ may be directed graphs.

2.3 Security

In this work we consider protocols that achieve a communication channel with *perfect security* between sender and receiver.

Reliable Communication. A reliable communication channel allows a sender S to send a message to a receiver R in a tamper-resilient manner.

Definition 1 (Reliable Communication). *A protocol achieves a* reliable *communication channel with perfect security between* S *and* R *in the presence of an adversary* $\mathcal{A}$ *if the following holds:*

Correctness. *The output message* m_{R} *of* R *is the input message* m *of* S*, i.e.* $\Pr[m_{\mathsf{R}} \neq m] = 0$ *where the randomness is over the coins of all honest parties and* $\mathcal{A}$*.*

[4] $\bar{\mathsf{G}}$ is often the fully-connected graph, so every party's neighborhood is then all of $\mathcal{P}$.

Private Communication. A private communication channel allows a sender S to send a message to a receiver R in a reliable and private manner, i.e. the adversary will not learn any information on the message.

Definition 2 (Private Communication). *A protocol achieves a* private *communication channel with perfect security in the presence of an adversary $\mathcal{A}$ if it achieves a* reliable *communication channel, and additionally the following holds:*

Privacy. *The view of adversary $\mathcal{A}$ can be simulated from the inputs and outputs of corrupted parties. In particular, for honest S, R the adversarial view is independent of the sender's input message m.*

2.4 Graphs

In this section we define the graph properties we consider for the communication network in our protocols.

Definition 3. *A (directed) graph is* (u, v)-k-connected *if for nodes u, v there exist k disjoint (directed) paths from u to v.*

Definition 4. *A (directed) graph is* k-connected *if it is (u, v)-k-connected for any pair (u, v) of nodes.*

A 1-connected graph is simply called *connected*. We say graph family $\mathcal{G}$ has property X if every graph in $\mathcal{G}$ has property X.

3 Private Communication with Passive Corruption

Damgård et al. [DRTY23] presented a private communication protocol that tolerates $t < n$ passive corruptions in a dynamic network with connectivity $k > t$. Both the round and communication complexity of this protocol scales linearly with the cardinality of a given set of paths Paths between sender and receiver. If Paths happens to be the set of all possible paths, its cardinality is exponentially large in n. Motivated by the goal of improving the efficiency of this protocol, we propose a private communication protocol in this setting where the round and communication complexity is polynomial in n.

Reliable multicast as building block. Before describing our protocol, we note that in the passive setting reliable multicast is easy to achieve using a flooding approach if $\mathcal{G}$ is connected. All parties that have seen the sender's message will repeatedly attempt to send it to all their neighbors. We will use this multicast primitive in our protocol for secure communication.

Lemma 1 (From [DRTY23]). *For any connected $\mathcal{G}$ there exists a protocol* $\mathtt{MultiCast}(\mathsf{P}, m)$ *that allows party P to safely distribute m in the presence of a passive adversary. The protocol runs for n rounds and has a total communication complexity of $\mathcal{O}(n^3|m|)$ bits, where $|m|$ denotes the number of message bits.*

Remark 1. The flooding protocol in [DRTY23] is used to construct a reliable communication channel from the sender to a specific receiver. However, we note that the flooding approach guarantees that at the end of the protocol every party will have the sender's message. So by modifying the protocol slightly, such that every party outputs the received message, the construction directly achieves multicast.

3.1 Private Communication on Directed Graphs

Assume that directed $\mathcal{G}$ has connectivity $k > t$, i.e. every graph in $\mathcal{G}$ has connectivity $k > t$. The main idea of Protocol 1 is as follows: In the first round, parties attempt to send a random field element to each of their potential neighbors. Then they each use `MultiCast` to announce the set of nodes from whom they actually received randomness. This publicly defines a (undirected) 'meta graph', say $\mathbb{G}$, whose edges correspond to pairs of nodes who now have shared randomness. Essentially, this serves as a means to freeze[5] the first graph chosen by the dynamic adversary. Now, the problem of private communication becomes much simpler as we can focus on this meta graph $\mathbb{G}$ which is guaranteed to have connectivity $k > t$. The sender S splits their message into $t+1$ sum shares which are now passed along $t+1$ disjoint paths in $\mathbb{G}$ between S and R. Communication along the edges in $\mathbb{G}$ is emulated using `MultiCast` and the shared randomness is used to encrypt messages.

Comparison with previous work. We point out that the idea of additively secret sharing the secret among a set of $t+1$ disjoint paths is similar to the protocol of Damgård et al. The crucial difference is that we depend on a *fixed set* of disjoint paths in the meta graph while their protocol considered *all possible sets* of disjoint paths over dynamic graphs. This allows us to achieve complexity that is polynomial in n.

Theorem 1. *Protocol 1 is a private communication protocol that achieves perfect security against $t < n$ passive corruptions for network $\mathcal{G}$ with connectivity $k > t$. The protocol communicates at most $n^2|m|$ bits over network $\mathcal{G}$, and sends at most $\mathcal{O}\big(n^2 \log(n) + n|m|\big)$ bits over multicast channels.*

Proof. We start with an observation on $\mathbb{G}$. Multicasting the sets In ensures that parties agree on the meta-graph $\mathbb{G}$. The connectivity $k > t$ of the first round graph ensures that there are at least $t+1$ disjoint paths from S to R in $\mathbb{G}$. We also note that due to Lemma 1 multicasting is possible as $\mathcal{G}$ is connected.

Correctness. There are $t+1$ disjoint paths from S to R in $\mathbb{G}$, thus $\mathsf{GoodPaths}$ exists. The agreement on $\mathbb{G}$ implies agreement on $\mathsf{GoodPaths}$. This ensures that parties agree on all necessary invocations of multicast in the second phase. For

[5] Technically, $\mathbb{G}$ is an undirected graph with the same node set as the first graph. The edge (u, v) is in $\mathbb{G}$ if and only if (u, v) or (v, u) are in the first graph.

each path p it holds that

$$s'_p = \left(\sum_{p_i \in \mathcal{N}_p \setminus \{\mathsf{R}\}} m_{p,p_i} \right) - o_{p_{\ell-1},p_\ell}$$
$$= s_p + o_{p_1,p_2} + \left(\sum_{p_i \in \mathcal{N}_p \setminus \{\mathsf{R},\mathsf{S}\}} o_{p_i,p_{i+1}} - o_{p_{i-1},p_i} \right) - o_{p_{\ell-1},p_\ell} = s_p.$$

This means the receiver will compute the right shares from the multicast messages and will output the sender's message m.

Privacy. The view of the adversary can be simulated from the output of an ideal secure channel between S and R. If sender or receiver are corrupted, the simulator knows the message m, and the selected graphs. This allows simulation of the adversarial view by executing the real protocol in the head. If neither the sender nor receiver are corrupted the view of the adversary can be simulated by executing the protocol in the head with $m = 0$ (or any other default value). The values in the first phase will be distributed exactly as in the real world. In the second phase there exists a path p in $\mathsf{GoodPaths}$ that consists of honest parties. The multicast messages for this path are, without knowledge of the involved shared randomness, distributed independent of the actual message. So the simulated view is indistinguishable from the real protocol. In particular, the adversary does not learn s_p. On the other paths the adversary may learn the shares. However, these shares are uniform random and independent of the message (if one does not know s_p), so the simulated view for those paths is again indistinguishable from the real protocol. *Complexity.* In the first round each party sends at most $n|m|$ bits. The first phase additionally sends n^2 bits over multicast channels. The second phase communicates at most $n(n\log(n) + |m|)$ over multicast channels. □

Corollary 1. *Protocol 1 runs for $2n+1$ rounds and has a total communication complexity of $\mathcal{O}(n^5 \log(n) + n^4|m|)$ bits per party if the multicast channels are instantiated with* `MultiCast`.

Proof. This follows from the numbers in Theorem 1 and Lemma 1. □

Observe that Protocol 1 only requires connectivity k in the first round; after that, connectivity 1 suffices.

Corollary 2. *Protocol 1 is a private communication protocol that achieves perfect security against $t < n$ passive corruptions for connected network $\mathcal{G}$ if the first round graph is guaranteed to be (S,R)-k-connected for $k > t$.*

Protocol 1: $\Pi^{\mathsf{prv}}_{\mathsf{perf},\mathsf{sh}}(\mathsf{S},\mathsf{R},m)$

The sender S and receiver R are public input. The sender S has message m as private input. Let $\bar{\mathsf{G}} = \bigcup_{\mathsf{G}\in\mathcal{G}} \mathsf{G}$.

Establishing meta graph and shared randomness. Each party P_i does the following:

- For each neighbor P_j in $\bar{\mathsf{G}}$, sample uniform random $r_{i,j}$ (of size $|m|$).
- In round 1, attempt to send $r_{i,j}$ to P_j using the communication network.
- Let In_i denote the set of parties P_j from whom P_i actually received randomness $r_{j,i}$ in round 1.
- Parties jointly invoke $\texttt{MultiCast}(\mathsf{P}_i, \mathsf{In}_i)$ where In_i is encoded as a n-bit vector.
- Build the meta graph $\mathbb{G}$ as follows: there is an edge from P_u to P_v if $\mathsf{P}_u \in \mathsf{In}_v$ or $\mathsf{P}_v \in \mathsf{In}_u$.
- For each neighbor P_j in $\mathbb{G}$ set $o_{i,j} \coloneqq r_{j,i} + r_{i,j}$. (Missing values $r_{\cdot,\cdot}$ are set to 0; that is, if $\mathsf{P}_j \notin \mathsf{In}_i$, $o_{i,j} = r_{i,j}$, and if $\mathsf{P}_i \notin \mathsf{In}_j$, $o_{i,j} = r_{j,i}$.)

Secure message transfer in $\mathbb{G}$. All the parties can now locally determine the same set of $t+1$ disjoint paths, say $\mathsf{GoodPaths}$, in $\mathbb{G}$ between S and R (using Ford-Fulkerson algorithm)[a]. Then message transfer is done as follows:

- The sender S selects $t+1$ uniform random shares $\{s_p\}_{p\in\mathsf{GoodPaths}}$, such that $m = \sum_{p\in\mathsf{GoodPaths}} s_p$.
- The parties $\mathcal{N}_p = \{\mathsf{P}_{p_1}, \ldots, \mathsf{P}_{p_\ell}\}$ on each path p do the following:
  - The sender $\mathsf{S} = \mathsf{P}_{p_1}$ computes $m_{p,p_1} = s_p + o_{p_1,p_2}$ and all parties jointly invoke $\texttt{MultiCast}(\mathsf{S}, (p, m_{p,p_1}))$ where p is encoded in $n\log(n)$ bits.
  - Each party $\mathsf{P}_{p_i} \in \mathcal{N}_p \setminus \{\mathsf{S},\mathsf{R}\}$ computes $m_{p,p_i} = o_{p_i,p_{i+1}} - o_{p_{i-1},p_i}$ and all parties jointly invoke $\texttt{MultiCast}(\mathsf{P}_{p_i}, (p, m_{p,p_i}))$ where p is encoded in $n\log(n)$ bits.
- For each path p the receiver R computes share

$$s'_p = \left(\sum_{p_i \in \mathcal{N}_p \setminus \{\mathsf{R}\}} m_{p,p_i} \right) - o_{p_{\ell-1},p_\ell}.$$

- The receiver outputs $m' = \sum_{p\in\mathsf{GoodPaths}} s'_p$.

[a] In case of multiple disjoint sets, we can assume a lexicographic ordering among them and choose the smallest one.

Fig. 1. An efficient perfectly-secure private communication protocol against $t < n$ passive corruptions in a network with connectivity $k > t$.

Protocol 2: $\Pi(\mathsf{S}, \mathsf{R}, m)$

The identities of the sender S and receiver R are public input. The sender S additionally has message m as private input.
Let $\bar{\mathsf{G}} = \bigcup_{\mathsf{G} \in \mathcal{G}} \mathsf{G}$.

Establishing sharing of message. Party P_i does the following:
- If $\mathsf{P}_i = \mathsf{S}$, set $m_i = m$. Otherwise, set $m_i = 0$.
- For each neighbor P_j in $\bar{\mathsf{G}}$, sample uniform random $r_{i,j} \in \mathbb{F}$.
- Attempt to send $r_{i,j}$ to P_j using the normal channels. Denote by In_i the set of all parties P_j from whom P_i actually received randomness $r_{j,i}$ in this round.
- Compute $r_{i,i} = m_i - \sum_{j \in \mathsf{In}_i} r_{i,j}$.
- Compute $r_i = r_{i,i} + \sum_{j \in \mathsf{In}_i} r_{j,i}$.

Send message shares to receiver. Parties jointly invoke an instance of $\texttt{MultiCast}(\mathsf{P}_i, r_i)$ for $\mathsf{P}_i \neq \mathsf{R}$.

Reconstruction of message. The receiver $\mathsf{P}_i = \mathsf{R}$ computes $m = \sum_{\mathsf{P}_i \in \mathcal{P}} r_i$.

Fig. 2. An efficient perfectly-secure private communication protocol against $t < n$ passive corruptions in an undirected network with connectivity $k > t$.

We can improve on the round complexity of Protocol 1 by optimizing the multicast protocol.

Lemma 2. *If $\mathcal{G}$ has connectivity k, then the flooding protocol from Lemma 1 requires at most $\lfloor \frac{n}{k} \rfloor + 1$ rounds and total communication at most $(\lfloor \frac{n}{k} \rfloor + 1)n|m|$ to distribute a message m among all parties.*

Proof. We show this by induction. In each round at least k parties learn the message, until the final round where all remaining parties learn the message. Consider any round r and a party P that did not receive the message before r. There must be at least k disjoint paths from the sender to P in the round r graph. Along each such path, there must be a party who knows the message (possibly the sender) and a party who does not (possibly P). This means that P learns the message, or at least k other parties learn the message. Thus, in round r at least k parties will learn the message or all parties will know the message after the round. As there are n parties, there can be at most $\lfloor \frac{n}{k} \rfloor$ rounds where at least k parties learn the message. This means after $\lfloor \frac{n}{k} \rfloor + 1$ rounds, every party knows the message. □

Corollary 3. *If $\mathcal{G}$ has connectivity k, and the multicast channels in Protocol 1 are instantiated with* `MultiCast` *as in Lemma 2, the protocol runs for $2\lfloor \frac{n}{k} \rfloor + 3$ rounds, and has total communication complexity of $\mathcal{O}\left(\frac{n^4}{k}(n \log(n) + |m|)\right)$.*

In particular, if the connectivity k is a constant fraction of n, the protocol achieves a constant round complexity.

3.2 Private Communication on Undirected Graphs

In this section we consider undirected $\mathcal{G}$ with connectivity $t+1$. The undirectedness allows us to save (almost) half the round complexity compared to Protocol 1 using the following idea. Each party starts with a local value. For the sender, this is the message m. Every other party starts with 0. As in Protocol 1 , parties attempt to send a random field element to each potential neighbor. This defines an (undirected) meta-graph $\mathbb{G}$ of parties that have successfully exchanged random values. Since the graph is undirected, parties know their own neighborhood.[6] In contrast to the above protocol, parties do not need to learn the full $\mathbb{G}$; it is enough to know one's own neighborhood. Parties use successfully sent random values to define a sharing of their local value in their closed neighborhood; i.e., their own share is their local value minus the sum of sent out random values. Next, parties add up all the shares they hold (including their own share of their local value). This establishes an additive sharing of the sender's message. Everyone but the receiver multicasts their shares, allowing the receiver to reconstruct the message.

Remark 2. This protocol can be extended to undirected graphs. All that is needed is for parties to learn if their sent random values were received. To achieves this, parties could multicast the set from whom they received a random value in the first round. However, this brings the round complexity back to the one of Protocol 1 .

Theorem 2. *Protocol 2 is a private communication protocol that achieves perfect security against $t < n$ passive corruptions for an undirected, connected network $\mathcal{G}$ with connectivity $k > t$. The protocol communicates at most $n^2|m|$ over $\mathcal{G}$ and sends at most $n|m|$ bits over multicast channels.*

Proof. Correctness. The symmetry of $\mathcal{G}$ ensures that if $\mathsf{P}_j \in \mathsf{In}_i$, then P_j has received a random value from P_i and will use it to compute r_j. This implies

$$\begin{aligned} m &= \sum_{\mathsf{P}_i \in \mathcal{P}} r_i = \sum_{\mathsf{P}_i \in \mathcal{P}} \left(r_{i,i} + \sum_{j \in \mathsf{In}_i} r_{j,i} \right) = \sum_{\mathsf{P}_i \in \mathcal{P}} \left(m_i - \sum_{j \in \mathsf{In}_i} r_{i,j} + \sum_{j \in \mathsf{In}_i} r_{j,i} \right) \\ &= \sum_{\mathsf{P}_i \in \mathcal{P}} m_i + \underbrace{\sum_{\mathsf{P}_i \in \mathcal{P}} \sum_{j \in \mathsf{In}_i} r_{j,i} - \sum_{\mathsf{P}_i \in \mathcal{P}} \sum_{j \in \mathsf{In}_i} r_{i,j}}_{=0} \\ &= m \end{aligned}$$

□

The last two double sums cancel out as one sums up all the received random values and the other all the sent random values which is the same set. The

[6] This is because if a party received randomness from another party, it can infer that the other party received the randomness sent by it as well.

multicast ($\mathcal{G}$ is connected) ensures that the receiver gets all r_i. The receiver will therefore output m.

Privacy. If the sender or the receiver are corrupted, the adversarial view can be simulated by running the actual protocol in the head.

Otherwise, observe that the only message that depends on m is r_i for $\mathsf{P}_i = \mathsf{S}$ (which is multicast). There must exist an all-honest path from S to R, as $\mathcal{G}$ has connectivity of at least $t+1$. Let $\mathsf{S} = \mathsf{P}_{i_1}, \ldots, \mathsf{P}_{i_l} = \mathsf{R}$ be the parties on that path. If all other parties are corrupt, what the adversary learns is

$$\begin{aligned}
x_1 &= m - r_{i_1,i_2} + r_{i_2,i_1}, \\
x_2 &= 0 - r_{i_2,i_1} - r_{i_2,i_3} + r_{i_1 i_2} + r_{i_3,i_2}, \\
&\ldots \\
x_{l-2} &= 0 - r_{i_{l-2},i_{l-3}} - r_{i_{l-2},i_{l-1}} + r_{i_{l-3},i_{l-2}} + r_{i_{l-1},i_{l-2}}, \\
x_{l-1} &= 0 - r_{i_{l-1},i_{l-2}} - r_{i_{l-1},i_l} + r_{i_{l-2},i_{l-1}} + r_{i_l,i_{l-1}}.
\end{aligned}$$

Note that the values $r_{i_l,i_{l-1}}$ and r_{i_{l-1},i_l} only appear as summands in x_{l-1}; these are both randomly chosen, and thus $r_{i_l,i_{l-1}} - r_{i_{l-1},i_l}$ perfectly masks $r_{i_{l-2},i_{l-1}} - r_{i_{l-1},i_{l-2}}$. That in turn perfectly masks $r_{i_{l-3},i_{l-2}} - r_{i_{l-2},i_{l-3}}$ in x_{l-2}, and so on; finally, we get that $r_{i_1,i_2} - r_{i_2,i_1}$ perfectly masks m in x_1. We can conclude that r_i for $i \in \{i_1, \ldots, i_l\}$ (where r_i contains x_i as a summand) are independent and uniform. Thus, the adversarial view can be simulated by running the actual protocol in the head with $m = 0$.

Complexity. In the first round each party sends at most $n|m|$ bits. Additionally, there is communication of $n|m|$ bits over multicast channels. □

Corollary 4. *Protocol 2 runs for $n+1$ rounds and has a total communication complexity of $\mathcal{O}(n^4|m|)$ if the multicast channels are instantiated with* `MultiCast` *from Lemma 1.*

Proof. This follows from the numbers in Theorem 2 and Lemma 1. □

Observe that Protocol 2 only requires connectivity k in the first round; after that, connectivity 1 suffices.

Corollary 5. *Protocol 2 is a private communication protocol that achieves perfect security against $t < n$ passive corruptions for undirected connected network $\mathcal{G}$ if the first round graph is guaranteed to have connectivity $t+1$.*

We can improve on the round complexity of Protocol 2 by optimizing the multicast protocol.

Corollary 6. *If $\mathcal{G}$ has connectivity k, and the multicast channels in Protocol 1 are instantiated with* `MultiCast` *as in Lemma 2, the protocol runs for $\lfloor \frac{n}{k} \rfloor + 2$ rounds, and has total communication complexity of $\mathcal{O}\left(\frac{n^4|m|}{k}\right)$.*

4 Reliable Communication with Active Corruption

In this section, we construct a network flooding protocol for reliable communication with security against active adversaries. In this kind of protocol, the best attack is essentially to select t parties to corrupt, and have these parties ignore the honest messages and transmit lies. This attack is limited in two ways: every lie must originate from one of only t parties, while if the graph is at least k-connected then there must be at least $k - t$ disjoint paths from the original sender along which the honest message can pass. Since the sender did not send the lie, every path the lie was purported to have been sent along must go through some corrupt party. [DRTY23] therefore proposed flooding the network until a message is heard from more than t disjoint paths, which will happen eventually if $k - t > t$ (i.e., $k > 2t$). Unfortunately, there can be exponentially many paths from the sender in the graph, so their protocol does not work in polynomial time or polynomial communication.[7]

We present a more efficient flooding protocol, based on the idea of tracking the (polynomial-size) communication graph along which a message has been sent, instead of all paths. As motivation, we start with a protocol for the case of static graphs. Recall that every path along which a lie was purported to have been transmitted from S to a R must go through a corrupted party. Rephrased in terms of graphs, this means that the corrupted parties must form a (S, R)-cut[8] of this graph, with size t. This suggests a protocol along these lines:

1. Flood the network with the sender's message m. Alongside the messages, transmit graphs that somehow representing where the message has come from.
2. If a receiver R notices that a particular message m is associated with a minimum (S, R)-cut greater than t, then output m.

There are three challenges with this approach:

1. Min-cut is only defined for a static graph, and we want a protocol that works for dynamic communication graphs.
2. Corrupt parties must be restricted in how they can influence the graph showing where a message came form. Otherwise, as we explain below, they might introduce a fake path from S to R that only goes through honest parties.
3. For efficiency, the graphs must have polynomial size.

Challenge 1 we overcome by introducing multiple vertices for each party—one for each round—with each vertex labeled by the party it represents, and by introducing the notion of a labeled cut. In a labeled cut, all vertices with the same label P can be cut at once, for a cost of 1. For Challenge 2, we additionally introduce multiple vertices for the same party even within the same round, representing different claims about how that party heard about the message. To see why this is needed, imagine that during round r an honest P_h hears l from a corrupt party P_j (who claims to have heard it from S), then in round $r + 1$,

[7] Even polynomial round complexity has not been proven.
[8] In this paper we only consider vertex cuts, not edge cuts.

P_h sends l to another honest party $\mathsf{P}_{h'}$ (who reports it to R in round $r+2$), while simultaneously P_j claims to R that in round r it heard l from P_h (who heard it from S). Even though l has only been claimed to have traveled down the two paths $\mathsf{S} \to \mathsf{P}_j \to \mathsf{P}_h \to \mathsf{P}_{h'} \to \mathsf{R}$ and $\mathsf{S} \to \mathsf{P}_h \to \mathsf{P}_j \to \mathsf{R}$, which both go through the corrupt party, if you treat all instances P_h in round r as a single node then you would add the path $\mathsf{S} \to \mathsf{P}_h \to \mathsf{P}_{h'} \to \mathsf{R}$ to the graph, which does not go through a corrupted party. Therefore, we treat the P_h who heard l from P_j as a separate node from the P_h who was claimed to have heard l from S. This means that if R hears l from P_h and P_i, it won't think that there was a path directly from S through P_h to R. Finally, for Challenge 3 we perform a kind of deduplication on the graphs to keep them polynomially sized, while still keeping separate the nodes representing different claims of message provenance.

4.1 Labeled Min-Cut

An (s,d)-(vertex)-cut of a directed graph G is a partition $\{S,C,D\}$ of the vertices of G where $s \in S$ and $d \in D$, such that there are no edges from S to D. The minimum (s,d)-cut problem is to find an (s,d)-cut that minimizes $|C|$. Now let G be a labeled directed graph, where we have a function l mapping from vertices of G to labels in some set L. One may think of L as the set of parties in our protocol. Recall that in the previous section we argued that we need graphs with several vertices labelled as one party. Similarly, we can now define an integral labeled (s,d)-cut as follows.

Definition 5. *Let G be a directed graph with labels L and labeling function l. An integral* labeled (s,d)*-(vertex)-cut of G is a partition $\{S,C',D\}$ of the vertices of G such that there are no edges from S to D and C' is the preimage $l^{-1}(C)$ for some set $C \subseteq L$.*

That is, we only want to consider cuts that correspond to an actual set of parties (i.e., labels). The integral labeled minimum (s,d)-cut problem is then to minimize $|C|$ over all such cuts. This is the vertex-labeled cut analog of the edge-labeled cuts introduced by [DHKM16].

In the protocol sketched above, the adversary will corrupt a subset C of the parties (i.e., labels), and any lie will have to traverse this subset. Letting S be the connected component of s after removing $C' = l^{-1}(C)$ and D be the remainder of the graph then gives an integral labeled (s,d)-cut. Indeed, our final protocol would still be secure if it used integral labeled cuts. However, unlike for unlabeled cuts, we do not know how to efficiently compute minimal integral labeled (s,d)-cuts. In fact the problem is NP-hard: there is an easy reduction to weighted monotone satisfiability, which is NP-hard (see, e.g., [DF98]).

To see why, view these problems as integer linear programs. In Fig. 3a we have written a linear program for unlabeled min-cut, which exactly matches the definition above if the variables $\{S_v, C_v, D_v\}$ are restricted to being integers. For each vertex v, exactly one of (S_v, C_v, D_v) will be 1 and the others will be zero, so these variables will define a partition of the vertices of G. The constraint that

$$\min \sum_{v \in G} C_v$$

subject to

$$\begin{aligned} S_v + D_w &\leq 1 && \forall (v,w) \in G \\ \\ S_v + D_v + C_v &= 1 && \forall v \in G \\ S_v, D_v &\in [0,1] && \forall v \in G \\ C_v &\in [0,1] && \forall v \in G \\ S_s = D_d &= 1 \end{aligned}$$

(a) Linear program for (s,d)-min-cut.

$$\min \sum_{\ell \in L} C_\ell$$

subject to

$$\begin{aligned} S_v + D_w &\leq 1 && \forall (v,w) \in G \\ S_v &\leq S_w && \forall (v,w) \in G, l(v) = l(w) \\ S_v + D_v + C_{l(v)} &= 1 && \forall v \in G \\ S_v, D_v &\in [0,1] && \forall v \in G \\ C_\ell &\in [0,1] && \forall \ell \in L \\ S_s = D_d &= 1 \end{aligned}$$

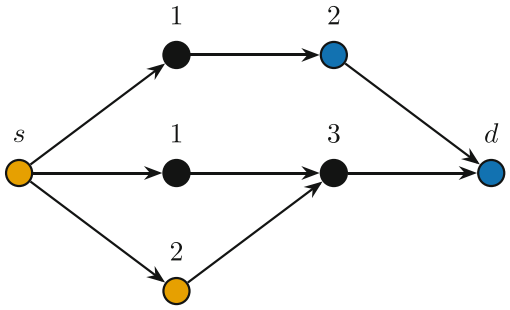
(b) Linear program for labeled (s,d)-min-cut.

Fig. 3. Linear programs for min-cut.

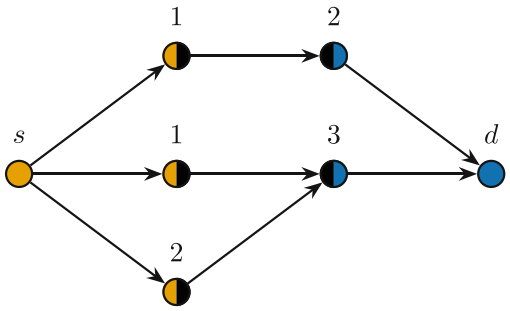
no edges connect S to D is enforced by $S_v + D_w \leq 1$ for all edges (v,w), i.e., we cannot have both $v \in S$ and $w \in D$. We have $s \in S$ and $d \in D$ because of the constraints $S_s = 1$ and $D_d = 1$.

While in general integer linear programs can be hard to solve, this particular one is easy. That is, the constraint matrix is totally unimodular (after eliminating the D_v variables using the equality constraint $D_v = 1 - S_v - C_v$), and constants in the constraints are integers, so every vertex of the polytope defined by the constraints has integer coordinates. Therefore, if you solve the linear programming relaxation of the problem (i.e., solve the same linear programming problem, but without requiring that the variables take integer values) then the minimal $\sum_{v \in G} C_v$ will be exactly the same as in the integer problem. This shows that min-cut can be solved in polynomial time [Kha79].

We present a similar linear program for labeled min-cut in Fig. 3b. The only significant change is that cut variables C_v are now defined on labels instead of



(a) An integral labeled min-cut.



(b) A relaxed labeled min-cut.

Fig. 4. Example graph where the integral labeled (s, d) min-cut has size 2, but the relaxed labeled min-cut has size $\frac{3}{2}$. The cut is black, the source component orange, and the sink component blue. Any subset of two of three labels $\{1, 2, 3\}$ gives an integral labeled min-cut, while the relaxed labeled min-cut is half of all three labels.(Color figure online)

vertices, to match the set C' of cut vertices being defined as $C' = l^{-1}(C)$ in this problem.[9] Unfortunately, this new system is not totally unimodular, because the same cut variable C_ℓ gets used with many different vertices. In fact, the linear programming relaxation can now have a min-cut with a fractional min $\sum_{\ell \in L} C_\ell$ (see Fig. 4). Because we desire a computationally efficient protocol, we will use only the relaxed labeled (s, d)-cuts, rather than integral labeled (s, d)-cuts. Our analysis will show that the protocol will still be secure, even though these relaxed labeled cuts are only an approximation to the integral labeled cuts that describe the possible attacks.

[9] There is another change, which is the added constraint $S_v \leq S_w$ for all edges $(v, w) \in G$ where v and w have the same label ℓ. This does not affect the integer linear program, as either $C_\ell = 0$, in which case it's implied by $S_v + D_w \leq 1$ and $D_w = 1 - S_w - C_\ell$, or $C_\ell = 1$, in which case $S_v = S_w = 1$. The constraint should be viewed as tightening the approximation of the linear program relaxation, as the constraint becomes non-trivial if C_ℓ is not an integer.

4.2 Multicast Protocol

We present our actively secure multicast Protocol 3. In each round, every party sends to its neighbors an associative array (i.e. a key-value store) containing all the messages they've heard so far (i.e. the keys), together with the labeled directed acyclic graphs representing the paths along which the message is purported to come from the sender S (i.e. the values). Each party then takes a kind of union of all the graphs it has heard (together with the graph from the previous round), where nodes representing the same message provenance are

Protocol 3: $\texttt{MultiCast}(\mathsf{S}, m)$

The sender S is public input. S has message $m \in \{0,1\}^*$. Denote by $\bar{\mathsf{G}} = \bigcup_{\mathsf{G} \in \mathcal{G}} \mathsf{G}$.

Initialization. Every party P_i has an associative array D_i mapping from possible messages (i.e, from $\{0,1\}^*$) to DAGs with nodes labeled by party identifiers.
- For all i, set D_i to be empty, so that $\mathsf{D}_i[m']$ is the empty graph for all $m' \in \{0,1\}^*$.
- For $\mathsf{P}_i = \mathsf{S}$, set $\mathsf{D}_i[m] := (\{s\}, \{s \mapsto \mathsf{S}\})$, i.e., the graph containing a single node labeled as S.

Network flooding. For ρ rounds, every party P_i does the following:
- For each neighbor P_j in $\bar{\mathsf{G}}$, attempt to send D_i to P_j.
- For each message $\mathsf{D}_{j,i}$ that received from P_j, and for each m' such that $\mathsf{D}_{j,i}[m']$ is non-empty:
  - Check that $\mathsf{D}_{j,i}[m']$ has exactly one sink, which must be labeled P_j.
  - If so, update $\mathsf{D}_i[m'] := \mathsf{D}_i[m'] + \mathsf{D}_{j,i}[m']$. (I.e., take the disjoint union.)
- For any m' where $\mathsf{D}_i[m']$ is not empty,
  - Connect every sink in $\mathsf{D}_i[m']$ to a new node d, labeled as P_i. Now d is a unique sink.
  - Compress the DAG with $\mathsf{D}_i[m'] := \texttt{Deduplicate}(\mathsf{D}_i[m'])$.

Output condition. Each party P_i locally decides when it knows the message sent by S. If there exists a $m' \in \{0,1\}^*$ such that $\mathsf{D}_i[m']$ is non-empty, output m' if
- $\mathsf{D}_i[m']$ has a source s labeled as S;[a] and
- The relaxed labeled (s, d) min-cut of $\mathsf{D}_i[m']$ is greater than t, where d is the unique sink (labeled as P_i) in $\mathsf{D}_i[m']$.

[a] s is unique if it exists, because of $\texttt{Deduplicate}$.

Function $\texttt{Deduplicate}(\mathsf{D})$

Repeat the following merger operation until a fixed point is reached.

- Find two nodes x, y of D such that x and y have the same label and the same direct predecessors.
- Merge x and y into a node z that has the same label and direct predecessors, and the union of the x's and y's direct successors.

Output D.

Fig. 5. Our actively secure multicast protocol.

identified. This is done by taking the disjoint union of the graphs, then running `Deduplicate` to merge nodes when they are labeled with the same party and have the same predecessor nodes. Finally, each party computes the relaxed labeled min-cut of each message's graph, which lower bounds the number of lies needed to cause this graph to appear for a message that was not sent by S.[10] The party determines that the message is correct and outputs it if this min-cut is greater than t.

Theorem 3. *Protocol 3 is a multicast protocol that achieves perfect security against t parties for networks $\mathcal{G}$ with connectivity $k > 2t$. It completes in $\rho = 1 + \lfloor \frac{n-t-2}{k-2t} \rfloor$ rounds, uses at most $\mathcal{O}\big(n\rho d_{avg}|m| + (M+1)n^2\rho^2 d_{avg}^2 \log_2(n\rho)\big)$ bits of total communication, and runs in polynomial time. Here, $|m|$ is an upper bound on the size of the message, d_{avg} is an upper bound on the average degree of the communication graph of every round, and the M is the total bits of all messages sent by corrupt parties.*

Proof. We must show that the protocol does not output a lie ("security"), that it eventually outputs the correct message ("correctness"), and that the communication cost is bounded ("efficiency").

Security. Security comes down to the following property: for every honest party P_i and for any $m' \neq m$, after every round all paths from source s to sink d in $\mathsf{D}_i[m']$ will go through a corrupted party. We call this the "corrupt paths property" for d. If this property holds, then cutting the t labels corresponding to corrupted parties in $\mathsf{D}_i[m']$ will disconnect s from d, so there is a labeled min-cut of size t, and so P_i won't output m'.

We prove this property by induction. In the base case, all $\mathsf{D}_i[m']$ is the empty graph, so the property holds vacuously. In each round, $\mathsf{D}_i[m']$ grows only by receiving $\mathsf{D}_{j,i}[m']$ from some other party P_j[11]. P_j can either be corrupt or honest. If it is corrupt, before adding $\mathsf{D}_{j,i}[m']$ to the graph, P_j checks that it's unique sink is labeled P_j. Therefore, every path in $\mathsf{D}_{j,i}[m']$ to this sink must go through a corrupt party, P_j. If P_j is honest, then by the induction hypothesis every path from s to the sink must go through a corrupt party. Next, a new sink d is added (labeled P_i), and all existing sinks are attached to it. Since the corrupt paths property was satisfied by all previous sinks, it will be satisfied by d.

Finally, P_i runs $\mathsf{D}_i[m'] := \texttt{Deduplicate}(\mathsf{D}_i[m'])$ to remove redundant nodes. We must show that this operation preserves the corrupt paths property. For any path from s' to d' in the deduplicated graph, there exists a corresponding path from s to d in the original graph, sharing exactly the same labels. To see this, follow the path in reverse, from d' to s'. Note d must not have been merged, because it is the unique sink in the original graph and the graph is acyclic, so

[10] Our protocol would also work with the integral labeled min-cut instead, which would give the exact minimum number of lies, except that this is hard to compute.

[11] Or if no new graphs are received for m', it grows only by adding a new sink node d; this is discussed later.

initially the correspondence is unambiguous. At each step, if x' has predecessor y' in the path, select a predecessor y of x such that y was merged into y' during `Deduplicate`. Such a node must exist because x is a node that was merged into x', and all nodes merged together shared the same set of direct predecessors. The path will have exactly the same labels because nodes are only merged if they have the same label. Since the corrupt paths property holds for the original graph, it therefore holds for the deduplicated graph. We have now proven that the hypothesis holds after the round is finished.

Correctness. Let the sequence of network graphs be $\mathsf{G}_1, \ldots, \mathsf{G}_\rho$. Remove the corrupt nodes, then combine them into a single acyclic graph $\mathbb{G}$ representing all honest communication paths that could occur. In more detail, construct $\mathbb{G}$ by creating vertices v_{ir} for all honest parties P_i and all rounds $r \in \{0, \ldots, \rho\}$. For each edge (P_i, j) in G_r, create an edge $(v_{i(r-1)}, v_{jr})$ in $\mathbb{G}$ to represent that P_i will send a message to P_j in round r. Additionally, add an edge $(v_{i(r-1)}, v_{ir})$ for all parties P_i and rounds r, since P_i preserves its state from rounds. Then we have:

Lemma 3. *The relaxed labeled* $(v_{\mathsf{S}0}, v_{\mathsf{R}\rho})$*-min-cut of* $\mathbb{G}$ *is greater than* t*, for any honest party* R*.*

Using this lemma, we will show that every honest party P_i in our protocol will output the correct message m by round ρ. That is, we will show that $\mathsf{D}_i[m]$ will contain a copy of the relevant subgraph of $\mathbb{G}$: all nodes that are on some path from $v_{\mathsf{S}0}$ to $v_{i\rho}$[12].

At the start, the only non-empty $\mathsf{D}_i[m]$ is when $\mathsf{P}_i = \mathsf{S}$, where it contains a solitary node. Call this node $v_{\mathsf{S}0}$. In every round, honest parties will send each other their graphs, take the disjoint union, deduplicate, and add new sinks d to their own graphs. Let v_{ir} be the sink added by P_i at the end of round r. It will have an edge from the previous sink $v_{i(r-1)}$, as well as edges from $v_{j(r-1)}$ for all honest P_j that P_i heard from in round r. Note that any such node v_{ir} will always get deduplicated, as there the honest parties do not modify the predecessors of nodes in their graphs before sending them on, so there will be at most one copy of any v_{ir} in any of these graphs. Since all paths in $\mathbb{G}$ are exactly the paths of honest parties are communicating along, P_i will have heard about v_{jr} by round ρ if and only if v_{jr} is on some path from $v_{\mathsf{S}0}$ to $v_{i\rho}$.

Nodes not on any path $v_{\mathsf{S}0}$ to $v_{i\rho}$ are irrelevant for determining the minimum cut, so by Lemma 3 each honest party P_i will output m by round ρ. It remains only to prove this lemma.

Proof (Proof of Lemma 3). Assume that there exists some relaxed labeled $(v_{\mathsf{S}0}, v_{\mathsf{R}\rho})$-cut $(\{C_{\mathsf{P}_i}\}_i, \{S_v, D_v\}_{v \in \mathbb{G}})$ where $\sum_i C_{\mathsf{P}_i} \leq t$. To avoid using double subscripts, we will write S_{ir} for $S_{v_{ir}}$, and similarly for C and D. Let $S_r = \sum_i S_{ir}$ represent the "progress" made in the flooding on round r. It is essentially the

[12] In general, it can contain much more, because the corrupted parties can make up whatever graph they choose. But it will always contain this subgraph of $\mathbb{G}$.

total number of nodes that are on the sender's side of the cut, except that the variables S_v need not be integers. For all r we have $S_{\mathsf{S}r} = D_{\mathsf{R}r} = 1$, as $S_{\mathsf{S}0} = D_{\mathsf{R}\rho} = 1$ by assumption that it is a $(v_{\mathsf{S}0}, v_{\mathsf{R}\rho})$-cut, and there are edges from $v_{\mathsf{R}(r-1)}$ to $v_{\mathsf{R}r}$ for all r. We also have $S_0 \geq S_{\mathsf{S}0} = 1$.

Next, for all rounds r there exists (S, R)-cut of the honest subgraph of G_r with size $t + S_r - S_{r-1}$: let $S'_i = S_{i(r-1)}$, $C'_i = C_i + S_{ir} - S_{i(r-1)}$, and $D'_i = D_{ir}$. We can now show that (S', C', D') is a cut, assuming that (S, C, D) is a labeled cut:

$$\begin{aligned} S'_i + D'_j &= S_{i(r-1)} + D_{jr} \leq S_{ir} + D_{jr} \leq 1 \\ S'_i + D'_i + C'_i &= S_{i(r-1)} + D_{jr} + C_i + S_{ir} - S_{i(r-1)} = S_{ir} + D_{jr} + C_i = 1 \\ S'_i &= S_{i(r-1)} \in [0, 1] \\ D'_i &= D_{jr} \in [0, 1] \\ C'_i &= C_i + S_{ir} - S_{i(r-1)} \geq C_i \geq 0 \\ C'_i &= 1 - S'_i - D'_i \leq 1. \end{aligned}$$

The size of C' is $\sum_i C'_i = \sum_i (C_i + S_{ir} - S_{i(r-1)}) \leq t + S_r - S_{r-1}$. While this is a relaxed cut, the relaxed min-cut coincides with the integral min-cut, so there exists some cut of honest subgraph of G_r with size at most $t + S_r - S_{r-1}$. By assumption, G_r is k-connected, so after removing the corrupted nodes it is $(k-t)$-connected. Therefore, G_r's honest subgraph has no cuts smaller than $k-t$. We now have $t + S_r - S_{r-1} \geq k - t$, or $S_r - S_{r-1} \geq k - 2t$.

We have lower bounded S_0, and lower bounded its increase in each round, so at the end we get

$$S_\rho \geq 1 + \rho(k - 2t) \geq 1 + \left(1 + \left\lfloor \frac{n - t - 2}{k - 2t} \right\rfloor\right)(k - 2t) > 1 + n - t - 2 = n - t - 1.$$

However, there are only $n - t - 1$ honest parties other than R, and $S_{\mathsf{R}\rho} = 0$, so $S_\rho \leq n - t - 1$. This is a contradiction.

Efficiency. We start by upper bounding the size of the graphs being sent when all parties are honest. Let $\mathsf{D}[m] = \mathtt{Deduplicate}(\sum_i \mathsf{D}_i[m])$ be the deduplication of the disjoint unions of the graphs of all honest parties. Every $\mathsf{D}_i[m]$ is then a subgraph of $\mathsf{D}[m]$, so the cost of sending $\mathsf{D}_i[m]$ is upper bounded by the cost of sending of $\mathsf{D}[m]$. This graph only grows as the protocol continues, so need only bound the size of the final value of $\mathsf{D}_i[m]$. In every round $\mathsf{D}[m]$ will increase by n nodes and $d_{\text{avg}}n$ edges, because every party will add a new sink connected to the sinks from the previous round of all parties it has heard from. Note that all of these honest nodes will always get deduplicated after the disjoint union operations, because all copies of these honest nodes will have the same predecessors.

We use a sparse representation of the graph, and compute its communication cost as $2E \log_2 V + V \log_2 n$, where E and V are the number of vertices. That is, each edge specifies its two endpoints, and each vertex specifies its label. Note

that $V \geq n$, so this communication cost is upper bounded by $(2E + V)\log_2 V$. If everybody is honest, this upper bounds the message size by

$$|m| + (2nd_{\text{avg}}\rho + n\rho + 1)\log_2(n\rho + 1) = |m| + (n\rho(2d_{\text{avg}} + 1) + 1)\log_2(n\rho + 1),$$

because the graph starts with one vertex (the sender) and zero edges. The total communication cost of graphs would then be $n\rho d_{\text{avg}}(n\rho(2d_{\text{avg}}+1)+1)\log_2(n\rho+1)$, as there are ρ rounds and at most nd_{avg} messages sent in each round.

Next, we consider how the corrupt parties could increase the communication cost. They can do some combination of sending lies (i.e., sending $m' \neq m$), increasing the number of graphs for honest parties to send, and sending graphs, increasing the size of each graph the honest parties send. The total communication increases linearly with the number of lies, as essentially the same protocol is run for lies as for m. The communication also increases linearly with the size of each graph sent by a corrupt party, as the corrupt graph will become part of $\mathsf{D}[m]$. However, sending lies increases the communication faster than sending graphs, as sending lies causes honest parties to add another $n\rho$ nodes to the graph themselves, while sending graphs only increases the size of an existing graph by the amount you sent. Therefore, the best attack is to spend the corrupt party's communication entirely on lies, multiplying the total communication for graphs by $M + 1$. Adding on the cost of sending every message on every round, this gives a total communication cost of

$$n\rho d_{\text{avg}}(M+1)(n\rho(2d_{\text{avg}}+1)+1)\log_2(n\rho+1) + n\rho d_{\text{avg}}(|m| + \text{M}).$$

Finally, note that based on these bounds all graphs remain polynomial in size. The relaxed labeled min-cut can be computed in polynomial time using linear programming, so the entire protocol then runs in polynomial time. □

5 Private Communication with Active Corruption

5.1 Feasibility of Perfect Security When $k > 2t$

In this section, we disprove the conjecture by Damgård et al. [DRTY23] by showing feasibility of perfectly-secure private communication when $3t \geq k > 2t$. First, we describe the building blocks used by Protocol 4.

Reliable Channel as a Building Block. We note that actively-secure reliable communication protocols that are secure against t corruptions in a dynamic network exist, as long as $k > 2t$. Thus, for simplicity, we use reliable channels as a primitive in Protocol 4 . These could be realized by using our reliable communication Protocol 3 or alternately by using the (more expensive) protocol of Damgård et al. [DRTY23]. The communication complexity of the former is upper bounded by $\mathcal{O}\big(n^3|m| + Mn^6\log_2(n)\big)$ where $|m|$ denotes the number of message bits and M is an upper bound on the number of bits communicated by corrupt parties.

Perfect Secure Message Transmission (PSMT). We use a perfect secure message transmission protocol, which informally speaking, allows a sender to send a private message securely to a receiver in a network where the sender and receiver are connected by n secure channels, among which up to t channels could be controlled by the adversary. More formally, PSMT can be defined as below.

Definition 6. *Assume there are k secure channels between a sender S and a receiver R. A protocol between S (with input message m) and R is a perfect secure message transmission protocol if it satisfies* privacy *and* correctness *as per Definition 2 against any adversary $\mathcal{A}$ corrupting at most t out of the k channels.*

Dolev et al. [DDWY93] showed the following:

Lemma 4 ([DDWY93]). *PSMT is achievable if and only if $k \geq 2t+1$.*

We observe that any such PSMT protocol should be secure even if there are potentially fewer than $2t+1$ channels (say, there are $\delta < 2t+1$ channels), as long as there are at least $t+1$ uncorrupted channels. This is because one can always augment this network with $2t+1-\delta$ dummy channels, where the dummy channels can be considered as being controlled by the adversary (as the adversary corruption budget of t allows for this). This observation allows us to state the below lemma.

Lemma 5. *Assume a party S and another party R are connected by $k \geq t+1$ secure channels, among which at least $t+1$ channels are uncorrupted. Then, there exists a protocol $\Pi_{\mathtt{SMT}}(\mathsf{S}, \mathsf{R}, m)$ that allows S to securely communicate a private message m to R.*

Looking ahead, in our Protocol 4 , we use the protocol $\Pi_{\mathtt{SMT}}$ in a non-black box fashion by replacing the *secure* channels used by such protocols with reliable channels and suitably masking the private message using established shared keys. Shared keys are established as in our protocols in the passive setting. Lastly, we point that $\Pi_{\mathtt{SMT}}$ can be instantiated using any existing efficient PSMT construction, such as the two round protocol of [SZ16] which has a communication complexity of $\mathcal{O}\big(n^2 \log n\big)$ bits to securely communicate a 1-bit secret.

Theorem 4. *Protocol 4 is a private communication protocol that achieves perfect security against $t < n$ active corruptions for network $\mathcal{G}$ with connectivity $k > 2t$.*

The protocol runs for $1 + 2\rho_{\mathtt{Rel}} + \rho_{\mathtt{SMT}}\rho_{\mathtt{Rel}}$ rounds, where $\rho_{\mathtt{SMT}}$ is the round complexity of the SMT protocol and $\rho_{\mathtt{Rel}}$ is the round complexity of the reliable communication protocol.

Its communication complexity is $\mathcal{O}\big(c_{\mathtt{Rel}}(n^2) + c_{\mathtt{SMT}}(|m|)(nc_{\mathtt{Rel}}(1+n\log(n)))\big)$ bits where $c_{\mathtt{Rel}}(x)$ denotes the communication complexity of reliably communicating x bits, $c_{\mathtt{SMT}}(x)$ denotes the communication complexity of SMT for x bits, and $|m|$ denotes the number of message bits.

Lemma 6. *If none of* $(\mathsf{P}_{i_1}, \ldots, \mathsf{P}_{i_\ell})$ *are corrupt, then the steps from Protocol 4 for communicating a (private) message* m' *from* S *to* R *over channel* Ch_p *achieve private and reliable communication of* m'*.*

The proof of Lemma 6 is identical to that of Theorem 2, with the graph limited to the single path in question.

Proof (Proof of Theorem 4). The security of Theorem 4 then follows from the security of the SMT protocol used: clearly, all connections between honest parties will be present in $\mathbb{G}$ and since the subgraph of honest parties is at least $t+1$-connected, $\mathbb{G}'$ must contain at least $t+1$ all-honest paths. Moreover, we only care about security if S is honest, and in that case all honest parties will be in agreement regarding the network $\mathbb{G}'$. We can therefore rely on the security of the SMT protocol (that assumes a public network).

The round complexity of the protocol follows from the fact that establishing shared randomness takes one round; establishing the metagraph involves two phases of (concurrent) reliable communications; and finally secure transmission involves executing the SMT protocol, where each message in the SMT protocol involves (multiple parallel) invocations of the reliable communication protocol.

The communication complexity of the protocol follows from the fact that establishing the metagraph incurs $c_{\mathtt{Rel}}(n^2)$ complexity (since the reliable communication protocol directly achieves multicast); and secure message transfer involves n instances of reliable communication (each involving communication of $c_{\mathtt{Rel}}(1 + n\log(n))$) corresponding to each bit sent in the underlying SMT protocol. □

Remark 3. If S is not honest, there are no requirements on security, but we can note that the protocol will still terminate and be efficient. In this case, parties may no longer agree on which instances of the protocol for reliable communication should be run in the last phase. So, effectively, the set of honest parties will be partitioned in subsets each trying to run their own set of instances. However, this will not affect the round complexity as each instance runs for a fixed number of rounds. As for the communication complexity, we can assume that honest parties will ignore messages from instances they do not think should be run, so we are effectively running at most a factor n more protocols in parallel. However, note that the complexity of the reliable communication protocol already incorporates a polynomial factor that depends on the behavior of the adversary (as indeed it must). This means that the adversary could make us work at least as hard by instead allowing parties to agree on $\mathbb{G}'$ but increase the number of incorrect messages it sends when the reliable communication protocols are run. Therefore, our expression for the communication complexity captures what can happen, even for a dishonest S.

Conjecture of Damgård et al. [DRTY23]. Lastly, recall that Damgård et al. conjectured that perfectly secure private communication in the active setting was impossible to achieve when $3t \geq k > 2t$. They gave evidence for the conjecture

by arguing that any protocol in the class they considered would fail in this case. The assumption on the protocol was that the receiver would receive values from different sets of $t+1$ disjoint paths and would then try to identify the honest set of $t+1$ disjoint paths. However, this was shown to be impossible with 0 error probability when $3t \geq k > 2t$, since in the dynamic setting, an adversary can always fabricate paths. While this argument is correct, it does not cover all protocols, as also pointed out in [DRTY23]. Indeed, our protocol does not fall in this class, as we crucially rely on fixing the meta graph using additional public communication, and such additional interaction was not considered in [DRTY23]. The metagraph essentially gives us a means to work with the static setting instead, where perfect private communication is known to be possible for $k > 2t$.

6 Communication with Less Connectivity

In the model we have used in the paper so far, the network graph is k-connected in every round. It is natural to ask if this is required for our protocols to work. As we shall see, the answer is no, in some cases we can work with the weakest possible network assumption.

Let us first consider the case of reliable communication. We will use the concept of a dynamic path, which is taken from [MTD15]. Consider some sequence of network graphs as chosen by the adversary. Now, informally, if there is a path in round 1 from party P_i to party P_j and in round 2 we have one from party P_j to party P_k, we say there is a dynamic path from P_i to P_k; and this generalizes in the natural way to any number of rounds. We say the network is dynamically connected if there is a dynamic path from any party to any other party. If this is the case, then for a passive adversary, the simple flooding protocol will allow any party to multicast a message to anyone else. On the other hand, if there is no dynamic path from P_i to P_j, nothing P_i says can reach P_j.

For an active adversary, we have shown an efficient multicast protocol working in at most n rounds if we have k-connectivity in every round. A first observation is that since the protocol is based on flooding, it will clearly also work if we run for $m > n$ rounds and we have k-connectivity in at least n of the m rounds.

However, we can even work with the weakest possible network model as defined in [MTD15]. For a given set Ω of dynamic paths between nodes P_i and P_j, they define the dynamic min-cut of Ω as the minimal number of nodes one needs to remove, in order to cut all paths in Ω. Finally, the dynamic min-cut between P_i and P_j is defined as the min-cut of the set of *all possible* dynamic paths from P_i to P_j. It is shown that reliable communication from P_i to P_j with t active corruptions is possible if and only if the dynamic min-cut between P_i and P_j is larger than $2t$.

We can rephrase this in terms of the notions we define in this paper, namely for a set of dynamic paths Ω as above, we define a labelled graph G_Ω as we did in Sect. 4, where for each party P_a and each round there is a node labelled as P_a. If a path in Ω connects P_a to P_b in some round r, we put an edge between the

node for round $r-1$ labelled as P_a and the one for round r labeled as P_b (also, each party is connected to itself in the next round). It is then straightforward to see that the labeled (integer) min-cut of G_Ω as defined in Sect. 4 equals the dynamic min-cut of Ω.

Protocol 4: $\Pi^{\mathsf{prv}}_{\mathsf{perf,mal}}(\mathsf{S},\mathsf{R},m)$

The identities of the sender S and receiver R are public input. The sender S has message $m \in \mathbb{F}$ as private input.
Let $\bar{\mathsf{G}} = \bigcup_{\mathsf{G}\in\mathcal{G}} \mathsf{G}$. We use the following building blocks:

- $\mathtt{Rel}(\mathsf{S},\mathsf{R},m)$, which allows reliable communication of a message m from S to R.
- $\Pi_{\mathtt{SMT}}(\mathsf{S},\mathsf{R},m)$: A perfect secure message transmission protocol that allows secure communication of a private message m over a network of $k \geq t+1$ secure channels, among which at least $t+1$ channels are uncorrupted.

Establishing shared randomness. Party P_i does the following:
- For each neighbor P_j in $\bar{\mathsf{G}}$, sample uniform random $r_{i,j}$ (of size $|m|$).
- In round 1, attempt to send $r_{i,j}$ to P_j using the communication network.
- Let In_i denote the set of parties P_j from whom P_i actually received randomness $r_{j,i}$ in round 1.
- For each neighbor P_j in $\mathbb{G}$ set $o_{i,j} := r_{j,i} + r_{i,j}$ where missing values $r_{\cdot,\cdot}$ are set to 0.

Establishing meta graph of shared keys. This phase consists of the following steps:
- For each party P_i, parties jointly invoke $\mathtt{Rel}(\mathsf{P}_i,\mathsf{S},\mathsf{In}_i)$, where In_i is encoded as a n-bit vector.
- The sender S builds the meta graph $\mathbb{G}$ as follows: Let $\mathcal{P}$ be the set of nodes. There is an edge between P_u and P_v if $\mathsf{P}_u \in \mathsf{In}'_v$ or $\mathsf{P}_v \in \mathsf{In}'_u$, where In'_u denotes the output of the instance $\mathtt{Rel}(\mathsf{P}_u,\mathsf{S},\mathsf{In}_u)$.
- For each party P_i, parties jointly invoke $\mathtt{Rel}(\mathsf{S},\mathsf{P}_i,\mathbb{G})$, where $\mathbb{G}$ is encoded in n^2 bits.

Secure message transfer in $\mathbb{G}$. All the parties can now locally determine the same set of disjoint paths, say $\mathsf{GoodPaths}$ in $\mathbb{G}$ between S and R (using the Ford-Fulkerson algorithm to get a maximal set of disjoint paths): If the algorithm returns at most $2t+1$ disjoint paths, use these as $\mathsf{GoodPaths}$, else use the first $2t+1$ paths returned. Consider a network $\mathbb{G}'$ where each disjoint path p in $\mathsf{GoodPaths}$ corresponds to a channel Ch_p. In order to emulate each step of an instance of $\Pi_{\mathtt{SMT}}(\mathsf{S},\mathsf{R},m)$ over $\mathbb{G}'$, parties do the following:
- If the step involves computation, it is done exactly as in the protocol $\Pi_{\mathtt{SMT}}$.
- If the step involves communicating a (private) message m' from S to R over a channel Ch_p:
  - Let $(\mathsf{P}_{i_1},\ldots,\mathsf{P}_{i_\ell})$ be the path corresponding to Ch_p, where $\mathsf{P}_{i_1} = \mathsf{S}$ and $\mathsf{P}_{i_\ell} = \mathsf{R}$.
  - The sender $\mathsf{S} = \mathsf{P}_{i_1}$ computes $m_{i_1} = m' + o_{i_1,i_2}$[a] and all parties jointly invoke $\mathtt{Rel}(\mathsf{S},\mathsf{R},(p,m'_{i_1}))$ where p is encoded in $n\log(n)$ bits.
  - Each party $\mathsf{P}_{i_j} \in (\mathsf{P}_{i_2},\ldots,\mathsf{P}_{i_{\ell-1}})$ computes $m'_{i_j} = o_{i_j,i_{j+1}} - o_{i_{j-1},i_j}$ and all parties jointly invoke $\mathtt{Rel}(\mathsf{P}_{i_j},\mathsf{R},(p,m'_{i_j}))$ where p is encoded in $n\log(n)$ bits.
  - The receiver R computes $m' = \left(\sum_{i_j\in(i_1,\ldots,i_{\ell-1})} m'_{i_j}\right) - o_{i_{\ell-1},i_\ell}$ as the value received via the channel Ch_p in this step.
- If the step involves communicating a (private) message m' from R to S over a channel Ch_p: Similar steps as above, with the roles of S and R interchanged.

- R returns the output of $\Pi_{\mathtt{SMT}}(\mathsf{S},\mathsf{R},m)$ over $\mathbb{G}'$.

[a] We abuse notation here; assume that the shared key $o_{i,j}$ established earlier contains enough randomness that can be used to mask the private messages sent throughout all steps of $\Pi_{\mathtt{SMT}}$ and only the relevant part of the shared key corresponding to this particular private message m_p is used here.

Fig. 6. A perfectly-secure private communication protocol in a network with connectivity $k > 2t$.

Now, since our actively secure multicast protocol works with labelled min-cuts, we can make a variant that will work making the minimal assumption that the dynamic min-cut between any pair of players is $> 2t$. The idea is to change the output condition such that the graph accompanying the message m' ($D_i[m']$) must have *integer* labelled min-cut greater than t – rather than the relaxed labelled min-cut that allows fractional values. Computing the non-relaxed labeled min-cut is not computationally efficient, but we still use only polynomial communication. This protocol therefore compares favorably to the protocol from [MTD15] for the same setting. That protocol tracks every individual path a message travels and so is exponential both in communication and computation.

Finally, we consider private communication. Observe that in both the passive and active case, we use one round to decide on a sufficiently connected graph according to which keys are shared pairwise between parties. Clearly, we could also run this phase for several rounds, where in each round players attempt to share keys with other players. As long as the *union* of the network graphs in all the rounds is sufficiently connected, this will be enough for the protocol to work. Once the keys have been established, reliable communication is used, so we only need that the network supports non-private communication in that last phase. We note that for passive security this means that we only need $k > t$ for the key sharing phase, and $k > 0$ (or a dynamically connected network) for the rest of the protocol.

References

[DDWY93] Dolev, D., Dwork, C., Waarts, O., Yung, M.: Perfectly secure message transmission. J. ACM **40**(1), 17–47 (1993)

[DF98] Downey, R.G., Fellows, M.R.: Parameterized Complexity. In: Monographs in Computer Science. Springer, New York (1998). https://doi.org/10.1007/978-1-4612-0515-9

[DHKM16] Dutta, T., Heath, L.S., Kumar, V.A., Marathe, M.V.: Labeled cuts in graphs. Theor. Comput. Sci. **648**, 34–39 (2016)

[Dol82] Dolev, D.: The byzantine generals strike again. J. Algorithms **3**(1), 14–30 (1982)

[DRTY23] Damgård, I., Ravi, D., Tschudi, D., Yakoubov, S.: Secure communication in dynamic incomplete networks. In: 4th Conference on Information-Theoretic Cryptography, ITC 2023 (2023)

[Kha79] Khachiyan, L.G.: A polynomial algorithm in linear programming. Dokl. Akad. Nauk SSSR **244**, 1093–1096 (1979)

[MTD15] Maurer, A., Tixeuil, S., Defago, X.: Communicating reliably in multihop dynamic networks despite byzantine failures. In: 2015 IEEE 34th Symposium on Reliable Distributed Systems (SRDS), pp. 238–245 (2015)

[SZ16] Spini, G., Zémor, G.: Perfectly secure message transmission in two rounds. In: Hirt, M., Smith, A. (eds.) TCC 2016. LNCS, vol. 9985, pp. 286–304. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53641-4_12

Adaptive Security, Erasures, and Network Assumptions in Communication-Local MPC

Nishanth Chandran[1], Juan Garay[2], Ankit Kumar Misra[3(✉)], Rafail Ostrovsky[3], and Vassilis Zikas[4]

[1] Microsoft Research, Bengaluru, India
nichandr@microsoft.com
[2] Texas A&M University, Texas, USA
garay@tamu.edu
[3] University of California, Los Angeles, USA
{ankitkmisra,rafail}@cs.ucla.edu
[4] Purdue University, West Lafayette, USA
vzikas@cs.purdue.edu

Abstract. The problem of reliable/secure all-to-all communication over low-degree networks has been essential for *communication-local* (CL) n-party MPC (i.e., MPC protocols where every party directly communicates only with a few, typically polylogarithmic in n, parties) and more recently for communication over *ad hoc* networks, which are used in blockchain protocols. However, a limited number of adaptively secure solutions exist, and they all make relatively strong assumptions on the ability of parties to act in some specific manner before the adversary can corrupt them. Two such assumptions were made in the work of Chandran *et al.* [ITCS '15]—parties can (a) *multisend* messages to several receivers simultaneously; and (b) *securely erase* the message and the identities of the receivers, before the adversary gets a chance to corrupt the sender (even if a receiver is corrupted).

A natural question to ask is: Are these assumptions necessary for adaptively secure CL MPC? In this paper, we characterize the feasibility landscape for all-to-all reliable message transmission (RMT) under these two assumptions, and use this characterization to obtain (asymptotically) tight feasibility results for CL MPC.

- First, we prove a strong impossibility result for a broad class of RMT protocols, termed here *store-and-forward* protocols, which includes all known communication protocols for CL MPC from standard cryptographic assumptions. Concretely, we show that no such protocol with a certain expansion rate can tolerate a constant fraction of parties being corrupted.
- Next, under the assumption of only a PKI, we show that assuming secure erasures, we can obtain an RMT protocol between all pairs of

R. Ostrovsky—This research was supported in part by NSF grants CNS-2246355, CCF-2220450, US-Israel BSF grant 2022370, and by Sunday Group.

E. Boyle and M. Mahmoody (Eds.): TCC 2024, LNCS 15367, pp. 293–326, 2025.
https://doi.org/10.1007/978-3-031-78023-3_10

parties with polylogarithmic locality (even without assuming multi-send) for the honest majority setting. We complement this result by showing a negative result for the setting of dishonest majority.
– Finally, and somewhat surprisingly, under stronger assumptions (i.e., trapdoor permutations with a reverse domain sampler, and compact and malicious circuit-private FHE), we construct a polylogarithmic-locality all-to-one RMT protocol, which is adaptively secure and tolerates any constant fraction of corruptions, without assuming either secure erasures or multisend. This last result uses a novel combination of adaptively secure (e.g., non-committing) encryption and (static) FHE to bypass the impossibility of compact adaptively secure FHE by Katz *et al.* [PKC'13], which we believe may be of independent interest. Intriguingly, even such assumptions do not allow reducing all-to-all RMT to all-to-one RMT (a reduction which is trivial in the non-CL setting). Still, we can implement what we call *sublinear output-set* RMT (SOS-RMT for short). We show how SOS-RMT can be used for SOS-MPC under the known bounds for feasibility of MPC in the standard (i.e., non-CL) setting assuming, in addition to SOS-RMT, an anonymous PKI.

Keywords: Secure Multiparty Computation · Communication Locality

1 Introduction

1.1 Communication Locality and Adaptive Security

Secure multi-party computation (MPC) [4,19,31,47] allows a set of n parties to securely compute a function on their joint private data. Initial work on MPC focused on feasibility, and it was followed by a series of works on improving round and communication complexity. Envisioning the potential need to deploy MPC on massive networks, novel works on *scalable* MPC (e.g., [20,21,25,34]) have investigated settings and techniques that allowed for protocols with communication complexity that grows (asymptotically) slower than the size of the player set. Boyle, Goldwasser, and Tessaro [8] put forth a different metric that is very relevant for the design of massive-scale MPC, namely, *communication locality* (CL). The CL of a party in a protocol is the number of parties that this party sends/receives messages to/from, via a direct point-to-point channel, through the execution of the protocol; as such, the CL of a protocol is the maximum CL of any party. Motivated by the potential application of MPC to privately executing sublinear algorithms in a distributed manner, [8] proposed a solution which achieves MPC with a sublinear (i.e., polylogarithmic in n) CL tolerating a (sub-optimal) number of $t < (1/3 - \epsilon)n$ actively corrupted parties, for $\epsilon > 0$.

The original solution in [8] only considered static corruptions and relied on the existence of a public-key infrastructure (PKI), a common reference string (CRS), semantically secure public-key encryption and existentially unforgeable signatures. Chandran *et al.* [15] improved on the above result to tolerate an asymptotically *optimal* number of $t < (1/2 - \epsilon)n$ *adaptive* active corruptions,

for an arbitrary small constant ϵ. Their construction relied on the same assumptions except for the CRS, which was replaced by a *hidden (random) graph*: A suitable random graph on n vertices (with sublinear degree) that is sampled by a trusted entity and where each party is given its neighborhood in this graph. However, parties do not know the other parties' neighborhoods, and, most importantly, the *adversary* does not know the (honest) neighbors of honest parties. As shown in [15], this random graph can be realized via a standard symmetric-key infrastructure (SKI)—wherein every two parties share a (secret) symmetric-key encryption key. The emulation is simple: Every two nodes locally use their symmetric key as a seed to a PRF to derive sufficiently long pseudorandomness that can be utilized to decide (locally and independently) whether or not the parties should have an edge between them in any given round. In fact, we mention that, as noted in [14], the above hidden graph (or, equivalently, SKI) assumption can be replaced by standard number-theoretic cryptographic assumptions (such as DDH and its resulting PKI), allowing non-interactive key exchange (NIKE, for short—cf. [24]). The simple idea is that the PKI can be used to non-interactively establish the SKI required in [15], which can be used to derive the hidden graph.

The core challenge associated with such sublinear CL protocols is propagating information and connecting any two parties using a sparse (i.e., sublinear degree) communication graph. In such a context, one needs to route messages through the induced (incomplete) communication graph so that the adversary cannot block (even indirect) communication between any two honest nodes, thus disconnecting the graph. Indeed, since any party can only directly communicate with a sublinear number of neighbors, the only way for it to reach all parties in the network is by means of a *gossiping* protocol. In [8], gossiping was done via a routing protocol based on hierarchical routing and sorting networks that cleverly knit the paths to ensure each message travels over sufficiently many paths, making it impossible for the adversary to block it.

The above gossiping protocol works for a static adversary corrupting $t < (1/3 - \epsilon)n$ parties. However, when one considers stronger adversaries, with an asymptotically optimal (for MPC) corruption threshold—i.e., $t < (1/2 - \epsilon)n$—and, most importantly, adaptive adversaries, the problem becomes even more challenging, as message routing through the incomplete graph turns into a "cat-and-mouse" game—more formally, a *graph discovery* game—with the adversary using an initial set of corrupted parties to try to discover possible message routes and block them. In [15], properties of a hidden Erdős-Rényi graph were used along with a clever use of edges in a disposable manner (where every edge was used only once) in order to win the above graph discovery game, and devise a sublinear-locality communication protocol for the problem of *reliable message transmission* (RMT) between any two honest nodes, which allowed every honest party to reach every other honest party. The protocol from [15] tolerates an arbitrary constant fraction of the parties being adaptively (and actively) corrupted. RMT protocols can then be used to trivially construct secure message transmission (SMT) protocols—informally, these are protocols which emulate a secure, i.e., private and authenticated, channel between a sender and a receiver—in the

model assuming a PKI, which can then be used to build communication local MPC protocols[1].

1.2 Erasures and Network Assumptions in CL Protocols

The protocol in [15] relied on the aforementioned setup and hardness assumptions, namely, a PKI, an SKI, and the existence of enhanced one-way permutations—the latter being a common minimal assumption for MPC. In addition, an assumption was made in [15] which is not essential for MPC, but, as we show in this paper, turns out to be *necessary* for sublinear communication when natural gossiping protocols are used. In more detail, the assumption of *secure erasures* [12]—namely, that honest parties can erase whichever part of their state they wish, in a way that if they are corrupted later on, the adversary cannot recover the erased information—is only needed for a subset of adaptively secure MPC protocols without the sublinear CL restriction [28]. The construction from [15], however, assumes not just erasures, but actually two levels of strengthening of the assumption: First, it also assumes an *atomic multisend* capability [33], which in a nutshell ensures that if a player p attempts to send a message to a subset $\mathcal{Q}$ of the player set in some given round, then either all honest parties will receive the message or none of them will[2]. However, even assuming secure erasures in addition to such a rushing adversary proves not to be sufficient for the protocol in [15]. The reason is that when a message is sent to a polylogarithmic (in n) number of parties, then one of these parties may be corrupted, in which case the adversary can corrupt the sender and learn who the other receivers were, *before* the sender has had a chance to erase their identities, and corrupt them too, thereby completely neutralizing the sender. In fact, the inability of the adversary to mount such an attack is essential in [15]'s security proof. In order to exclude this attack, [15] also assumes that multisend and erase can jointly be done as an atomic operation—i.e., p can send his message so that it is received by all the parties in $\mathcal{Q}$ and erase their identitites before the adversary is able to corrupt him.

1.3 Our Results

The above state of affairs leaves open several questions regarding the minimal assumptions required for sublinear locality in all-to-all communication (and therefore also in MPC) in the adaptive security setting. In particular, it leaves open the question of the necessity and sufficiency of secure erasures, atomic multisend, and their atomic combination as mentioned above. In this paper we

[1] As a side note, an interesting side effect of the recent popularity of blockchain protocols is that, as they also rely on gossiping for communication, results on the feasibility of sublinear-communication protocols provide insights on basic feasibility questions in the blockchain context as well (see related work for further details).

[2] As shown in [33,36], this property is impossible to obtain from simple point-to-point communication in the standard adaptive and rushing adversary setting [10].

provide a characterization of this landscape, as depicted in Table 1. The impossibility results in Table 1 are for adaptively secure all-to-all reliable message transmission, i.e., the task of allowing every party p_i to send a (potentially different) message to every party p_j in a reliable, i.e., authenticated manner—where p_j becomes aware that the message was sent by p_i—so that the adversary cannot block or alter the message exchanges between any pair of honest parties. As noted earlier, this serves as a building block for SMT and CL MPC protocols. These results apply to a broad class of protocols which we call *store-and-forward* (SF) protocols, which, intuitively, allow intermediate parties to only store and forward previously received messages, and (for the non-erasure case) under an *expansion-rate* assumption, which mandates that messages originating from any neighbors of a sender will reach a large (polylogarithmic size) set relatively fast (i.e., before they reach their respective receiver) (see Definitions 1 and 2). This includes several natural message propagation and gossiping protocols, and in particular all those used in the CL literature (see Sect. 2 for a discussion).

The positive results in Table 1 are for all-to-one RMT, i.e., there is one receiver who everyone wishes to send messages to. As we shall show, assuming erasures, the feasibility results can be extended to all-to-all RMT. However, intriguingly, in the non-erasure setting, the protocol can only be extended to allow for a sublinear (polylogarithmic) set of receivers. Even so, the assumption of an additional setup, namely, an *anonymous* PKI, allows us to rescue the situation: We show that we can implement a new notion of MPC, which we term *sublinear output-set* MPC (SOS-MPC for short). Likewise, we use the term SOS-RMT to refer to the formerly obtained notion of RMT with a sublinear set of receivers. Intuitively, in an anonymous PKI setting, parties have access to a PKI but do not know which public-key corresponds to which party. Such a setup is common in YOSO-style MPC protocols which have become highly relevant in the blockchain literature [30]. Our newly defined SOS-MPC is similar to standard MPC (i.e., the inputs of all parties are accounted for in the computation) but only a (random) subset of the parties of sublinear size receives the output from the computation. We note that SOS-MPC is sufficient for the motivating applications of CL MPC, namely, secure computation of sublinear algorithms, where the output is by definition far smaller than the input (cf. [9]). This leaves open the question of feasibility or impossibility of (all-to-all RMT and) standard MPC in this setting.

Our positive results are of two flavors. Assuming secure erasures under standard assumptions (i.e., one-way functions), we provide SF protocols for RMT tolerating the (asymptotically) optimal number of corruptions, as implied by the impossibility result of Theorem 3. This leaves open the following important question:

> *Is it possible to construct an RMT protocol in the no-erasures setting that is adaptively secure against a constant fraction of corruptions?*

We answer this question in the affirmative, albeit using strong cryptographic assumptions. Concretely, assuming trapdoor permutations with a reversed domain sampler and the existence of a malicious, compact, and circuit-private

Table 1. A characterization of feasibility of reliable communication (RMT) under the different assumptions: Atomic multisend (A-MS), secure erasures (Erasures), and multisend and secure erasures as an atomic operation (A-MSE). All negative results are for all-to-all RMT. The positive results are for all-to-one RMT; but all except Theorem 4 are extended to all-to-all RMT, whereas Theorem 4 is extended to SOS-RMT. Note that SF stands for store-and-forward as defined in Definition 1, and expansion rate is as defined in Definition 2.

A-MS	Erasures	A-MSE	
✓	✓	✓	[15]: Assuming *PKI and one-way functions*, there exists an (SF) sublinear locality RMT protocol tolerating $t < (1-\epsilon)n$ corruptions, for any $\epsilon > 0$.
✓	✗	✗	**Theorem 1:** There exists no SF polylogarithmic locality RMT protocol with expansion rate $(\mathsf{polylog}(n), \frac{k \log n}{c \log \log n})$ tolerating a constant fraction of corruptions, for any $k < 1$, where the degree of the communication graph is $O(\log^c n)$.
✗	✓	✗	**Theorem 2:** Assuming *PKI and one-way functions*, there exists an (SF) polylogarithmic locality RMT protocol tolerating $t < (\frac{1}{2} - \epsilon)n$ corruptions, for any $0 < \epsilon < \frac{1}{2}$. **Theorem 3:** There exists no SF polylogarithmic locality RMT protocol tolerating $t > (\frac{1}{2} + \epsilon)n$ corruptions, for any $0 < \epsilon < \frac{1}{2}$.
✗	✗	✗	**Theorem 4:** Assuming *PKI, trapdoor permutations with a reversed domain sampler* and *compact and malicious circuit-private FHE*, there exists a polylogarithmic locality RMT protocol tolerating $t < (1-\epsilon)n$ corruptions, for any $\epsilon > 0$.

fully-homomorphic encryption (FHE) scheme [42], we can construct a protocol, which is not SF, and thus circumvents our impossibility result, allowing for RMT tolerating any constant fraction of corruptions. Its construction relies on a novel combination of adaptively secure (e.g., non-committing) encryption [12] and (statically) secure FHE to obtain a homomorphic encryption scheme that, although not fully homomorphic, allows to compute a class of circuits sufficient for our RMT, while providing adaptive security (including deniability), a property which is known to be impossible for general FHE [37].

Regarding our MPC feasibility results, using techniques from [15], we can "lift" all the above all-to-all RMT feasibility results on adaptive all-to-all communication to adaptively secure MPC, under the assumption of enhanced trapdoor permutations, or any other assumption that would allow for corruption-optimal adaptively secure MPC over a complete point-to-point network. Similarly, we can lift the SOS-RMT results to SOS-MPC.

1.4 Related Work

Although introduced as a notion explicitly for standard MPC by Boyle, Goldwasser, and Tessaro [9], the idea of low communication locality was already implicit in a number of works on *almost-everywhere* secure (communication, Byzantine agreement, and computation) protocols [17,18,23,27,39,40,44]. Such protocols can operate over incomplete networks (or under-utilize a complete

network to achieve low CL as we do here) but "give up" security for a number of parties; such so-called *doomed* parties might loose their input privacy, and/or contribute a false input to or receive a false output from the computation. The goal of such almost-everywhere secure protocols is then to achieve optimal tradeoffs between number of corruptions and number of doomed parties. We refer to [16] for a recent formal treatment of and detailed literature review of almost-everywhere secure protocols.

As already mentioned, Chandran *et al.* [15] improved on the resiliency of the protocol from [9] and brought adaptive security to the model, at the cost, however, of the strong atomic erase-and-multisend assumption, which restricts the ability of an adaptive adversary to attack the protocol, as we discuss in detail in Sect. 2. The results from [15] relied on a hidden-graph setup which, by construction, was an expander graph. The follow-up work by Boyle *et al.* [6] provided the first solutions to the problem for PRAM-based MPC, which in addition achieved some load balancing properties. Following that, Boyle *et al.* [7] investigated the question of whether an expander is in fact needed for sublinear locality MPC, answering it in the negative.

Also related to our goals are works that explicitly target sublinear per-party communication complexity. In this context, Dani *et al.* [22] presented a statically secure information-theoretic MPC protocol with a per-party communication complexity of $O(\sqrt{n})$ tolerating $t < n/3$ corruptions. King and Saia [38] showed how to construct a Byzantine agreement (BA) protocol that is secure against adaptive corruptions, where the communication complexity of every party is $\tilde{\mathcal{O}}(\sqrt{n})$, which leads to a BA protocol with $\tilde{\mathcal{O}}(n)$ communication locality tolerating $t < (\frac{1}{3} - \epsilon)n$ corruptions.

Also related to our work is the work of Matt *et al.* [41], who consider a weakening of the adversary's adaptivity, which they term *delayed adaptive corruption*. Here the adversary who wants to corrupt a party needs to first indicate its intention, say, in round r, but the actual corruption does not take effect until a few rounds later. Despite being useful for making statements about the load balance and delivery guarantees of blockchain-inspired message propagation protocols, in the context of sublinear locality we are considering, this assumption trivializes feasibility questions. Indeed, the latter protocols tend to use parties (network nodes) as "disposable" relays, i.e., once a party successfully relays its message to its neighbors, corrupting it does not buy the adversary anything. This fact, in combination with the delayed adaptive corruption assumption (which would imply that the adversary's ability to corrupt is **slower** than the message propagation), would prevent the adversary from adaptively blocking discovered paths.

2 Model

Notation. Here we present some basic notation used throughout the paper. We denote by $[n]$ the set $[n] = \{1, \ldots, n\}$. $\mathcal{P} = \{p_1, \ldots, p_n\}$ denotes the set of parties participating in the MPC protocol. We will often refer to parties as nodes in a network. We will assume that the adversary is able to corrupt a number $t < n$

of the parties; it will be convenient for our exposition to express corruption in terms of a fraction τ of the total number of parties; hence $t = \tau n$, for $0 < \tau < 1$.

In any directed graph over $\mathcal{P}$, we will use $\rho_{i,j}$ to denote the length of the shortest path from node $i \in \mathcal{P}$ to node $j \in \mathcal{P}$. Further, we will denote by $\Gamma_q(u)$ the set of all nodes (not including u) that are at forward distance $\leq q$ from node $u \in \mathcal{P}$, and we will define $\gamma_q(u) = |\Gamma_q(u)|$. Hence, $\Gamma_1(u)$ denotes the set of all outgoing neighbors of u. Analogously, we will denote by $\Gamma_q^{\text{in}}(u)$ the set of all nodes $v \in \mathcal{P}$ such that $\in \Gamma_q(v)$. Hence, $\Gamma_1^{\text{in}}(u)$ denotes the set of all incoming neighbors of u.

2.1 Adversarial Model

Next, we turn to defining the communication and adversary model for adaptively secure computation with communication locality. We note that most works in this area leave several of the model assumptions implicit or unspecified. Instead, since our goal is to provide a complete feasibility landscape given such assumptions, we need to take a more rigorous and detailed approach to the specification of the model.

Consistently with the classical MPC literature, we assume that parties are connected with each other via a complete network of secure (i.e., authenticated and private) point-to-point channels [5,32]. However, since each party can "talk" to only a sublinear (i.e., polylogarithmic in n) number of other parties, no party will be using all its point-to-point channels. The communication is synchronous, which means all parties advance in a round-based manner, where whenever the round switches everyone is informed, and messages sent in any round r are guaranteed to be delivered by the beginning of the following round $r+1$—unless the sender gets corrupted during r and no *multi-send* capability is assumed (see below). We will consider an adaptive and rushing adversary who might actively corrupt parties during the protocol execution.

The combination of sublinear communication locality and synchrony with such an adversary brings up a number of modeling challenges, described below.

Localized Notification. In classical synchronous point-to-point networks, the adversary is always informed when a party p_i sends a message to another party p_j via their direct point-to-point channel. Notifying the adversary about such a transmission implicitly captures the assumption that the adversary has a **global** view of the entire network, including runtime-observable events, such as messages being transmitted. This gives him the ability to induce a worst-case (arbitrary/adversarial) scheduling of messages that makes for stronger security.

However, when we shift to settings with vastly large sets of parties—these are the settings where sublinear-locality protocols become relevant—the assumption that the adversary has a complete view of all the events that occur in the network might be too strong. In fact, it is impossible to achieve adaptive security with sublinear locality in this worst-case setting. Indeed, in low-locality settings[3], a

[3] Here "low" specifically means asymptotically smaller than the adversary's corruption budget.

party might only communicate with a small number of its neighbors. Hence, if the adversary is able to detect that an honest party p attempts to send a message to another honest party, then he can simply corrupt the receiver and block this transmission path; by performing this attack on every transmission of p the adversary will be able to isolate the party from the rest of the network, making *all-to-all* communication (and therefore "full" MPC) impossible[4]. Hence, for such settings, it is natural and relevant to limit the visibility to the adversary in message transmission events to only events happening near his neighborhood, i.e., assume that the adversary only observes message transmissions on channels that are incident to a neighborhood where he is present—e.g., channels in the immediate neighborhoods of parties currently under his control. In our model, we make this assumption—which is necessary for sublinear-locality communication and MPC, and therefore implicit in all relevant works—explicit by assuming that the adversary is only able to observe a transmission when the sender or the receiver of that transmission is corrupted.

Adaptive Adversarial Scheduling. The above localized notification assumption is natural in large networks (and necessary for adaptive corruption), but it does create a challenge with allowing the adversary to perform worst-case scheduling: Since the adversary does not have a full view of which honest-to-honest channels are used in each round (recall that only a small, sublinear per party, number is utilized), how can he induce a worst-case scheduling? In fact, although the above localized-notification assumption was implicit in [15], this delicate issue was not addressed.

To address it, we look back at the classical way of defining scheduling for an adaptive and rushing adversary in [10]: Every round is split in "mini-rounds", where in each mini-round the adversary can have one party p_i send his message to another party p_j. If in the protocol every party communicates with all other parties in each round (a common protocol structure in feasibility results) then this means that each round has n^2 mini-rounds. The adaptive adversary is able to corrupt parties between any two mini-rounds. Note that, as explicitly discussed in [10], this means that a worst-case adversary would first deliver all messages sent to corrupted receivers and then schedule the remaining messages in an adaptive manner. To allow for worst-case scheduling in our model, we rely on the exact same idea: In each round r, the adversary operates in n^2 mini-rounds, where each such mini-round corresponds to a unique (ordered) pair of parties (p_i, p_j), allowing p_i to send its rth round message to p_j. The only difference here is that if and while both p_i and p_j are honest, the adversary does not learn whether or not a message was sent on the (p_i, p_j) channel in that round. (Of course, if either p_i or p_j gets corrupted down the line, then the adversary will find out at the point of corruption whether a message was exchanged in round r, unless the corrupted party has had a chance to erase before becoming corrupted—see discussion about erasures below.)

[4] Here, we use "full" MPC to refer to the standard MPC formulation where no party is left out; this is in contrast to "almost-everywhere" MPC [27], where some of the parties are not given any correctness and privacy guarantees.

Trusted Setup Assumptions. We assume classical correlated randomness setups: The parties have access to a public-key infrastructure (PKI), which they can use for digital signatures and public-key encryption. In addition, the parties are given an appropriately sampled hidden random graph setup with polylogarithmic degree [15]. As discussed in [15], and already mentioned earlier, under standard hardness assumptions (i.e., existence of pseudo-random generators) this graph setup can be replaced by a different correlated randomness setup, namely, a secret key infrastructure (SKI); alternatively, if the PKI allows for non-interactive key exchange (NIKE), then any other setup assumption is not needed since an SKI can be created by pairwise invocation of the NIKE protocol—since this is non-interactive, it does not result increase communication locality.

Secure Erasures. Secure erasures are common (and, in fact, necessary in SF protocols) for adaptively secure MPC and, as we prove here, for point-to-point communication over a sublinear-degree network graph. The assumption is that (honest) parties are able to erase any part of their internal state (including parts of their setup and/or randomness) so that if they get corrupted later on, the adversary does not have access to the erased information. We note that in a model where such erasures are possible, such actions take place responding to protocol instructions, and therefore the adversary corrupting a party is allowed to learn that an erasure was performed by the party in the past.

2.2 Atomicity Assumptions

Atomicity of Actions. One of the most important parameters of any adaptive adversary setting, which is often left as implicit, is the question of atomicity of operations for the honest party. A block of operations is considered atomic if, once an honest party starts performing them, it is allowed to complete them before the adversary gets a chance to act (e.g., perform additional corruptions). Clearly, the higher the number of operations that are bundled in an atomic block, the harder the job of the adversary becomes. One of the standard uses of atomicity in the distributed computing and cryptographic protocols literature is the so-called *atomic multi-send*, where if in a given round a party is supposed to send a message to multiple parties, it is allowed to do so without any in-between adversarial interference. One can view this assumption as a way to restrict the rushing ability of the adaptive adversary. As such, in the recent literature the (setting of an) adaptive adversary over a network of standard (point-to-point) channels (i.e., with non-atomic multisend) is at times referred to as *strongly rushing* [1,46] and the term "rushing" is used to refer to the setting where the adaptive adversary operates over a network with atomic-multisend channels.

In this work we use the term "rushing" consistently with [10,11,26,33] to characterize the adversary (i.e., its ability to schedule the delivery honest parties' messages) rather than the setting of network and adversary. Hence, atomic multisend is a network/protocol-related atomicity assumption: When the protocol instructs a party p to deliver a message to several parties in some set $\mathcal{Q}$ *in the same round* (or even a vector $(m_{p_1}, \ldots, m_{p_\mathcal{Q}})$ of messages, where each m_{p_i} is

to be sent to $p_i \in \mathcal{Q}$), then the party can send all those messages as an atomic operation, meaning that these messages will be delivered to their intended recipients at the beginning of the following round; in particular, the adversary cannot corrupt p in the middle of the transmission and enforce that some $p_i \in \mathcal{Q}$ receives the message m_{p_i} from p while some other $p_j \in \mathcal{Q}$ does not.

Atomicity Assumptions Used in This Work. The way different feasibility assumptions are bound together in an atomic operation has an impact on the feasibility of reliable communication (and therefore MPC) that one is able to prove. As our goal is to investigate the different relevant assumptions for sublinear- (polylogarithmic-) locality MPC, we next include a detailed discussion on the ways these assumptions can be bound (and have been bound in prior literature):

- *No erasures/no (atomic) multisend (NE-NAMS):* This is the worst-case network and erasures model (fourth row of Table 1): (1) No honest party is allowed to erase its internal state and the adversary, upon corrupting a party, learns that party's entire prior and current state, and (2) the parties send their messages one by one as discussed in the adaptive scheduling above.
- *No erasures/(atomic) multisend (NE-AMS):* This corresponds to the second row of Table 1): (1) (Lack of) erasures are as above, and (2) the parties can atomically multisend their messages.
- *Erasures/no (atomic) multisend (E-NAMS):* This corresponds to the third row of Table 1. We assume erasures but no atomic multisend. Hence: (1) In each mini-round where a party p_i is allowed to "speak" (we say the party is *activated*); i.e., in each miniround corresponding to a pair (p_i, p_j) for some p_j, the party p_i can first erase and then send, but it cannot erase the message it is about to send (including the identity of the receiver) until the next mini-round when the party is activated, and (2) messages are sent in a one-by-one manner (in mini-rounds as above), where between any two mini-rounds the adversary can act.
- *Erasures/(atomic) multisend (E-AMS):* (1) Erasures are as in the previous case, and (2) when a party is activated for sending, it is allowed to erase and then send to a set $\mathcal{Q}$ of parties (but as above, it cannot erase this set or the messages it sends until the next time it is activated). Note that this case allows an adversary who has corrupted a party in $\mathcal{Q}$ to learn the message, corrupt the sender, learn the identities of all other parties in $\mathcal{Q}$ (since the sender is not given a chance to erase before) and corrupt all of these parties thereby blocking this message.
- *Atomic erasures* **and** *multisend (AE-AMS):* This is the strongest of the atomicity assumpions (first row of Table 1) and corresponds to the model considered in past works on sublinear locality MPC with adaptive corruptions (e.g., [7,15]). In this case, whenever a party is activated for sending a message, it is allowed to send to a set $\mathcal{Q}$ of parties (where all are guaranteed to receive their messages at the beginning of the following round) *and* perform erasures after sending is complete and before the adversary has a chance to corrupt this party. This, in particular, means that even if the adversary controls one

of the parties in $\mathcal{Q}$, he is still unable to learn who the other parties in $\mathcal{Q}$ are even by corrupting the sender (as before corruption the sender is able to erase their identities).

3 Technical Overview

Impossibility of Store-and-Forward Without Erasures. Our first technical contribution (see Sect. 4) is an impossibility result for all-to-all (in fact, even one-to-all) store-and-forward RMT with a high expansion rate, if we do not assume erasures. In particular, we show this impossibility for expansion rate $(\log^z(n), \frac{k \log n}{c \log \log n})$, for all $z > 1$ and $k < 1$, where the degree of the communication graph is $O(\log^c n)$. Intuitively, our definition of (L, ℓ) expansion rate (see Definition 2) captures the constraint that, in an honest execution of the protocol, when the sender's message reaches parties up to a distance ℓ from himself, his message also reaches at least L parties through each of his neighbors. To our knowledge, this property can be shown to hold for all CL MPC protocols in the current literature. The result holds independently of whether or not we assume atomic multisend. The proof utilizes a combination of graph-theoretic and protocol results and can be summarized as follows: The first step in the proof shows that, due to the polylogarithmic locality assumption, there must exist a pair (p_s, p_r) of sender and receiver that are far enough from one another, concretely, in distance greater than $\frac{k \log n}{c \log \log n}$ (Lemma 2). Looking ahead, this will be the pair the adversary will try to disconnect. The expansion rate assumption will then ensure that for each forward neighbor p of p_s in the RMT to p_r, the graph rooted at p that is created by coloring the communication graph as the message of p_s (intended for p_r) passes though the intermediate nodes, will have sufficiently many colored nodes before it reaches p_r, so that with high probability one of them can be adversarial. Once this happens, the adversary will be able to follow the thread backwards all the way to this neighbor p and then corrupt p and all parties that p has sent the message to, thereby eliminating p as a possible relayer. Since each such p can be eliminated with high probability, and p_s had at most polylogarithmically many neighbors, the above adversary will be able to capture all these neighbors (and the colored graphs rooted at them) before the message reaches p_r (and without corrupting p_s). This adversary can drop all the message passing from captured nodes, thereby disconnecting p_s and p_r and violating the security of the RMT. The detailed proof is given in Sect. 4.

We remark that, although the above SF subclass might appear somewhat simple, it captures the structure of several natural message-propagation and gossiping protocols—in particular those used in the context of blockchains, as well as in so-called *store-and-forward* (switching) networks (cf. [3]). In fact, as demonstrated in Lemma 1, the class of SF RMT protocols with expansion-rate parameter that fall within our impossibility range includes all known message-propagation protocols in the sublinear locality MPC realm. We stress that our impossibility is not intended as a way to tightly characterize the feasibility landscape of CL RMT in the non-erasure setting, but rather to abstract the core of

the above protocols that makes them inadequate against adaptive adversaries in this setting. Notwithstanding, we are not aware of any technique that yields a protocol in the non-erasure setting, as even common approaches for anonymous communication, e.g., onion-routing-based protocols, seem insufficient in this highly adversarial setting (as discussed in Sect. 1.4).

The Power of Secure Erasures for CL Protocols. As discussed above, the above impossibility (for SF RMT) protocols renders existing communication protocols in the CL MPC (and blockchain-via-gossip) literature insecure when erasures are not assumed. Continuing our exploration of the landscape, we turn to the question of how far can the secure-erasures assumption take us in terms of feasibility of RMT. Recall that [15] proved that if erasures and multisend can be performed/bundled in an atomic operation, then any adversary corrupting any constant fraction of the parties can be tolerated in RMT. Here we ask what happens if we unbounded these two assumptions, i.e., assume either only erasures, or erasures and multi-send as separate operations. And we shoot for the strongest possible results: (1) Feasibility even without atomic multisend and (2) Impossibility even with atomic multisend.

For the first (feasibility) result, our starting point is the RMT protocol from [15], which at a high level operates as follows: The sender sends his signed inputs to his hidden graph Round-1 neighbors,[5] and whenever a party receives a message in some round rnd, he relays it (in the next round, $rnd + 1$) to his $(rnd + 1)$-round hidden-graph (forward) neighbors[6].

The above protocol does not work here, since the communicating parties can be cut off by the attack described in Sect. 1.2; namely, a corrupted node p who receives a message from an honest neighbor q can corrupt q and all of q's neighbors before any chance of erasure. To make the above protocol secure when (just) erasures are assumed, we make the following modification: Every round is assigned to exactly one party in $\mathcal{P}$, who, if he has a message to send, sends this message to exactly one of his hidden graph (forward) neighbors at a time, and then (in the next activation/round) erases both the fact that he sent it and the relevant (used) edge from his hidden graph setup (i.e., the edge pointing to the neighbor he just contacted). Importantly, unlike [15], where as soon as a party relays a message he does not need to do any more relaying, we require every party that has received a message to keep relaying it to its hidden graph(s) neighbors. By forwarding messages to neighbors one at a time and erasing in between, the adversary is prevented from corrupting the neighbors of honest relayers who get corrupted, enabling a message to propagate into the network even after a past relayer is compromised. Note that to allow for worst-case attacks, one needs to devise a careful structure that gives the adversary sufficient attack-opportunities. We ensure this by forcing on our protocol a structure that makes erasures slow enough, so that the adversary is given enough time to attack (see Sect. 5.1.1).

[5] Since we will be running at most polylogarithmic-round protocols, we can assume wlog that the hidden graph is a multi-graph consisting of polylogarithmic, independent copies of polylogarithmic-degree hidden graph setup.

[6] Recall that the hidden graph is directed.

The above protocol is clearly SF, and it even has an expansion rate that matches the parameter of our non-erasure impossibility theorem. Hence, one might be tempted to believe that if the adversary corrupts a constant fraction of the parties, then with good probability he will be able to block the message in a similar way as in our no-erasures impossibility proof. However, contrary to the above intuition, we show that any adversary corrupting at most $t < (1/2-\epsilon)n$ of the parties can be tolerated (with overwhelming probability). The proof follows a careful probabilistic argument: Since, from a Chernoff bound, more than half of the hidden graph neighbors of any party are honest, we can show that the above protocol will create an avalanche effect: With good probability, for several rounds from the start of the above protocol the set of honest parties that has received the message will keep growing and it will become large enough to be guaranteed to include a party who is one hop (in the hidden graph) from the RMT receiver. Once this happens, it is game over for the adversary since this party will relay the message to the RMT receiver in the following round.

The above result establishes (one-to-one) RMT assuming an honest majority and erasures only (Theorem 2) and as a corollary, (one-to-one) RMT assuming honest majority, erasures, and multisend, not necessarily bulked as a single atomic operation (Corollary 1). Furthermore, it can be trivially extended to the all-to-all RMT case (where every party wants to send a message reliably to every other party) by using batching techniques from [15] (Corollary 2). This settles the feasibility question.

We next turn to impossibility. Here we use an argument that can be seen as mirroring the proof of Theorem 2: We prove that if the majority of the parties can be corrupted (in particular, $t > (1/2 + \epsilon)n$ for any constant ϵ) then in *any* RMT-protocol candidate, the adversary can with noticeable probability make the expansion of the set of parties that learn the sender's input shrink, resulting in the message dying in the network before it reaches the receiver. This yields a strong impossibility of RMT in the erasures model, which as we show holds even if we assume multisend (Theorem 3).

Beyond Store-and-Forward. Next, we turn our attention to the question of whether one can design RMT protocols in the non-erasure model, by devising non-SF protocols, thereby circumventing our impossibility. The answer to this question is far from simple, which further underpins the challenges that CL protocol design poses.

Our key idea is to hide the store-and-forward procedure under the hood of fully homomorphic encryption (FHE). This will hide the message and signature (and in particular, origin and path) for transmitted messages. But as as we shall see, this seemingly simple intuition needs several modification to work.

The protocol structure is similar to the previous protocol (which assumed secure erasures), with the main difference being that the sender encrypts his original message and signature with the receiver's HE public key, and this ciphertext is what is diffused through the low locality network. In particular, instead of checking signatures in the clear, every party uses HE to check if any of the received ciphertexts is an encoding of a message signed by the sender. If so,

it (the circuit evaluated by HE) outputs a new, rerandomized HE ciphertext that encrypts the sender's message and signature (if several such messages are received then use any of them); otherwise encrypt the all-zero message along with a default signature on it. We denote the above protocol by $\Pi_{\mathsf{FHE}}^{\mathsf{RMT}}$.

In order for the above approach to work we need the HE scheme to satisfy several properties that are common in the fully homomorphic encryption (FHE) literature, namely *compactness* and *malicious circuit-privacy* [29,43,45]. Informally, compactness ensures that the ciphertext size depends only on the plaintext and the security parameter (in particular it does not grow when the $\mathtt{Eval}_{\mathtt{FHE}}$ operation is applied). On the other hand, malicious circuit privacy ensures that the ciphertext that is computed by $\mathtt{Eval}_{\mathtt{FHE}}$ leaks no information about the circuit that was homomorphically evaluated, even when the ciphertext on which $\mathtt{Eval}_{\mathtt{FHE}}$ is computed is maliciously formed. This last property will ensure that applying $\mathtt{Eval}_{\mathtt{FHE}}$ automatically rerandomizes the ciphertext. We note that both of these properties are common in standard FHE schemes (cf. [29,42,43,45]).

A caveat in the above idea is that it is known that adaptively secure FHE which satisfies compactness is impossible [37]. Nonetheless, we construct a compact adaptively secure FHE scheme which allows for the homomorphic evaluation of the specific function needed by our RMT protocol. In particular the specific function we require does not actually modify the contents of the underlying message that was encrypted (but only checks validity of a signature "under-the-hood" and then retains or disregards the underlying message). We stress that existence of such a scheme does not contradict the impossibility result of Katz *et al.* [37], as the circuits we consider are not in the class considered there.

Next, we describe the idea to circumvent the above impossibility. We consider only the single-pair RMT setting, where one sender u wishes to send a message to one receiver v. In order to ensure that ciphertexts can be simulated without knowledge of the plaintext and, if, later on, the sender and/or receiver is corrupted, the simulator can present randomness matching this ciphertext, we use the following idea: The sender first encrypts the message m it wishes to send with the receiver's public key, using an adaptively secure encryption scheme (e.g., the non-committing encryption of Canetti *et al.* [13]). (Looking ahead, this will allow a simulator to equivocate the message if needed.) Next, the sender signs the resulting ciphertext c, and encrypts the resulting pair (c, σ) using the receiver's public key for a (*statically secure*) FHE scheme. For brevity, we will refer to the above operation of encrypting with FHE an authenticated version of the adaptively encrypted plaintext as *adaptively authenticated homomorphic encryption* (aaHE for short). Then the sender propagates the aaHE ciphertext through the hidden communication graph, identically to the original protocol described in Sect. 5.1.

However, unlike the original protocol, parties that are at distance > 1 from the sender cannot tell if the aaHE ciphertext they receive is encrypting a valid message (i.e., one actually originating from the sender). Looking ahead, this property is the key part where our protocol deviates from the SF protocol structure. One solution would be for relayers to propagate all messages they receive.

This, however, would result in an exponentially growing message and could compromise the security of the protocol, as the number of relayed messages would leak information about the position of the relayer in the hidden graph.

This is where FHE comes to the rescue. Upon receiving several such ciphertexts, a relayer homomorphically evaluates the circuit which on input all the (plaintexts of the) received ciphertexts and the sender's verification key, checks if any of these plaintexts is of the form (c, σ), where σ is a valid sender's signature on c, and if it finds one it outputs (an FHE ciphertext $\tilde{c}$ encrypting) it. (If more than one such message is found then output the first one encountered in the above search.) This will make sure that every relayer sends out ciphertexts of the same length which encrypt either (c, σ), in case the sender was honest and the relayer is on an honest path between the sender and the receiver, or some arbitrary pair $(*, *)$ (chosen by the adversary) otherwise. In other words, the above scheme ensures that the information transmitted by the above scheme is exactly an aaHE encryption of the message m which our original SF protocol from Sect. 5.1 would propagate, except that since this information is encrypted, the adversary cannot use it to trace the message/aaHE ciphertext back to the sender. We stress that the properties of compactness as well as malicious circuit privacy, of FHE play a crucial role here, as we elaborate in the full version.

However, there is still a way the adversary can obtain information in the above scheme that allows him to potentially link the sender to the transmitted aaHE ciphertext, by observing which parties "speak" in which round. This can easily be mitigated by decoy traffic: every party sends some message in every round, where in the first round, parties other than the actual sender create and send to their neighbors an aaHE ciphertext using dummy strings d and s in place of m and σ respectively, with $|d| = |m|$ and $|s| = |\sigma|$.[7]

The final missing piece in the protocol is to ensure that we can use the above protocol for transmitting multiple messages (as is needed in MPC). It is not hard to see that for this to be possible, we need to exclude replay attacks. Indeed, although the adversary cannot create an aaHE of a new message corresponding to honest senders, he can replay past aaHE of messages created by the sender. The way to mitigate this is, as is common in the security literature, to use unique publicly agreed identifiers—e.g., round-number and/or message ID—and make sure that the circuit which $\mathtt{Enc}_{\mathtt{FHE}}$ is run on, also takes as input (and checks) the corresponding identifier. Yet, one needs to be careful about where the identifier is placed inside the aaHE. If one encrypts $(m, \mathsf{msg\_ID})$ with aaHE then $\mathtt{Eval}_{\mathtt{FHE}}$ will be unable to check the message ID $\mathsf{msg\_ID}$ under the (adaptively secure) encryption. Therefore, the actual message which is encrypted with FHE will be of the form $((c, \mathsf{msg\_ID}), \sigma)$, where σ is a signature on the pair $(c, \mathsf{msg\_ID})$.

This completes the high-level protocol description. In Theorem 4 we prove that the above protocol is an adaptively secure single-pair RMT protocol with polylogarithmic locality. Intuitively, the fact that the message is transmitted successfully between two honest parties follows from the fact that the protocol view of the adversary in this case is fully simulatable, hence any attack by an

[7] WLOG, we assume that valid RMT messages are from a fixed-size domain.

adaptive adversary who does not corrupt the sender or receiver can be reduced to the static case proven in [15]. If, on the other hand, the adversary corrupts the sender or the receiver, then the only challenge for the simulator is to be able to come up with coins that are consistent with the actual input m of the sender. But this is straightforward in aaHE as the FHE ciphertext (i.e., encrypting $((c, \mathsf{msg\_ID}), \sigma)$) is generated by the simulator and hence he can simply reveal the keys used for this encryption. Having these coins, the simulator needs to simply show how to open c to the message m. But for that, he can use the adaptive security of the underlying encryption scheme.

From One-to-One CL RMT to Many-to-Many CL RMT. The FHE-based protocol described above is for one-to-one RMT. One might might be tempted to assume that having such a protocol gives us also all-to-all RMT in this model. Indeed, at first thought, it appears that one can achieve this by simply running $\frac{n(n-1)}{2}$ instances of $\Pi_{\mathsf{FHE}}^{\mathsf{RMT}}$ in parallel over the same hidden graph (i.e., a joint state) on separate slots, one for each sender-receiver pair, following the idea of Sect. 5.1.2. However, the following major flaw breaks such a protocol.

Let u and v be honest parties in the network, and z be a party corrupted by adaptive adversary $\mathcal{A}$, with u being a sender while v and z act as receivers. Further, suppose the shortest honest path from u to z is shorter than that from u to v; i.e., z receives her message from u before (in an earlier round than) v. Now, adversary $\mathcal{A}$ can decrypt z's message in that same round and learn that it is a valid message originating from u. Moreover, $\mathcal{A}$ can corrupt the neighbor who sent her this valid message and decrypt that neighbor's previously received set of messages for z. In this manner, $\mathcal{A}$ can corrupt everyone who this message has passed through by following the inverse path the message travelled. Clearly, this is identical to the adversarial strategy employed in Sect. 4 to prove the impossibility of SF RMT in the $\mathsf{NE\text{-}NAMS}$ model, and it is easy to see that the same reasoning breaks the protocol here.

Intuitively, the root of the problem is that using FHE under the receiver's key renders messages "unlinkable" for everyone *but* the receiver. In single-pair RMT, there is a single receiver and correctness is trivial if he is corrupted, so $\Pi_{\mathsf{FHE}}^{\mathsf{RMT}}$ suffices. But with multiple receivers, corrupted receivers can evidently cause trouble for honest ones.

The above discussion, points to a restricted class of all-to-all RMT, which we call *sublinear-output-set RMT* (in short, SOS-RMT) to come to the rescue. SOS RMT allows every party (a sender) to send a (potentially different) message to every receiver *in a subset of* $\mathcal{P}$ *of size* $o(n)$—in our case this will be of polylogarithmic size. Here is how we proceed towards the design of SOS RMT:

1. First we observe that the above one-to-one RMT can be trivially turned into an all-to-one RMT, by having honest parties replace their decoy messages with the actual message they want to send to the (single) receiver and adjusting the homomorphic operation to keep (one copy of) all these messages as the ciphertext is diffused through the network. We provide the specification of this operation and the corresponding statement of security in Sect. 6.2. Note

that this protocol uses the same edges of the underlying communication graph as the one-to-one RMT protocol it "piggybacks" on.
2. Having such an all-to-one RMT it is straightforward to turn it to an SOS RMT by using an independent (part of the) hidden-graph setup for each of the receivers. Since there are polylogarithmic receivers and each of these hidden (sub-)graphs has polylogarithmic degree, the resulting protocol will also have polylogarithmic locality (see Corollary 4).

From CL RMT to CL MPC. Last but not least, we show how to turn the above feasibility results on RMT into feasibility for CL multi-party computation (CL MPC). For the erasures case, we can use the same approach as the one used in [15] for this reduction: Use a constant-round MPC, e.g., [2], where calls to a broadcast channels are replaced by a polylogarithmic-round (in the worst-case) byzantine broadcast protocol. Because the expected constant-round protocol of [35] is guaranteed to terminate with overwhelming probability after polylogarithmically many round, we can simply employ this protocol. This will result in polylogarithmically many invocations of the all-to-all RM from Sect. 5.1, which consumes a polylogarithmic hidden graph setup, thereby yielding a CL MPC protocol in the secure-erasures model (with or without atomic multisend), which is secure under an honest-majority (adaptive) adversary. We refer to Sect. 7.1 for details.

The more challenging case is the non-erasures setting. Here, we do not have an all-to-all CL RMT, so we cannot hope for standard CL MPC—as the latter would imply the former. Instead, we go for (CL) SOS MPC, which as with SOS RMT, computes a function with inputs from all parties, but only distributes the output to a sublinear (polylogarithmic) set of parties. We believe that the notion of SOS MPC is interesting in is own accord, as it appears to be a best-possible security notion in the CP non-erasure setting. Furthermore this notion is already reasonable for the core application of CL MPC, namely computing sublinear algorithms. Indeed, such algorithms typically have output asymptotically smaller than n. In this case, having the output-set of SOS MPC distribute the otputs to the whole player set (using the complete graph) does not incur a big overhead in communication complexity.

To implement CL SOS MPC, we assume an additional setup, namely, anonymous PKIs for the FHE, NCE, and signature schemes used in our non-erasures RMT protocol: Parties are given public keys but they do not know who has the corresponding secret key. Given this setup and SOS RMT, SOS MPC can be designed as follows: Let C denote a polylogarithmic size subset of the (owners of the secret keys for) the anonymous public keys. (Any subset will do, but for simplicity we can assume that this is the first polylog(n) public key in a lexicographic order). The parties use SOS RMT (where the FHE and NCE encryptions, and the underlying signatures are are with the anonymous PKIs) to share their input to C. Then the parties in C run an MPC over SOS RMT (again with these anonymous keys). However, to avoid leaking their identities through communication pattern, *all* n parties participate in these RMTs, where parties not in C simply send decoy traffic as in our one-to-one version of the

non-erasure RMT protocol. The details on this construction and security proof are given in the end of Sect. 7.1.

4 Impossibility in the NE-NAMS and NE-AMS Models

In this section, we show an impossibility result for a natural class of reliable message transmission (RMT) protocols (and therefore also MPC) which we term store-and-forward protocols, with low communication locality in NE-NAMS and NE-AMS models—i.e., a model where secure erasures are not allowed—tolerating an adaptive adversary corrupting a linear number of parties. As discussed above, this class includes most if not all, gossip-style communication protocols which have been used in the CL MPC and CL communication literature.

Let us first define the class of store-and-forward RMT protocols:

Definition 1 (Store-and-Forward). *A PKI-hybrid RMT protocol with sender p_s and receiver p_r, using parties in $\mathcal{P}$ who communicate over point-to-point channels, is a* store-and-forward *(in short, SF) protocol if it has the following structure:*

- *In the first round of the protocol, p_s sends the message m (that he wishes to transmit to p_r) to any neighbors of his choosing, along with his signature σ_s on m. p_s does not participate in any other transmission.*
- *In every round $j > 1$, any party who has received a pair (m, σ_s), where σ_s is a valid signature (with respect to p_s's verification key) on m, may forward the pair (m, σ_s) to any neighbors of his choosing.*

We remark that the above definition allows parties to selectively decide (using their current state) when and to whom they forward the pair (m, σ_s). As such it also captures protocols that use delays to hide communication patterns. Furthermore, the assumption that in the first round, p_s sends all his protocol messages does not pose a restriction with respect to such scheduling—i.e., on when p_s sends his message to each neighbor—as such delays can be trivially simulated by p_s telling his neighbors (in the first round) to apply the intended delay. We also point out that the above single-pair RMT can be trivially extended to all-to-all RMT, by allowing each relayer p to forward vectors of pairs $((m_{i_1}, \sigma_{i_1}), \ldots, (m_{i_\ell}, \sigma_{i_\ell}))$, where each (m_{i_j}, σ_{i_j}) is a (message, p_{i_j}-signature)-pair that p heard in a previous round.

Our impossibility result assumes a restriction on the above class of SF protocols, which relates the maximum length of a path traversed by the sender's message to the size of the set of parties that have seen the message. To define this, we introduce the following graph theoretic notation.

Notation. We will denote by $G_{s,r}$ the *labeled* graph with vertices $V = \mathcal{P}$ which corresponds to a protocol's execution in the following manner: an edge (w_1, w_2) is added to $G_{s,r}$ when $w_1 \in V$ sends the message (m, σ_s) to $w_2 \in V$ through their point-to-point channel, in the RMT protocol between sender p_s and receiver p_r. The label l_{w_1,w_2} of each such edge (w_1, w_2) is defined as the round of the RMT

protocol in which this edge was added to $G_{s,r}$. Further, we denote by $G_{s,r}^{\mathsf{rnd}}$ the subgraph of $G_{s,r}$ that only contains edges (w_1, w_2) having labels $l_{w_1,w_2} \leq \mathsf{rnd}$.

Definition 2 (Expansion Rate of SF RMT Protocols). *We say that an SF RMT protocol has expansion rate (L, ℓ), where $L \in [n]$ and $\ell \in \mathbb{N}$, if the following property holds at every round* rnd *in the protocol execution: If the maximum size of a path in $G_{s,r}^{\mathsf{rnd}}$ from p_s to a sink (i.e., a node which has out-degree 0 in $G_{s,r}^{\mathsf{rnd}}$) is ℓ, then for any (forward) neighbor p of p_s in $G_{s,r}^{\mathsf{rnd}}$, the number of nodes in the subgraph of $G_{s,r}^{\mathsf{rnd}}$ (with in-degree at least 1) rooted at p is at least L.*

The above definition can easily be extended to all-to-all SF RMT protocols (resp. one-to-all), by requiring that for every pair (p_s, p_r) (resp. for sender p_s and all receivers p_r) the expansion rate of the transmission is as above.

Looking ahead, we will prove our impossibility result for SF RMT protocols with expansion rate $(\log^z(n), \frac{k \log n}{c \log \log n})$ for all $k < 1$ and $z > 1$, where the degree of the communication graph is $O(\log^c n)$. To get a better intuition of how the expansion rate affects the security of RMT protocols against adaptive adversaries, it is worth looking at the simpler case with expansion rate $(d^{\xi-1}, \xi)$, where $d = O(\log^c n)$ is the degree of the underlying communication graph and ξ is a constant. We note that this seemingly simple case corresponds to (the first of ξ rounds of) the "vanilla" SF strategy which, to our knowledge, is employed by all CL MPC protocols in the literature.

Lemma 1. *Assuming no erasures, there exists no polylogarithmic-locality SF one-to-all RMT protocol with sender p_s that has expansion rate $((\log^c n)^2, 3))$ for some constant $c > 1$, and tolerates an adaptive adversary corrupting a constant fraction of the parties.*

Note that this is the case with $\xi = 3$. We give a proof sketch of the above lemma in the full version. Since this is just a special case of the general theorem (Theorem 1) proved later in this section, we keep the proof at an informal level, to allow the reader to grasp the main ideas, and refer to the remainder of this section for formal claims (that even cover the more general case).

The Case of Expansion Rate $(\log^z n, \frac{k \log n}{c \log \log n})$. We assume that the adversary is able to corrupt a constant fraction τ of the nodes; hence $t = \tau n$. Our impossibility result holds for any constant $\tau > 0$. Our proof relies on a series of lemmas (see the full version for proofs) on the (polylogarithmic-degree) communication graph and how the adversary attacks any RMT protocol π over such a graph. Concretely, towards our impossibility result, we prove the following:

1. First, we will show (Lemma 2) that there exists a sender p_s and a receiver p_r, such that the length of the shortest path between p_s and p_r (in the communication graph $G_{s,r}$ of π) is strictly greater than q, where $q = \frac{k \log n}{c \log \log n}$, for any $k < 1$ and where the communication locality of the RMT protocol is $O(\log^c n)$. In other words, let rnd be the round of π in which the (p_s, p_r)-RMT message first reaches a distance q from p_s; then, p_r is not connected to any node in $G_{s,r}^{\mathsf{rnd}}$.

2. The remainder of our proof strategy is as follows: Consider an execution of RMT with sender p_s and receiver p_r as above; the communication graph after rnd rounds is $G^{\mathsf{rnd}}_{s,r}$. The goal of the adversary (and what we will prove he can achieve) is to corrupt each of the neighbors of p_s in this graph, and also corrupt everyone to whom they have (directly or indirectly) conveyed information on p_s's message, *before* this information reaches the receiver p_r. Thus, our proof focuses on each of the neighbors of s individually, and shows that the above is achieved with overwhelming probability. Using the fact that the total number of neighbors of p_s is $O(\log^c n)$, and by the choice of q, we can then prove that the probability of an adversary successfully attacking all of p_s's neighbors and cutting p_s off before (information on) his intended message reaches p_r is noticeable (Lemma 3). This forms a successful attack on the RMT protocol π, as it disconnects p_s and p_r, completing the proof.

The formal statement of Lemma 2 now follows.

Lemma 2. *For any given SF RMT protocol π with communication locality $O(\log^c n)$, let $q = \frac{k \log n}{c \log \log n}$ for any $k < 1$. Consider a sender p_s. There exists a receiver p_r such that p_r is not connected to any node in $G^{\mathsf{rnd}}_{s,r}$, where rnd denotes the round of π in which the (p_s, p_r)-RMT message first reaches distance q from p_s.*

The most interesting step which captures the essence of our proof is Lemma 3 below. The intuition of the proof is that because the distance between p_s and p_r is larger than q, each p_s-neighbor in $G_{s,r}$ will have distance of at least q to p_r in $G_{s,r}$. Thus, any message originating from such a p_s-neighbor needs at least q hops to reach the receiver. Now consider the subgraph of $G_{s,r}$ which grows from a forward neighbor p of p_s only, i.e., the graph consisting of p, his neighbours, his neighbors' neighbors and so on. To prevent p_s from communicating with p_r via p, the adversary will first corrupt nodes at random with the hope of corrupting at least one node in this subgraph. More specifically, let us consider an adversary that initially corrupts a $\beta < \frac{\tau}{4}$ fraction of random nodes in the whole graph $G_{s,r}$, and show that such an adversary leaves both p_s and p_r initially uncorrupted—and looking ahead, will avoid corrupting p_s and p_r after this initial step. The first observation is that the message sent by p_s and relayed by p will, with overwhelming probability, hit some party in this initially corrupted set before it hits the receiver p_r. Once this happens, the adversary corrupts everyone who this message has passed through by following the inverse path the message travelled. This is feasible because nodes cannot erase any information, and in particular where messages came from and where they were relayed. If the adversary is able to do this for every p_s-neighbor, then he successfully cuts p_s off from p_r. To complete the proof we need to argue that this strategy can be launched within the adversary's corruption budget. Intuitively, this is the case, because in each step the total set of parties who have received information about the sender's message grows by a polylogarithmic factor. Hence, by the above choice of q we are guaranteed that the size of the set remains sublinear. Since the adversary

has only spent a fraction of his linear budget in his initial corruption choice, he still has sufficiently many corruptions to perform the above attack. The formal statement follows.

Lemma 3. *For any given SF RMT protocol π with communication locality $O(\log^c n)$, let $q = \frac{k \log n}{c \log \log n}$ for any $k < 1$. Further, let π have expansion rate $(\log^z n, q)$ for any $z > 1$. Consider a sender p_s and a receiver p_r, and an adversary $\mathcal{A}$ who corrupts each node in $\mathcal{P}$ at random with constant probability $\beta < \frac{\tau}{4}$. Then, with noticeable probability $\frac{1}{\mathsf{poly}(n)}$, $\mathcal{A}$ (i) does not corrupt more than $\frac{\tau n}{2}$ nodes in total; (ii) does not corrupt nodes p_s and p_r; but (iii) corrupts at least one node in $\{p\} \bigcup \Gamma_{(q-1)}(p)$ for every $p \in \Gamma_1(p_s)$.*

We now combine Lemmas 2 and 3 to obtain our impossibility result, in the following theorem, the proof of which is given in the full version.

Theorem 1. *In NE-NAMS and NE-AMS models (i.e. models not assuming erasures), there exists no SF protocol, with $(\log^z n, q)$ expansion-rate, for all-to-all RMT with polylogarithmic (i.e., $O(\log^c n)$) communication locality tolerating an adaptive adversary corrupting a linear number $t = \tau n$ (for any constant τ) of parties, where $q = \frac{k \log n}{c \log \log n}$, for any $k < 1$ and $z > 1$. The statement holds even assuming an arbitrary correlated randomness setup, atomic multisend, and any cryptographic hardness assumptions.*

5 Positive Results in the E-NAMS/E-AMS Models

In this section, we assume that parties can erase their state. In our positive result, we do not assume that parties have an atomic multisend operation available to them, and the operations of sending a message and erasing state are not atomically bound either. This corresponds to the E-NAMS model. We will first show an all-to-all RMT protocol in this model with polylogarithmic locality tolerating $t < (\frac{1}{2} - \epsilon)n$ corruptions (Sect. 5.1); this automatically also implies a protocol in the stronger E-AMS model. To complement this result, we also show that tolerating $t > (\frac{1}{2} + \epsilon)n$ corruptions is impossible in the E-NAMS model (Sect. 5.2).

5.1 Polylogarithmic Locality RMT in the E-NAMS Model

Our protocol is a standard RMT protocol that allows a sender p_s to reliably transmit a message to a remote recipient p_r over a polylogarithmic degree graph. The all-to-all RMT is then obtained by having each pair use this RMT simultaneously. In addition to a PKI (for digital signatures) our protocol uses a hidden graph setup as in [15] as follows: the setup picks a directed random Erdős-Rényi graph $G(n, p) = (V, E)$, where $V = \mathcal{P}$ is the vertex set and E is the set of edges in G, and for every $i, j \in V$, $\Pr[(i, j) \in E] = p$. This graph is given to the parties such that every party learns its incoming and outgoing edges in G (and nothing else). From Sect. 2, recall that for any node $u \in V$, we denote its set of outgoing neighbors by $\Gamma_1(u)$. The set of nodes at distance $\leq i$ on a forward (directed) path starting from a node u are denoted by $\Gamma_i(u)$.

5.1.1 Single-Pair RMT Protocol in the E-NAMS Model

We now describe our RMT protocol from honest sender p_s to honest receiver p_r, where $p_s, p_r \in V$, denoted by Π_{RMT}. Our protocol assumes a PKI, a hidden graph setup, existentially unforgeable digital signatures (equivalently, one-way functions), worst-case secure erasures (as discussed in Sect. 2), and no atomic multisend. As a corollary of our statement, at the end of the section we prove that the protocol with a minor tweak works even if we assume atomic multisend instead of just point-to-point communication. (The latter corollary is not that surprising, as atomic multisend restricts the adversary's power, but needs to be nonetheless done with care to make sure the adversary cannot abuse it to discover the hidden graph.)

Protocol Structure. To make the protocol and proof simplest, we will (implicitly) induce the following structure: The protocol advances in blocks of two sequential (mini-)rounds, where in the first of these two minirounds a specific sender gets a chance to send a message to a specific receiver, and in the second one that sender gets a chance to erase his state (e.g., the information of the previous receiver). These miniround-blocks are advanced in a round-robin fashion: the first n blocks of such minirounds are with sender p_1 and receiver each party p_j in the party set; the next n blocks of minirounds are for sender p_2 and receiver each party p_j in the party set, and so on; after n such sets of n blocks, the $(n+1)$st set of n blocks is again with sender p_1, etc. Thus, a sequence of $2n^2$ minirounds (where all parties have had a chance to send all the messages they have for any other parties) constitutes a round in the protocol. We induce the above structure as it makes the influence of an adaptive adversary clean, no matter what model one is used to. Recall that we consider an adversary $\mathcal{A}$ who can adaptively corrupt up to a τ fraction of nodes in the network. Hence, $\mathcal{A}$ is allowed to order the blocks of minirounds within a single round (worst-case adaptive scheduling), and he can corrupt any party in between any two minirounds. Observe that this protocol structure does not increase the CL of the protocol, as a party will utilize its associated minirounds if and only if it has something to send to the corresponding receiver.

More concretely, the protocol starts at round 0, and we call the entire block below a round in the protocol.

- For $\mathsf{spkr} = 1$ to n, do the following:
  - If Party p_{spkr} has a message to send to p_0 it will do so.
  - Party p_{spkr} is given a chance to erase.
  - If Party p_{spkr} has a message to send to p_1 it will do so.
  - Party p_{spkr} is given a chance to erase.

    $\vdots$
  - If Party p_{spkr} has a message to send to p_{n-1} it will do so.
  - Party p_{spkr} is given a chance to erase.

The protocol then proceeds to the next round, completing a total of R rounds. The protocol Π_{RMT} itself is defined as follows.

RMT Protocol Between p_s and p_r. Our protocol proceeds for a total of $R = \log^{\tilde{c}} n$ rounds, for some constant $\tilde{c} > 1$ (where rounds are defined as above). All the verification keys of all nodes (denoted vk_w for each party $w \in \mathcal{P}$, with corresponding signing key sk_w) are known to all parties.

1. First, p_s signs (m, p_r) with sk_{p_s}. Denote the signed message (which comprises of the (m, p_r) as well as the signature on it) by μ_m. Party p_s also initializes ctr_{p_s} to the index of a random neighbor in $\Gamma_1(p_s)$.
2. Now, at every round $0 \leq j \leq R$, every node w does the following:
   - w checks if he possesses a single valid message μ_m - i.e., a message of the form (m, p_r) that has been signed by p_s. If so, then w sends μ_m to $\Gamma_1(w)[\mathsf{ctr}_w]$ and sets $\mathsf{ctr}_w = \mathsf{ctr}_w + 1$. (This constitutes a mini-round, and is immediately followed by another mini-round in which w is given a chance to erase the information of the node in $\Gamma_1(w)[\mathsf{ctr}_w]$ he sent to in the previous miniround.) Node w repeats the above over d many neighbors in $\Gamma_1(w)$ by iteratively incrementing ctr_w, where $d = O(\log^c n)$ (for some $c > 1$) is the communication locality. Otherwise, if he possesses no valid message μ_m, then he does nothing.
   - w disregards all messages from $w^* \notin \Gamma_1^{\text{in}}(w)$.

We now prove that the above protocol is an RMT between p_s and p_r. Define set, GOOD_j, $0 \leq j \leq R$ to be the set of nodes, who at the beginning of round j of the protocol are a) honest and b) are in possession of the message μ_m. Let $g_j = |\mathsf{GOOD}_j|$ for all j. The idea of our proof is that no matter what the adversary does, the set of parties that know the message (and will therefore forward it in the next round) grows multiplicatively in each round of the protocol. Hence, in $\log^{\tilde{c}} n$ rounds, the message will reach a large enough honest set, so that one of them will be a neighbor of p_r and will therefore forward the message to p_r. Once this happens, RMT will have succeeded.

To this end, we prove a series of lemmas, which can be found in the full version. We first use a probabilistic argument to show that the good set GOOD_1 of honest parties that have seen message μ_m at the beginning of round 1 has size $\mathsf{polylog}(n)$, with overwhelming probability. Next, we show that, as long as the adversary corrupts a minority of parties, the size of the good set increases multiplicatively in every round by a constant greater than 1. Finally, we prove that the above constant expansion will ensure that, in $\log^{\tilde{c}} n$ rounds, μ_m will arrive at its intended receiver p_r. This implies the following theorem.

Theorem 2. *Assuming a PKI, a hidden graph setup as above, secure erasures, one-way functions (for existentially unforgeable signatures) and an adaptive adversary corrupting at most $t < (\frac{1}{2} - \epsilon)n$ parties for any $0 < \epsilon < \frac{1}{2}$, the protocol Π_{RMT} realizes reliable message transmission from p_s to p_r. The statement holds in the E-NAMS as well as E-AMS models.*

Since atomic multisend is a stronger model than the non-atomic multisend setting considered above, the possibility result applies also to this case. Indeed, given atomic multisend one can trivially simulate point-to-point communication

between a sender p_s and receiver p_r, by having p_s multisend the vector that includes the intended message to the location corresponding to p_r and 0 to all other parties in its outgoing neighborhood of the hidden graph setup. This proves the following statement about the protocol $\Pi_{RMT}^{(MS)}$ which results from instantiating Π_{RMT} via the above invocation of the atomic multisend primitive.

Corollary 1. *Assuming a PKI, a hidden graph setup as above, secure erasures, one-way functions (for existentially unforgeable signatures), atomic multisend, and an adaptive adversary corrupting at most $t < (\frac{1}{2} - \epsilon)n$ parties for any $0 < \epsilon < \frac{1}{2}$, the protocol $\Pi_{RMT}^{(MS)}$ described above realizes reliable message transmission from p_s to p_r in the E-AMS model.*

5.1.2 All-Pairs (aka All-to-All) RMT in the E-NAMS Model

We now describe our protocol for RMT between all pairs of parties, denoted by Π_{a2aRMT}. This will allow every party u to send a message to every other party v in a total of R rounds (where a round is defined as earlier). At a high level, Π_{a2aRMT} works as follows. We will execute a total of $\frac{n(n-1)}{2}$ instances of protocol Π_{RMT} from Sect. 5.1.1 in parallel. For every receiver v, every sender u signs (m, v) with sk_u. Denote the signed message (which comprises of the (m, v) as well as its signature) by $\mu_{u,v}$. Every party w will maintain $\frac{n(n-1)}{2}$ slots, each corresponding to one (u, v) pair. Now, at every round $0 \leq j \leq R$, w checks if it possesses any valid message that has been sent by sender u to receiver v (i.e. a message $\mu_{u,v}$ of the form (m, v) that has been signed by u). It places this message in the slot corresponding to the pair (u, v). w then sends (potentially) all $\frac{n(n-1)}{2}$ messages to $\Gamma_1(w)[\mathsf{ctr}_w]$ and sets $\mathsf{ctr}_w = \mathsf{ctr}_w + 1$. It is easy to see that the communication locality of any party does not increase through this process – only the number of messages sent by a party at a time increases from a single message to a collection of $\frac{n(n-1)}{2}$ messages. Applying a union bound over all pairs of senders and receivers, one can obtain the following corollary to Theorem 2.

Corollary 2. *Assuming a PKI, a hidden graph setup as above, secure erasures, one-way functions (for existentially unforgeable signatures) and an adaptive adversary corrupting at most $t < (\frac{1}{2} - \epsilon)n$ parties for any $0 < \epsilon < \frac{1}{2}$, the protocol Π_{a2aRMT} realizes reliable message transmission between all pairs of parties $(u \in \mathcal{P}, v \in \mathcal{P})$. The statement holds in the E-NAMS as well as E-AMS models.*

5.2 Impossibility of Dishonest Majority in the E-NAMS Model

In this section, we shall show that it is impossible to construct SF reliable message transmission (RMT) protocols (and therefore also MPC) with low communication locality, in a model even with erasures, if the corruption threshold is $\frac{1}{2} + \epsilon$ for any constant $\epsilon > 0$. To do so, we shall prove that an adversary can break correctness of any RMT protocol between some pair of honest nodes in any such protocol. The proof idea can be seen as symmetric to the proof of Theorem 2. In

particular, in Theorem 2 we showed that if the adversary corrupts $t < (\frac{1}{2} - \epsilon)n$ parties, for any constant $0 < \epsilon < 1/2$, then the set of honest nodes that learns (and forwards) the sender's message grows exponentially fast, and therefore in $\log^{\tilde{c}} n$ rounds it will be large enough to hit a neighbor of the receiver. Here we prove that if $t \geq (\frac{1}{2} + \epsilon)n$ for any constant $0 < \epsilon < 1/2$, then there is a strategy making the above set shrink exponentially fast, which can make the message disappear before reaching the receiver p_r.

Consider an RMT protocol between an honest sender p_s and an honest receiver p_r. We shall show that for any SF RMT protocol from p_s to p_r, an adversary that corrupts $\frac{1}{2} + \epsilon$ parties can prevent the message m from reaching p_r in any polynomial number of rounds. As in Sect. 5.1, let the RMT protocol from p_s to p_r begin at round 0, and define GOOD_j, $0 \leq j \leq R$ to be the set of nodes, who at round j of the protocol are a) honest and b) are in possession of the message μ_m. Let $g_j = |\mathsf{GOOD}_j|$ for all j.

Adversarial Strategy. Our adversarial strategy is as follows: First, corrupt nodes in the graph uniformly at random (i.e., every node is corrupted with probability $\frac{1}{2} + \frac{\epsilon}{4}$). Next, if an adversarial node receives a message (that was a part of the RMT protocol between p_s and p_r) from some node w (other than p_s), then corrupt w. Do not forward any messages.

We now prove a series of lemmas (see full version) to show that our adversary violates the assumed security of RMT. We first show that our adversary corrupts at most $\frac{1}{2} + \epsilon$ fraction of nodes, with overwhelming probability. We next show that after the initial round, only a small set of honest nodes are in possession of the message. Finally, we show that after every round in the protocol, the number of honest nodes having the message reduces. This implies the following theorem.

Theorem 3. *In the E-NAMS model (i.e., without atomic multisend), there exists no all-to-all store-and-forward RMT protocol with polylogarithmic CL tolerating an adaptive adversary corrupting $t \geq (\frac{1}{2} + \epsilon)n$ (for any constant $0 < \epsilon < \frac{1}{2}$) parties. The statement holds even assuming an arbitrary correlated randomness setup, secure erasures, and any cryptographic hardness assumption.*

Remark 1. We note that the above argument holds if we do not assume atomic multisend (i.e., in E-NAMS). Indeed, in the stronger E-AMS model, the nodes might be able to do some smart multisend-based relay that prevents the set of parties that know the message from shrinking, or slows down the rate. We leave this interesting question as a future research direction.

6 Polylogarithmic Locality RMT in the NE-NAMS Model

In this section, we propose RMT protocols which are not store-and-forward, and can therefore circumvent the impossibility result from Sect. 4. Our key idea is to remove the ability of the adversary to identify the sender by looking at intermediate messages, with the use of fully homomorphic encryption (FHE) to hide the contents (and in particular, origin and path) of transmitted messages.

The resulting protocol for single-pair RMT is described in Sect. 6.1. However, we subsequently note that the same protocol loses its security when composed in parallel for the purpose of all-pairs RMT. In the following Sect. 6.2, we show how the protocol can be extended to obtain RMT from all senders to $\mathsf{polylog}(n)$ receivers, which we term *sublinear output-set* RMT (or SOS-RMT) and later use directly to achieve communication-local SOS-MPC in Sect. 7.2.

6.1 Single-Pair RMT Using Fully Homomorphic Encryption

We next provide a description of our (one-to-one) RMT protocol in the non-erasure case under strong cryptographic assumptions. Here we denote the sender by u and the receiver by v. See full version for the proof of Theorem 4.

Single-pair RMT Protocol $\Pi_{\mathsf{FHE}}^{\mathsf{RMT}}$ Between u and v from FHE and Adaptively Secure (Non-committing) Encryption. Similarly to the protocol from Sect. 5.1, our protocol proceeds for a total of $R = \log^{\tilde{c}} n$ rounds for any constant $\tilde{c} > 1$ (where rounds are as defined in Sect. 5.1). The protocol assumes setup for the following schemes:

- An existentially unforgeable digital signature scheme $(\mathtt{KeyGen}, \mathtt{Sign}, \mathtt{Verify})$. Denote by vk_u the verification key of the sender u and by sk_u the corresponding signing key.
- A non-committing encryption scheme $(\mathtt{KeyGen}_{\mathtt{NCE}}, \mathtt{Enc}_{\mathtt{NCE}}, \mathtt{Dec}_{\mathtt{NCE}})$. Denote by $\mathsf{pk}_v^{\mathtt{NCE}}$ and $\mathsf{sk}_v^{\mathtt{NCE}}$ the encryption and decryption keys of the receiver.
- A compact and malicious circuit-private FHE scheme $(\mathtt{KeyGen}_{\mathtt{FHE}}, \mathtt{Enc}_{\mathtt{FHE}}, \mathtt{Eval}_{\mathtt{FHE}}, \mathtt{Dec}_{\mathtt{FHE}})$. Denote by $\mathsf{pk}_v^{\mathtt{FHE}}$ and $\mathsf{sk}_v^{\mathtt{FHE}}$ the encryption and decryption keys of the receiver, respectively.

The protocol also assumes that the parties have agreed on unique public message IDs $\mathsf{msg\_ID}$ for the transmitted messages (this will include the protocol ID, the party ID, and the current round). The protocol proceeds as follows:

1. Computation of each party when the protocols starts (to compute the first message they will send):
   - *Code for the sender u:* First, u encrypts m with v's (non-committing) encryption key $\mathsf{pk}_v^{\mathtt{NCE}}$; denote the resulting ciphertext by c. Then, u signs $(c, v, \mathsf{msg\_ID})$ with sk_u; denote the corresponding signature by σ. Finally, u encrypts the pair $((c, v, \mathsf{msg\_ID}), \sigma)$ with v's FHE encryption key $\mathsf{pk}_v^{\mathtt{FHE}}$; denote the resulting ($\mathsf{aaHE}$) ciphertext by $\tilde{c}_u$.
   - *Code for each party $w \neq u$:* Party w computes c as an encryption of the all-zero message of size $|m|$ with v's (non-committing) encryption key $\mathsf{pk}_v^{\mathtt{NCE}}$ and sets σ to the all-zero string of same size as the actual signature of u above. Then, w encrypts $((c, v, \mathsf{msg\_ID}), \sigma)$ with v's FHE encryption key $\mathsf{pk}_v^{\mathtt{FHE}}$; denote the ciphertext by $\tilde{c}_w$.
2. Next, at any round $0 \leq j \leq R$, every node w does the following: Let $C_{w,j} = \{\tilde{y}_1, \ldots, \tilde{y}_q\}$ be the ciphertexts that party w has received in the previous rounds ($C_{w,j} = \{\tilde{c}_w\}$ if no messages have been received yet.)

- w applies the homomorphic evaluation function $\mathtt{Eval}_{\mathtt{FHE}}$ on input the ciphertexts in $C_{w,j}$, the verification key vk_u of the sender u, and the pre-agreed message ID $\mathsf{msg\_ID}$ to compute the following function: If any $\tilde{y} \in C_{w,j}$ can be parsed as $((c, v, \mathsf{msg\_ID}), \sigma)$, where σ is a valid signature on $(c, v, \mathsf{msg\_ID})$ according to the sender's verification key vk_u, then output $((c, v, \mathsf{msg\_ID}), \sigma)$. (If there are multiple such $\tilde{y}$, output the one with the smallest c.) Party w denotes the resulting FHE ciphertext by $\tilde{c}_{w,j}$, sends it to $\Gamma_1(w)[\mathsf{ctr}_w]$ and sets $\mathsf{ctr}_w = \mathsf{ctr}_w + 1$.
- w disregards all messages from $w^* \notin \Gamma_1^{\text{in}}(w)$.

3. At round R, v uses his FHE decryption key to decrypt each FHE ciphertext in $C_{v,R}$ (i.e., all ciphertexts received in the protocol). If any $\tilde{y} \in C_{v,R}$ can be parsed as $((c, v, \mathsf{msg\_ID}), \sigma)$, where σ is a valid signature on $(c, v, \mathsf{msg\_ID})$ according to the sender's verification key vk_u, then v uses his NCE decryption key $\mathsf{sk}_v^{\mathsf{NCE}}$ to decrypt c and outputs the corresponding message as the one sent by u (if more than one such c exists, then v takes the one corresponding to the smallest message m). Otherwise, v outputs 0 as the message received from u.

Theorem 4. *Assuming a PKI, hidden graph setup, trapdoor permutations with a reversed domain sampler, and compact and malicious circuit-private FHE [42], protocol* $\Pi_{\mathsf{FHE}}^{\mathsf{RMT}}$ *securely realizes single-pair RMT, tolerating an adaptive adversary who corrupts* $t < \epsilon n$ *parties for any* $0 \leq \epsilon < 1$*, in the* NE-NAMS *model.*

6.2 Multi-sender RMT

While $\Pi_{\mathsf{FHE}}^{\mathsf{RMT}}$ cannot be composed in parallel to achieve all-pairs RMT as discussed in Sect. 3, we show in this section that simple joint state parallel composition of single-pair RMT is sufficient to construct an all-to-one RMT protocol $\Pi_{\mathsf{FHE}}^{\mathsf{a21RMT}}$, and the usage of multiple all-to-one RMT instances over independent hidden graphs can further extend this to an SOS-RMT protocol $\Pi_{\mathsf{FHE}}^{\mathsf{SOS-RMT}}$. See the full version for detailed protocols and proofs.

Corollary 3. *Assuming a PKI, a hidden graph setup, trapdoor permutations with a reversed domain sampler, and compact and malicious circuit-private FHE,* $\Pi_{\mathsf{FHE}}^{\mathsf{a21RMT}}$ *securely realizes all-to-one RMT, tolerating an adaptive adversary who corrupts* $t < \epsilon n$ *parties for any constant* $0 \leq \epsilon < 1$*, in the* NE-NAMS *model.*

Corollary 4. *Assuming a PKI, a hidden graph setup, trapdoor permutations with a reversed domain sampler, and compact and malicious circuit-private FHE,* $\Pi_{\mathsf{FHE}}^{\mathsf{SOS-RMT}}$ *securely realizes SOS-RMT, tolerating an adaptive adversary who corrupts* $t < \epsilon n$ *parties for any constant* $0 \leq \epsilon < 1$*, in the* NE-NAMS *model.*

7 Communication-Local MPC

We finally turn to the question of adaptively secure MPC with polylogarithmic communication locality. In Sect. 7.1, we show that all-pairs RMT can be used to realize CL MPC, and we outline our impossibility and feasibility results for CL MPC in the NE-AMS, E-NAMS, and E-AMS models. In Sect. 7.2, we state our final feasibility result for CL SOS-MPC in the NE-NAMS model.

7.1 CL MPC in the NE-AMS, E-NAMS, and E-AMS Models

All our negative results on all-to-all RMT trivially apply to MPC, since the former is a special case. Next, we show the feasibility of CL MPC, under the combination of the feasibility bounds of all-pairs RMT proven here and the classical $t < n/2$ bound, necessary and sufficient for standard (non-CL) MPC.

The idea for the above is as follows: Execute the MPC protocol where the point-to-point communication is replaced by encrypting the message with the public key of the receiver and sending it using a fresh all-pairs RMT execution (as constructed here). As noticed in [15], to achieve adaptive security, each round of the MPC protocol will require RMT on a new hidden graph setup which, in the worst case, induces an additive polylogarithmic increase in the CL in every round. To keep the overall CL of the MPC polylogarithmic, one needs to be careful that the total number of point-to-point rounds in the MPC protocol is at most polylogarithmic. To this direction, we adopt the following solution from [15]: Invocations to the (typically round-intensive) broadcast channel are replaced by a polylogarithmic-round broadcast protocol provided in [15]. This protocol can be used within an adaptively secure constant-round MPC protocol (e.g., [2]) to get an overall polylogarithmic-round MPC protocol.

Theorem 5. *Assuming a PKI, a polylogarithmic-degree hidden graph setup*[8]*, and trapdoor permutations with a reversed domain sampler, the following feasibility and impossibility statements hold for the existence of a* store-and-forward *protocol for securely evaluating any given n-party function against an adaptive t-adversary satisfying the following two conditions with overwhelming probability:*

- **Locality.** *Every party communicates with at most* $\mathcal{O}(\log^{1+\epsilon} n)$ *other parties, for some constant* $\epsilon > 0$.
- **Rounds.** *The protocol terminates after* $\mathcal{O}(\log^{\epsilon'} n)$ *rounds, for some constant* $\epsilon' > 0$.

1. *In the NE-AMS and NE-NAMS models, i.e. if we do not assume erasures, then no such MPC exists if* $t = O(n)$ *and the protocol has an expansion rate of* $(\log^z n, \frac{k \log n}{(1+\epsilon) \log \log n})$*, for some* $k < 1$ *and* $z > 1$.
2. *In the E-NAMS and E-AMS models, i.e., if we assume erasures (with or without atomic multisend), then there exists such an MPC protocol if* $t < (\frac{1}{2} - \epsilon'')n$ *for some constant* ϵ''.
3. *In the E-NAMS model, no such MPC exists if* $t > (\frac{1}{2} + \epsilon'')n$ *for some constant* ϵ''.[9]

[8] Recall that this can be replaced by an SKI or a NIKE scheme assuming the PKI supports it.

[9] Note that Theorem 3 implies that if we assume erasures as an atomic operation and no atomic multisend, then no MPC as in the above theorem exists if $t > (\frac{1}{2} + \epsilon'')n$ for some constant ϵ''. However, this is anyway implied by the tightness of the condition $t < n/2$ for adaptive security even in the complete (i.e., non-CL) point-to-point channels setting, and is therefore omitted.

7.2 CL Sublinear-Output-Set (SOS) MPC in the NE-NAMS Model

Since we do not have all-pairs RMT in the NE-NAMS model, we propose a protocol for sublinear output-set MPC (SOS-MPC), where a sublinear (here polylogarithmic) set of parties learns the output. A high-level outline is as follows.

We first select a committee of some $\mathsf{polylog}(n)$ parties; these parties perform the actual MPC and learn the output. The first task here is to select the committee members and keep their identities hidden from the adversary, while still allowing parties to send messages to committee members. We resolve this by introducing an *anonymous* PKI setup, for all three schemes involved, namely the signature scheme, the non-committing encryption scheme, and the FHE scheme. Each party receives its secret keys, while the public keys are made known to everyone without disclosing identities. The setup also selects some $\mathsf{polylog}(n)$ parties at random to form the committee, and publishes their public keys.

Next, each party creates $\mathsf{polylog}(n)$ secret shares of his MPC input, and distributes these shares to the committee with a single instance of SOS-RMT (using $\Pi_{\mathsf{FHE}}^{\mathsf{SOS-RMT}}$). Finally, the committee members simulate an arbitrary MPC protocol to obtain the output. This can be realized by simulating each round of communication in the MPC protocol via a new instance of SOS-RMT, wherein each committee member sends the appropriate message for the MPC protocol to other committee members, while other parties just send a dummy all-zeros message to each committee member. After every RMT instance, each committee member decrypts messages received from other committee members.

A detailed description of our final protocol can be found in the full version, and the following theorem is immediate.

Theorem 6. *Assuming an anonymous PKI, a polylogarithmic-degree hidden graph setup, trapdoor permutations with a reversed domain sampler, and compact and malicious circuit-private FHE, there exists a protocol, satisfying the following two constraints with overwhelming probability:*

- **Locality.** *Every party communicates with at most* $\mathcal{O}(\log^{1+\epsilon} n)$ *other parties, for some constant* $\epsilon > 0$*, and*
- **Rounds.** *The protocol terminates after* $\mathcal{O}(\log^{\epsilon'} n)$ *rounds, for some constant* $\epsilon' > 0$*,*

which securely evaluates any given n-party function against an adaptive t-adversary corrupting up to $t < n/2$ *parties in the NE-NAMS model, and delivers the output to any* $\mathcal{O}(\log^{\epsilon''} n)$ *parties, for constant* $\epsilon'' > 0$*.*

References

1. Abraham, I., et al.: Communication complexity of byzantine agreement, revisited. In: Robinson, P., Ellen, F. (eds.) Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing, PODC 2019, Toronto, ON, Canada, 29 July–2 August 2019, pp. 317–326. ACM (2019). https://doi.org/10.1145/3293611.3331629
2. Beaver, D., Micali, S., Rogaway, P.: The round complexity of secure protocols (extended abstract), pp. 503–513 (1990). https://doi.org/10.1145/100216.100287
3. Beck, M., et al.: Logistical computing and internetworking: middleware for the use of storage in communication. In: 3rd Annual International Workshop on Active Middleware Services (AMS 2001), 6 August 2001, San Francisco, CA, USA, pp. 12–21. IEEE Computer Society (2001). https://doi.org/10.1109/AMS.2001.993716
4. Ben-Or, M., Goldwasser, S., Wigderson, A.: Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In: Simon, J. (ed.) Proceedings of the 20th Annual ACM Symposium on Theory of Computing, 2–4 May, 1988, Chicago, Illinois, USA, pp. 1–10. ACM (1988). https://doi.org/10.1145/62212.62213
5. Ben-Or, M., Goldwasser, S., Wigderson, A.: Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract), pp. 1–10 (1988). https://doi.org/10.1145/62212.62213
6. Boyle, E., Chung, K.-M., Pass, R.: Large-scale secure computation: multi-party computation for (parallel) RAM programs. In: Gennaro, R., Robshaw, M. (eds.) CRYPTO 2015. LNCS, vol. 9216, pp. 742–762. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48000-7_36
7. Boyle, E., Cohen, R., Data, D., Hubáček, P.: Must the communication graph of MPC protocols be an expander? In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018. LNCS, vol. 10993, pp. 243–272. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-96878-0_9
8. Boyle, E., Goldwasser, S., Tessaro, S.: Communication locality in secure multi-party computation. In: Sahai, A. (ed.) TCC 2013. LNCS, vol. 7785, pp. 356–376. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-36594-2_21
9. Boyle, E., Goldwasser, S., Tessaro, S.: Communication locality in secure multi-party computation. In: Sahai, A. (ed.) TCC 2013. LNCS, vol. 7785, pp. 356–376. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-36594-2_21
10. Canetti, R.: Security and composition of multiparty cryptographic protocols **13**(1), 143–202 (2000). https://doi.org/10.1007/s001459910006
11. Canetti, R.: Universally composable security: a new paradigm for cryptographic protocols, pp. 136–145 (2001). https://doi.org/10.1109/SFCS.2001.959888
12. Canetti, R., Feige, U., Goldreich, O., Naor, M.: Adaptively secure multi-party computation. In: Miller, G.L. (ed.) Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing, Philadelphia, Pennsylvania, USA, 22–24 May, 1996, pp. 639–648. ACM (1996). https://doi.org/10.1145/237814.238015
13. Canetti, R., Feige, U., Goldreich, O., Naor, M.: Adaptively secure multi-party computation, pp. 639–648 (1996). https://doi.org/10.1145/237814.238015
14. Chandran, N., Chongchitmate, W., Garay, J.A., Goldwasser, S., Ostrovsky, R., Zikas, V.: Optimally resilient and adaptively secure multi-party computation with low communication locality. IACR Cryptol. ePrint Arch, vol. 2014, p. 615 (2014). http://eprint.iacr.org/2014/615
15. Chandran, N., Chongchitmate, W., Garay, J.A., Goldwasser, S., Ostrovsky, R., Zikas, V.: The hidden graph model: communication locality and optimal resiliency with adaptive faults, pp. 153–162 (2015).https://doi.org/10.1145/2688073.2688102

16. Chandran, N., Forghani, P., Garay, J.A., Ostrovsky, R., Patel, R., Zikas, V.: Universally composable almost-everywhere secure computation. In: Dachman-Soled, D. (eds.) 3rd Conference on Information-Theoretic Cryptography, ITC 2022. LIPIcs, 5–7 July, 2022, Cambridge, MA, USA, vol. 230, pp. 14:1–14:25. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2022). https://doi.org/10.4230/LIPIcs.ITC.2022.14
17. Chandran, N., Garay, J., Ostrovsky, R.: Improved fault tolerance and secure computation on sparse networks. In: Abramsky, S., Gavoille, C., Kirchner, C., Meyer auf der Heide, F., Spirakis, P.G. (eds.) ICALP 2010. LNCS, vol. 6199, pp. 249–260. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14162-1_21
18. Chandran, N., Garay, J., Ostrovsky, R.: Edge fault tolerance on sparse networks. In: Czumaj, A., Mehlhorn, K., Pitts, A., Wattenhofer, R. (eds.) ICALP 2012. LNCS, vol. 7392, pp. 452–463. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-31585-5_41
19. Chaum, D., Crépeau, C., Damgård, I.: Multiparty unconditionally secure protocols (extended abstract). In: Simon, J. (eds.) Proceedings of the 20th Annual ACM Symposium on Theory of Computing, 2–4 May, 1988, Chicago, Illinois, USA, pp. 11–19. ACM (1988). https://doi.org/10.1145/62212.62214
20. Damgård, I., Ishai, Y.: Scalable secure multiparty computation. In: Dwork, C. (ed.) CRYPTO 2006. LNCS, vol. 4117, pp. 501–520. Springer, Heidelberg (2006). https://doi.org/10.1007/11818175_30
21. Damgård, I., Ishai, Y., Krøigaard, M.: Perfectly secure multiparty computation and the computational overhead of cryptography. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 445–465. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-13190-5_23
22. Dani, V., King, V., Movahedi, M., Saia, J.: Brief announcement: breaking the $O(nm)$ bit barrier, secure multiparty computation with a static adversary. In: ACM Symposium on Principles of Distributed Computing, PODC 2012, Funchal, Madeira, Portugal, 16–18 July 2012, pp. 227–228 (2012)
23. Dwork, C., Peleg, D., Pippenger, N., Upfal, E.: Fault tolerance in networks of bounded degree (preliminary version), pp. 370–379 (1986). https://doi.org/10.1145/12130.12169
24. Freire, E.S.V., Hofheinz, D., Kiltz, E., Paterson, K.G.: Non-interactive key exchange. In: Kurosawa, K., Hanaoka, G. (eds.) PKC 2013. LNCS, vol. 7778, pp. 254–271. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-36362-7_17
25. Garay, J., Ishai, Y., Ostrovsky, R., Zikas, V.: The price of low communication in secure multi-party computation. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017. LNCS, vol. 10401, pp. 420–446. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63688-7_14
26. Garay, J.A., Katz, J., Kumaresan, R., Zhou, H.: Adaptively secure broadcast, revisited. In: Gavoille, C., Fraigniaud, P. (eds.) Proceedings of the 30th Annual ACM Symposium on Principles of Distributed Computing, PODC 2011, San Jose, CA, USA, 6–8 June, 2011, pp. 179–186. ACM (2011). https://doi.org/10.1145/1993806.1993832
27. Garay, J.A., Ostrovsky, R.: Almost-everywhere secure computation. In: Smart, N. (ed.) EUROCRYPT 2008. LNCS, vol. 4965, pp. 307–323. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-78967-3_18

28. Garg, S., Sahai, A.: Adaptively secure multi-party computation with dishonest majority. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 105–123. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-32009-5_8
29. Gentry, C.: Fully homomorphic encryption using ideal lattices, pp. 169–178 (2009). https://doi.org/10.1145/1536414.1536440
30. Gentry, C., et al.: YOSO: you only speak once. In: Malkin, T., Peikert, C. (eds.) CRYPTO 2021. LNCS, vol. 12826, pp. 64–93. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-84245-1_3
31. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game or a completeness theorem for protocols with honest majority. In: Aho, A.V. (eds.) Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 1987, New York, New York, USA, pp. 218–229. ACM (1987). https://doi.org/10.1145/28395.28420
32. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game or a completeness theorem for protocols with honest majority, pp. 218–229 (1987). https://doi.org/10.1145/28395.28420
33. Hirt, M., Zikas, V.: Adaptively secure broadcast. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 466–485. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-13190-5_24
34. Ishai, Y., Prabhakaran, M., Sahai, A.: Founding cryptography on oblivious transfer – efficiently. In: Wagner, D. (ed.) CRYPTO 2008. LNCS, vol. 5157, pp. 572–591. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-85174-5_32
35. Katz, J., Koo, C.-Y.: On expected constant-round protocols for byzantine agreement. In: Dwork, C. (eds.) CRYPTO 2006. LNCS, vol. 4117, pp. 445–462. Springer, Heidelberg (2006). https://doi.org/10.1007/11818175_27
36. Katz, J., Maurer, U., Tackmann, B., Zikas, V.: Universally composable synchronous computation. In: Sahai, A. (ed.) TCC 2013. LNCS, vol. 7785, pp. 477–498. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-36594-2_27
37. Katz, J., Thiruvengadam, A., Zhou, H.-S.: Feasibility and infeasibility of adaptively secure fully homomorphic encryption. In: Kurosawa, K., Hanaoka, G. (eds.) PKC 2013. LNCS, vol. 7778, pp. 14–31. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-36362-7_2
38. King, V., Saia, J.: Breaking the $o(n^2)$ bit barrier: scalable byzantine agreement with an adaptive adversary. In: Proceedings of the 29th Annual ACM Symposium on Principles of Distributed Computing, PODC 2010, Zurich, Switzerland, 25–28 July 2010, pp. 420–429 (2010)
39. King, V., Saia, J., Sanwalani, V., Vee, E.: Scalable leader election. In: SODA, pp. 990–999 (2006)
40. King, V., Saia, J., Sanwalani, V., Vee, E.: Towards secure and scalable computation in peer-to-peer networks. In: FOCS, pp. 87–98 (2006)
41. Matt, C., Nielsen, J.B., Thomsen, S.E.: Formalizing delayed adaptive corruptions and the security of flooding networks. In: Dodis, Y., Shrimpton, T. (eds.) Advances in Cryptology, CRYPTO 2022. LNCS, vol. 13508, pp. 400–430. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-15979-4_14
42. Ostrovsky, R., Paskin-Cherniavsky, A., Paskin-Cherniavsky, B.: Maliciously circuit-private FHE. In: Garay, J.A., Gennaro, R. (eds.) CRYPTO 2014. LNCS, vol. 8616, pp. 536–553. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-44371-2_30

43. Smart, N.P., Vercauteren, F.: Fully homomorphic encryption with relatively small key and ciphertext sizes. In: Nguyen, P.Q., Pointcheval, D. (eds.) PKC 2010. LNCS, vol. 6056, pp. 420–443. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-13013-7_25
44. Upfal, E.: Tolerating linear number of faults in networks of bounded degree. In: PODC, pp. 83–89 (1992)
45. van Dijk, M., Gentry, C., Halevi, S., Vaikuntanathan, V.: Fully homomorphic encryption over the integers. In: Gilbert, H. (eds.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 24–43. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-13190-5_2
46. Wan, J., Xiao, H., Devadas, S., Shi, E.: Round-efficient byzantine broadcast under strongly adaptive and majority corruptions. In: Pass, R., Pietrzak, K. (eds.) TCC 2020. LNCS, vol. 12550, pp. 412–456. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-64375-1_15
47. Yao, A.C.C.: Protocols for secure computations (extended abstract), pp. 160–164 (1982). https://doi.org/10.1109/SFCS.1982.38

Information-Theoretic Cryptography

Perfectly-Secure MPC with Constant Online Communication Complexity

Yifan Song[1,2(✉)] and Xiaxi Ye[1]

[1] Tsinghua University, Beijing, China
yfsong@mail.tsinghua.edu.cn, yexx23@mails.tsinghua.edu.cn
[2] Shanghai Qi Zhi Institute, Shanghai, China

Abstract. In this work, we study the communication complexity of perfectly secure MPC protocol with guaranteed output delivery against $t = (n-1)/3$ corruptions. The previously best-known result in this setting is due to Goyal, Liu, and Song (CRYPTO, 2019) which achieves $O(n)$ communication per gate, where n is the number of parties.

On the other hand, in the honest majority setting, a recent trend in designing efficient MPC protocol is to rely on packed Shamir sharings to speed up the online phase. In particular, the work by Escudero et al. (CCS 2022) gives the first semi-honest protocol that achieves a constant communication overhead per gate across all parties in the online phase while maintaining overall $O(n)$ communication per gate. We thus ask the following question: "Is it possible to construct a perfectly secure MPC protocol with GOD such that the online communication per gate is $O(1)$ while maintaining overall $O(n)$ communication per gate?"

In this work, we give an affirmative answer by providing an MPC protocol for computing an arithmetic circuit C over a finite field of size at least $2n$ with communication complexity $O(|C| + \mathsf{Depth} \cdot n + n^5 \cdot \log n)$ elements for the online phase, and $O(|C| \cdot n + \mathsf{Depth} \cdot n^2 + n^5 \cdot \log n)$ elements for the preprocessing phase, where $|C|$ is the circuit size and Depth is the circuit depth.

1 Introduction

Secure multiparty computation (MPC) allows a set of mutually distrustful parties to jointly compute a common function on their private inputs. Very informally, the protocol guarantees that each party can only learn his own input and output but nothing else. Since the notion of MPC was introduced by Yao [Yao82], early feasibility results on MPC were obtained by Yao [Yao82] and Goldreich et al. [GMW87] in the computational setting, where the adversary is assumed to have bounded computational resources. Subsequent works [BGW88, CCD88] considered the unconditional (or information-theoretic) setting and showed positive results up to $t < n/3$ corrupted parties assuming point-to-point communication channels. If one assumes a broadcast channel in addition, it was shown in [RB89, Bea89] how to obtain positive results in the information-theoretic setting for up to $t < n/2$ corrupted parties.

E. Boyle and M. Mahmoody (Eds.): TCC 2024, LNCS 15367, pp. 329–361, 2025.
https://doi.org/10.1007/978-3-031-78023-3_11

In this work, we are interested in the communication complexity of perfectly secure MPC with guaranteed output delivery (GOD) over point-to-point channels. At a high level, perfect security requires that any (computationally unbounded) adversary by controlling t corrupted parties cannot learn any information about honest parties' inputs even if corrupted parties can arbitrarily deviate from the protocol. Guaranteed output delivery, on the other hand, requires that the protocol should always succeed. Indeed, perfect security with GOD is the best possible security one can hope. It has been shown in [BGW88] that perfectly secure MPC with GOD is impossible to achieve when $t \geq n/3$. On the other hand, when $t < n/3$, [BGW88] gives the first positive result that can compute any computable functions.

Communication complexity is an important measurement of the efficiency of an MPC protocol, especially in the information-theoretic setting. This is because unconditionally secure MPC protocols usually have light weight local computation, often just a series of linear operations. On the other hand, known constructions for general functions still require at least a linear communication complexity in the circuit size. Thus in the real world, the efficiency of an unconditional MPC protocol is dominated by its communication complexity.

There is a rich line of works studying the communication complexity of perfectly secure MPC with GOD. Most noticeably, the work [BH08] gives the first result with linear communication complexity per multiplication gate. To be more concrete, the achieved communication complexity is $O(|C|\cdot n+\mathsf{Depth}\cdot n^2+n^3)$ elements. In 2019, Goyal et al. [GLS19] show how to remove the quadratic communication overhead in the circuit depth and achieve $O(|C|\cdot n+n^3)$ elements. Both of these works are based on the party elimination framework [HMP00], a generic approach to achieve GOD efficiently. On the other hand, this approach inherently requires $O(n)$ evaluation rounds. Therefore, the round complexity of these two works are $O(\mathsf{Depth}+n)$. Another line of works [ALR11, AAY22, AAPP23] focuses on improving the communication complexity without the $O(n)$ overhead in the round complexity. And in the recent work [AAPP23], a linear communication overhead per gate in the number of parties is also achieved in this setting. Concretely the achieved communication complexity is $O(|C|\cdot n+\mathsf{Depth}\cdot n^2+n^4)$. It is not clear whether a linear communication overhead per gate is inherent in general in perfect malicious security setting despite there are negative results for special cases with an additional requirement that the protocol be two-phase [DS20].[1]

We note that in the honest majority setting where $t < n/2$, the communication complexity of the best-known semi-honest protocol [DN07, GLO+21] is also linear in the number of parties per gate. However, a recent work [EGPS22] gives a novel construction that achieves $O(1)$ communication per gate in the online phase while preserving $O(n)$ overall communication per gate in the preprocessing phase. Having $O(1)$ communication per gate in the online phase is interesting since

[1] The negative result in [DS20] requires that protocols are UC secure and have a standard 2-phase structure where the inputs are committed in the first phase, before the output is computed in the second phase.

– In practice, people care more about the efficiency in the online phase as the preprocessing phase can be done in the idle time before knowing the inputs.
– It means that the amortized communication complexity per party decreases as the increase of the number of parties and one may speed up the protocol by having more parties![2]

Unfortunately, to the best of our knowledge, such a result is not known in the perfect security setting. Thus, we ask the following question:

"Is it possible to construct a perfectly secure MPC protocol with GOD such that the online communication complexity per gate is $O(1)$ while the overall communication remains $O(n)$?"

1.1 Our Contribution

In this work, we answer the above question affirmatively. Our main result is summarized in the following theorem.

Theorem 1. *Let n denote the number of parties. Let $\mathbb{F}$ be a finite field of size $|\mathbb{F}| \geq 2n$. For an arithmetic circuit C over $\mathbb{F}$, there exists an information-theoretic MPC protocol that computes C against a fully malicious adversary controlling at most $t = \frac{n-1}{3}$ corrupted parties with perfect security. The communication cost of the protocol is expected $O(|C| + \mathsf{Depth} \cdot n + n^5 \cdot \log n)$ elements for the online phase and expected $O(|C| \cdot n + \mathsf{Depth} \cdot n^2 + n^5 \cdot \log n)$ elements for the preprocessing phase, where Depth is the circuit depth. The round complexity of the protocol is $O(\mathsf{Depth} + n^2)$ in expectation for the online phase and $O(n^2)$ in expectation for the preprocessing phase.*

We note that the previously best-known results [GLS19, AAPP23] both require linear communication complexity per gate in the online phase. Thus, our result gives a factor of $O(n)$ improvements over the previous results in the online phase.

To achieve our result, our idea is to compile the semi-honest protocol in [EGPS22] to achieve perfect security. While the semi-honest protocol in [EGPS22] gives us an efficient way to compute the circuit in the online phase, we note that the main difficulty of achieving perfect security is to efficiently verify the computation and locate the errors. In particular, we identify a security issue that does not occur in the semi-honest setting when compiling [EGPS22] to achieve perfect security. We note that this security issue is very similar to the one observed in [GLS19]. Compared with [GLS19], we give a much simpler solution to address this issue which can potentially be used to improve the construction in [GLS19]. In Sect. 2, we give an overview of our solution towards tackling these difficulties.

[2] Our construction requires the field size to be $O(n)$. We leave the extension of our result to constant-size fields to future work.

Comparison with Previous Works. As we have mentioned above, our work achieves the same overall asymptotic communication complexity as [GLS19, AAPP23] while we achieve constant online communication per gate.

On the other hand,

- Both of our work and [AAPP23] suffer an additive communication overhead depending on the circuit depth, $O(\mathsf{Depth} \cdot n^2)$, while [GLS19] does not have this term.
- For round complexity, when using an expected constant round BC protocol, our protocol requires $O(\mathsf{Depth} + n^2)$ rounds in expectation due to the use of the dispute control framework. This is in contrast to [AAPP23] that achieves expected $O(\mathsf{Depth})$ rounds and [GLS19] that achieves expected $O(\mathsf{Depth} + n)$ rounds.

This additional communication overhead depending on the circuit depth and additional round complexity may be viewed as the tradeoff of achieving constant online communication per gate. An interesting future direction is to achieve the best among these three works.

When focusing on the non-optimal corruption setting, [DIK10] achieves sublinear communication and computation in the number of parties. As for its techniques, [DIK10] first transforms a general circuit into a SIMD-like circuit to overcome the issue of network routing where the input wires for each group of gates need to be aligned. However, this transformation increases the circuit size by a factor of $\log |C|$. To evaluate the circuit after transformation, [DIK10] designs a protocol for a much smaller corruption threshold and use the party virtualization technique to boost the corruption threshold. However, the technique of party virtualization only works in the non-optimal corruption setting which does not work for the optimal 1/3 corruption setting we target for. We note that if only focusing on the communication complexity, one can potentially achieve overall $O(|C|)$ communication by adapting the protocol in [GPS21] that addresses network routing with constant overhead and combining it with our verification technique. However, one main challenge in our case is that, to achieve $O(|C|)$ online communication, we have to use packed Shamir sharing with larger degree $d(> n/3)$ to pack $O(n)$ secrets which leads to the loss of error correction property. In contrast, both [DIK10] and the above adaption of [GPS21] can use Shamir sharings with degree $d = n/3-1$ while still packing $O(n)$ secrets and take advantage of the error correction property to achieve a much simpler protocol.

2 Technical Overview

Our work uses both the standard Shamir sharings and Packed Shamir sharings. We use $[x]_d$ to represent a degree-d Shamir sharing of x, which corresponds to a degree-d polynomial f such that the i-th share is $f(i)$, and $f(0) = x$. We will also use $[x|j]_d$ to denote a degree-d Shamir sharing of x where the secret value is stored at $f(-j+1)$ rather than $f(0)$. For a vector $\boldsymbol{x} = (x_1, \ldots, x_k) \in \mathbb{F}^k$, we use $[\boldsymbol{x}]_d$ to represent a degree-d packed Shamir sharing of $\boldsymbol{x}$, which corresponds

to a degree-d polynomial f such that the i-th share is $f(i)$ and for all $j \in [k]$, $f(-j+1) = x_j$.

2.1 Efficient Online Protocol via Preprocessing

In this work, our goal is to design a perfectly secure MPC protocol with guaranteed output delivery (GOD) against a fully malicious adversary who controls up to $t = \frac{n-1}{3}$ corrupted parties such that the online communication complexity per multiplication gate is constant among all parties. Our starting point is the recent semi-honest protocol [EGPS22] in the honest majority setting that achieves constant communication complexity in the online phase relying on the packed Shamir sharing scheme and preprocessing.

The notion of packed Shamir sharing was introduced by Franklin and Yung in [FY92]. At a high level, packed Shamir sharings allow us to compute a batch of $O(n)$ gates of the same kind simultaneously but only at cost $O(n)$, the same as using the Shamir sharings to compute a single gate. This allows us to bring the cost per gate to $O(1)$. In [EGPS22], the authors rely on preprocessing to prepare packed Beaver triples (the packed version of the standard Beaver triples) which are used in the online phase to achieve constant online communication. On the other hand, the overall communication complexity of the construction in [EGPS22] remains $O(n)$ per gate, which matches the best-known semi-honest protocol [DN07, GLO+21].

Inspired by [EGPS22], our idea is to rely on packed Shamir sharings to achieve perfect security with GOD with constant online communication per gate. At a very high level, our idea is to (1) use techniques in [BH08] to compile the preprocessing phase of [EGPS22] to prepare packed Beaver triples, and (2) use party elimination framework [HMP00] to compile the online protocol that uses packed Beaver triples. However, the second step is much more difficult to achieve than it looks. In the following, we give an overview of our construction and demonstrate the technical difficulties we have to address. For simplicity, we first focus on a SIMD circuit, which computes many copies of the same sub-circuit. We will discuss how to move to a general circuit later.

Overview of Our Protocol. Let k be the number of secrets packed in a single sharing. We will discuss the choice of k at a later point. We set $d = t + k - 1$ and use degree-d packed Shamir sharings to ensure privacy against t corrupted parties. A packed Beaver triple contains three degree-d packed Shamir sharings $([\boldsymbol{a}]_d, [\boldsymbol{b}]_d, [\boldsymbol{c}]_d)$ such that $\boldsymbol{c} = \boldsymbol{a} * \boldsymbol{b}$, where $*$ denotes the coordinate-wise multiplication.

In the preprocessing phase of [EGPS22], a random packed Beaver triple is prepared as follows.

- First, for all $i \in [k]$, a standard Beaver triple with secrets stored at position $-i+1$ is prepared: $([a_i|i]_t, [b_i|i]_t, [c_i|i]_t)$.
- Then, all parties locally convert such a group of k Beaver triples to a packed Beaver triple. This is done by computing $[\boldsymbol{a}]_d = \sum_{i=1}^{k} [\boldsymbol{e}_i]_{k-1} \cdot [a_i|i]_t$, where

$\boldsymbol{e}_i$ is the i-th unit vector (i.e., the i-th entry is 1 while all other entries are 0). To see why it works, just note that the secrets of $[\boldsymbol{e}_i]_{k-1} \cdot [a_i|i]_t$ is equal to $a_i \cdot \boldsymbol{e}_i$. Thus, we have $\boldsymbol{a} = \sum_{i=1}^{k} a_i \cdot \boldsymbol{e}_i$.

Using techniques in [BH08], we can prepare standard Beaver triples with perfect security. Thus, following [EGPS22], we can efficiently prepare packed Beaver triples with perfect security.

With packed Beaver triples in hand, in the online phase, all parties evaluate a group of k gates in parallel each time. For a group of k addition gates with input sharings $[\boldsymbol{x}]_d, [\boldsymbol{y}]_d$, all parties locally compute $[\boldsymbol{z}]_d = [\boldsymbol{x}]_d + [\boldsymbol{y}]_d$. For a group of k multiplication gates, all parties run the following steps:

1. All parties locally compute $[\boldsymbol{x} + \boldsymbol{a}]_d = [\boldsymbol{x}]_d + [\boldsymbol{a}]_d$ and $[\boldsymbol{y} + \boldsymbol{b}]_d = [\boldsymbol{y}]_d + [\boldsymbol{b}]_d$ and send them to a common party P_{king}.
2. P_{king} reconstructs $\boldsymbol{x} + \boldsymbol{a}, \boldsymbol{y} + \boldsymbol{b}$, and distributes $[\boldsymbol{x} + \boldsymbol{a}]_{k-1}, [\boldsymbol{y} + \boldsymbol{b}]_{k-1}$ to all parties.
3. All parties locally compute

$$[\boldsymbol{z}]_{d+k-1} = [\boldsymbol{x} + \boldsymbol{a}]_{k-1} \cdot [\boldsymbol{y} + \boldsymbol{b}]_{k-1} - [\boldsymbol{x} + \boldsymbol{a}]_{k-1} \cdot [\boldsymbol{b}]_d - [\boldsymbol{a}]_d \cdot [\boldsymbol{y} + \boldsymbol{b}]_{k-1} + [\boldsymbol{c}]_d.$$

   To obtain a degree-d packed Shamir sharing of $\boldsymbol{z}$, all parties in addition prepare $([\boldsymbol{r}]_d, [\boldsymbol{r}]_{d+k-1})$. Such a pair of random sharings can also be prepared by using techniques in [BH08].
4. All parties locally compute $[\boldsymbol{z} + \boldsymbol{r}]_{d+k-1} = [\boldsymbol{z}]_{d+k-1} + [\boldsymbol{r}]_{d+k-1}$ and send it to P_{king}.
5. P_{king} reconstructs $\boldsymbol{z} + \boldsymbol{r}$ and distributes $[\boldsymbol{z} + \boldsymbol{r}]_{k-1}$ to all parties.
6. All parties locally compute $[\boldsymbol{z}]_d = [\boldsymbol{z} + \boldsymbol{r}]_{k-1} - [\boldsymbol{r}]_d$.

In this way, all parties can evaluate every group of gates with constant communication.

When corrupted parties deviate from the protocol, however, the above protocol can easily go wrong. This is because the packed Shamir sharings are of degree d which is greater than t, and we cannot hope to use the error correction of Reed Solomon Code as [BH08] to ensure the correctness of the computation in the online phase. To achieve perfect security, one may hope to use the party elimination framework as [GLS19]. At a high level, the whole computation task is first divided into $O(n)$ segments. Each time one segment is computed as above. Then all parties together check the correctness of the computation. If the computation is correct, all parties move to the next segment. Otherwise, all parties together find a pair of dispute parties which ensures at least one party is corrupted. Such a dispute pair is removed and all parties re-evaluate the current segment. This way, we can achieve perfect security without blowing up the communication complexity.

Unfortunately, this idea does not work in our case. This is because in the worst case, the party elimination framework may remove $2t$ parties and only $t + 1$ parties left. However, in the above multiplication protocol, to allow P_{king} to reconstruct $[\boldsymbol{z}]_{d+k-1}$, at least $d + k = t + 2k - 1 > t + 1$ parties are needed.

To resolve this issue, our construction uses the dispute control method [BH06]. The high-level idea of dispute control is similar to party elimination framework except that each time a dispute pair is found, we will not remove these two parties. Instead, we will ensure that these two parties never talk to each other in the following computation. In this way, we could avoid finding the same dispute pair in the future evaluation. A party is only removed if he is disputed with more than t parties, in which case this party is definitely a corrupted party. By using dispute control, we will only remove corrupted parties. Thus, at least $2t+1$ (honest) parties are active. To ensure that P_{king} can always receive enough shares for $[\boldsymbol{z}]_{d+k-1}$, which requires $d+k = t+2k-1$ shares, we need $t+2k-1 \leq 2t+1$. Thus, our construction sets $k = \frac{t}{2}+1 = O(n)$, which is sufficient to achieve our goal.

Note that so far we haven't discussed how to check the correctness of the computation and how to find a dispute pair if the computation goes wrong. In particular,

- The verification of computation should be done with constant communication per gate as well. Note that this difficulty does not appear in [GLS19] since their construction only achieves linear communication per gate in the online phase. So it suffices for them to also pay linear cost per gate in the verification. This difficulty does not appear in [EGPS22] either since their malicious version is in the honest majority setting where one cannot hope to achieve perfect security. However, allowing errors with negligible probability simplifies the task of verification as one can even achieve sublinear cost in the verification [GSZ20].
- Degree-d packed Shamir sharings are insufficient to identify dispute pairs when the computation goes wrong. This is unlike degree-t Shamir sharings where even if all corrupted parties provide incorrect shares, we can always reconstruct the whole sharing. In fact, this is a much more severe issue since if the input packed Shamir sharings are found to be incorrect, then even if all parties follow the protocol, the computation will always fail and we are not able to find a dispute pair.

In the following sections, we will address these two difficulties.

2.2 Boosting Verification

As we discussed above, we have to design a verification protocol with constant communication per gate. We first examine where the multiplication protocol may go wrong.

- **Issue 1**. In Step 1 and 4, parties may send incorrect shares to P_{king} so that P_{king} cannot reconstruct the secrets of degree-d or degree-$(d+k-1)$ packed Shamir sharings.
- **Issue 2**. In Step 2 and 5, P_{king} may maliciously distribute incorrect packed sharings where either the degree of packed sharings is not $k-1$ or the secrets are incorrect.

In the first case, we show that P_{king} can detect such issues and directly announce to others that the computation is incorrect. Recall that we set k, the number of secrets packed in a single sharing, to be $\frac{t}{2}+1$. Then $(d+k-1)+1 = t+2k-1 = 2t+1$. Therefore, a degree-$d$ or degree-$(d+k-1)$ packed Shamir sharing is fully determined by the shares of honest parties and corrupted parties can only cause the sharing to be inconsistent but not change the secret. Thus, P_{king} can detect errors by checking whether the received shares lie on a valid degree-d or degree-$(d+k-1)$ polynomial.

In the second case, we can abstract the verification task as follows. All parties hold a pair of packed Shamir sharings $([\boldsymbol{u}]_{d+k-1}, [\boldsymbol{u}]_{k-1})$, where the first sharing is the one all parties send to P_{king} and the second sharing is the one all parties receive from P_{king} (note that we can always view a degree-d packed Shamir sharing as a degree-$(d+k-1)$ packed Shamir sharing). The goal is to check that (1) the second sharing is a valid degree-$(k-1)$ packed Shamir sharing, and (2) both sharings have the same secrets. To verify such a pair, we let each party receive the shares from all parties and check the above two points accordingly. Here we rely again on the fact that a degree-$(d+k-1)$ packed Shamir sharing is fully determined by the shares of honest parties. Thus corrupted parties cannot make honest parties accept the verification by sending wrong shares.

However this way of checking $([\boldsymbol{u}]_{d+k-1}, [\boldsymbol{u}]_{k-1})$ would require $O(n^2)$ communication per pair. To amortize the communication complexity, we adapt the technique in [BH08] to efficiently check a batch of $2t+1$ such pairs, say $\{([\boldsymbol{u}_i]_{d+k-1}, [\boldsymbol{u}_i]_{k-1})\}_{i=1}^{2t+1}$. Let $\boldsymbol{M}$ be a Vandermonde matrix of size $n \times (2t+1)$. All parties first expand such $2t+1$ pairs to n pairs by locally computing

$$([\boldsymbol{v}_i]_{d+k-1}, [\boldsymbol{v}_i]_{k-1})_{i=1}^{n} = \boldsymbol{M} \cdot ([\boldsymbol{u}_i]_{d+k-1}, [\boldsymbol{u}_i]_{k-1})_{i=1}^{2t+1}.$$

By the property of Vandermonde matrices, there is a one-to-one linear map between any subset of $2t+1$ pairs in $\{([\boldsymbol{v}_i]_{d+k-1}, [\boldsymbol{v}_i]_{k-1})\}_{i=1}^{n}$ and $\{([\boldsymbol{u}_i]_{d+k-1}, [\boldsymbol{u}_i]_{k-1})\}_{i=1}^{2t+1}$. Thus if $2t+1$ pairs in $\{([\boldsymbol{v}_i]_{d+k-1}, [\boldsymbol{v}_i]_{k-1})\}_{i=1}^{n}$ are correct, this implies that $\{([\boldsymbol{u}_i]_{d+k-1}, [\boldsymbol{u}_i]_{k-1})\}_{i=1}^{2t+1}$ are all correct. Therefore, after expansion, we let each party P_i check a single pair $([\boldsymbol{v}_i]_{d+k-1}, [\boldsymbol{v}_i]_{k-1})$. If every party is happy with the pair he checked, then at least $2t+1$ pairs are verified by honest parties, which ensures that the original $2t+1$ pairs are correct. Finally, all parties run a Byzantine Agreement protocol to reach an agreement on whether the verification passes or not. Note that the communication remains to be $O(n^2)$ but we check $O(n)$ pairs each time.

We extend the above idea to check any linear secret sharing scheme which satisfies that the whole sharing is determined by the shares of honest parties. The functionality $\mathcal{F}_{\mathsf{VerifyPub}}$ will be formally described in Subsect. 4.1 and the protocol $\Pi_{\mathsf{VerifyPub}}$ instantiating $\mathcal{F}_{\mathsf{VerifyPub}}$ appears in the full version of this paper [SY24] (Subsect. 5.1).

2.3 Identifying Dispute Pair

After the verification, if the check fails, we reach a scenario where

- Either P_{king} claims that the degree-d or degree-$(d+k-1)$ packed Shamir sharing he received is incorrect.
- Or some party P_i claims that $([\boldsymbol{v}_i]_{d+k-1}, [\boldsymbol{v}_i]_{k-1})$ he received is incorrect.

Then we know the computation of the current segment fails and we have to identify a dispute pair of parties. For the latter case, since $([\boldsymbol{v}_i]_{d+k-1}, [\boldsymbol{v}_i]_{k-1})$ is computed from $\{([\boldsymbol{u}_i]_{d+k-1}, [\boldsymbol{u}_i]_{k-1})\}_{i=1}^{2t+1}$ which are known by P_{king}, P_{king} can provide a correct version to P_i so that P_i can cross check which party deviates from the protocol. However, what if P_{king} claims that a degree-d or degree-$(d+k-1)$ packed Shamir sharing is incorrect? Since $d > t$, we cannot hope to always reconstruct the correct sharing and identify the party who sends the wrong share.

To enable identification of dispute pair, we have to resort to degree-t Shamir sharings. Our idea is to compute a degree-t Shamir sharing for every value and all parties hold these degree-t Shamir sharings silently. In this way, whenever a degree-d or degree-$(d+k-1)$ packed Shamir sharing is incorrect, we can always come back to the degree-t sharings to identify a dispute pair. However, we have to achieve this with constant online communication.

Computing Degree-t Sharings with Constant Online Communication. Our observation is that in the preprocessing phase of [EGPS22], all parties can obtain a degree-d packed Shamir sharing $[\boldsymbol{a}]_d$ via local computation from $\{[a_i|i]_t\}_{i=1}^k$. This inspires us to compute degree-t Shamir sharings in the online phase with a similar form so that when we compute multiplication gates, we can pack the degree-t Shamir sharings on demand. To be more concrete, for every group of k multiplication gates, all parties hold $\{[x_i|i]_t, [y_i|i]_t\}_{i=1}^k$ in the beginning. They first locally transform these $2k$ degree-t Shamir sharings to $[\boldsymbol{x}]_d, [\boldsymbol{y}]_d$ and then run the multiplication protocol.

Note that we also have to unpack the output sharing $[\boldsymbol{z}]_d$ to $\{[z_i|i]_t\}_{i=1}^k$. Fortunately, this can be achieved with a small modification of the preprocessing data: We require all parties to also prepare $\{[r_i|i]_t\}_{i=1}^k$ when preparing $([\boldsymbol{r}]_d, [\boldsymbol{r}]_{d+k-1})$. Then after receiving $[\boldsymbol{z}+\boldsymbol{r}]_{k-1}$ from P_{king}, since $[\boldsymbol{z}+\boldsymbol{r}]_{k-1}$ can be viewed as a degree-$(k-1)$ Shamir sharing of $z_i + r_i$ stored at $-i+1$, i.e., $[z_i + r_i|i]_{k-1}$, all parties can compute $[z_i|i]_t = [\boldsymbol{z}+\boldsymbol{r}]_{k-1} - [r_i|i]_t$.

Identifying Dispute Pair When $[\boldsymbol{x}+\boldsymbol{a}]_d$ is Incorrect. Now come back to the problem of identifying dispute pair. If P_{king} claims that $[\boldsymbol{x}+\boldsymbol{a}]_d$ is incorrect, all parties can provide $\{[x_i+a_i|i]_t\}_{i=1}^k$ (note that $[a_i|i]_t$ is generated when preparing packed Beaver triples in the preprocessing phase). In this way, P_{king} can robustly reconstruct each degree-t Shamir sharing and compute the correct $[\boldsymbol{x}+\boldsymbol{a}]_d$.

Identifying Dispute Pair When $[\boldsymbol{z}+\boldsymbol{r}]_{d+k-1}$ is Incorrect. If P_{king} claims that $[\boldsymbol{z}+\boldsymbol{r}]_{d+k-1}$ is incorrect, however, the above approach does not work. Recall that

$$[\boldsymbol{z}+\boldsymbol{r}]_{d+k-1} = [\boldsymbol{x}+\boldsymbol{a}]_{k-1} \cdot [\boldsymbol{y}+\boldsymbol{b}]_{k-1} - [\boldsymbol{x}+\boldsymbol{a}]_{k-1} \cdot [\boldsymbol{b}]_d - [\boldsymbol{a}]_d \cdot [\boldsymbol{y}+\boldsymbol{b}]_{k-1} + [\boldsymbol{c}]_d + [\boldsymbol{r}]_{d+k-1},$$

where $[\boldsymbol{x}+\boldsymbol{a}]_{k-1}$ and $[\boldsymbol{y}+\boldsymbol{b}]_{k-1}$ are distributed by P_{king}. Also recall that all parties have prepared $\{([a_i|i]_t, [b_i|i]_t, [c_i|i]_t)\}_{i=1}^k$ in the preprocessing phase. To

allow P_{king} robustly reconstructing $[\boldsymbol{z}+\boldsymbol{r}]_{d+k-1}$, we further change the way of preparing $[\boldsymbol{r}]_{d+k-1}$ as follows: All parties prepare $2k-1$ random degree-t Shamir sharings $\{[r_i|i]_t\}_{i=1}^{2k-1}$. Let $\boldsymbol{r}' = (r_1, \ldots, r_{2k-1})$. Following the observation in [EGPS22], all parties can locally transform $\{[r_i|i]_t\}_{i=1}^{2k-1}$ to a degree-$(d+k-1)$ packed Shamir sharing $[\boldsymbol{r}']_{d+k-1}$ (that stores $2k-1$ secrets). Here we utilize the fact that $d+k-1 = t+(2k-1)-1$ by our choices of d and k. Note that if we only focus on the first k secrets of $\boldsymbol{r}'$, denoted by $\boldsymbol{r} = (r_1, \ldots, r_k)$, $[\boldsymbol{r}']_{t+2k-2}$ can be directly viewed as a degree-$(d+k-1)$ packed Shamir sharing $[\boldsymbol{r}]_{d+k-1}$ (that stores k secrets), which is what we need.

Now if all parties send $\{([a_i|i]_t, [b_i|i]_t, [c_i|i]_t)\}_{i=1}^{k}$ and $\{[r_i|i]_t\}_{i=1}^{2k-1}$ to P_{king}, P_{king} can reconstruct each degree-t Shamir sharing and compute a correct degree-$(d+k-1)$ packed Shamir sharing $[\boldsymbol{z}+\boldsymbol{r}]_{d+k-1}$, and therefore can identify a dispute pair. However, doing this would reveal $\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}$ to P_{king}. To resolve this issue, we use the following tricks.

1. In the preprocessing phase, all parties prepare random degree-t Shamir sharings $\{([a'_i|i]_t, [b'_i|i]_t, [c'_i|i]_t)\}_{i=1}^{k}$ and $\{[r'_i|i]_t\}_{i=1}^{2k-1}$ as random masks. Note that each c'_i is a random value rather than $a'_i \cdot b'_i$.
2. All parties compute $[\boldsymbol{a}']_d, [\boldsymbol{b}']_d, [\boldsymbol{c}']_d, [\boldsymbol{r}']_{2k-1}$ locally. Then all parties send $[\boldsymbol{v}]_{d+k-1} = [\boldsymbol{x}+\boldsymbol{a}]_{k-1} \cdot [\boldsymbol{y}+\boldsymbol{b}]_{k-1} - [\boldsymbol{x}+\boldsymbol{a}]_{k-1} \cdot [\boldsymbol{b}']_d - [\boldsymbol{a}']_d \cdot [\boldsymbol{y}+\boldsymbol{b}]_{k-1} + [\boldsymbol{c}']_d + [\boldsymbol{r}']_{d+k-1}$ to P_{king}. Basically, $[\boldsymbol{v}]_{d+k-1}$ is computed in the same way as $[\boldsymbol{z}+\boldsymbol{r}]_{d+k-1}$ except that we use $[\boldsymbol{a}']_d, [\boldsymbol{b}']_d, [\boldsymbol{c}']_d, [\boldsymbol{r}']_{d+k-1}$.
3. Now if P_{king} complains about $[\boldsymbol{v}]_{d+k-1}$, all parties send $\{([a'_i|i]_t, [b'_i|i]_t, [c'_i|i]_t)\}_{i=1}^{k}$ and $\{[r'_i|i]_t\}_{i=1}^{2k-1}$ to P_{king}. These are independent of the actual wire values.
4. Otherwise, it means that $[\boldsymbol{z}+\boldsymbol{r}]_{d+k-1}+[\boldsymbol{v}]_{d+k-1}$ is not a valid degree-$(d+k-1)$ packed Shamir sharing. Note that we have

$$\begin{aligned}
&[\boldsymbol{z}+\boldsymbol{r}]_{d+k-1} + [\boldsymbol{v}]_{d+k-1} \\
&= 2 \cdot [\boldsymbol{x}+\boldsymbol{a}]_{k-1} \cdot [\boldsymbol{y}+\boldsymbol{b}]_{k-1} - [\boldsymbol{x}+\boldsymbol{a}]_{k-1} \cdot ([\boldsymbol{b}]_d + [\boldsymbol{b}']_d) \\
&\quad -([\boldsymbol{a}]_d + [\boldsymbol{a}']_d) \cdot [\boldsymbol{y}+\boldsymbol{b}]_{k-1} + ([\boldsymbol{c}]_d + [\boldsymbol{c}']_d) + ([\boldsymbol{r}]_{d+k-1} + [\boldsymbol{r}']_{d+k-1}).
\end{aligned}$$

   All parties send $\{([a_i+a'_i|i]_t, [b_i+b'_i|i]_t, [c_i+c'_i|i]_t)\}_{i=1}^{k}$ and $\{[r_i+r'_i|i]_t\}_{i=1}^{2k-1}$ to P_{king}. Again those sharings are independent of the actual wire values because of the random masks.

In this way, P_{king} can identify a party who sends incorrect shares.

2.4 Security Issue of the Current Approach

One issue we omitted so far is that the multiplication above is actually not secure against malicious corrupted parties: A malicious P_{king} can learn some partial information of $\boldsymbol{y}$ from the shares of $[\boldsymbol{z}+\boldsymbol{r}]_{d+k-1}$ by sending incorrect $[\boldsymbol{x}+\boldsymbol{a}]_{k-1}$ and $[\boldsymbol{y}+\boldsymbol{b}]_{k-1}$. Consider the following so-called double-dipping attack [GBO+23, DEN24] which has been pointed out in [GLS19]:

1. Without loss of generality, assume the first $2t+1$ parties are honest. After P_{king} reconstructs $\boldsymbol{e} = \boldsymbol{x} + \boldsymbol{a}$, P_{king} chooses another vector $\boldsymbol{e}'$ such that the first shares of $[\boldsymbol{e}]_{k-1}$ and $[\boldsymbol{e}']_{k-1}$ are identical. Then P_{king} sends the shares of $[\boldsymbol{e}]_{k-1}$ to $\{P_1, \ldots, P_{t+1}\}$, and sends the shares of $[\boldsymbol{e}']_{k-1}$ to $\{P_{t+2}, \ldots, P_{2t+1}\}$.
2. Let $[\boldsymbol{z}+\boldsymbol{r}]_{d+k-1}$ denote the correct sharings computed from $[\boldsymbol{e}]_{k-1}$ and $[\boldsymbol{z}'+\boldsymbol{r}]_{d+k-1}$ denote the incorrect sharings computed from $[\boldsymbol{e}']_{k-1}$. Then the parties in $\{P_1, \ldots, P_{t+1}\}$ hold shares of $[\boldsymbol{z}+\boldsymbol{r}]_{d+k-1}$ while parties in $\{P_1, P_{t+2}, \ldots, P_{2t+1}\}$ hold shares of $[\boldsymbol{z}'+\boldsymbol{r}]_{d+k-1}$. Note that corrupted parties, i.e., $\{P_{2t+2}, \ldots, P_{3t+1}\}$, can compute their shares of both $[\boldsymbol{z}+\boldsymbol{r}]_{d+k-1}$ and $[\boldsymbol{z}'+\boldsymbol{r}]_{d+k-1}$.
3. After receiving the shares of $[\boldsymbol{z}+\boldsymbol{r}]_{d+k-1}$ (or $[\boldsymbol{z}'+\boldsymbol{r}]_{d+k-1}$) from all honest parties, P_{king} learns $2t+1$ shares of both $[\boldsymbol{z}+\boldsymbol{r}]_{d+k-1}$ and $[\boldsymbol{z}'+\boldsymbol{r}]_{d+k-1}$. Thus P_{king} can reconstruct $\boldsymbol{z}+\boldsymbol{r}$ and $\boldsymbol{z}'+\boldsymbol{r}$ and compute $\boldsymbol{z}'-\boldsymbol{z} = (\boldsymbol{e}'-\boldsymbol{e}) * \boldsymbol{y}$, which leaks the information about $\boldsymbol{y}$.

We point out that such an issue does not appear in [EGPS22] since in their setting, they set $d+k-1 = n-1$ so that P_{king} needs all shares to reconstruct $\boldsymbol{z}+\boldsymbol{r}$. In our case, however, we need $d+k-1 \leq 2t$ to ensure that P_{king} can detect the errors in $[\boldsymbol{z}+\boldsymbol{r}]_{d+k-1}$.

We note that a similar issue has been pointed out in [GLS19]. We follow the approach in [GLS19] to resolve this issue. Before sending $[\boldsymbol{z}+\boldsymbol{r}]_{d+k-1}$ to P_{king}, all parties prepare a random degree-$(n-1)$ packed Shamir sharing of $\mathbf{0}$, denoted by $[\mathbf{0}]_{n-1}$, and add it with $[\boldsymbol{z}+\boldsymbol{r}]_{d+k-1}$. In this way, even if a malicious P_{king} may distribute incorrect $[\boldsymbol{x}+\boldsymbol{a}]_{k-1}$ and $[\boldsymbol{y}+\boldsymbol{b}]_{k-1}$, he no longer gains any information from shares of $[\boldsymbol{z}+\boldsymbol{r}]_{n-1}$. Intuitively, this is because $[\boldsymbol{z}+\boldsymbol{r}]_{n-1} = [\boldsymbol{z}]_{d+k-1} + ([\boldsymbol{r}]_{d+k-1} + [\mathbf{0}]_{n-1})$ where the second part is a random degree-$(n-1)$ packed Shamir sharing and every share is just a random value. Effectively, every party uses a uniform value to hide his share of $[\boldsymbol{z}]_{d+k-1}$.

While this approach prevents a corrupted P_{king} from gaining information from $[\boldsymbol{z}+\boldsymbol{r}]_{n-1}$, an honest P_{king} cannot detect errors in $[\boldsymbol{z}+\boldsymbol{r}]_{n-1}$ either. In [GLS19], the authors introduce the so called 4-consistent sharings to detect errors in $[\boldsymbol{z}+\boldsymbol{r}]_{n-1}$. In our work, we give a much simpler approach for this task, which can also be used to simplify the construction in [GLS19] and avoid the use of 4-consistent sharings.

We notice that the key point of the above attack is that P_{king} may distribute $[\boldsymbol{x}+\boldsymbol{a}]_{k-1}$ and $[\boldsymbol{y}+\boldsymbol{b}]_{k-1}$ that are not of degree $(k-1)$. On the other hand, if P_{king} is guaranteed to distribute degree-$(k-1)$ packed Shamir sharings, even if the secrets are incorrect, P_{king} cannot learn any information from $[\boldsymbol{z}+\boldsymbol{r}]_{d+k-1}$ since in this case it is a valid degree-$(d+k-1)$ packed Shamir sharing and the potentially incorrect secrets $\boldsymbol{z}$ are masked by $\boldsymbol{r}$. Thus, to allow P_{king} to detect errors in $[\boldsymbol{z}+\boldsymbol{r}]_{n-1}$,

1. All parties first check that P_{king} indeed shares degree-$(k-1)$ packed Shamir sharings. This check can be done by the verification protocol $\Pi_{\mathsf{VerifyPub}}$ we briefly sketched in Subsect. 2.2.
2. All parties together open the mask sharing $[\mathbf{0}]_{n-1}$. This is done by letting each party send all messages when generating $[\mathbf{0}]_{n-1}$ to P_{king}. Note that given

that P_{king} shares valid degree-$(k-1)$ packed Shamir sharings, $[\mathbf{0}]_{n-1}$ is safe to open. Now P_{king} can detect errors in $[\boldsymbol{z}+\boldsymbol{r}]_{d+k-1} = [\boldsymbol{z}+\boldsymbol{r}]_{n-1} - [\mathbf{0}]_{n-1}$ again.

In our construction, we will also use a random packed Shamir sharing of $\mathbf{0}$ when reconstructing $[\boldsymbol{x}+\boldsymbol{a}]_d$ and $[\boldsymbol{y}+\boldsymbol{b}]_d$ to P_{king}. In this way, we can first evaluate multiplication gates in multiple layers without check, and then verify the correctness of multiplication gates at the end. We point out that without doing this, a similar security issue occurs when evaluating multiplication gates in multiple layers without check [GLS19].

To summarize, we use techniques in [GLS19] to allow computation across layers and only verify the computation at the end of each segment. To achieve this, we add a degree-$(n-1)$ packed Shamir sharings of $\mathbf{0}$ to remove the redundancy of the sharings reconstructed to P_{king} in order to prevent P_{king} from applying the double-dipping attack by distributing inconsistent reconstruction results to all parties. However, when the computation fails, P_{king} cannot pin-point which party misbehaved due to the lack of redundancy. Our observation is that the high-degree packed Shamir sharings of $\mathbf{0}$ is to prevent P_{king} from applying the double-dipping attack and once this has been checked (i.e., P_{king} indeed sends consistent reconstruction results to all parties), it is safe for all parties to de-mask the random sharings of $\mathbf{0}$. Hence, P_{king} could again rely on the redundancy of lower degree Shamir sharings for verification.

Using Our Technique in [GLS19]. The construction in [GLS19] uses the standard degree-t Shamir sharing and random Beaver triples for online computation. To compute the multiplication of two input sharings $[x]_t, [y]_t$ with Beaver triple $([a]_t, [b]_t, [c]_t)$, all parties first reconstruct $x+a$ and $y+b$ by sending their shares of $[x+a]_t, [y+b]_t$ to P_{king} and then asking P_{king} to distribute back the reconstruction results. To prevent the above mentioned double-dipping attack, $[x+a]_t$ and $[y+b]_t$ are masked by random degree-$(n-1)$ Shamir sharings of 0, denoted by $[o_1]_{n-1}$ and $[o_2]_{n-2}$. However these random masks also prevent P_{king} from detecting and error-correcting incorrect shares when reconstructing $x+a$ and $y+b$. As a result, even if a single incorrect share may cause the whole computation incorrect.

To solve this, after checking that P_{king} does not play the double-dipping attack, all parties transform $[x+a]_{n-1} := [x+a]_t + [o_1]_{n-1}$ to a tuple of 4 degree-t Shamir sharings such that each share of $[x+a]_{n-1}$ can be robustly recovered from one of the 4 newly generated degree-t Shamir sharings. This technique is known as the 4-consistent sharings [GLS19].

We note that following our approach, the above problem can be easily addressed by asking all parties send all messages when generating $[o_1]_{n-1}$ to P_{king}. In this way, P_{king} can de-mask $[x+a]_{n-1}$ and obtain $[x+a]_t$, which allows him to find incorrect shares using error correction.

2.5 Towards General Circuits

When evaluating a general circuit, one issue is to prepare the packed Shamir sharings for the next group of multiplication gates. We note that the packed

Shamir sharings only allow us to do coordinate-wise multiplications. It requires that the secrets of the two input sharings are correctly aligned. This holds automatically for SIMD circuits. However, when dealing with a general circuit, the secrets may not be in the order we want. Or even worse, the secrets may not be in a single packed Shamir sharing. This problem is referred to as *network routing* [GPS21, GPS22].

In [GPS21, GPS22], the authors reduce the problem of network routing to the following sharing transformation problem. Suppose all parties hold a packed Shamir sharing $[\boldsymbol{x}]_d$ and want to perform a linear map $L : \mathbb{F}^k \to \mathbb{F}^k$ to the secrets $\boldsymbol{x}$. I.e., the goal is to compute a packed Shamir sharing $[L(\boldsymbol{x})]_d$. Following [GPS21], this task can be achieved as follows.

1. All parties prepare $([\boldsymbol{r}]_d, [L(\boldsymbol{r})]_d)$.
2. All parties compute $[\boldsymbol{x} + \boldsymbol{r}]_d$ and send their shares to P_{king}.
3. P_{king} reconstructs $\boldsymbol{x} + \boldsymbol{r}$ and distributes $[L(\boldsymbol{x} + \boldsymbol{r})]_{k-1}$.
4. All parties locally compute $[L(\boldsymbol{x})]_d = [L(\boldsymbol{x} + \boldsymbol{r})]_{k-1} - [\boldsymbol{r}]_d$.

There are two difficulties we have to address. First, how should parties prepare $([\boldsymbol{r}]_d, [L(\boldsymbol{r})]_d)$ efficiently? Indeed this is the main technical difficulty resolved in [GPS21, GPS22]. This task is not simple because each time we may need to perform a different linear map. Known solutions from [DN07, BH08] only allow us to prepare such random sharings for the same linear map many times. In [GPS22], the authors show how to perform any linear transformation efficiently (where the underlying secret sharing scheme can be any linear secret sharing scheme). Instead of using the general linear transformation, we give an efficient solution towards this task that is tailored for our case.

Second, how should parties check the correctness of P_{king}? We note that the verification functionality $\mathcal{F}_{\mathsf{VerifyPub}}$ only allows us to check the same linear maps many times, which is not sufficient since each time we may have to perform a different linear map. We design an efficient solution to resolve this issue which will be introduced later.

Efficient Preprocessing for Sharing Transformation. Our idea is to prepare random sharings $\{([\boldsymbol{r}_i]_d, [L_i(\boldsymbol{r}_i)]_d)\}_{i=1}^k$ for k different linear maps in a batch way, where recall that k is the number of secrets packed in a single sharing. Let $\boldsymbol{u}_i = L_i(\boldsymbol{r}_i)$. Then for all $j \in [k]$, $u_{i,j}$ is a linear combination of $r_{i,1}, \ldots, r_{i,k}$.

In the beginning, all parties prepare k random degree-d packed Shamir sharings $\{[\boldsymbol{r}_i]_d\}_{i=1}^k$. We may list the secrets in a matrix as follows:

$$\begin{bmatrix} r_{1,1} & \cdots & r_{1,k} \\ \vdots & \ddots & \vdots \\ r_{k,1} & \cdots & r_{k,k} \end{bmatrix}$$

Then the secrets in the same row are stored in a single packed Shamir sharing.

We observe that the packed Shamir sharings support efficient linear operations over secrets that are stored in the same positions, i.e., we can efficiently

compute any linear combination of $r_{1,i}, \ldots, r_{k,i}$, which are stored at position $-i+1$. However, the sharing transformation requires us to do linear operations over secrets that are all stored in different positions. Thus, a natural idea is to re-share the matrix so that the secrets in the *same column* are stored in a single packed Shamir sharing, i.e., $\{[\boldsymbol{r}_{*,j}]_d\}_{j=1}^k$. This can be viewed as a "transpose" operation. Now all parties could efficiently compute $\{[\boldsymbol{u}_{*,j}]_d\}_{j=1}^k$. This is because the i-th secret of $[\boldsymbol{u}_{*,j}]_d$, which is $u_{i,j}$, is a linear combination of $r_{i,1}, \ldots, r_{i,k}$, which are the i-th secrets of $\{[\boldsymbol{r}_{*,j}]_d\}_{j=1}^k$. To obtain $\{[\boldsymbol{u}_i]_d\}_{i=1}^k$, we simply perform another "transpose" operation on $\{[\boldsymbol{u}_{*,j}]_d\}_{j=1}^k$.

We note that the "transpose" operation can also be viewed as one type of sharing transformation and the only difference is that it acts on the secrets of k packed Shamir sharings. Since we have to perform the same "transpose" operation many times, this can be handled efficiently by solutions from [DN07, BH08].

Efficient Verification of P_{king}. We may abstract the verification task as follows. Given $([\boldsymbol{u}]_d, [\boldsymbol{v}]_{k-1})$ and a linear map L, we want to check that $[\boldsymbol{v}]_{k-1}$ is a valid degree-$(k-1)$ packed Shamir sharing and $\boldsymbol{v} = L(\boldsymbol{u})$.

To achieve efficient verification, we follow a similar approach to that when preparing random sharings for sharing transformation. This becomes even simpler since P_{king} knows all sharings and he can help all parties perform the "transpose" operations. We sketch our solution as follows:

1. For each group of k pairs $\{([\boldsymbol{u}_i]_d, [\boldsymbol{v}_i]_{k-1}), L_i\}_{i=1}^k$, P_{king} shares the "transpose" sharings $\{[\boldsymbol{u}_{*,j}]_{k-1}\}_{j=1}^k$, $\{[\boldsymbol{v}_{*,j}]_{k-1}\}_{j=1}^k$.
2. All parties use $\Pi_{\mathsf{VerifyPub}}$ to check the correctness of all "transpose" operations. Note that these are just the same operations, which can be handled by our verification protocol.
3. All parties compute from $\{[\boldsymbol{u}_{*,j}]_{k-1}\}_{j=1}^k$ to $\{[\boldsymbol{v}_{*,j}]_{k-1}\}_{j=1}^k$ and check whether the secrets of the resulting sharings are the same as those shared by P_{king}. Again the last check can be handled by our verification protocol as well.

2.6 Summary of Our Construction

Putting all components together, we obtain a perfectly secure MPC protocol with constant online communication. In the preprocessing phase, we prepare correlated random degree-t Shamir sharings and Beaver triples. We show how to use the techniques in [BH08] to achieve this task. The communication complexity in the preprocessing phase is $O(n)$ elements per gate.

In the online phase, we follow the dispute control method and divide the circuit into $O(n^2)$ segments of equal size. Since there are at most $O(n^2)$ different dispute pairs, in the worst case, we may need to re-evaluate $O(n^2)$ segments. By having $O(n^2)$ segments, even in the worst case, the asymptotic communication complexity remains unchanged.

In each segment, we first evaluate the circuit by using packed Shamir sharings without check. This includes (1) performing proper linear transformations to prepare input packed Shamir sharings for each layer, and (2) using packed Shamir

sharings to evaluate each group of gates efficiently. Note that to protect against a malicious P_{king}, every sharing that is reconstructed to P_{king} is masked by a degree-$(n-1)$ packed Shamir sharing of $\mathbf{0}$.

After evaluation, all parties together verify whether P_{king} distributes valid degree-$(k-1)$ packed Shamir sharings. After this check, all parties can reveal the mask sharing $[\mathbf{0}]_{n-1}$. Now P_{king} checks whether the shares he received are valid degree-d or degree-$(d+k-1)$ sharings, and all parties check whether P_{king} honestly follows the protocol. If all check passes, all parties proceed to the next segment. Otherwise, all parties rely on the degree-t Shamir sharings prepared in the preprocessing phase to identify a dispute pair. Then the whole segment is re-evaluated.

As a conclusion, we have the following theorem.

Theorem 1. *Let n denote the number of parties. Let $\mathbb{F}$ be a finite field of size $|\mathbb{F}| \geq 2n$. For an arithmetic circuit C over $\mathbb{F}$, there exists an information-theoretic MPC protocol that computes C against a fully malicious adversary controlling at most $t = \frac{n-1}{3}$ corrupted parties with perfect security. The communication cost of the protocol is expected $O(|C| + \mathsf{Depth} \cdot n + n^5 \cdot \log n)$ elements for the online phase and expected $O(|C| \cdot n + \mathsf{Depth} \cdot n^2 + n^5 \cdot \log n)$ elements for the preprocessing phase, where Depth is the circuit depth. The round complexity of the protocol is $O(\mathsf{Depth} + n^2)$ in expectation for the online phase and $O(n^2)$ in expectation for the preprocessing phase.*

3 Preliminary

3.1 The Model

We consider a set of n parties $\{P_1, P_2, \ldots, P_n\}$ where each party can provide inputs, receive outputs, and participate in the computation. For every pair of parties, there exists a secure (private and authenticated) synchronous channel so that they can directly send messages to each other. The communication complexity is measured by the number of bits via private channels.

We focus on functions which can be represented as arithmetic circuits over a finite field $\mathbb{F}$ (with $|\mathbb{F}| \geq 2n$) with input, addition, multiplication, and output gates. Let $\kappa = \log |\mathbb{F}|$ be the size of an element in $\mathbb{F}$.

In this work, we consider the standard simulation-based definition of MPC [Can00]. An adversary is able to corrupt at most $t = \frac{n-1}{3}$ parties, provide inputs to corrupted parties, and receive all messages sent to the corrupted parties. Corrupted parties can deviate from the protocol arbitrarily. We denote the set of corrupted parties by $\mathcal{C}$. We consider perfect security with guaranteed output delivery. That is the protocol is guaranteed to succeed with no error.

3.2 Byzantine Agreement

Our MPC protocol uses Byzantine agreement for both broadcast and consensus. Broadcast channels are authenticated and synchronous which allow a sender to

distribute a message with a guarantee that all parties will receive the same value. Consensus allows the parties who hold an individual input x_i, to reach agreement on a value x' such that $x' = x$ if every honest party holds $x_i = x$. Focusing on perfect security with corruption threshold $t < \frac{n}{3}$, to instantiate Byzantine agreement and broadcast channels, we use an expected constant round broadcast protocol proposed in [AC24] which achieves communication complexity of $O(n \cdot L)$ bits plus expected $O(n^3 \cdot \log^2 n)$ bits for broadcasting a message of size L bits.

We denote the communication complexity of a protocol by $\mathsf{P2P}(M) + N_1 \times \mathsf{BA}(L_1) + N_2 \times \mathsf{BC}(L_2)$, which means the protocol costs M bits in total over the point-to-point private channels, calls the Byzantine agreement N_1 times to reach an agreement on a message of L_1 bits, and calls the broadcast channel N_2 times with a message of L_2 bits.

3.3 Packed Shamir Secret Sharing

We use the packed secret sharing technique introduced by Franklin and Yung [FY92]. This is a generalization of the standard Shamir secret sharing scheme [Sha79]. Let $\mathbb{F}$ be a finite field of size $|\mathbb{F}| \geq 2n$. Let n be the number of parties and k be the number of secrets that are packed in one sharing. A degree-d ($d \geq k-1$) packed Shamir sharing with secret $\boldsymbol{x} = (x_1, \ldots, x_k) \in \mathbb{F}^k$ is a vector $(w_1, \ldots, w_n)$ for which there exists a polynomial $f(\cdot) \in \mathbb{F}[X]$ of degree at most d such that $f(-i+1) = x_i$ for all $i \in [k]$, and $f(i) = w_i$ for all $i \in \{1, 2, \ldots, n\}$. The i-th share w_i is held by party P_i. Reconstructing a degree-d packed Shamir sharing requires $d+1$ shares and can be done by Lagrange interpolation. For a random degree-d packed Shamir sharing of $\boldsymbol{x}$, any $d-k+1$ shares are independent of the secret $\boldsymbol{x}$.

In our work, we use $[\boldsymbol{x}]_d$ to denote a degree-d packed Shamir sharing of $\boldsymbol{x} \in \mathbb{F}^k$. In the following, operations (addition and multiplication) between two packed Shamir sharings are coordinate-wise. We recall two properties of the packed Shamir sharing scheme:

- Linear Homomorphism: For all $d \geq k-1$ and $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{F}^k$, $[\boldsymbol{x}+\boldsymbol{y}]_d = [\boldsymbol{x}]_d + [\boldsymbol{y}]_d$.
- Multiplicativity: Let $*$ denote the coordinate-wise multiplication operation. For all $d_1, d_2 \geq k-1$ subject to $d_1+d_2 < n$, and for all $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{F}^k$, $[\boldsymbol{x}*\boldsymbol{y}]_{d_1+d_2} = [\boldsymbol{x}]_{d_1} \cdot [\boldsymbol{y}]_{d_2}$.[3]

These two properties directly follow from computing the underlying polynomials.

Note that the second property implies that, for all $\boldsymbol{x}, \boldsymbol{c} \in \mathbb{F}^k$, all parties can locally compute $[\boldsymbol{c} * \boldsymbol{x}]_{d+k-1}$ from $[\boldsymbol{x}]_d$ and the public vector $\boldsymbol{c}$. To see this, all parties can locally transform $\boldsymbol{c}$ to degree-$(k-1)$ packed Shamir sharing $[\boldsymbol{c}]_{k-1}$. Then they can use the property of the packed Shamir sharing scheme to compute $[\boldsymbol{c}*\boldsymbol{x}]_{d+k-1} = [\boldsymbol{c}]_{k-1} \cdot [\boldsymbol{x}]_d$. This property is referred to as multiplication-friendliness in [GPS22].

[3] We remark that the resulting sharing $[\boldsymbol{x} * \boldsymbol{y}]_{d_1+d_2}$ is not a random degree-(d_1+d_2) secret sharing with the secret $\boldsymbol{x} * \boldsymbol{y}$.

Recall that t is the number of corrupted parties. Also recall that a degree-d packed Shamir secret sharing scheme is secure against $d-k+1$ corrupted parties. In our work, we set $k = (t+2)/2 = (n+5)/6$ and $d = t+k-1$. As we have discussed in Sect. 2.1, during the computation, all parties need to reconstruct degree-$(d+k-1)$ packed Shamir sharings. Using the above choices of k and d, we have $(d+k-1)+1 = t+2k-1 = 2t+1$, which ensures that even if all corrupted parties have been removed when using dispute control (introduced below), we still have enough shares to reconstruct degree-$(d+k-1)$ packed Shamir sharings.

Standard Shamir Secret Sharing. When $k = 1$, $[x]_d$ is a standard degree-d Shamir sharing of $x \in \mathbb{F}$. In our work, for all $i \in \{1, \ldots, n\}$, we use $[x|i]_d$ to denote a degree-d Shamir sharing with secret stored at position $-i+1$ rather than 0.

3.4 The Generalization of Party Elimination: Dispute Control

Our work uses the dispute control technique introduced in [BH06]. The crux for dispute control is to divide the whole computation into several segments and do it segment by segment. As for the current segment, we first evaluate it and then detect efficiently whether the evaluation is correct. In the case of success, all parties move to the next segment. In the case of failure, all parties run another protocol to localize a pair of two dispute parties containing at least one corrupted party. Then the current segment will be evaluated again. To avoid the dispute pairs that have been detected to disrupt the execution again, before re-evaluating the current segment, we find an intermediate party P_r which is not disputed with both P_i and P_j to help pass messages between P_i and P_j for each dispute pair (P_i, P_j) who have been already detected and are both active. As a result, the number of failures is bounded by $O(n^2)$ since each failure leads to finding a new dispute pair.

Hence, dividing uniformly the whole circuit into n^2 segments, each of size $|C|/n^2$, since there are totally at most n^2 failures, the whole communication complexity at most doubles. We refer the readers to the full version of this paper [SY24] for more details regarding the dispute control technique.

We denote the set of currently active parties by $\mathcal{P}$ and the set of recorded dispute pairs of parties by $\mathcal{D}$. We use n' to denote the size of $\mathcal{P}$ and t' to denote the number of corrupted parties in $\mathcal{P}$. Then each time a corrupted party is eliminated, it results in $n' := n'-1, t' := t'-1$.

3.5 Enabling Preprocessing

In this part, we briefly summarize the functionalities that will be used in our main construction. We refer the readers to the full version of this paper [SY24] for the descriptions of these functionalities and the realizations using techniques in [BH08].

Preparing Random Degree-t Shamir Sharings. Our work needs to prepare the following kinds of correlated random degree-t Shamir sharings in the preprocessing phase.

- A random degree-t Shamir sharing $[r|i]_t$. We use $\Sigma_{1,i}$ to denote this kind of random sharings.
- A pair of random degree-t Shamir sharings $([r|i]_t, [r|j]_t)$ of the same random value. We use $\Sigma_{2,i,j}$ to denote this kind of random sharings.

For all $\Sigma \in \{\Sigma_{1,i}, \Sigma_{2,i,j}\}_{i,j=1}^{n}$, $\mathcal{F}_{\mathsf{RandSh}}(\Sigma)$ prepares N random Σ-sharings. Using techniques in [BH08], the communication complexity for N random Σ-sharings is $\mathsf{P2P}(O(N \cdot n + n^4))$ elements plus $O(n^2) \times \mathsf{BA}(O(1))$ bits, which results in total communication of $\mathsf{P2P}(O(N \cdot n + n^5 \cdot \log n))$ elements in expectation.

Preparing Random Beaver Triples. To prepare random packed Beaver triples, we also make use of the functionality $\mathcal{F}_{\mathsf{Triples}}(i)$ that generates N random degree-t Beaver triples with secrets stored at position $-i+1$. Using techniques in [BH08], the communication complexity for N degree-t Beaver triples is $\mathsf{P2P}(O(N \cdot n + n^4))$ elements plus $O(n^2) \times \mathsf{BA}(O(1))$ bits, which results in total communication of $\mathsf{P2P}(O(N \cdot n + n^5 \cdot \log n))$ elements in expectation.

4 Circuit Evaluation

Recall that, we use n to denote the number of parties, n' to denote the number of active parties and t' to denote the number of active corrupted parties. Also recall the corruption threshold $t = (n-1)/3$ and the packing parameter $k = (t+2)/2 = (n+5)/6$.

4.1 Useful Building Block for Verification

Let Σ be a linear secret sharing scheme such that a Σ sharing is fully determined by the shares of honest parties. We consider the scenario where a party P_{king} distributes N Σ-sharings, denoted by $U_1, \ldots, U_N$ to all parties (via a relay if P_{king} and P_i are disputed). These N Σ-sharings need not to be private. All parties want to check whether they hold valid Σ-sharings. We assume that an honest P_{king} always distribute valid Σ-sharings to all parties.

We introduce a functionality $\mathcal{F}_{\mathsf{VerifyPub}}$ to accomplish this task. $\mathcal{F}_{\mathsf{VerifyPub}}$ either outputs `accept` to all parties, indicating that all (honest) parties hold valid Σ-sharings, or outputs a new dispute pair that contains at least one corrupted party. To instantiate $\mathcal{F}_{\mathsf{VerifyPub}}$, we follow the high-level idea in Subsect. 2.2 which utilizes the property of Vandermonde matrix to expand $(2t+1)$ sharings to be checked to n sharings with each verified by a different party. We refer the readers to the full version of this paper [SY24] (Subsect. 5.1) in which we describe the protocol $\Pi_{\mathsf{VerifyPub}}$ and show that $\Pi_{\mathsf{VerifyPub}}$ realizes $\mathcal{F}_{\mathsf{VerifyPub}}$ with communication complexity $\mathsf{P2P}(O((N \cdot n + n^2) \cdot |\Sigma| + n^3 \cdot \log n))$ elements in expectation if the check passes, where $|\Sigma|$ is the share size.

Functionality 1 : $\mathcal{F}_{\mathsf{VerifyPub}}(\Sigma)$

1. $\mathcal{F}_{\mathsf{VerifyPub}}$ receives the set $\mathcal{C}$ of corrupted parties' identities and the set $\mathcal{D}$ of disputed pairs.
2. $\mathcal{F}_{\mathsf{VerifyPub}}$ receives honest parties' shares of $U_1, \ldots, U_N$ from the honest parties and sends them to the adversary.
3. $\mathcal{F}_{\mathsf{VerifyPub}}$ checks if each group of shares form a valid Σ-sharing.
   - If all the checks are passed, $\mathcal{F}_{\mathsf{VerifyPub}}$ receives an instruction from the adversary:
     - If $\mathcal{F}_{\mathsf{VerifyPub}}$ receives `reject` from the adversary, it further receives a new pair of dispute parties containing at least one corrupted party and outputs `reject` together with the receiving pair to all parties.
     - If $\mathcal{F}_{\mathsf{VerifyPub}}$ receives `accept` from the adversary, it outputs `accept` to all parties.
   - If one of the checks fails, $\mathcal{F}_{\mathsf{VerifyPub}}$ receives a new pair of dispute parties containing at least one corrupted party and outputs `reject` together with the receiving pair to all parties.

4.2 Evaluating Multiplication Gates

To evaluate a group of k multiplication gates with input sharings $\{[x_i|i]_t, [y_i|i]_t\}_{i=1}^k$, we follow the technique of packed Beaver triples together with degree reduction.

In the preprocessing phase, all parties prepare random Beaver triples $\{[a_i|i]_t, [b_i|i]_t, [c_i|i]_t\}_{i=1}^k$ such that $c_i = a_i \cdot b_i$ for all $i \in [k]$, and random sharings $\{[r_i|i]_t\}_{i=1}^{2k-1}$. Before the evaluation of each segment, all parties also together prepare random degree-$(n'-1)$ packed Shamir sharings of $\mathbf{0} \in \mathbb{F}^k$ by invoking $\Pi_{\mathsf{zeroSharing}}$. The protocol $\Pi_{\mathsf{zeroSharing}}$ shown in the full version of this paper [SY24] allows all parties to prepare $O(n)$ degree-$(n'-1)$ packed Shamir sharings of $\mathbf{0}$ with communication cost $\mathsf{P2P}(O(n^2))$ elements. We give the description of Π_{Mult} below. The communication complexity per batch of k multiplication gates is $\mathsf{P2P}(O(n))$ elements.

Protocol 1 : Π_{Mult}

For a batch of k multiplication gates, all parties hold input sharings $\{[x_i|i]_t, [y_i|i]_t\}_{i=1}^k$. In addition, all parties hold the following sharings prepared in the preprocessing phase.

- A group of Beaver triples $\{[a_i|i]_t, [b_i|i]_t, [c_i|i]_t\}_{i=1}^k$;
- A group of random degree-t Shamir sharings $\{[r_i|i]_t\}_{i=1}^{2k-1}$;

All parties also hold the following sharings prepared right before evaluating the current segment: three degree-$(n'-1)$ packed Shamir sharings

$[\mathbf{0}_1]_{n'-1}, [\mathbf{0}_2]_{n'-1}, [\mathbf{0}_3]_{n'-1}$. All parties run the following steps to compute $\{[z_i|i]_t\}_{i=1}^k$.

1. All parties locally compute $[\boldsymbol{x}]_{t+k-1}, [\boldsymbol{y}]_{t+k-1}, [\boldsymbol{a}]_{t+k-1}, [\boldsymbol{b}]_{t+k-1}, [\boldsymbol{c}]_{t+k-1}$ from $\{[x_i|i]_t, [y_i|i]_t\}_{i=1}^k$ and $\{[a_i|i]_t, [b_i|i]_t, [c_i|i]_t\}_{i=1}^k$.
2. All parties locally compute $[\boldsymbol{x}+\boldsymbol{a}]_{n'-1} = [\boldsymbol{x}]_{t+k-1} + [\boldsymbol{a}]_{t+k-1} + [\mathbf{0}_1]_{n'-1}$ and $[\boldsymbol{y}+\boldsymbol{b}]_{n'-1} = [\boldsymbol{y}]_{t+k-1} + [\boldsymbol{b}]_{t+k-1} + [\mathbf{0}_2]_{n'-1}$, and send their shares to P_{king}.
3. P_{king} reconstructs $\boldsymbol{x}+\boldsymbol{a}$ and $\boldsymbol{y}+\boldsymbol{b}$, distributes $[\boldsymbol{x}+\boldsymbol{a}]_{k-1}, [\boldsymbol{y}+\boldsymbol{b}]_{k-1}$ to all parties.
4. All parties locally compute

$$[\boldsymbol{z}]_{t+2k-2} = [\boldsymbol{x}+\boldsymbol{a}]_{k-1} \cdot [\boldsymbol{y}+\boldsymbol{b}]_{k-1} - [\boldsymbol{x}+\boldsymbol{a}]_{k-1} \cdot [\boldsymbol{b}]_{t+k-1}$$

$$-[\boldsymbol{a}]_{t+k-1} \cdot [\boldsymbol{y}+\boldsymbol{b}]_{k-1} + [\boldsymbol{c}]_{t+k-1}$$

   and a random degree-$(t+2k-2)$ packed Shamir sharing $[\boldsymbol{r}]_{t+2k-2}$ from $\{[r_i|i]_t\}_{i=1}^{2k-1}$.
   Finally, all parties locally compute $[\boldsymbol{z}+\boldsymbol{r}]_{n'-1} = [\boldsymbol{z}]_{t+2k-2} + [\boldsymbol{r}]_{t+2k-2} + [\mathbf{0}_3]_{n'-1}$ and send their shares to P_{king}.
5. P_{king} reconstructs $\boldsymbol{z}+\boldsymbol{r}$ and distributes $[\boldsymbol{z}+\boldsymbol{r}]_{k-1}$ to all parties.
6. All parties locally compute $[z_i|i]_t = [\boldsymbol{z}+\boldsymbol{r}]_{k-1} - [r_i|i]_t$ for all $i \in [k]$.

4.3 Handling Sharing Transformations

In this section, we show how to efficiently perform sharing transformations. This will be an important component to evaluate a general circuit using packed Shamir sharings. The task of sharing transformation is to transform a degree-$(t+k-1)$ packed Shamir sharing $[\boldsymbol{x}]_{t+k-1}$ to $[L(\boldsymbol{x})]_{t+k-1}$, where $L : \mathbb{F}^k \to \mathbb{F}^k$ is a linear map.

Preparing Correlated Random Sharings for Linear Maps. As we discussed in Subsect. 2.5, the sharing transformation is done by first preparing a pair of random sharings $([\boldsymbol{r}]_{t+k-1}, [L(\boldsymbol{r})]_{t+k-1})$.

We first introduce a protocol $\Pi_{\mathsf{Transpose}}$ that allows all parties to transform $\{[x_{i,j}|j]_t\}_{i,j=1}^k$ to $\{[x_{j,i}|j]\}_{i,j=1}^k$. The communication complexity of $\Pi_{\mathsf{Transpose}}$ is $\mathsf{P2P}(O(n^2))$ elements before running the consensus in step 6 in the case of success. Note that all parties run the consensus on their `happy-bit`s only once after they finish all the executions of the protocol $\Pi_{\mathsf{Transpose}}$ to perform the 'transpose' operations in each segment.

Protocol 2 : $\Pi_{\mathsf{Transpose}}$ in $\mathcal{F}_{\mathsf{VerifyPub}}$-hybrid model

All parties hold input sharings $\{[x_{i,j}|j]_t\}_{i,j=1}^k$. All parties in addition hold the following random sharings prepared in the preprocessing phase: $\{[r_{i,j}|j]_t, [r_{j,i}|j]_t\}_{i,j=1}^k$. All parties run the following steps to compute $\{[x_{j,i}|j]_t\}_{i,j=1}^k$.

1. For all $i \in [k]$, all parties locally compute $[\boldsymbol{x}_i]_{t+k-1}$ and $[\boldsymbol{r}_i]_{t+k-1}$ from $\{[x_{i,j}|j]_t\}_{j=1}^k$ and $\{[r_{i,j}|j]_t\}_{j=1}^k$.
2. For all $i \in [k]$, all parties locally compute $[\boldsymbol{x}_i + \boldsymbol{r}_i]_{t+k-1} = [\boldsymbol{x}_i]_{t+k-1} + [\boldsymbol{r}_i]_{t+k-1}$ and send their shares to P_{king}.
3. P_{king} checks whether all received degree-$(t+k-1)$ packed Shamir sharings are valid. If not, supposing the i-th sharing is inconsistent, P_{king} broadcasts a complain to all parties. Then all parties send their shares of $\{[x_{i,j} + r_{i,j}|j]_t\}_{j=1}^k$ to P_{king}.
   P_{king} reconstructs $(x_{i,j} + r_{i,j})_{j=1}^k$ and computes $[(x_{i,j} + r_{i,j})_{j=1}^k]_{t+k-1}$. Then P_{king} identifies the cheating party P_i and broadcasts (i, x, x'), where P_i should have sent x to P_{king}, but P_{king} receives $x' \neq x$. Then P_i and the relay between P_i and P_{king} broadcast the messages they believe. Set two adjacent parties broadcasting differently to be the dispute pair.
4. Otherwise, P_{king} reconstructs $(x_{i,j} + r_{i,j})_{i,j=1}^k$. For all $i \in [k]$, P_{king} distributes $[\boldsymbol{x}_{*,i} + \boldsymbol{r}_{*,i}]_{k-1}$ to all parties.
5. For all $i, j \in [k]$, all parties locally compute $[x_{j,i}|j]_t = [\boldsymbol{x}_{*,i} + \boldsymbol{r}_{*,i}]_{k-1} - [r_{j,i}|j]_t$.
6. All parties invoke $\mathcal{F}_{\mathsf{VerifyPub}}$ to check whether P_{king} correctly distributes valid degree-$(k-1)$ packed Shamir sharings. If not, all parties take a new dispute pair as output. Otherwise, all parties take $\{[x_{j,i}|j]_t\}_{i,j=1}^k$ as output.

We will use $\Pi_{\mathsf{Transpose}}$ to prepare correlated random sharings for sharing transformations in $\Pi_{\mathsf{PrepTrans}}$. The communication complexity of $\Pi_{\mathsf{PrepTrans}}$ is $\mathsf{P2P}(O(N \cdot n + n^3))$ elements plus $O(1) \times \mathsf{BA}(O(1))$ bits accounting for total communication of expected $\mathsf{P2P}(O(N \cdot n + n^3 \cdot \log n))$ elements to prepare random sharings for N linear transformations if succeeds.

Protocol 3 : $\Pi_{\mathsf{PrepTrans}}$ in $\mathcal{F}_{\mathsf{VerifyPub}}$-hybrid model

All parties together hold N linear maps $L_1, \ldots, L_N$ and the goal is to prepare $\{[r_{i,j}|j]_t, [L_{i,j}(\boldsymbol{r}_i)|j]_t\}_{j=1}^k$ for all $i \in [N]$. Here $L_{i,j}(\cdot)$ denote the linear function that outputs the j-th value of $L_i(\cdot)$.
For each group of k linear maps, say $L_1, \ldots, L_k$, in the beginning, all parties hold the following random sharings prepared in the preprocessing phase: $\{[r_{i,j}|j]_t, [r_{j,i}|j]_t\}_{i,j=1}^k$. All parties do the following.

1. All parties locally compute

$$[L_{j,i}(\boldsymbol{r}_j)|j]_t = L_{j,i}([r_{j,1}|j]_t, \ldots, [r_{j,k}|j]_t), \forall i, j \in [k].$$

2. All parties invoke $\Pi_{\mathsf{Transpose}}$ on $\{[L_{j,i}(\boldsymbol{r}_j)|j]_t\}_{i,j=1}^k$ to obtain shares of $\{[L_{i,j}(\boldsymbol{r}_i)|j]_t\}_{i,j=1}^k$.

Finally, if all parties succeed in $\Pi_{\mathsf{Transpose}}$, all parties invoke $\mathcal{F}_{\mathsf{VerifyPub}}$ to check whether P_{king} correctly performs the transpose operation in $\Pi_{\mathsf{Transpose}}$. This is done as follows. In $\Pi_{\mathsf{Transpose}}$, P_{king} receives $\{[\boldsymbol{x}_i + \boldsymbol{r}_i]_{t+k-1}\}_{i=1}^k$ from all parties and distributes $\{[\boldsymbol{x}_{*,i} + \boldsymbol{r}_{*,i}]_{k-1}\}_{i=1}^k$ to all parties. We may effectively think that P_{king} distributes $\{[\boldsymbol{x}_i + \boldsymbol{r}_i]_{t+k-1}\}_{i=1}^k$ and $\{[\boldsymbol{x}_{*,i} + \boldsymbol{r}_{*,i}]_{k-1}\}_{i=1}^k$ to all parties. All parties invoke $\mathcal{F}_{\mathsf{VerifyPub}}$ to check the correctness of the transpose operations.

– If the check fails, all parties take a new dispute pair as output. Otherwise, all parties take $\{[r_{i,j}|j]_t, [L_{i,j}(\mathbf{r}_i)|j]_t\}_{i,j=1}^{N,k}$ as output.

Performing Sharing Transformations. With $\{[r_j|j]_t, [L_j(\boldsymbol{r})|j]_t\}_{j=1}^k$, we show how to transform $\{[x_j|j]_t\}_{j=1}^k$ to $\{[L_j(\boldsymbol{x})|j]_t\}_{j=1}^k$ in Π_{ShTrans}. The communication complexity of Π_{ShTrans} is $\mathsf{P2P}(O(n))$ elements.

Protocol 4 : Π_{ShTrans}

All parties take $\{[x_j|j]_t\}_{i=1}^k$ and a linear map $L = (L_1, \ldots, L_k)$ as input. All parties also hold the following random sharings prepared right before the evaluation of the current segment:

– Two groups of correlated random sharings $\{[r_j|j]_t, [L_j(\boldsymbol{r})|j]_t\}_{j=1}^k$,
– A degree-$(n'-1)$ packed Shamir sharing $[\mathbf{0}]_{n'-1}$.

All parties run the following steps to compute $\{[L_j(\boldsymbol{x})|j]_t\}_{j=1}^k$.

1. All parties locally compute $[\boldsymbol{x}]_{t+k-1}$ and $[\boldsymbol{r}]_{t+k-1}$ from $\{[x_j|j]_t\}_{j=1}^k$ and $\{[r_j|j]_t\}_{j=1}^k$.
2. All parties locally compute $[\boldsymbol{x}+\boldsymbol{r}]_{n'-1} = [\boldsymbol{x}]_{t+k-1} + [\boldsymbol{r}]_{t+k-1} + [\mathbf{0}]_{n'-1}$ and send their shares to P_{king}.
3. P_{king} reconstructs $\boldsymbol{x}+\boldsymbol{r}$, computes $L(\boldsymbol{x}+\boldsymbol{r})$, and distributes $[L(\boldsymbol{x}+\boldsymbol{r})]_{k-1}$ to all parties.
4. All parties locally compute $[L_j(\boldsymbol{x})|j]_t = [L(\boldsymbol{x}+\boldsymbol{r})]_{k-1} - [L_j(\boldsymbol{r})|j]_t$ for all $j \in [k]$.

4.4 Summary of the Evaluation Phase

To compute a general circuit with packed Shamir sharings, we utilize the techniques in [GPS21] for network routing. At a high level, for any circuit C, we first divide gates of the same type in each layer into groups of size k. After we obtain output packed Shamir sharings for each group of gates in the current layer, the authors in [GPS21] show that the input packed Shamir sharings in the next layer can be obtained as follows:

- For each output packed Shamir sharings in the current layer, we perform the fan-out operation to copy each secret enough number of times. For example, if a packed Shamir sharing $[x_1, x_2, x_3]_d$ satisfies that x_1, x_2, x_3 will be used by $(2, 3, 1)$ times in future layers, then all parties compute $[x_1, x_1, x_2]_d$ and $[x_2, x_2, x_3]_d$. Note that each new packed Shamir sharing can be obtained by performing a proper linear transformation to the original packed Shamir sharing.
- After the fan-out operations, for each obtained packed Shamir sharing, perform a proper permutation on the secrets, which is also a linear transformation.
- After doing the above two steps, we move to prepare the input packed Sharings we need in the next layer. The main property that is achieved in [GPS21] is that, now for every packed Shamir sharing $[\boldsymbol{x}]_{t+k-1}$ we want to prepare, the previous steps have generated k packed Shamir sharings $\{[\boldsymbol{x}^{(i)}]_{t+k-1}\}_{i=1}^{k}$ such that there exists a permutation $p : \{1, \ldots, k\} \to \{1, \ldots, k\}$ and $x_i = x^{(i)}_{p(i)}$. In other words, all secrets we want to collect all come from different positions. In this way, all parties can efficiently collect secrets they want and obtain $[\boldsymbol{x}']_{t+k-1}$ without changing the positions of the secrets, i.e., $x_i = x^{(i)}_{p(i)} = x'_{p(i)}$. We note that in our case this task is even simpler. This is because for every $[\boldsymbol{x}^{(i)}]_{t+k-1}$, we actually prepare $\{[x^{(i)}_j | j]_t\}_{j=1}^{k}$. Thus, we just need to pick $\{[x_i | p(i)]_t\}_{i=1}^{k} = \{x^{(i)}_{p(i)} | p(i)\}_{i=1}^{k}$.
- Finally, to obtain $[\boldsymbol{x}]_{t+k-1}$, we permute the secrets in $[\boldsymbol{x}']_{t+k-1}$, which again is a linear transformation.

Since our work uses [GPS21] in a black box way, we refer the readers to [GPS21] for how to find these linear maps.

We assume that for output packed Shamir sharings from the last segment, the first two steps have been done. We will maintain the invariant in the current segment. We summarize our evaluation protocol as Π_{Eval} below and analyze the number of required different kinds of preprocessing data as follows, where we suppose there are N linear maps and M multiplication gates needed to be handled in this segment and thus $N = O(|C|/n^3), M = O(|C|/n^2)$.

- A degree-$(n' - 1)$ packed Shamir sharing $[\mathbf{0}]_{n'-1}$: $\frac{3M}{k} + N$.
- A group of Beaver triples $\{[a_i | i]_t, [b_i | i]_t, [c_i | i]_t\}_{i=1}^{k}$: $\frac{M}{k}$ groups.
- A group of random degree-t Shamir sharings $\{[r_i | i]_t\}_{i=1}^{2k-1}$: $\frac{M}{k}$ groups.
- A group of Shamir sharings $\{[r_{i,j} | j]_t, [r_{j,i} | j]_t\}_{i,j=1}^{k}$: $\frac{2N}{k}$ groups.

Protocol 5 : Π_{Eval}

For each group of k wires before the current segment, all parties hold $\{[x_i|i]_t\}_{i=1}^k$. All parties run the following steps.

1. Suppose the linear maps we need to perform in this segment is $L_1, \ldots, L_N$. All parties invoke $\Pi_{\mathsf{PrepTrans}}$ to prepare $\{[r_{i,j}|j]_t, [L_{i,j}(\boldsymbol{r}_i)|j]_t\}_{i\in[N],j\in[k]}$. All parties invoke $\Pi_{\mathsf{zeroSharing}}$ to prepare $[\mathbf{0}]_{n'-1}$.
2. All parties evaluate the current segment layer by layer.
   (a) For each group of input wires $\boldsymbol{x}$, all parties locally collect secrets and obtain $\{[x_i|p(i)]_t\}_{i=1}^k$, where p is a permutation over $[k]$. This step is guaranteed by the network routing protocol in [GPS21].
   (b) For each group of input wires $\boldsymbol{x}$, All parties invoke Π_{ShTrans} on $\{[x_i|p(i)]_t\}_{i=1}^k$ to obtain $\{[x_i|i]_t\}_{i=1}^k$.
   (c) For each group of addition gates with input packed Shamir sharings $\{[x_i|i]_t\}_{i=1}^k$ and $\{[y_i|i]_t\}_{i=1}^k$, all parties locally compute $[z_i|i]_t = [x_i|i]_t + [y_i|i]_t$ via local computation for $i \in [k]$.
   (d) For each group of multiplication gates with input packed Shamir sharings $\{[x_i|i]_t\}_{i=1}^k$ and $\{[y_i|i]_t\}_{i=1}^k$, all parties invoke Π_{Mult} to compute $\{[z_i|i]_t\}_{i=1}^k$.
   (e) For each output packed Shamir sharing $\{[z_i|i]_t\}_{i=1}^k$ and the linear map L that is needed to perform on $\boldsymbol{z}$, all parties invoke Π_{ShTrans} on $\{[z_i|i]_t\}_{i=1}^k$ to obtain $\{[L_i(\boldsymbol{z})|i]_t\}_{i=1}^k$. (Fan-out gates and permutations are handled here.)

5 Efficient Verification

In this section, we study how to perform efficient verification after completing evaluating the current segment and handling sharing transformations.

5.1 Verifying Multiplication Gates

For now, all parties already complete the computation of the current segment including evaluating all groups of multiplication gates inside the current segment and handling sharing transformations.

To verify the evaluation above, all parties first invoke $\mathcal{F}_{\mathsf{VerifyPub}}$ to check whether P_{king} distributes valid degree-$(k-1)$ packed Shamir sharings in Π_{Mult}. If the verification passes, all parties open their shares of degree-$(n'-1)$ packed Shamir sharings of $\mathbf{0}$ prepared in $\Pi_{\mathsf{zeroSharing}}$. After subtracting the shares of degree-$(n'-1)$ packed Shamir sharings of $\mathbf{0}$, P_{king} obtains the original messages including 1) two degree-$(t+k-1)$ sharings $[\boldsymbol{x}+\boldsymbol{a}]_{t+k-1}, [\boldsymbol{y}+\boldsymbol{b}]_{t+k-1}$ and 2) a degree-$(t+2k-2)$ sharing $[\boldsymbol{z}+\boldsymbol{r}]_{t+2k-2}$ receiving from other parties.

Notice P_{king} is able to detect whether other parties send these shares correctly following the high-level idea in Subsect. 2.2. In the case of inconsistency, P_{king} localizes a dispute pair following the high-level idea in Subsect. 2.3.

The protocol $\Pi_{\mathsf{VerifyMult}}$ described below requires all parties to prepare randomness $\{[a'_j|j]_t, [b'_j|j]_t, [c'_j|j]_t\}_{j=1}^{k}$, $\{[r'_j|j]_t\}_{j=1}^{2k-1}$ in the preprocessing phase. The communication of verifying N multiplication gates is $\mathsf{P2P}(O(N+n))$ elements plus $O(1) \times \mathsf{BA}(O(1))$ bits accounting for total communication of expected $\mathsf{P2P}(O(N + n^3 \cdot \log n))$ elements in the case of success.

Protocol 6 : $\Pi_{\mathsf{VerifyMult}}$ **in** $\mathcal{F}_{\mathsf{VerifyPub}}$**-hybrid model**

All parties hold random degree-t Shamir sharings prepared in the preprocessing phase $\{[a'_j|j]_t, [b'_j|j]_t, [c'_j|j]_t\}_{j=1}^{k}$ and $\{[r'_j|j]_t\}_{j=1}^{2k-1}$.

1. All parties invoke $\mathcal{F}_{\mathsf{VerifyPub}}$ to check whether P_{king} sends valid degree-$(k-1)$ packed Shamir sharings in step 3 and step 5 of Π_{Mult}.
   - If the verification passes, then all parties send all messages in $\Pi_{\mathsf{zeroSharing}}$ to P_{king}. Then P_{king} checks whether all parties honestly follow $\Pi_{\mathsf{zeroSharing}}$.
     - If not, P_{king} either broadcasts (ℓ, v, v', P_i, P_j) if P_i and P_j do not agree on the same message, or $(P_i, \text{corrupt})$ if P_i does not follow the protocol. Then follow Step 4.(e) in $\Pi_{\mathsf{VerifyPub}}$ (in the full version of this paper [SY24]) to identify a new dispute pair. All parties take the new dispute pair as output.
     - Otherwise P_{king} subtracts the degree-$(n'-1)$ packed Shamir sharings of $\mathbf{0}$ from the sharings he received in Π_{Mult}.
   - Otherwise, all parties take the new dispute pair as output.
2. P_{king} checks if one of other parties sends wrong share $[\boldsymbol{x}+\boldsymbol{a}]_{t+k-1}, [\boldsymbol{y}+\boldsymbol{b}]_{t+k-1}$ in step 2 of Π_{Mult}: P_{king} only needs to check whether these shares are of degree $t+k-1$.
   - If it is inconsistent, every party P_i sends $\{[x_j|j]_t + [a_j|j]_t, [y_j|j]_t + [b_j|j]_t\}_{j=1}^{k}$ to P_{king}, $\forall i \in [n']$. P_{king} reconstructs $\boldsymbol{x}+\boldsymbol{a}, \boldsymbol{y}+\boldsymbol{b}$, compares it with $[\boldsymbol{x}+\boldsymbol{a}]_{t+k-1}, [\boldsymbol{y}+\boldsymbol{b}]_{t+k-1}$ to detect who is cheating and broadcasts (i, x, x'), where P_i should have sent x to P_{king}, but P_{king} claims to have received $x' \neq x$. P_{king} asks P_i and their relay to broadcast the messages they believe. Set two adjacent parties broadcasting differently to be the dispute pair. All parties take the new dispute pair as output.
3. All parties invoke $\mathcal{F}_{\mathsf{VerifyPub}}$ to check whether P_{king} sends degree-$(k-1)$ sharings with correct secret value $\boldsymbol{x}+\boldsymbol{a}, \boldsymbol{y}+\boldsymbol{b}$ in step 3 of Π_{Mult}.
   - If not, all parties take the new dispute pair as output.
4. P_{king} checks if it receives a valid deg-$(t+2k-2)$ Shamir secret sharing in step 4 of Π_{Mult}.

– If it is inconsistent, all parties localize a dispute pair as follows, using fresh random sharings $\{[a'_j|j]_t, [b'_j|j]_t, [c'_j|j]_t\}_{j=1}^{k}$ and $\{[r'_j|j]_t\}_{j=1}^{2k-1}$.

(a) Every party locally computes a degree-$(t+2k-2)$ sharing $[\boldsymbol{v}]_{t+2k-2}$ as

$$[\boldsymbol{v}]_{t+2k-2} = [\boldsymbol{x}+\boldsymbol{a}]_{k-1} \cdot [\boldsymbol{y}+\boldsymbol{b}]_{k-1} - [\boldsymbol{x}+\boldsymbol{a}]_{k-1} \cdot [\boldsymbol{b'}]_{t+k-1} - [\boldsymbol{a'}]_{t+k-1} \cdot [\boldsymbol{y}+\boldsymbol{b}]_{k-1} + [\boldsymbol{c'}]_{t+k-1} + [\boldsymbol{r'}]_{t+2k-2},$$

where $[\boldsymbol{a'}]_{t+k-1}, [\boldsymbol{b'}]_{t+k-1}, [\boldsymbol{c'}]_{t+k-1}, [\boldsymbol{r'}]_{t+2k-2}$ are locally computed from $\{[a'_j|j]_t, [b'_j|j]_t, [c'_j|j]_t\}_{j=1}^{k}$ and $\{[r'_j|j]_t\}_{j=1}^{2k-1}$. Then all parties send $[\boldsymbol{v}]_{t+2k-2}$ to P_{king}.

(b) P_{king} checks if the received shares in step 4(a) satisfy a degree-$(t+2k-2)$ Shamir secret sharing.

- If it is inconsistent, every party P_i sends $\{[a'_j|j]_t, [b'_j|j]_t, [c'_j|j]_t\}_{j=1}^{k}$ and $\{[r'_j|j]_t\}_{j=1}^{2k-1}$ to P_{king} who localizes a dispute pair as follows. P_{king} reconstructs all degree-t Shamir sharings and computes $[\boldsymbol{a'}]_{t+k-1}, [\boldsymbol{b'}]_{t+k-1}, [\boldsymbol{c'}]_{t+k-1}, [\boldsymbol{r'}]_{t+2k-2}$. Then P_{king} computes $[\boldsymbol{v}]_{t+2k-2}$ and detect who is cheating. The rest is the same as step 2.

(c) If it is consistent, every party P_i sends $\{[a'_j|j]_t + [a_j|j]_t, [b'_j|j]_t + [b_j|j]_t, [c'_j|j]_t + [c_j|j]_t\}_{j=1}^{k}$ and $\{[r'_j|j]_t + [r_j|j]_t\}_{j=1}^{2k-1}$ to P_{king} who localizes a dispute pair as in step 4(b).

5. All parties invoke $\mathcal{F}_{\mathsf{VerifyPub}}$ to check whether P_{king} sends a degree-$(k-1)$ sharing with correct secret value $\boldsymbol{z}+\boldsymbol{r}$ in step 5 of Π_{Mult}.
   – If not, all parties take the new dispute pair as output.

5.2 Verifying Sharing Transformations

Following the verification of multiplication gates, after degree-$(n'-1)$ packed Shamir sharings of $\mathbf{0}$ are opened, P_{king} checks whether all parties send correct shares $[\boldsymbol{x}+\boldsymbol{r}]_{t+k-1}$. Then all parties check whether P_{king} honestly shares $[L(\boldsymbol{x}+\boldsymbol{r})]_{k-1}$.

The protocol $\Pi_{\mathsf{VerifyShTrans}}$ is described below. The communication of verifying N sharing transformations is $\mathsf{P2P}(O(N \cdot n + n^2))$ elements plus $O(1) \times \mathsf{BA}(O(1))$ bits accounting for total communication of expected $\mathsf{P2P}(O(N \cdot n + n^3 \cdot \log n))$ elements in the case of success.

Protocol 7 : $\Pi_{\mathsf{VerifyShTrans}}$ in $\mathcal{F}_{\mathsf{VerifyPub}}$-hybrid model

1. All parties check whether P_{king} distributes a degree-$(k-1)$ Shamir sharing in step 3 of Π_{ShTrans} in the same way as step 1 in $\Pi_{\mathsf{VerifyMult}}$. Then P_{king} checks if one of other parties sends wrong share of $[\boldsymbol{x}+\boldsymbol{r}]_{t+k-1}$ in step 2 of Π_{ShTrans} in the same way as step 2 in $\Pi_{\mathsf{VerifyMult}}$.
2. All parties check whether P_{king} sends valid sharing of degree $k-1$ with correct secret $L(\boldsymbol{x}+\boldsymbol{r})$ as follows. For each group of k linear transformations, say $L_1, L_2, \ldots, L_k$, all parties hold $\{[\boldsymbol{x}_i+\boldsymbol{r}_i]_{t+k-1}, [L_i(\boldsymbol{x}_i+\boldsymbol{r}_i)]_{k-1}\}_{i=1}^{k}$. Let $[\boldsymbol{u}_i]_{t+k-1}$ denote $[\boldsymbol{x}_i+\boldsymbol{r}_i]_{t+k-1}$ and $[\boldsymbol{v}_i]_{k-1}$ denote $[L_i(\boldsymbol{x}_i+\boldsymbol{r}_i)]_{k-1}$.
   (a) P_{king} distributes $\{[\boldsymbol{u}_{*,i}]_{k-1}\}_{i=1}^{k}$ and $\{[\boldsymbol{v}_{*,i}]_{k-1}\}_{i=1}^{k}$ to all parties.
   (b) All parties invoke $\mathcal{F}_{\mathsf{VerifyPub}}$ to check whether P_{king} honestly perform the transpose operations for $\{[\boldsymbol{u}_i]_{t+k-1}\}_{i=1}^{k}$. This step is done for all groups of k linear transformations.
      – If not, all parties take a new dispute pair as output.
   (c) All parties invoke $\mathcal{F}_{\mathsf{VerifyPub}}$ to check whether P_{king} honestly perform the transpose operations for $\{[\boldsymbol{v}_i]_{k-1}\}_{i=1}^{k}$. This step is done for all groups of k linear transformations.
      – If not, all parties take a new dispute pair as output.
   (d) For all $i \in [k]$, all parties locally compute

$$[\boldsymbol{o}_i]_{2k-2} = [\boldsymbol{v}_{*,i}]_{k-1} - \sum_{j=1}^{k} [\boldsymbol{e}_j]_{k-1} \cdot L_{j,i}([\boldsymbol{u}_{*,1}]_{k-1}, \ldots, [\boldsymbol{u}_{*,k}]_{k-1}),$$

   where $\boldsymbol{e}_j \in \mathbb{F}^k$ is the j-th unit vector (i.e., the j-th value is 1 while all other values are 0). Note that P_{king} can also compute $\{[\boldsymbol{o}_i]_{2k-2}\}_{i=1}^{k}$. Effectively, we view these sharings are distributed by P_{king}.
   (e) All parties invoke $\mathcal{F}_{\mathsf{VerifyPub}}$ to check whether $\{[\boldsymbol{o}_i]_{2k-2}\}_{i=1}^{k}$ are degree-$(2k-2)$ packed secret sharings of $\mathbf{0}$.
      – If not, all parties take a new dispute pair as output.

6 Main Protocol with Perfect Security

In this section, we will conclude our main protocol and discuss the instantiation of preprocessing phase.

6.1 Input Layer

We sketch the protocol Π_{Input} that allows a party P_s to share his inputs to all parties and refer its formal description to the full version of this paper [SY24]. At the end of Π_{Input}, either all parties hold degree-t Shamir sharings of P_s's input, or a new dispute pair is identified and P_s will re-share his input.

For each group of k inputs $\boldsymbol{x}$ of P_s, all parties prepare random degree-t Shamir sharings $\{[r_j|j]_t\}_{j=1}^k$ in the preprocessing phase. Then in the online phase, all parties locally compute $[\boldsymbol{r}]_{t+k-1}$ from $\{[r_j|j]_t\}_{j=1}^k$ and send their shares to P_s. P_s checks whether the received shares form a valid degree-$(t+k-1)$ packed Shamir sharing. If not, all parties send their shares of $\{[r_j|j]_t\}_{j=1}^k$ to P_s and P_s helps identify a dispute pair. Otherwise, P_s distributes $[\boldsymbol{x}+\boldsymbol{r}]_{k-1}$ to all parties. Then all parties use $\mathcal{F}_{\mathsf{VerifyPub}}$ to check whether P_s distributes a valid degree-$(k-1)$ packed Shamir sharing. If not, all parties receive a new dispute pair from $\mathcal{F}_{\mathsf{VerifyPub}}$. Otherwise, all parties locally compute $[x_j|j]_t = [\boldsymbol{x}+\boldsymbol{r}]_{k-1} - [r_j|j]_t$.

The communication of handling N input gates is $\mathsf{P2P}(O(N+n))$ elements plus $O(1) \times \mathsf{BA}(O(1))$ bits accounting for total communication of expected $\mathsf{P2P}(N + n^3 \cdot \log n)$ elements in the case of success.

6.2 Output Layer

After completing the entire computation segment by segment, all parties come to the output phase. To reconstruct the output $\boldsymbol{z} \in \mathbb{F}^k$ of party P_s towards P_s, all parties send their shares of $[\boldsymbol{z}]_{t+k-1}$ to P_s. Then after checking the consistency, P_s is able to reconstruct its output.

We formally describe Π_{Output} in the full version of this paper [SY24]. The communication is $\mathsf{P2P}(O(N+n))$ elements to handle N output gates in the case of success.

6.3 Main Protocol

Now we are ready to present our main protocol. First, we apply the deterministic circuit transformation algorithm proposed in [GPS21] to ensure the resulting circuit is friendly to the packed secret sharing technique. In dispute control framework, we keep recording the set of active parties $\mathcal{P}$ which contains all parties initially, and the set of dispute pair of parties $\mathcal{D}$ which is empty initially. In the **preprocessing phase**, all parties invoke $\mathcal{F}_{\mathsf{RandSh}}$ and $\mathcal{F}_{\mathsf{Triples}}$ to prepare sufficient amount of random sharings of each type. In the **input phase**, all parties invoke Π_{Input} to share their groups of inputs. Before computing the circuit, we divide uniformly the whole circuit into $O(n^2)$ segments satisfying the requirements stated in Remark 1 and start to compute all the segments sequentially. When evaluating each segment, to tackle with the fan-out gates in the current segment, we first handle the fan-out gates whose output wires will be used in the current segment, then divide the rest of them into several segments also satisfying the requirements in Remark 1, and compute the segments sequentially. We refer the readers to the full version of this paper [SY24] for the detailed requirements of circuit transformation and the formal algorithm for segment division.

In the **computation phase**, each segment is first evaluated by Π_{Eval}. Then all parties together check the correctness of the computation by running $\Pi_{\mathsf{VerifyMult}}$ and $\Pi_{\mathsf{VerifyShTrans}}$. All parties either agree on the success of the computation, in which case they move on to evaluate the next segment, or receive a new dispute pair. In the latter case, the whole segment is re-evaluated.

Ultimately, all parties have already had their shares of each output. Then in the **output phase**, all parties invoke Π_{Output} to reconstruct the output towards the corresponding party. We describe the main protocol Π_{Main} below.

Protocol 8 : Π_{Main} **in** $\mathcal{F}_{\mathsf{RandSh}}, \mathcal{F}_{\mathsf{Triples}}, \mathcal{F}_{\mathsf{VerifyPub}}$**-hybrid model**

Circuit Transformation Phase. We adopt the deterministic algorithm in [GPS21].
Let $\mathcal{D} = \{(P_i, P_j) | P_i \text{ and } P_j \text{ are disputed}\}$ denote the set of pairs of disputed parties which initially is an empty set. Let $\mathcal{P}$ of size n' denote the set of parties remained to participate the computation which contains all n parties initially.
Preprocessing Phase. All parties invoke $\mathcal{F}_{\mathsf{RandSh}}, \mathcal{F}_{\mathsf{Triples}}$ to receive correlated randomness that will be used in the online phase.
Input Phase. All parties invoke Π_{Input} to share their inputs.
Segment Division Phase. We perform the algorithm for segment division to divide the circuit into $O(n^2)$ segments which will then be evaluated sequentially.
Computation Phase. For every segment of the circuit:

1. All parties in $\mathcal{P}$ invoke Π_{Eval} to evaluate this segment.
2. All parties in $\mathcal{P}$ invoke $\Pi_{\mathsf{VerifyMult}}$ and $\Pi_{\mathsf{VerifyShTrans}}$ to verify the correctness of the computation.

If a dispute pair of parties is identified, **re-evaluate*** the current segment. Otherwise, all parties perform the remaining fan-out gates. This can be viewed as a segment that only contains fan-out gates. Thus, it can be evaluated in the same way as described above.
Output Phase. All parties invoke Π_{Output} to reconstruct their outputs. If a dispute pair of parties is identified, **re-evaluate*** the current segment.

***Re-evaluation**. Each time when faults occur and a dispute pair is identified, add this pair to $\mathcal{D}$, assign each dispute pair in $\mathcal{D}$ who are both active an intermediate party for relaying, and re-evaluate the current segment. (We omit this in protocol description for simplicity.)

Remark 1 (Segment Division). To upper bound the overhead of re-evaluating the segments due to the faults caused by dispute pairs, we enforce our segment division to satisfy the following three requirements: 1) the circuit size of each segment is $O(\frac{|C|}{n^2})$ 2) the circuit depth of each segment is $O(\frac{\mathsf{Depth}}{n^2})$ and 3) gates belonging to one group should be contained in the same segment. To achieve this, following the topology of the circuit, we include the gates into one segment until the circuit size of the current segment exceeds $\frac{|C|}{n^2}$ or the circuit depth of

the current segment exceeds $\frac{\mathsf{Depth}}{n^2}$ while keeping the number of gates of each kind a multiple of k, which results in at most $3n^2$ segments in the online phase.

Analysis of the Communication Complexity of Π_{Main}. Let I denote the input size, G denote the number of gates, and O denote the output size, Depth denote the depth of the circuit. Then $|C| \geq I+G+O$. We set $T = n-t, k = (n+5)/6$. In summary, the overall communication complexity for online phase is $\mathsf{P2P}(O(|C| \cdot \frac{n}{k} + (\mathsf{Depth}+n) \cdot n + n^5))$ elements plus $O(n^2) \times \mathsf{BA}(O(1))$ bits, where the term n^5 is caused by redoing the segment when encountering faults. Thus, if we apply an expected constant round broadcast protocol like [AC24], the online communication becomes expected $\mathsf{P2P}(O(|C| \cdot \frac{n}{k} + (\mathsf{Depth}+n) \cdot n + n^5 \cdot \log n))$ elements.

Analysis of the Round Complexity of Π_{Main}. For each segment of circuit, its depth is bounded by $O(\frac{\mathsf{Depth}}{n^2})$. The round complexity is $O(\frac{\mathsf{Depth}}{n^2}) + O(1) \times \mathsf{BA}$ and $O(\frac{\mathsf{Depth}}{n^2}) + O(1) \times \mathsf{BA} + O(1) \times \mathsf{BC}$ in the case that faults occur. Hence, the online round complexity is $O(\mathsf{Depth}+n^2) + O(n^2) \times \mathsf{BA} + O(n^2) \times \mathsf{BC}$. Thus, if we apply an expected constant round broadcast protocol like [AC24], the online round complexity becomes $O(\mathsf{Depth}+n^2)$.

6.4 Summary

In the preprocessing phase, the preprocessing randomness can be divided into two classes: (1) degree-t Shamir secret sharings satisfying some specific requirements and (2) packed Beaver triples, which both can be dealt with using the techniques in [BH08] as mentioned in Subsect. 3.5. In total, the communication for preprocessing phase is $\mathsf{P2P}(O((|C| + k(n + \mathsf{Depth}))n + n^4))$ elements plus $O(n^2) \times \mathsf{BA}(O(1))$ bits, which becomes expected $\mathsf{P2P}(O((|C|+k(n+\mathsf{Depth}))n + n^5 \cdot \log n))$ elements assuming an expected constant round broadcast protocol like [AC24] is used. Moreover, the round complexity for preprocessing phase is $O(n^2) + O(n^2) \times \mathsf{BA} + O(n^2) \times \mathsf{BC}$ and becomes $O(n^2)$ in expectation assuming an expected constant round broadcast protocol like [AC24] is used.

To conclude, combining the preprocessing phase and the online phase, we get a perfectly secure protocol to compute a circuit C with online communication of expected $O(|C| \cdot \frac{n}{k} + (\mathsf{Depth}+n) \cdot n + n^5 \cdot \log n)$ elements which is linear in the circuit size, offline communication of expected $O((|C|+k(n+\mathsf{Depth}))n+n^5 \cdot \log n$ elements, online round complexity $O(\mathsf{Depth}+n^2)$ in expectation and offline round complexity $O(n^2)$ in expectation.

Functionality 2 : $\mathcal{F}_{\mathsf{Main}}$ in $\mathcal{F}_{\mathsf{RandSh}}, \mathcal{F}_{\mathsf{Triples}}, \mathcal{F}_{\mathsf{VerifyPub}}$-hybrid model

1. $\mathcal{F}_{\mathsf{Main}}$ receives the input from all parties. Let x denote the input and C denote the circuit.
2. $\mathcal{F}_{\mathsf{Main}}$ computes $C(x)$ and distributes the output to all parties.

Lemma 1. *Protocol* Π_{Main} *securely computes* $\mathcal{F}_{\mathsf{Main}}$ *in* $\mathcal{F}_{\mathsf{RandSh}}, \mathcal{F}_{\mathsf{Triples}}, \mathcal{F}_{\mathsf{VerifyPub}}$*-hybrid model against a fully malicious adversary who controls at most* $t < n/3$ *parties.*

We refer the readers to the full version of this paper [SY24] for the proof of Lemma 1 and the detailed communication complexity analysis. Combining the complexity analysis and Lemma 1, we obtain the following theorem.

Theorem 1. *Let* n *denote the number of parties. Let* $\mathbb{F}$ *be a finite field of size* $|\mathbb{F}| \geq 2n$*. For an arithmetic circuit* C *over* $\mathbb{F}$*, there exists an information-theoretic MPC protocol that computes* C *against a fully malicious adversary controlling at most* $t = \frac{n-1}{3}$ *corrupted parties with perfect security. The communication cost of the protocol is expected* $O(|C| + \mathsf{Depth} \cdot n + n^5 \cdot \log n)$ *elements for the online phase and expected* $O(|C| \cdot n + \mathsf{Depth} \cdot n^2 + n^5 \cdot \log n)$ *elements for the preprocessing phase, where* Depth *is the circuit depth. The round complexity of the protocol is* $O(\mathsf{Depth} + n^2)$ *in expectation for the online phase and* $O(n^2)$ *in expectation for the preprocessing phase.*

Acknowledgments. Y. Song was supported in part by the National Basic Research Program of China Grant 2011CBA00300, 2011CBA00301, the National Natural Science Foundation of China Grant 61033001, 61361136003.

References

AAPP23. Abraham, I., Asharov, G., Patil, S., Patra, A.: Detect, pack and batch: perfectly-secure MPC with linear communication and constant expected time. In: Hazay, C., Stam, M. (eds.) EUROCRYPT 2023. LNCS, vol. 14005, pp. 251–281. Springer, Cham (2023). https://doi.org/10.1007/978-3-031-30617-4_9

AAY22. Abraham, I., Asharov, G., Yanai, A.: Efficient perfectly secure computation with optimal resilience. J. Cryptol. **35**(4), 27 (2022)

AC24. Asharov, G., Chandramouli, A.: Perfect (parallel) broadcast in constant expected rounds via statistical VSS. In: Joye, M., Leander, G. (eds.) EUROCRYPT 2024. LNCS, vol. 14655, pp. 310–339. Springer, Cham (2024). https://doi.org/10.1007/978-3-031-58740-5_11

ALR11. Asharov, G., Lindell, Y., Rabin, T.: Perfectly-secure multiplication for any tâĂĽ<âĂĽn/3. In: Rogaway, P. (ed.) CRYPTO 2011. LNCS, vol. 6841, pp. 240–258. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22792-9_14

Bea89. Beaver, D.: Multiparty protocols tolerating half faulty processors. In: Brassard, G. (ed.) CRYPTO 1989. LNCS, vol. 435, pp. 560–572. Springer, New York (1990). https://doi.org/10.1007/0-387-34805-0_49

BGW88. Ben-Or, M., Goldwasser, S., Wigderson, A.: Completeness theorems for non-cryptographic fault-tolerant distributed computation. In: Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing, pp. 1–10. ACM (1988)

BH06. Beerliová-Trubíniová, Z., Hirt, M.: Efficient multi-party computation with dispute control. In: Halevi, S., Rabin, T. (eds.) TCC 2006. LNCS, vol. 3876, pp. 305–328. Springer, Heidelberg (2006). https://doi.org/10.1007/11681878_16

BH08. Beerliová-Trubíniová, Z., Hirt, M.: Perfectly-secure MPC with linear communication complexity. In: Canetti, R. (ed.) TCC 2008. LNCS, vol. 4948, pp. 213–230. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-78524-8_13

Can00. Canetti, R.: Security and composition of multiparty cryptographic protocols. J. Cryptol. **13**, 143–202 (2000)

CCD88. Chaum, D., Crépeau, C., Damgard, I.: Multiparty unconditionally secure protocols. In: Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing, pp. 11–19. ACM (1988)

DEN24. Dalskov, A., Escudero, D., Nof, A.: Fully secure MPC and zk-FLIOP over rings: new constructions, improvements and extensions. In: Reyzin, L., Stebila, D. (eds.) CRYPTO 2024. LNCS, vol. 14927, pp. 136–169. Springer, Cham (2024). https://doi.org/10.1007/978-3-031-68397-8_5

DIK10. Damgård, I., Ishai, Y., Krøigaard, M.: Perfectly secure multiparty computation and the computational overhead of cryptography. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 445–465. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-13190-5_23

DN07. Damgård, I., Nielsen, J.B.: Scalable and unconditionally secure multiparty computation. In: Menezes, A. (ed.) CRYPTO 2007. LNCS, vol. 4622, pp. 572–590. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-74143-5_32

DS20. Damgård, I., Schwartzbach, N.I.: Communication lower bounds for perfect maliciously secure MPC. Cryptology ePrint Archive, Paper 2020/251 (2020)

EGPS22. Escudero, D., Goyal, V., Polychroniadou, A., Song, Y.: TurboPack: honest majority MPC with constant online communication. In: Yin, H., Stavrou, A., Cremers, C., Shi, E. (eds.) ACM CCS 2022: 29th Conference on Computer and Communications Security, Los Angeles, CA, USA, 7–11 November, pp. 951–964. ACM Press (2022)

FY92. Franklin, M.K., Yung, M.: Communication complexity of secure computation (extended abstract). In: 24th Annual ACM Symposium on Theory of Computing, Victoria, BC, Canada, 4–6 May, pp. 699–710. ACM Press (1992)

GBO+23. Gama, M., Beni, E.H., Orsini, E., Smart, N.P., Zajonc, O.: MPC with delayed parties over star-like networks. In: Guo, J., Steinfeld, R. (eds.) ASIACRYPT 2023. LNCS, vol. 14438, pp. 172–203. Springer, Singapore (2023). https://doi.org/10.1007/978-981-99-8721-4_6

GLO+21. Goyal, V., Li, H., Ostrovsky, R., Polychroniadou, A., Song, Y.: ATLAS: efficient and scalable MPC in the honest majority setting. In: Malkin, T., Peikert, C. (eds.) CRYPTO 2021. LNCS, vol. 12826, pp. 244–274. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-84245-1_9

GLS19. Goyal, V., Liu, Y., Song, Y.: Communication-efficient unconditional MPC with guaranteed output delivery. In: Boldyreva, A., Micciancio, D. (eds.) CRYPTO 2019, Part II. LNCS, vol. 11693, pp. 85–114. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-26951-7_4

GMW87. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game. In: Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing, pp. 218–229. ACM (1987)

GPS21. Goyal, V., Polychroniadou, A., Song, Y.: Unconditional communication-efficient MPC via Hall's Marriage Theorem. In: Malkin, T., Peikert, C. (eds.) CRYPTO 2021, Part II. LNCS, vol. 12826, pp. 275–304. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-84245-1_10

GPS22. Goyal, V., Polychroniadou, A., Song, Y.: Sharing transformation and dishonest majority MPC with packed secret sharing. In: Dodis, Y., Shrimpton, T. (eds.) CRYPTO 2022, Part IV. LNCS, vol. 13510, pp. 3–32. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-15985-5_1

GSZ20. Goyal, V., Song, Y., Zhu, C.: Guaranteed output delivery comes free in honest majority MPC. In: Micciancio, D., Ristenpart, T. (eds.) CRYPTO 2020. LNCS, vol. 12171, pp. 618–646. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-56880-1_22

HMP00. Hirt, M., Maurer, U., Przydatek, B.: Efficient secure multi-party computation. In: Okamoto, T. (ed.) ASIACRYPT 2000. LNCS, vol. 1976, pp. 143–161. Springer, Heidelberg (2000). https://doi.org/10.1007/3-540-44448-3_12

RB89. Rabin, T., Ben-Or, M.: Verifiable secret sharing and multiparty protocols with honest majority. In: Proceedings of the Twenty-First Annual ACM Symposium on Theory of Computing, pp. 73–85. ACM (1989)

Sha79. Shamir, A.: How to share a secret. Commun. Assoc. Comput. Mach. **22**(11), 612–613 (1979)

SY24. Song, Y., Ye, X.: Perfectly-secure MPC with constant online communication complexity. Cryptology ePrint Archive, Paper 2024/242 (2024)

Yao82. Yao, A.C.: Protocols for secure computations. In: 23rd Annual Symposium on Foundations of Computer Science, SFCS 2008, pp. 160–164. IEEE (1982)

Statistical Layered MPC

Giovanni Deligios[1(✉)], Anders Konring[2], Chen-Da Liu-Zhang[3], and Varun Narayanan[4]

[1] ETH Zurich, Zürich, Switzerland
gdeligios@ethz.ch
[2] Espresso Systems, Menlo Park, USA
[3] Lucerne University of Applied Sciences and Arts and Web3 Foundation, Zug, Switzerland
chen-da.liuzhang@hslu.ch
[4] University of California, Los Angeles, USA

Abstract. The seminal work of Rabin and Ben-Or (STOC '89) showed that the problem of secure n-party computation can be solved for $t < n/2$ corruptions with guaranteed output delivery and statistical security. This holds in the traditional static model where the set of parties is fixed throughout the entire protocol execution.

The need to better capture the dynamics of large scale and long-lived computations, where compromised parties may recover and the set of parties can change over time, has sparked renewed interest in the proactive security model by Ostrovsky and Yung (PODC '91). This abstraction, where the adversary may periodically uncorrupt and corrupt a new set of parties, is taken even a step further in the more recent YOSO and Fluid MPC models (CRYPTO '21) which allow, in addition, disjoint sets of parties participating in each round. Previous solutions with guaranteed output delivery and statistical security only tolerate $t < n/3$ corruptions, or assume a random corruption pattern plus non-standard communication models. Matching the Rabin and Ben-Or bound in these settings remains an open problem.

In this work, we settle this question considering the unifying Layered MPC abstraction recently introduced by David et al. (CRYPTO '23). In this model, the interaction pattern is defined by a layered acyclic graph, where each party sends secret messages and broadcast messages only to parties in the very next layer. We complete the feasibility landscape of layered MPC, by extending the Rabin and Ben-Or result to this setting. Our results imply maximally-proactive MPC with statistical security in the honest-majority setting.

1 Introduction

Setting. In the problem of secure multi-party computation (MPC) [2,6,17,27,29] a set of mutually distrusting parties jointly computes a function of their private data, so that nothing about their data beyond the function output is leaked.

MPC protocols are traditionally designed assuming a *static* set of n parties that stay online throughout the entire protocol execution. Security is guaranteed

E. Boyle and M. Mahmoody (Eds.): TCC 2024, LNCS 15367, pp. 362–394, 2025.
https://doi.org/10.1007/978-3-031-78023-3_12

as long as the number of parties compromised at any point throughout the entire protocol execution is less than a specified threshold. In this setting, assuming bilateral secure channels, Ben-Or, Goldwasser and Wigderson [2] and Chaum, Crepeau and Damgard [7] showed protocols to compute *any* function with *perfect* security against computationally unbounded adversaries corrupting less than $n/3$ of participants. Rabin and Ben-Or [27] showed that assuming ideal broadcast and settling for statistical (up to some error probability) rather than perfect security, the resilience can be improved to the optimal threshold of less than $n/2$.

However, this static-participant model is ill suited for capturing large-scale long-lived secure computations, where the assumption that servers remain online throughout the protocol is hard to satisfy. For longer applications, it is reasonable to assume that parties that are compromised at some point in the protocol execution may be able to recover, especially if the corruption capability of the adversary is tied to some scarce resource. The gap between the reach of traditional models and the needs of some use-cases has revived interest in the mobile adversary model by Ostrovsky and Yung [26]. Here, the adversary can corrupt different sets of parties in each round, as long as the number of per-round corruptions does not exceed a fixed threshold.

Recently, models have been proposed which in addition address the need for a more dynamic participation of parties: the YOSO [16] and Fluid MPC [8] models. They take the mobile-adversary model a step further by allowing a different set of n parties (called a committee) to participate in each round of the computation.

In [16] a protocol with statistical security and guaranteed output delivery against a dishonest minority is presented. However, this protocol *assumes* ideal secure channels (with strong properties) between parties that come online at different times, and the adversary corrupts parties independently and with a constant probability $\tau < 1/2$. In contrast, the Fluid model [8] assumes a worst-case corruption model, where the adversary can choose to corrupt any $t < n/2$ out of the n committee members in each round. In this case, the original paper only provides protocols that fall short of full security (guaranteed output delivery) and achieve the weaker notion of security with abort instead. Recently in [10], the authors showed a protocol for *perfect* security and guaranteed output delivery, but tolerating only up to $t < n/3$ corruptions. In this work, we investigate this model further in the statistical setting striving for the optimal resilience of $t < n/2$ corruptions. More concretely, we ask the following question:

Is it possible to solve the MPC problem with statistical security and optimal resiliency of $n/2$ corruptions in the setting of dynamic committees?

Our Contributions. We answer the question in the affirmative. We construct the first MPC protocol supporting dynamic committees that achieves guaranteed output delivery and statistical security tolerating up to $t < n/2$ corruptions per round. We remark that prior works in this setting only tolerate $t < n/3$ corruptions or assume non-standard communication and adversary models.

We consider the *Layered MPC* abstraction recently introduced in [10]. This model (which is equivalent to Fluid MPC with maximal-fluidity [8]) provides

a clean and elegant framework to treat MPC with dynamic committees. The $N = dn$ parties are partitioned in d numbered *layers*, with each layer consisting of n parties. Parties have access to bilateral secure communication channels, but the communication pattern of a protocol is restricted as follows: a party in any layer can send messages to any party in the following layer. Additionally, the parties have access to secure broadcast channels, but again the use of these channels is restricted and each party can only broadcast messages to all the parties in the next layer. The adversary is restricted to corrupt a subset of up to t out of n parties in each layer, a specialization of the notion of general adversary structure [18]. In this model, we prove the following theorem.

Theorem 1. *Let f be an n-party function computed by an arithmetic circuit C of depth d over a finite field $\mathbb{F}$. Then, for any $t < n/2$, there is a polynomial-time and polynomial-communication (in n and $|C|$) layered protocol over a $O(d)$-layered network that computes f with statistical t-security.*

This result finally settles the feasibility landscape of optimally-resilient MPC with guaranteed output delivery in the layered model, complementing analogous results in the settings of perfect and computational security. Furthermore, as proven in ([10], Lemma 1), layered MPC implies maximally proactive MPC ([10], Definition 3), so that our result also implies the existence of a statistically-secure maximally-proactive MPC protocols in the honest majority setting for a *fixed* set of n parties.

Corollary 1. *Let f be an n-party function computed by an arithmetic circuit over a finite field $\mathbb{F}$ with depth d. Then, for any $t < n/2$, there is a t-resilient polynomial-time and polynomial-communication maximally proactive MPC protocol computing f in $O(d)$ rounds with statistical security.*

We provide explicit ideal functionalities for all the primitives we build, and formally reduce the security of our protocols to the security of their most basic building blocks (like linear information-theoretic message authentication codes), whose security is captured through formal definitions. Considering the significant complexity overhead that by design affects constructions with dynamic committees, we consider this modelling effort a contribution in its own right.

Related Work. Due to space constraints, we refer the reader to the full version of this paper for an exhaustive treatment of related work. We summarize the landscape of relevant results in Table 1, updated from [10].

2 Technical Overview

Challenges. An approach that has repeatedly proven successful in the literature is to transform protocols that provide security with abort into protocols achieving full security (guaranteed output delivery). Popular techniques such as player elimination or dispute control [1,19] rely on detecting and excluding pairs of parties when at least one is known to be corrupted, and then restarting the protocol

Table 1. Protocols realizing primitives in the most extreme proactive settings. (*protocol security relies on the adversary only doing probabilistic corruption, †assumes access to ideal target-anonymous channels for future messaging)

Maximally Proactive MPC with Dynamic Committees				
Functionality	Reference	Level	Security	Threshold
VSS	[10]	perfect	full	$t < n/3$
	[3]	computational	full	$t < n/4^*$
	[16] (YOSO)	statistical	full (w/setup†)	$t < n/2^*$
	This work	statistical	full	$t < n/2$
MPC	[26]	perfect	full	$t < n/d$
	[10]	perfect	full	$t < n/3$
	[10]	computational	full	$t < n/2$
	[16] (YOSO)	statistical	full (w/setup†)	$t < n/2^*$
	[8] (Fluid)	statistical	w/abort	$t < n/2$
	This work	statistical	full	$t < n/2$

from a previous step, thus reducing the number of parties while maintaining the same corruption ratio. Unfortunately, these techniques are not applicable to the layered setting, where new committees consist of entirely different parties.

An alternative approach is to build upon sub-protocols, such as verifiable secret sharing, which tolerate malicious adversaries from the outset. However, the only information-theoretic protocols that provide guaranteed output delivery in the layered setting are resilient against $t < n/3$ corruptions, and the techniques used fail in the honest-majority setting with $t \geq n/3$ corruptions. In particular, the efficient verifiable secret sharing scheme in [10] is based on the perfectly secure VSS from Gennaro et al. [14], tailored to the $t < n/3$ setting. Conversely, the computational protocol from [10] (see [11] for details) inherently utilizes linearly-homomorphic cryptographic commitments.

Our construction follows a blueprint first adapted to the setting of dynamic committees in the original YOSO paper [16]: an implementation of the well-known BGW paradigm. Initially, clients in the first layer distribute their inputs using a verifiable secret sharing scheme. Then, servers in subsequent layers process the circuit gate by gate; for each gate, servers in consecutive layers compute shares of the output wire from shares of the input wires of that gate. Ultimately, a layer of clients holds the output of the circuit.

However, adapting this paradigm to the layered setting presents a unique challenges, as the protocol outlined in [16] relies on a couple of crucial assumptions. The first is ideal point-to-point *committing* channels to the future. These channels allow parties in any layer to transmit messages to parties in *any* subsequent layer. More importantly, when a corrupt party in $\mathcal{L}_0$ sends a message to a party in a later layer $\mathcal{L}_k$ for $k > 1$, the channel does not allow the sender to modify the message in a later layer $\mathcal{L}_{k'}$ where $k' > 0$. Such channels trivially

solve issues arising from rushing and future causal dependency attacks, because a corrupted sender cannot adjust their message based on information acquired throughout the intermediate layers until $\mathcal{L}_k$. Addressing these attacks across all the primitives we construct (including our own constructions of channels to future committees, distributed information-theoretic signatures, distributed commitments) constitutes the primary challenge in the layered model.

Secondly, the YOSO model assumes that the adversary corrupts parties independently and at random. To illustrate why this model is weaker, consider an example where an honest party $P \in \mathcal{L}_0$ wants to distribute a secret s to the future. For this purpose, P generates k additive shares $s = s_1 + \cdots + s_k$ and transmits s_i to $P_i \in \mathcal{L}_1$. The probability of the adversary learning s in this scenario is 2^{-k}. In contrast, in our model, the adversary can simply corrupt the first k parties in $\mathcal{L}_1$ and learn the secret with probability 1.

Our Approach. In this section we go through each of the building blocks for our layered MPC protocol, explaining the novel ideas needed to overcome these challenges.

Robust Linear Secret Sharing. Our protocol follows the share-evaluate-reconstruct paradigm, so that naturally linear secret sharing serves as a basic building block. In the $t < n/3$ setting, even simple secret sharing schemes (think of Shamir sharing) achieve the important property of *robustness*: that is, if the dealer samples the shares honestly, even if an adversary modifies up to t out of n shares, the original secret can be correctly reconstructed. In our $t < n/2$ setting, robustness can also be achieved, but only up to some error probability.

We employ a simple secret sharing scheme that achieves robustness when up to $t < n/2$ are swapped *independently* from honest ones, is linear, and only requires a single round for reconstruction. The construction follows well-known techniques that date back to Rabin and Ben-Or or earlier, (also used in [12]) based on linear information theoretic message authentication codes (MAC). The secret m is shared as $(m_1, \ldots, m_n)$ using a plain t-out-of-n secret sharing scheme (such as classical Shamir [28]). Then, each share m'_i is authenticated using n MAC keys $(k_{1i}, \ldots, k_{ni})$ producing MAC tags $(t_{1i}, \ldots, t_{ni})$. The actual shares of the robust scheme are then defined as $m_i = (m'_i, (k_{i1}, \ldots, k_{in}), (t_{1i}, \ldots, t_{ni}))$, containing the non-robust share m'_i, the tags authenticating m'_i, as well as one key used to authenticate every other non-robust share $m'_j \neq m'_i$. An adversary holding up to t of the authenticated shares cannot forge MAC tags for the remaining $n - t$ keys, from which the robustness of the scheme follows. Hence, shares for which only up to t MAC checks succeed, can be dropped to ensure that only correct shares are used during reconstruction. This scheme is only linear if the keys for the MACs are sampled according to an appropriate distribution, and therefore it does *not* provide linearity among sharings performed by different dealers. Due to space constraints details are provided in the full version of this paper.

Tolerating attacks by a rushing adversary that can modify corrupt shares after having observed the honest ones is also important for later construc-

tions, but achieving this guarantee is postponed until our future broadcast functionality.

Future Messaging. This primitive effectively establishes secure channels to future layers. More specifically, future messaging enables any sender from an initial layer, say layer $\mathcal{L}_0$, to transmit a message m to any receiver in layer $\mathcal{L}_k$ for $k > 0$. If the sender is honest, the view of an adversary corrupting up to $t < n/2$ parties in *all* intermediate layers is independent from m. It is important to note that this functionality is considerably weaker compared to the channels assumed in the YOSO model: if the sender is corrupted, the adversary can alter the message until the final layer $\mathcal{L}_k$. This is in contrast to the channels assumed in YOSO, where the adversary must fix the message at the time of transmission, independently from its view in later layers.

When sender and receiver are in adjacent layers (case $k = 1$), they can simply communicate via the provided point-to-point secure channels. When sender and receiver are separated by exactly one layer of parties (case $k = 2$) future messaging reduces to the problem of one-way secure message transmission (SMT) [13]. In this special case, an easy solution is for the sender to distribute a t-out-of-n robust secret sharing of m to layer $\mathcal{L}_1$, so that each party in this layer holds a distinct share. Then, each party in layer $\mathcal{L}_1$ forwards their share to the receiver in layer $\mathcal{L}_2$, who reconstructs the secret. In this last step, it is crucial that the reconstruction procedure of the secret sharing scheme is non-interactive. Since all communication happens between parties in adjacent layers, we can take advantage of the provided secure point-to-point channels. Note that, because the receiver is honest (we are not interested in providing any security guarantees for corrupted receivers), we are in the best-case scenario outlined above: the adversary can modify corrupt shares but only independently from the honest ones. Therefore, the receiver can recover the message m thanks to the robustness of the secret sharing scheme.

The general case ($k > 2$) is tackled recursively: the robust secret sharing of the message is distributed by the sender towards some intermediate layer $\mathcal{L}_{k'}$ for $0 < k' < k$. The shares are then forwarded by parties in layer $\mathcal{L}_{k'}$ to the receiver in layer $\mathcal{L}_k$. However, if $k' \geq 2$ or similarly if $k - k' \geq 2$, we cannot take advantage of provided point-to-point channels. To overcome this problem, each share is treated by the sender as a new message for a receiver in layer $\mathcal{L}_{k'}$, and the procedure is iterated in a recursive fashion, with the base of the iteration being resolved by making use of point-to-point channels between adjacent layers. Choosing $k' = \lfloor \frac{k}{2} \rfloor$ results in $O\big((C \cdot n)^{\log(k)}\big)$ communication complexity for a small constant C, so that as long as the sender and receiver are separated by a constant number of layers, as in all our constructions, the protocol is efficient.

Future Broadcast. Our secret sharing scheme fails to provide robustness against a rushing adversary. This primitive achieves this, and in addition it provides an agreement guarantee (hence the name *broadcast*) when the sender (dealer) is corrupted: all receivers agree on the same value.

Specifically, future broadcast allows a party in layer $\mathcal{L}_0$ to send a message m securely to a set of recipients in layer $\mathcal{L}_k$. Each honest recipient agrees on *the*

same message m', even if the sender is corrupted, and $m' = m$ when the sender is honest. Moreover, the primitive allows an auxiliary layer $\mathcal{L}_{k'}$ for $0 < k' < k$ to decide whether to deliver the message m or not. Future broadcast also guarantees linearity among messages from the same sender: if $\mathcal{L}_{k'}$ can deliver messages m_1 and m_2 from the same sender, it can also deliver any linear combination $am_1 + bm_2$ of these messages. For a single recipient, in contrast to future messaging the auxiliary layer decides whether to reveal the message or not.

In our construction, the dealer samples n independent robust sharings of m and provides them to $\mathcal{L}_{k'}$ using future messaging. Then, each of these states are reconstructed towards a distinct party in a buffer layer. Each party in the buffer layer finally broadcasts the value they have reconstructed, and a majority decision over all broadcast values is taken. Because the final decision is over public values, the agreement property follows easily. What is more, the honest majority in the buffer layer ensures robustness, as all honest parties in the buffer layer reconstruct the correct m if the sender is honest. Note that it is not enough to distribute a single sharing to $\mathcal{L}_{k'}$ and have every party in this layer broadcast their share, because a rushing adversary sees the shares broadcasted by honest parties before broadcasting corrupt shares. Our robust secret sharing scheme provides no guarantees in this case. However, it is easy to observe that if the n sharings of the message m are independent this rushing attack does not apply for an honest receiver in the buffer layer.

For context, in the non-layered model, our future broadcast protocol effectively realizes a robust linear secret sharing scheme secure against a rushing adversary, with a reconstruction procedure consisting of two communication rounds. We did not optimize the efficiency of our protocol: to share an m-bit secret, the resulting share size is $O(n \cdot m \cdot \kappa)$, where κ is the statistical security parameter. This is to be compared with the most efficient robust protocols in the non-layered setting [24], where the share size is $m + O(\kappa \cdot \log n(\log n + \log m))$. However, efficient constructions sacrifice the linearity properties crucial in MPC applications, and it is anyway unclear how to use them in our model, as there is no generic way to adapt their interactive reconstruction procedures to the layered setting. In contrast, our construction is linear, conceptually simple, and secure in the layered setting. Details can be found in Sect. 5.

Information Theoretic Signature. The primitives described until now provide no guarantees when the sender is dishonest. They allow a dishonest sender to correlate their message with the information learned in the layers previous to (and including) the output layer. As a result, the sender is not committed to a message until the time of its delivery. Information theoretic signature (IT signature) takes the first step in the direction of limiting this freedom.

In an IT signature protocol, a sender $\mathbf{S} \in \mathcal{L}_0$ entrusts an intermediary $\mathbf{M} \in \mathcal{L}_k$ with a signed message which it can securely and verifiably reveal to receiver(s) $\mathbf{R}$ in $\mathcal{L}_{k''}$. When $\mathbf{S}$ is honest, the receivers will reject a corrupt $\mathbf{M}$ who attempts to reveal a value different from the sender's message. When $\mathbf{M}$ is honest, a corrupt $\mathbf{S}$ is committed to a message when the protocol reaches a so called auxiliary layer, in that, the states of parties in the auxiliary layer fix a message m such

that receivers output m at the end of the protocol. The protocol provides *no guarantees* when both $\mathbf{S}$ and $\mathbf{M}$ are corrupt: adversary can choose the message based on its view until the receiving layer.

Our protocol uses the same blueprint as that of the implementation of IT signature in the YOSO model:

- The sender transfers the signed message to the intermediary.
- The intermediary verifies the validity of the signature with the help of the parties in the subsequent layers.
- If the signature is verified to be valid, the state held by the intermediary defines a message that honest receivers will accept. If the signature check fails, the sender is forced to reveal the actual message to the auxiliary layer, thereby committing to the message.
- The receivers accept the message if the signature is valid. Since the signed message is given only to intermediary, if the intermediary is honest the privacy of the sender's input is preserved until the message is revealed to the intended receiver.

$\mathbf{S}$ with input m computes MAC tags $t_{i,j} = \mathsf{Aut}(k_{i,j}, m)$ with respect to keys k_{ij} for $i \in [n]$ and $j \in [\kappa]$, where κ is a security parameter. The message along with the MAC tags effectively constitute a one time signature of the message which is privately communicated to $\mathbf{M} \in \mathcal{L}_k$. To enable the verification of the signature, $\mathbf{S}$ sends the keys $\{k_{i,j}\}_j$ to $\mathcal{L}_{k+1}$ so that $P_i \in \mathcal{L}_{k+1}$ receives $k_{i,j}$ for all $j \in [\kappa]$. We will refer to the parties in this layer as key-holders.

A corrupt sender may supply malformed signatures to have the receivers reject an honest $\mathbf{M}$. To avoid this, $\mathbf{M}$ and the key-holders check the validity of the MACs provided by $\mathbf{S}$. This is achieved using cut and choose: each key-holder P_i announces a random subset $\mathrm{IND}_i \subseteq [\kappa]$ on which both P_i and the sender reveal their "versions" of $k_{i,j}$. The intermediary validates the MAC using the key revealed by the sender for each $j \in \mathrm{IND}_j$. If any of the checks fail, $\mathbf{M}$, now convinced that $\mathbf{S}$ is corrupt, complains forcing $\mathbf{S}$ to reveal m to parties in layer $\mathcal{L}_{k'}$. Otherwise, if some key-holder P_i's keys are different from the sender's, the vote of P_i is counted towards accepting the intermediary's message.

The cut and choose protocol ensures that a corrupt $\mathbf{S}$ will either be detected by the $\mathbf{M}$ forcing it to reveal the message or every honest key-holder will vote for the intermediary's message. To see this, consider any honest key-holder P_i. Consider the event in which the sender does not disqualify the sender while doing the MAC check for $k_{i,j}, j \in \mathrm{IND}_i$ and the vote of P_i is not counted towards intermediary's message. This occurs only if MAC $t_{i,j}$ sent by the corrupt sender is consistent for all $j \in \mathrm{IND}_i$ and inconsistent for all $j \notin \mathrm{IND}_i$. Since IND_i is chosen uniformly at random unknown to $\mathbf{S}$, this occurs with probability that is negligible in κ. Thus, an honest $\mathbf{M}$ will successfully call out a corrupt $\mathbf{S}$, in which case $\mathbf{S}$ reveals the message during the checking phase, or the message of $\mathbf{M}$ will be accepted by a receiver that accepts the message if a strict majority of key-holders vote for the message. Finally, a corrupt $\mathbf{M}$ that uses a value $m' \neq m$ will fail to receive a vote from any of the honest key-holders by the unforgeability of MAC, causing the receiver to reject the message.

Porting the above blueprint into a layered protocol, we encounter all the inherent challenges of any framework dealing with dynamic committees (including YOSO). In particular, the sender cannot be "present" to reveal the keys $\{k_{i,j}\}_{j\in \mathrm{IND}_i}$ after the key-holder P_i broadcasts the set IND_i. Clearly, **S** has already used up its communication round to send, among other things, the keys $\{k_{i,j}\}_{j\in[\kappa]}$ to key-holder P_i. We solve these challenges in the same way as all constructions dealing with dynamic committees: parties who need to speak multiple times cache their messages using future broadcast, and later layers conditionally reveal only those messages that are required to be opened.

However, the layered model brings up many subtle challenges that are not encountered in YOSO. It is crucial that the message and MACs are chosen by **S** *before* each honest key-holder P_i reveals their set IND_i. Otherwise, a corrupt **S** can choose the MAC tags $t_{i,j}$ to be consistent with $k_{i,j}$ for all $j \in \mathrm{IND}_i$ and inconsistent for all $j \notin \mathrm{IND}_i$. Since none of our communication primitives commit the sender to their messages, this is possible only if the layer in which the key-holders make their random sets public comes after the layer in which **M** is placed. Note, that this is not the case in YOSO: thanks to the *assumed* channels to the future, the messages from **S** to **M** are fixed at the time of sending. This allows **M**, to perform the MAC checks locally after the random sets and the keys in those sets are revealed, and only speak after this check. In the layered protocol, on the other hand, the MAC checks–a non-linear operation–need to be carried out by a future layer that learns IND_i and has access to cached values of the message and MAC tags $m, t_{i,j}$ provided by **M**, as well as the keys $k_{i,j}$ provided by **S**. We construct this by having the sender's keys made public first, and then using them to securely compute the appropriate linear combination $m, t_{i,j}$ that yields 0 if and only if the verification succeeds. We crucially use the fact that m is not revealed in this secure computation when **S** and **M** are honest.

Finally, because the receivers perform the MAC checks using the keys provided by the key-holders, to ensure security against a dishonest **M**, the message and MACs are crucially revealed by **M** *before* the keys are revealed. We encountered the same challenge in future broadcast, and address it similarly: the message and MACs published before the keys, by adding some additional *dummy* layers.

Distributed Commitment. Our information theoretic signature primitive commits a dishonest sender to their input *only* when the intermediary is honest. In contrast, the distributed commitment primitive commits a sender to their input unconditionally, by leveraging the honest majority in each layer. More specifically, we realize the following functionality: a party from an initial layer can commit to one (or multiple) values towards many future layers, who can then decide whether to open (any linear combination of) the committed values. Commitments can be opened publicly (towards all parties in one layer) or privately towards a single party. The opening of commitments made by an honest party never fails, but the adversary can prevent the opening of commitments of dishonest parties.

The protocol we present follows a natural blueprint in which the committer produces a Shamir sharing of their input, and uses one instance of our information theoretic signature to sign each share; in this step, it is crucial that the intermediaries involved in the signing of each share are distinct parties in the same layer: this guarantees that at most t of them are corrupted, preserving the privacy of the committed message. Furthermore, there are at least $n - t \geq t + 1$ honest intermediaries, and the information theoretic signature primitive guarantees that their shares will be accepted. Therefore, even if the committer is corrupted, these $t + 1$ shares uniquely determine a committed value. By controlling the dishonest shares, a corrupt dealer can still open the value $\perp$, or in other words refuse to open their commitment. Details are in Sect. 7.

Verifiable Secret Sharing. The last ring in this chain of primitives with increasingly strong commitment guarantees is verifiable secret sharing (VSS). This can be thought of as a distributed commitment primitive in which *even a dishonest* committer cannot prevent their commitment from being opened.[1]

As none of the few VSS constructions in the setting of dynamic committees can be adapted to the layered setting with $t < n/2$, we design an entirely new protocol. Our construction makes black-box use of a linearly homomorphic (for commitments generated by the same party) distributed commitment primitive to construct a full-fledged VSS. We reduce the security of VSS to that of the underlying distributed commitment perfectly: meaning that this construction does not introduce any further error probability. To the best of our knowledge, this simple black-box compiler is of independent interest even in the non-layered setting, where parties can send messages in multiple rounds. We provide a description of its non-layered version below. The layered version of the protocol is presented in Sect. 8.

An important technical point is that the resulting VSS is homomorphic even across sharings (commitments) dealt by *different* dealers, despite the fact that the underlying distributed commitment protocol is only homomorphic with respect to sharings dealt by the same dealer. This is crucial in our circuit evaluation protocol. The sharing phase works as follows.

1. The dealer uniformly samples a random bi-variate polynomial $F(x, y)$ of degree at most t in each variable conditioned on $F(0, 0) = s$, and sends to each P_i the vertical projection $F(i, y)$. The dealer also commits to (all coefficients of) the polynomial $F(x, y)$ via the distributed commitment primitive.
2. Each P_i commits to (all coefficients of) the received polynomial $v_i(y)$ via the distributed commitment primitive.
3. Using the homomorphism of the distributed commitments, parties privately open towards each party P_j the j-th horizontal projection $F(x, j)$ committed

[1] We are faced with a dichotomy of languages: a VSS protocol can be thought of as a *strong distributed commitment*. From this perspective, the party providing input is a *committer*, producing *commitments* that can be *opened*. More often, the language of *secret sharing* is used. Here, the party providing input is a *dealer*, producing *sharings* that can be *reconstructed*. We oscillate between these two abstractions.

by the dealer, and the j-th evaluation point $v_i(j)$ of every P_i's committed polynomial.

4. If the private reconstructions do not match, i.e. $F(i,j) \neq v_i(j)$ (or both fail), then P_j broadcasts a complain message $(\mathsf{complain}, i, j)$.
5. For each complaint $(\mathsf{complain}, i, j)$ parties publicly open the commitments to the two points, which we denote $\overline{F}(i,j)$ from the dealer and $\overline{v}_i(j)$ from P_i. If the dealer's opening fails, disqualify the dealer. And if P_i's opening does not match the dealer's point (or fails), then add P_i to a global set $\mathcal{I}$ of parties, and publicly open the projection $F(i,y)$. The dealer is disqualified if $|\mathcal{I}| > t$.

If the dealer is not disqualified, each party $P_i \notin \mathcal{I}$ has a committed polynomial $v_i(y)$ which is the same as the vertical projection of the polynomial $F'(x,y)$ committed by the dealer in the first step. If this was not the case, the two polynomials would differ in at least $t+1$ points, and therefore one honest party P_j would have complained in Step 4, a complaint that would have led to including P_i to the set $\mathcal{I}$, a contradiction. Moreover, if the dealer is honest, it is easy to see that $F'(x,y) = F(x,y)$. Further note that the sharing phase is homomorphic across different dealers, since the distributed vertical polynomials used for reconstruction are dealt by each of the recipients, or publicly known.

To ensure reconstruction (i.e. to open the dealer' commitment), parties simply open the vertical projections committed by each $P_i \notin \mathcal{I}$, and use any $t+1$ polynomials $\{v_{i_1}(y), \ldots, v_{i_{t+1}}(y)\}$ that are either reconstructed in this step, or were revealed in Step 5, to interpolate the original secret.

Multi-Party Computation. The circuit evaluation protocol proceeds gate-by-gate, with gates in the same layer of the circuit being evaluated in parallel. We maintain the following invariant: there is layer of parties, say $\mathcal{L}_0$, holding a state encoding the input values a and b for each gate g in a certain level of the circuit C and a layer $\mathcal{L}_k$ holding a state encoding the output of g. The state encoding an input value a to a gate g has three components:

1. A (t,n)-Shamir sharing of a: each party $P_i^0 \in \mathcal{L}_0$ holds share $a_i = f_a(i)$.
2. A (t,n)-Shamir sharing of a random value r, which is wasted to compute multiplication gates. This can be easily achieved using our VSS protocol, by letting all parties in a layer verifiably share a random value and taking the sum.
3. Verifiable sharings (aka commitments) to the coefficients of the polynomial $f_a(x)$ used for the Shamir Sharing of a.

Input Gates. Each client needs to produce the state described above (encoding their input) towards the layer tasked with computing the first level of gates in the circuit. To achieve this, a client with input m simply commits via VSS towards two different layers to each coefficient of a degree t polynomial $f(x)$ with $f(0) = m$. Then, the intermediary layer reconstructs each $f(i)$ (exploiting the linearity of the VSS) towards party P_i in the second layer.

Addition Gates. Since the invariant state is linear, addition gates can be performed locally. However, as captured by the parallel functionality $\mathcal{F}_{\mathsf{VSS}}$ in Sect. 8,

our VSS only allows to add sharings made by dealers in the *same layer*. The evaluation of multiplication gates, on the other hand, requires several rounds of interaction. We need the output of the addition gates to also be processed in the same number of rounds, and to avoid introducing extra machinery, we simply multiply the output of each addition gate by 1.

*Multiplication Gates.*The multiplication follows the classical blueprint of [15], adapted to the layered setting and the described state invariant. Suppose that layer $\mathcal{L}_0$ holds the states for the inputs a and b of the multiplication gate g. Each party P_i^0 locally multiplies their Shamir shares to compute $c_i = a_i \cdot b_i$. To reproduce the state for the value c_i, the party simply performs the client input routine described above. Note that $c = a \cdot b$ is a linear combination of the c_i's, so to compute the invariant state for c it is enough that the parties produce correct states for each c_i.

To show that the party P_i^0 actually produced a state for the right value c_i, the party provides a distributed zero-knowledge proof. Assume that the commitments to the coefficients of $f_a(x)$ and $f_b(x)$ are also available towards any auxiliary layer $\mathcal{L}_{k'}$ for $0 \leq k' \leq k$. Then, P_i^0 produces new verifiable secret sharings to $\widehat{a}_i, \widehat{b}_i$ and $\widehat{c}_i$, and proves towards parties in $\mathcal{L}_{k'}$: 1) that $\widehat{a}_i = a_i$, 2) that $\widehat{b}_i = b_i$, and 3) that $\widehat{a}_i \cdot \widehat{b}_i = \widehat{c}_i$. If the proof fails, parties in $\mathcal{L}_{k'}$ simply reveal values a_i and b_i to parties in layer $\mathcal{L}_k$.

For the proof of equality, party P_i^0 verifiably shares $\widehat{r}_i$ towards all future layers until $\mathcal{L}_k$. Then, parties in some later layer (who also hold commitments to a_i, r_i and $\widehat{a}_i$) sample a public common random value ρ, and the values $r_i + \rho a_i$ and $\widehat{r}_i + \rho\widehat{a}_i$ are opened publicly. If they are different, the proof fails. Note that if $a_i \neq \widehat{a}_i$ or $r_i \neq \widehat{r}_i$, there is only one value ρ that makes the proof succeed.

The proof of correct multiplication showing $\widehat{a}_i \cdot \widehat{b}_i = \widehat{c}_i$ is an adaptation from [9]. For this, party P_i^0 samples a random value β and verifiably shares β and $b\beta$. Now, parties in some later layer (who also hold commitments to $\widehat{a}_i, \widehat{b}_i$ and $\widehat{c}_i$) sample a random value ρ, and publicly open the value $\rho' = \rho a + \beta$. Finally, a later layer publicly opens and checks that $\rho' b - b\beta - rc = 0$. Note that if $\widehat{a}_i \cdot \widehat{b}_i \neq \widehat{c}_i$, only one value ρ makes the proof succeed, which happens with negligible probability. See details in Sect. 9.

3 Preliminaries and Model

A layered MPC protocol can be viewed as a special case of standard MPC with a general adversary structure, specialized in the following way: 1) the interaction pattern is defined by a layered graph, and 2) the adversary can corrupt at most t parties in each layer.

Definition 1 (Layered Protocol). *Let n, t, d be positive integers. An (n, t, d)-*layered protocol *is a synchronous protocol Π over secure point-to-point channels and a broadcast channel, with the following special features.*

- **Parties.** *There are* $N = n(d+1)$ *parties partitioned into* $d+1$ *layers* $\mathcal{L}_i$, $0 \leq i \leq d$, *where* $|\mathcal{L}_i| = n$. *Parties in the first layer* $\mathcal{L}_0$ *and the last layer* $\mathcal{L}_d$ *are referred to as* input clients *and* output clients, *respectively.*
- **Interaction pattern.** *The interaction consists of* d *rounds, where in round* i *parties in* $\mathcal{L}_{i-1}$ *may send messages to parties in* $\mathcal{L}_i$ *over secure point-to-point channels. We additionally allow each party in* $\mathcal{L}_{i-1}$ *to send a broadcast message to all parties in* $\mathcal{L}_i$.
- **Functionalities.** *We consider functionalities* f *that take inputs from input clients and deliver outputs to output clients.*
- **Adversaries.** *We consider adversaries who may corrupt any number of input and output clients, and additionally corrupt* t *parties in each intermediate layer* $\mathcal{L}_i$, $0 < i < d$. *We consider active, rushing and non-retroactive adaptive adversaries*[2].

We say that a protocol Π *is a layered protocol for* $\mathcal{F}$ *if it UC-realizes* $\mathcal{F}$ *in the setting of general adversary structures* [4,5,18]. *We consider statistical security (with guaranteed output delivery) where* κ *denotes the security parameter and* $\mathbb{F}$ *is a finite field of size* $1/\mathsf{negl}(\kappa)$.

A Note on Synchronous Universal Composability. We are interested in realizing functionalities f that take input from the input clients in layer $\mathcal{L}_0$ by default and deliver outputs to the output clients in the last layer (layer $\mathcal{L}_k$) of a layered network. We develop a synchronous protocol for computing general functionalities in the UC model. The standard UC model is asynchronous by default, but there have been a number of works that modeled synchronous universally composable frameworks [5,20,21,23,25], and our protocols can be described in any of those models. Very roughly, one usually considers a clock functionality that keeps track of the activation pattern and also advances a round whenever a round-robin of activations happen. For simplicity and ease of exposition, our descriptions omit the clock and it is understood that protocols and ideal functionalities know the current round number.

On Ideal Broadcast Channels. Without assuming broadcast, information theoretic MPC for general functionalities is provably impossible for $t \geq n/3$ [22]. We assume a broadcast channel that broadcasts messages from a certain layer to parties in all later layers. Note that this is equivalent to assuming broadcast to the immediate next layer in the honest majority setting, since broadcast messages can be propagated through layers via bilateral channels and sequential majority decisions, at the price of an additional quadratic factor in communication.

[2] For simplicity, we consider the notion of "non-retroactive" (see [8], Definition 3) adaptive adversaries, who chooses at each round r a set of up to t parties from layer $\mathcal{L}_r$ to corrupt. Since our protocols are information-theoretic, we conjecture that they are also secure against the stronger notion of retroactive-adaptive adversaries that can corrupt parties in previous layers.

4 Future Messaging

Future Messaging Functionality. As discussed in the technical overview, a basic challenge in the layered setting is for a party in a layer $\mathcal{L}_0$ to communicate securely with parties in a later layer $\mathcal{L}_k$ for $k > 0$. If $k = 1$, communication happens via provided point-to-point secure channels. However, if $k \geq 2$ secure channels must be emulated via an appropriate layered protocol. Our parallel future messaging functionality allows each party in layer $\mathcal{L}_0$ to send a message to each party in a layer $\mathcal{L}_k$ for any $k \geq 1$. We remark that the guarantees provided by the functionality are quite weak, as the adversary is allowed to fix the messages from corrupted parties in $\mathcal{L}_0$ to honest parties in $\mathcal{L}_k$ *after* having received the messages sent by honest parties in $\mathcal{L}_0$ to corrupted parties in $\mathcal{L}_k$.

Parallel Future Messaging Functionality $\mathcal{F}^k_{\mathsf{FutureMsg}}$

Public Parameters. Senders $\mathbf{S}_1, \ldots, \mathbf{S}_n \in \mathcal{L}_0$, receivers $\mathbf{R}_1, \ldots, \mathbf{R}_n \in \mathcal{L}_k$ where $k \geq 1$. The domain $M_{i,j}$ of message from $\mathbf{S}_i$ to $\mathbf{R}_j$.

Secret Inputs. For each $\mathbf{S}_i$ messages $m_{i,j} \in M_{i,j}$ for $j \in [n]$ to be sent to each $\mathbf{R}_j$.

Layer $\mathcal{L}_0$:

- For each honest $\mathbf{S}_i \in \mathcal{L}_0 \setminus \mathcal{I}_0$ and each $\mathbf{R}_j \in \mathcal{L}_k$, receives message $m_{i,j}$ from $\mathbf{S}_i$ to $\mathbf{R}_j$.

Layer $\mathcal{L}_k$:

- For each honest $\mathbf{S}_i \in \mathcal{L}_0 \setminus \mathcal{I}_0$ and corrupt $\mathbf{R}_k \in \mathcal{I}_k$, forward $m_{i,j}$ to the (ideal) adversary.
- For each corrupt $\mathbf{S}_i \in \mathcal{I}_0$ and each $\mathbf{R}_j \in \mathcal{L}_k$, receive from the (ideal) adversary the message $m_{i,j}$ that $\mathbf{S}_i$ wants to send to $\mathbf{R}_j$.
- For each $\mathbf{S}_i \in \mathcal{L}_0$ and $\mathbf{R}_j \in \mathcal{L}_k$, send $m_{i,j}$ to $\mathbf{R}_j$ as message from $\mathbf{S}_i$.

Future Messaging Protocol. Our protocol realizing $\mathcal{F}^k_{\mathsf{FutureMsg}}$ is described informally in Sect. 2. Due to space constraint, a formal description of the protocol and a proof of lemma 1 can be found in the full version of this paper.

Lemma 1. *If* $(\mathsf{Sh}, \mathsf{Rec})$ *is a* $(\mathcal{D}, t, \mathsf{negl}(\kappa))$*-robust* (t, n)*-secret-sharing scheme, then for any* $k' \in [k-1]$ *the* (n, t, k)*-layered protocol* $\Pi^k_{\mathsf{FutureMsg}}$ *realizes functionality* $\mathcal{F}^k_{\mathsf{FutureMsg}}$ *with* $(\mathsf{negl}(\kappa), t)$*-statistical security in the* $\left(\mathcal{F}^{k'}_{\mathsf{FutureMsg}}, \mathcal{F}^{k-k'}_{\mathsf{FutureMsg}}\right)$*-hybrid model.*

5 Future Broadcast

Future Broadcast Functionality. This primitive allows a sender to broadcast any linear combination of some input values to a later layer (or one single party in a later layer), and guarantees that 1) the messages (or their wanted linear combination) remains secret until the decision to reveal them is taken, and 2) even when the sender is dishonest, all honest parties in the receiving layer agree on a single message. The decision to reveal a message (or not) can be taken by honest parties in a certain layer depending on public information. Again, this primitive provides no commitment guarantees when the sender is corrupt, as adversary is allowed to modify the message up until the moment of delivery.

Linear Future Broadcast Functionality $\mathcal{F}^{k}_{\mathsf{FutureBC}}$

Public Parameters. Sender $\mathbf{S} \in \mathcal{L}_0$. Auxiliary layer $\mathcal{L}_k$ deciding which messages are revealed. Layer $\mathcal{L}_{k'}$ onto which the messages are broadcast. The domain M of the messages from $\mathbf{S}$. The maximum number of messages ℓ to be broadcast by each sender.

Secret Inputs.

- Messages $(m_1, \ldots, m_\ell) \in M$ from $\mathbf{S}$ to be broadcast to $\mathcal{L}_{k'}$.
- A public value (L, r) agreed up on by all honest P_j^k, where
  - $L : M^\ell \to M$ is a linear operator.
  - $r \in \mathcal{L}_{k'} \cup \{\mathcal{L}_{k'}\}$ is the intended recipient of $L(m_1, \ldots, m_\ell)$: either some specific party $P_s^{k'}$ in $\mathcal{L}_{k'}$ or all the parties in layer $\mathcal{L}_{k'}$.

Layer $\mathcal{L}_0$: If $\mathbf{S} \notin \mathcal{I}_0$ receive messages $(m_1, \ldots, m_\ell)$ from $\mathbf{S}$.

Layer $\mathcal{L}_k$: For each honest $P_i^k \in \mathcal{L}_k \setminus \mathcal{I}_k$, receive *the same* input (L, r).

Layer $\mathcal{L}_{k'-1}$: If $r = \mathcal{L}_{k'}$, forward $(L, L(m_1, \ldots, m_\ell))$ to the (ideal) adversary.

Layer $\mathcal{L}_{k'}$:

- If $r = P_s^{k'} \in \mathcal{I}_{k'}$, and $\mathbf{S} \notin \mathcal{I}_0$, forward $\big(L, L(m_1, \ldots, m_\ell)\big)$ to the (ideal) adversary.
- If $\mathbf{S} \in \mathcal{I}_0$ receive from the (ideal) adversary message $l_{\mathcal{A}}$ that $\mathbf{S}$ wants to broadcast.
- If $r = P_s^{k'} \notin \mathcal{I}_{k'}$, and $\mathbf{S} \notin \mathcal{I}_0$, send the value $L(m_1, \ldots, m_\ell)$ to $P_s^{k'}$.
- If $r = P_s^{k'} \notin \mathcal{I}_{k'}$, and $\mathbf{S} \in \mathcal{I}_0$, send the value $l_{\mathcal{A}}$ to $P_s^{k'}$.
- If $r = \mathcal{L}_{k'}$, and $\mathbf{S} \notin \mathcal{I}_0$, send the value $L(m_1, \ldots, m_\ell)$ to all parties in $\mathcal{L}_{k'}$.
- If $r = \mathcal{L}_{k'}$, and $\mathbf{S} \in \mathcal{I}_0$, send the value $l_{\mathcal{A}}$ to to all parties in $\mathcal{L}_{k'}$.

Future Broadcast Protocol. The first solution that comes to mind to realize $\mathcal{F}_{\mathsf{FutureBC}}$ is the following: to broadcast a message m onto layer $\mathcal{L}_{k'}$, simply provide, using future messaging, a robust sharing of s to layer $\mathcal{L}_k$, and ask parties in layer $\mathcal{L}_k$ to broadcast their shares using the provided broadcast channels. This construction is secure if there is only one recipient (notice that we do not provide any guarantees for dishonest recipients). However, when the robust shares

are broadcast, this construction is insecure, because *if the the dealer is honest* a rushing adversary can wait to see the shares broadcast by honest parties in $\mathcal{L}_k$ before broadcasting shares of corrupted parties. As briefly mentioned in Sect. 2, the robustness guarantees provided by the secret sharing scheme are insufficient in this scenario, because the corrupted shares can depend on the honest shares. For instance, in the robust secret sharing scheme outlined in Sect. 2, a rushing adversary would be able to make the reconstruction fail by first observing the keys broadcast by honest parties, and only then computing new valid MAC tags (with respect to these observed keys) for new arbitrary values. To avoid this, we instead have the dealer set up n independent robust sharings of m and provide them to $\mathcal{L}_k$ using future messaging. Then, each of these states are reconstructed towards distinct parties in some auxiliary layer (no rushing attack applies if the recipient is honest), and then we leverage the honest majority in the auxiliary layer to agree on a single value: each party in the auxiliary layer simply broadcasts the value they have reconstructed. In this construction, the adversary learns the broadcasted value one layer before the intended target layer. Note, our functionality matches this protocol since the message is leaked to the adversary in the auxiliary layer.

We get around this shortcoming by ensuring that the adversary gains no advantage by having learned the broadcast message one layer in advance. This is arranged by having all the other protocols run in parallel with the future broadcast completely ignore the auxiliary layer. Consequently, the view of the adversary in any set of layers including the auxiliary layer and the output layer of a future broadcast is identical to that in the subset excluding the auxiliary layer. We formally describe the protocol below, and argue about its security, captured by Lemma 2, only in the full version of this paper, due to space constraints.

Future Broadcast Protocol $\Pi^k_{\mathsf{FutureBC}}$

Public Parameters. Sender $\mathbf{S} \in \mathcal{L}_0$. Receiving layer $\mathcal{L}_k$. The domain M of $\mathbf{S}$'s messages.

Secret Inputs. Messages $(m_1, \ldots, m_\ell) \in M$ from sender $\mathbf{S}$.

Resources. A $(\mathcal{D}, t, \mathsf{negl}(\kappa))$-robust (t, n)-secret sharing scheme $(\mathsf{Sh}, \mathsf{Rec})$ with message space $\mathcal{M} = M$, randomness space $\mathcal{R}$ and share space $\mathcal{S}$; functionalities $\mathcal{F}^k_{\mathsf{FutureMsg}}$, $\mathcal{F}^{k'-k-1}_{\mathsf{FutureMsg}}$, and $\mathcal{F}^{k'-k}_{\mathsf{FutureMsg}}$.

Layer $\mathcal{L}_0$: $\mathbf{S} \in \mathcal{L}_0$ does

- Sample $(r_{1,j} \ldots, r_{\ell,j}) \leftarrow_{\mathcal{D}} \mathcal{R}$ for all $j \in [n]$.
- Compute $(m^1_{i,j}, \ldots, m^n_{i,j}) \leftarrow \mathsf{Sh}(m_i, r_{i,j})$ for each $i \in [\ell]$ and $j \in [n]$.
- For each $r \in [n]$, set m_r as the message [a] to P^k_r in $\mathcal{F}^k_{\mathsf{FutureMsg}}$, where m_r is the matrix of values $m^r_{i,j}$ for all $i \in [\ell]$ and $j \in [n]$.

Subroutine 1: Revealing $L(m_1, \ldots, m_\ell)$ to $P^{k'}_s$:

Layer $\mathcal{L}_k$: Each $P_r^k \in \mathcal{L}_k$ receives (from $\mathcal{F}_{\mathsf{FutureMsg}}^k$) values $\widehat{m_{i,j}^r}$ for all $i \in [\ell]$ and $j \in [n]$ and set input $m_{r,s}$ towards $P_s^{k'}$ in $\mathcal{F}_{\mathsf{FutureMsg}}^{(k'-k)}$ to the value $L\left(\widehat{m_{1,1}^r}, \ldots, \widehat{m_{\ell,1}^r}\right)$.

Layer $\mathcal{L}_{k'}$: Party $P_s^{k'}$ receives (from $\mathcal{F}_{\mathsf{FutureMsg}}^{k'-k}$) values $\widehat{l^r}$ for all $r \in [n]$ and computes their output $m_s \leftarrow \mathsf{Rec}\left(\widehat{l^1}, \ldots, \widehat{l^n}\right)$.

Subroutine 2: Revealing $L(m_1, \ldots, m_\ell)$ to all parties in $\mathcal{L}_{k'}$:

Layer $\mathcal{L}_k$: Each $P_r^k \in \mathcal{L}_k$ receives (from $\mathcal{F}_{\mathsf{FutureMsg}}^k$) values $\widehat{m_{i,j}^r}$ for all $i \in [\ell]$ and $j \in [n]$ and sets input $m_{r,j}$ towards $P_j^{k'-1}$ in $\mathcal{F}_{\mathsf{FutureMsg}}^{(k'-k-1)}$ to $L\left(\widehat{m_{1,j}^r}, \ldots, \widehat{m_{\ell,j}^r}\right)$.

Layer $\mathcal{L}_{k'-1}$: Each $P_j^{k'-1}$ receives (from $\mathcal{F}_{\mathsf{FutureMsg}}^{k'-k-1}$) values $\widehat{l_j^r}$ for all $r \in [n]$ and broadcasts (using the provided ideal broadcast channels) value $l_j \leftarrow \mathsf{Rec}\left(\widehat{l_j^1}, \ldots, \widehat{l_j^n}\right)$.

Layer $\mathcal{L}_{k'}$: Each party in $\mathcal{L}_{k'}$ outputs the value which was broadcast the most times by $\mathcal{L}_{k'-1}$.

[a] Since we are using the parallel $\mathcal{F}_{\mathsf{FutureMsg}}$ functionality with only one sender we drop the indices for clarity.

Lemma 2. *If $t < n/2$ and $(\mathsf{Sh}, \mathsf{Rec})$ is a $(\mathcal{D}, t, \mathsf{negl}(\kappa))$-robust (t, n)-secret-sharing scheme, then for all $k \in [k']$ the (n, t, k')-layered protocol $\Pi_{\mathsf{FutureBC}}^{k,k'}$ realizes $\mathcal{F}_{\mathsf{FutureBC}}^{k,k'}$ with $(\mathsf{negl}(\kappa), t)$-statistical security in the $\mathcal{F}_{\mathsf{FutureMsg}}$-hybrid model.*

6 Information Theoretic Signature

Information Theoretic Signature Functionality. This functionality allows a sender to entrust an intermediary with a message that a later layer can then reliably reveal to a receiver. The functionality ensures that an honest intermediary will always be able to convince the honest receiver to accept the message that the intermediary received from the sender. If the sender is honest, the functionality ensures that a potentially corrupt intermediary cannot convince an honest receiver to accept a distinct message than the one it received from the sender. When both the sender and intermediary are corrupt, the functionality provides no guarantees. Notice that in the layered setting, the message is not *actually* revealed by the intermediary itself, but rather by a later layer holding an appropriate state provided by the intermediary.

The functionality described below describes a generalization useful in our later constructions in which the message receiver(s) can be selected from a set of possible layers $\{\mathcal{L}_{r_1}, \ldots, \mathcal{L}_{r_v}\}$ by one among a set of auxiliary layers $\{\mathcal{L}_{k_1}, \ldots, \mathcal{L}_{k_v}\}$. Note, the sender is forced to commit to the message by the first among the auxiliary layers.

Information Theoretic Signature Functionality $\mathcal{F}_{\mathsf{ITSig}}^{k_1,\ldots,k_w}$

Public Parameters. Sender $\mathbf{S} \in \mathcal{L}_0$. Intermediary $\mathbf{M} \in \mathcal{L}_k$ for $k \geq 1$. Auxiliary layers $\mathcal{L}_{k_1}, \ldots, \mathcal{L}_{k_w}$ for $k_w > \ldots > k_1 \geq k + 8$. Candidate receiver layers $\mathcal{L}_{r_1}, \ldots, \mathcal{L}_{r_v}$, $r_1 > k_1$. The $\mathbb{F}$-vector space M, domain of $\mathbf{S}$'s messages.

Inputs. From $\mathbf{S}$, secret messages $(m_1, \ldots, m_\ell) \in M^\ell$. The same public input (L, r) from all honest parties from a unique auxiliary layer $\mathcal{L}_{k_i}$, $i \in [w]$.

- $L : M^\ell \to M$ is the linear operator.
- $r \in \mathcal{L}_{r_i} \cup \{\mathcal{L}_{r_i}\}$ for some $i \in [v]$ is the intended recipient of $L(m_1, \ldots, m_\ell)$: either a specific party in $\mathcal{L}_{r_i}$ or all parties in layer $\mathcal{L}_{r_i}$.

Layer $\mathcal{L}_0$:
– If $\mathbf{S} \notin \mathcal{I}_0$, receive messages $(m_1, \ldots, m_\ell)$ from $\mathbf{S}$.

Layer $\mathcal{L}_k$:

– If $\mathbf{M} \in \mathcal{I}_k$ is corrupt, reveal $(m_1, \ldots, m_\ell)$ to the (ideal) adversary.

Layer $\mathcal{L}_{k_1}$: If $\mathbf{M} \notin \mathcal{I}_k$ is honest and $\mathbf{S} \in \mathcal{I}_0$ is corrupt, receive messages $(m_1, \ldots, m_\ell)$ from the (ideal) adversary.

Layer $\mathcal{L}_{k_i}$ for $i \in [w]$:

– Receive (the same) message (r, L) or $\perp$ from each honest party in $\mathcal{L}_{k_i}$. Ignore messages from $\mathcal{L}_{k_{i'}}, i' > i$ if (r, L) is received in $\mathcal{L}_{k_i}$.

Layer $\mathcal{L}_{r_i-3}$ for $i \in [v]$:

– If $r = \mathcal{L}_{r_i}$ (i.e., specifically, the set of receivers is the entire $\mathcal{L}_{r_i}$ instead of an individual receiver $P_j^{r_i} \in \mathcal{L}_{r_i}$ for some $j \in [n]$), and both $\mathbf{M} \notin \mathcal{I}_k$ and $\mathbf{S} \notin \mathcal{I}_0$ are honest, deliver $L(m_1, \ldots, m_\ell)$ to the ideal adversary.

Layer $\mathcal{L}_{r_i}$ for $i \in [v]$:

1. If $r = \mathcal{L}_{r_i}$ or $r = P_j^{r_i} \in \mathcal{L}_{r_i}$:
   - If $\mathbf{M}$ is honest deliver $L(m_1, \ldots, m_\ell)$ to r.
   - If $\mathbf{M}$ is corrupt then:
     - If $\mathbf{S}$ is corrupt receive m' from the (ideal) adversary and forward it to r.
     - If $\mathbf{S}$ is honest receive boolean $\mathsf{reveal} \in \{0, 1\}$ from the (ideal) adversary. If $\mathsf{reveal} = 1$ then deliver $L(m_1, \ldots, m_\ell)$ to r.

Information Theoretic Signature Protocol. For improved legibility, we describe a protocol realizing the above functionality for $\ell = 1$, $w = 1$ and $v = 1$, meaning the auxiliary layer is fixed to $\mathcal{L}_{k'}$ and receiver layer is $\mathcal{L}_{k''}$. We only consider the more involved construction in which the message is revealed to *all* the parties in $\mathcal{L}_{k''}$. We will describe later how the protocol can be modified to realize the general functionality for arbitrary, finite ℓ, w and v. The security of the protocol, captured by Lemma 3, is proven in the full version of this paper, due to space constraints. The construction follows the description presented in the technical overview.

Information Theoretic Signature Protocol $\Pi^{k'}_{\mathsf{ITSig}}$

Public Parameters. Sender $\mathbf{S} \in \mathcal{L}_0$, intermediary $\mathbf{M} \in \mathcal{L}_k$, a committing layer k' and receiver layer $\mathbf{R} = \mathcal{L}_{k''}$ where $1 \leq k < k' < k''$. The message domain M which is a finite field. A security parameter κ.

Secret Inputs. **S** has a message $m \in M$ to be sent to **R** via **M**.

Resources. Functionality $\mathcal{F}_{\mathsf{FutureMsg}}$; functionality $\mathcal{F}_{\mathsf{FutureBC}}$; a message authentication code in which key $(a, b) \leftarrow_{\$} M^2$ and $\mathsf{Aut}(m, (a, b)) = a \cdot m + b$ for any message in M.

Layer $\mathcal{L}_0$: The sender **S** does:

1. Sample keys $k_{i,j} \leftarrow_{\$} \mathbb{F}^2$ for each $i \in [n]$, $j \in [\kappa]$, and compute $t_{i,j} = \mathsf{Aut}(k_{i,j}, m)$ for all $i \in [n]$ and $j \in [\kappa]$.
2. Send $(m, \{t_{i,j}\}_{i\in[n],j\in[\kappa]})$ to $\mathbf{M} \in \mathcal{L}_k$ using $\mathcal{F}_{\mathsf{FutureMsg}}$.
3. Send $\{k_{i,j}\}_{i\in[n],j\in[\kappa]}$ to each $P_i \in \mathcal{L}_{k+1}$ using $\mathcal{F}_{\mathsf{FutureMsg}}$ for all $i \in [n]$.
4. Invoke $\mathcal{F}_{\mathsf{FutureBC}}$ with $k_{i,j}$ as input and $\mathcal{L}_{k+2}$ for all $i \in [n]$ and $j \in [\kappa]$.
5. Invoke $\mathcal{F}_{\mathsf{FutureBC}}$ with m as input and $\mathcal{L}_{k+6}$ as auxiliary layer.

Layer $\mathcal{L}_k$: The intermediary **M** does:

1. Invoke $\mathcal{F}_{\mathsf{FutureBC}}$ with input $(m, t_{i,j})$ and $\mathcal{L}_{k+4}$ as auxiliary layer for all $i \in [n]$ and $j \in [\kappa]$.
2. Invoke $\mathcal{F}_{\mathsf{FutureBC}}$ with input m and $\mathcal{L}_{k'}$ as auxiliary layer.
3. Invoke $\mathcal{F}_{\mathsf{FutureBC}}$ with input $t_{i,j}$ and $\mathcal{L}_{k'}$ as auxiliary layer for all $i \in [n]$ and $j \in [\kappa]$.

Layer $\mathcal{L}_{k+1}$: Each $P_i^{k+1} \in \mathcal{L}_{k+1}$ does:

1. Choose a random subset $\mathrm{IND}_i \subset [\kappa]$ of size $\kappa/2$, and broadcasts IND_i.
2. Broadcast $k_{i,j}$ for all $j \in \mathrm{IND}_i$
3. Invoke $\mathcal{F}_{\mathsf{FutureBC}}$ with $k_{i,j}$ as input and $\mathcal{L}_{k+4}$ as auxiliary layer for all $j \notin \mathrm{IND}_i$.

Layer $\mathcal{L}_{k+2}$:

1. For each $i \in [n], j \in \mathrm{IND}_i$, to $\mathcal{F}_{\mathsf{FutureBC}}$ with **S** as sender, $k_{i,j}$ as message, and $\mathcal{L}_{k+2}$ as auxiliary layer, all parties send $(L \leftarrow (1), r \leftarrow \mathcal{L}_{k+4})$ as input.

Layer $\mathcal{L}_{k+4}$:

1. For each $i \in [n], j \in \mathrm{IND}_i$, each party stores $k_{i,j}$ sent by $\mathbf{S}$ using $\mathcal{F}_{\mathsf{FutureBC}}$ as $\bar{k}_{i,j}$, and that sent by $P_i \in \mathcal{L}_{k+2}$ as $\tilde{k}_{i,j}$.
2. Each party computes and broadcasts

$$\texttt{votes} = \{i \in [n] : \exists j \in \mathrm{IND}_i, \bar{k}_{i,j} \neq \tilde{k}_{i,j}\}.$$

3. For each $i \in [n], j \in \mathrm{IND}_i$, invoke $\mathcal{F}_{\mathsf{FutureBC}}$ with $\mathbf{M}$ as sender, $(m, t_{i,j})$ as message, and $\mathcal{L}_{k+4}$ as auxiliary layer, all parties send $(L \leftarrow (a_{i,j}, b_{i,j}), r \leftarrow \mathcal{L}_{k+4})$, where $(a_{i,j}, b_{i,j}) = \bar{k}_{i,j}$, as input.
4. For each $i \notin \texttt{votes}, j \notin \mathrm{IND}_i$, invoke $\mathcal{F}_{\mathsf{FutureBC}}$ with $P_i \in \mathcal{L}_{k+1}$ as sender, $k_{i,j}$ as message, and $\mathcal{L}_{k+4}$ as auxiliary layer, all parties send $(L \leftarrow (1), r \leftarrow \mathcal{L}_{k''})$ as input.

Layer $\mathcal{L}_{k+6}$:

1. If there exists $i \in [n]$ and $j \in \mathrm{IND}_i$ such that the output of $\mathcal{F}_{\mathsf{FutureBC}}$ with $\mathbf{M}$ as sender and $(m, t_{i,j})$ as message, is non-zero, then set $\texttt{success} = 0$, otherwise set it to 1. Broadcast $\texttt{success}$.
2. If $\texttt{success} = 0$, call $\mathcal{F}_{\mathsf{FutureBC}}$ with $\mathbf{S}$ as sender, m as message, and $\mathcal{L}_{k+6}$ as auxiliary layer, parties send $(L \leftarrow (1), r \leftarrow \mathcal{L}_{k'})$ as input.

Layer $\mathcal{L}_{k'}$:

1. If $\texttt{success} = 0$, each party receives m' as the output of $\mathcal{F}_{\mathsf{FutureBC}}$ with $\mathbf{S}$ as sender, m as message, and $\mathcal{L}_{k+6}$ as auxiliary layer. Each party broadcasts m'.
2. If $\texttt{success} = 1$:
   (a) Invoke $\mathcal{F}_{\mathsf{FutureBC}}$ with $\mathbf{M}$ as sender, m as message, and $\mathcal{L}_{k'}$ as auxiliary layer, parties send $(L \leftarrow (1), r \leftarrow \mathcal{L}_{k''-2})$ as input.
   (b) For each $i \notin \texttt{votes}, j \notin \mathrm{IND}_i$, to $\mathcal{F}_{\mathsf{FutureBC}}$ with $\mathbf{M}$ as sender, $t_{i,j}$ as message, and $\mathcal{L}_{k'}$ as auxiliary layer, parties send $(L \leftarrow (1), r \leftarrow \mathcal{L}_{k''-2})$ as input.

Layer $\mathcal{L}_{k''-2}$:

1. Each party recovers $\tilde{m}$ as the output of $\mathcal{F}_{\mathsf{FutureBC}}$ with $\mathbf{M}$ as sender, m as message, and $\mathcal{L}_{k'}$ as auxiliary layer.
2. For each $i \notin \texttt{votes}$ and $j \notin \mathrm{IND}_j$, each party recovers $\tilde{t}_{i,j}$ as the output of $\mathcal{F}_{\mathsf{FutureBC}}$ with $\mathbf{M}$ as sender, $t_{i,j}$ as message, and $\mathcal{L}_{k'}$ as auxiliary layer.
3. Each party broadcasts $\tilde{m}$ and $\{\tilde{t}_{i,j}\}_{i \in [n], j \in \mathrm{IND}_i}$.

Layer $\mathcal{L}_{k''}$:

1. If $\texttt{success} = 0$, the parties output m' broadcasted by parties in $\mathcal{L}_{k'}$.
2. If $\texttt{success} = 1$:
   (a) For each $i \notin \texttt{votes}, j \notin \mathrm{IND}_i$, each party recovers $\tilde{k}_{i,j}$ as the output of $\mathcal{F}_{\mathsf{FutureBC}}$ with $P_i \in \mathcal{L}_{k+1}$ as sender, $k_{i,j}$ as message, and $\mathcal{L}_{k+4}$ as auxiliary layer
   (b) For each $i \notin [\texttt{votes}]$, and $j \notin \mathrm{IND}_i$, using $\tilde{m}$ and $\tilde{t}_{i,j}$ broadcast by the $\mathcal{L}_{k''-2}$, and $\tilde{k}_{i,j}$, each party checks if $\mathsf{Vfy}(\tilde{t}_{i,j}, \tilde{m}, \tilde{k}_{i,j}) = 1$; if so, $\texttt{votes} \leftarrow \texttt{votes} \cup \{i\}$.
   (c) If $|\texttt{votes}| \geq t + 1$, each party outputs $\tilde{m}$; else outputs $\perp$.

Lemma 3. *If $t < n/2$ and $(\mathsf{Aut}, \mathsf{Vfy})$ is a $(\mathcal{D}, t, \mathsf{negl}(\kappa))$-secure MAC, then the (n, t, k'')-layered protocol Π_{ITSig} realizes functionality $\mathcal{F}_{\mathsf{ITSig}}$ with $\ell = 1$, $w = 1$ and $v = 1$ with $(\mathsf{negl}(\kappa), t)$-statistical security in the $(\mathcal{F}_{\mathsf{FutureMsg}}, \mathcal{F}_{\mathsf{FutureBC}})$-hybrid model.*

Generalizing the construction to implement general $\mathcal{F}_{\mathsf{ITSig}}$. Due to space constraints, we discuss the modifications that are needed to adapt the above protocol to handle a message vector from the sender, support multiple auxiliary layers, and choose among multiple receiver layers only in the full version of this paper. These modifications yield the following result.

Lemma 4. *If $t < n/2$ and assuming a linear $(\mathcal{D}, t, \mathsf{negl}(\kappa))$-secure* MAC*, there exists a protocol Π_{ITSig} realizing functionality $\mathcal{F}_{\mathsf{ITSig}}$ $(\mathsf{negl}(\kappa), t)$-statistical security in the $(\mathcal{F}_{\mathsf{FutureMsg}}, \mathcal{F}_{\mathsf{FutureBC}})$-hybrid model for a security parameter κ.*

7 Distributed Commitment

Distributed Commitment Functionality. We describe a distributed commitment functionality that allows parties to commit to values that can then be opened towards future layers. There might be multiple intermediate layers with the right to open a value. The functionality also allows for linear combinations of values to be opened. If the dealer was corrupted by the adversary at the time of commitment, then when an open request is submitted, the adversary is given the option to open the $\perp$ value (analogously as what happens with a traditional cryptographic commitment, where the dealer can always decide not to open a value). Notice the main differences between $\mathcal{F}_{\mathsf{DistCommit}}$ and $\mathcal{F}_{\mathsf{FutureBC}}$:

1. In $\mathcal{F}_{\mathsf{DistCommit}}$, even when the committer $\mathbf{C}$ is corrupted, then the adversary $\mathcal{A}$ must decide its inputs after some fixed number of layers, unlike in $\mathcal{F}_{\mathsf{FutureBC}}^{k,k'}$ where $\mathcal{A}$ can choose the value of a corrupt $\mathbf{S}$ until the very last moment. In other words, the adversary is *committed* to its inputs in $\mathcal{F}_{\mathsf{DistCommit}}$.
2. Functionality $\mathcal{F}_{\mathsf{DistCommit}}$ provides the ability for *multiple* layers to open the *same* commitments. While this can be achieved by $\mathcal{F}_{\mathsf{FutureBC}}$ when $\mathbf{S}$ is honest,

in $\mathcal{F}_{\mathsf{DistCommit}}$ one has the guarantee that, even when $\mathbf{C}$ is corrupt, if two different layers open the *same* linear combination of commitments, in both cases the opened value will be the *same* (that is, if in both cases the opened value is not $\perp$).

Linear Distributed Commitment Functionality $\mathcal{F}_{\mathsf{DistCommit}}^{k_1,\ldots,k_w}$

Public Parameters. Committer $\mathbf{C} \in \mathcal{L}_0$. Auxiliary layers $\mathcal{L}_{k_i}$ $i \in [w]$ deciding which messages are opened. The domain M of the messages from $\mathbf{C}$. The maximum number of messages ℓ to be committed by $\mathbf{C}$.

Secret Inputs.

- From $\mathbf{C}$ messages $(m_1, \ldots, m_\ell) \in M$ to be committed.
- From each $P_j^{k_i}$ $i \in [w]$ message (L, r):
  - $L : M^\ell \to M$ the linear combination of $\mathbf{C}$'s messages to compute.
  - $r \in \mathcal{L}_{k'} \cup \{\mathcal{L}_{k'}\}$ is the intended recipient of this linear combination: either some specific party in $\mathcal{L}_{k'}$ or all the parties in layer $\mathcal{L}_{k'}$.

Layer $\mathcal{L}_0$: If $\mathbf{C} \notin \mathcal{I}_0$ receive messages $(m_1, \ldots, m_\ell)$ from $\mathbf{C}$.

Layer $\mathcal{L}_{k_i}$ for $i \in [w]$:
- If $\mathbf{C} \in \mathcal{I}_0$ receive from the (ideal) adversary messages $(m_1, \ldots, m_\ell)$ that $\mathbf{C}$ wants to commit.
- For each honest $P_i^{k_i} \in \mathcal{L}_{k_i} \setminus \mathcal{I}_{k_i}$, receive *the same* input (L, r).

Layer $\mathcal{L}_{k'-3}$: If $r = \mathcal{L}_{k'}$, then forward $(L, L(m_1, \ldots, m_\ell))$ to the (ideal) adversary.

Layer $\mathcal{L}_{k'}$:

- If $r = P_s^{k'} \in \mathcal{I}_{k'}$, then forward $\big(L, L(m_1, \ldots, m_\ell)\big)$ to the (ideal) adversary.
- If $\mathbf{C} \in \mathcal{I}_0$ receive from the (ideal) adversary boolean $\mathsf{open} \in \{0, 1\}$.
- If $\mathbf{C} \in \mathcal{I}_0$ and $\mathsf{open} = 1$, or if $\mathbf{C} \notin \mathcal{I}_0$:
  - If $r = P_s^{k'}$, send $L(m_1, \ldots, m_\ell)$ to $P_s^{k'}$.[a]
  - If $r = \mathcal{L}_{k'}$, send $L(m_1, \ldots, m_\ell)$ to all parties in $\mathcal{L}_{k'}$.
- If $\mathbf{C} \in \mathcal{I}_0$ and $\mathsf{open} = 0$:
  - If $r = P_s^{k'}$, send $\perp$ to $P_s^{k'}$.
  - If $r = \mathcal{L}_{k'}$, send $\perp$ to all parties in $\mathcal{L}_{k'}$.

[a] The projection map onto a component of $(m_1, \ldots, m_\ell)$ is linear, so that parties can decide to reconstruct exactly one of $\mathbf{C}$'s inputs.

Distributed Commitment Protocol. The distributed commitment protocol we present takes full advantage of the guarantees provided by $\mathcal{F}_{\mathsf{ITSig}}^{k_1,\ldots,k_w}$ from Sect. 6. To commit to value m, because no single intermediary can be trusted (they could be corrupted), a party creates a (t, n)-shamir sharing of m and then invokes $\mathcal{F}_{\mathsf{ITSig}}^{k_1,\ldots,k_w}$ with a different intermediary (in the same layer) for each share. Intuitively, only the shares corresponding to corrupted intermediaries (at most t) are leaked to the adversary. This is not a problem thanks to the t-privacy of the secret sharing scheme. Furthermore, even a dishonest dealer, or *committer*, is committed to the shares entrusted to honest intermediaries. Since the latter are at least $t + 1$, they determine a unique polynomial and the dealer is now committed to the unique value identified by the honest intermediaries' shares.

Clearly, since $\mathcal{F}_{\mathsf{ITSig}}^{k_1,\ldots,k_w}$ provides no guarantees when both the dealer and an intermediary are corrupt, the reconstruction of the sharing might still fail, as the shares corresponding to dishonest intermediaries can be fixed arbitrarily by the adversary. However, if the reconstruction succeeds, the output of the reconstruction will be the unique value defined by the shares of honest intermediaries, providing the commitment property of our construction. The security of the protocol below, captured in Lemma 5, is proven in the full version of this paper, due to space constraints.

Linear Distributed Commitment Protocol $\Pi_{\mathsf{DistCommit}}^{k_1,\ldots,k_w}$

Public Parameters. Committer $\mathbf{C} \in \mathcal{L}_0$. Auxiliary layers $\mathcal{L}_{k_i}$ for $i \in [w]$ deciding which messages are opened. The domain M of the messages from $\mathbf{C}$. The maximum number of messages ℓ to be committed by $\mathbf{C}$. Latest layer $\mathcal{L}'_k$ onto which messages can be opened.

Secret Inputs. From $\mathbf{C}$ messages $(m_1, \ldots, m_\ell) \in M$.

Resources. Functionality $\mathcal{F}_{\mathsf{ITSig}}^{k_1,\ldots,k_w}$.

Layer $\mathcal{L}_0$: The committer $\mathbf{C}$ does:

- Sample a polynomial $f_i(x)$ of degree at most t such that $f_i(0) = m_i$ and do $(m_{i,1}, \ldots, m_{i,n}) \leftarrow (f_i(1), \ldots, f_i(n))$ for all $i \in [\ell]$.
- Input $(m_{1,j} \ldots, m_{\ell,j})$ to $\mathcal{F}_{\mathsf{ITSig}}^{k_1,\ldots,k_w}[j]$[a] for $j \in [n]$ with intermediary $P_j^k \in \mathcal{L}_k$ for all $j \in [n]$.

Revealing $L(m_1, \ldots, m_\ell)$ to $r \in \mathcal{L}_{k'} \cup \{\mathcal{L}_{k'}\}$:

Layer $\mathcal{L}_{k_i}$ for any $i \in [w]$: Each $P_s^{k_i}$ inputs (L, r) to $\mathcal{F}_{\mathsf{ITSig}}^{k_1,\ldots,k_w}[j]$ for all $j \in [n]$.

Layer $\mathcal{L}_{k'}$: Party $r = P_s^{k'}$ (or all parties in $\mathcal{L}_{k'}$ if $r = \mathcal{L}_{k'}$) does:

- Receive value l_j from $\mathcal{F}_{\mathsf{ITSig}}^{k_1,\ldots,k_w}[j]$ for all j.

- Interpolate a polynomial $\widehat{f}(x)$ through any of the $t+1$ points (i, l_i) for $l_i \neq \perp$ and compute $\widehat{l} = \widehat{f}(x)$ and output l.

[a] This notation is used to identify the n distinct parallel instances of $\mathcal{F}_{\mathsf{ITSig}}^{k_1,\ldots,k_w}[j]$ for $j \in [n]$.

Lemma 5. *If $t < n/2$ then the (n, t, k')-layered protocol $\Pi_{\mathsf{DistCommit}}^{k_1,\ldots,k_w}$ realizes $\mathcal{F}_{\mathsf{DistCommit}}^{k_1,\ldots,k_w}$ with $(0, t)$-statistical security in the $\mathcal{F}_{\mathsf{ITSig}}^{k_1,\ldots,k_w}$-hybrid model.*

8 Parallel Linear VSS

Parallel Linear VSS Functionality. In this section we describe our parallel VSS functionality $\mathcal{F}_{\mathsf{VSS}}^{k_1,\ldots,k_w}$. The functionality can be thought of as a strong distributed commitment functionality: parties in an initial layer $\mathcal{L}_0$ can input (commit to) values, and then later parties (in layers $k_1, \ldots, k_w$) can decide to perform linear operations on the values and reveal them as long as the majority of parties in the layer agree (from which the distributed nature of the commitment). Parties are *strongly committed* to the values they input, in the sense that they cannot abort the opening of these values, or linear combinations of them, at a later time: this is the first big difference between this functionality and $\mathcal{F}_{\mathsf{DistCommit}}$. The second major difference is that parties in layers $k_1, \ldots, k_w$ can perform linear operations on values committed by *different* parties (in the functionality, this is modeled by the linear function L). This stronger linearity property is crucial to perform secure addition and multiplication.

Parallel Linear VSS Functionality $\mathcal{F}_{\mathsf{VSS}}^{k_1,\ldots,k_w}$

Public Parameters. Committers $\mathbf{C}_1, \ldots, \mathbf{C}_n \in \mathcal{L}_0$. Auxiliary layers $\mathcal{L}_{k_i}$ for $i \in [w]$ deciding which messages are opened. The domain M of the messages from $\mathbf{C}_i$ for all $i \in [n]$. The maximum number of messages ℓ to be committed by each committer.

Secret Inputs.

- From each honest $\mathbf{C}_i$ messages $(m_{1,i}, \ldots, m_{\ell,i}) \in M$ to be committed.
- From each $P_j^{k_i}$ $i \in [w]$ message $(L, L_1, \ldots, L_n; r)$:
  - $L_i : M^\ell \to M$ the linear combination of $\mathbf{C}_i's$ inputs to compute.
  - $L : M^n \to M$ the linear combination of these linear combinations to compute.
  - $r \in \mathcal{L}_{k'} \cup \{\mathcal{L}_{k'}\}$ is the intended recipient of this linear combination: either some specific party in $\mathcal{L}_{k'}$ or all the parties in layer $\mathcal{L}_{k'}$.

Layer $\mathcal{L}_0$: For each $\mathbf{C}_i \notin \mathcal{I}_0$ receive messages $(m_{1,i}, \ldots, m_{\ell,i})$ from $\mathbf{C}_i$.

Layer $\mathcal{L}_{k_i}$ for $i \in [w]$:
- For each $\mathbf{C}_i \in \mathcal{I}_0$ receive from the (ideal) adversary messages $(m_{1,i}, \ldots, m_{\ell,i})$ that $\mathbf{C}_i$ wants to commit.
- For each honest $P_i^k \in \mathcal{L}_k \setminus \mathcal{I}_k$, receive *the same* input $(L, L_1, \ldots, L_n; r)$.

Layer $\mathcal{L}_{k'-3}$:

- Let $l = L\big(L_1(m_{1,1}, \ldots, m_{\ell,1}), \ldots, L(m_{1,n}, \ldots, m_{\ell,n}))\big)$.
- If $r = \mathcal{L}_{k'}$, then forward $(L, L_1, \ldots, L_n; l)$ to the (ideal) adversary.

Layer $\mathcal{L}_{k'}$:

– If $r = P_s^{k'}$, send the value l to $P_s^{k'}$.
– If $r = \mathcal{L}_{k'}$, send the value l to all parties in $\mathcal{L}_{k'}$.

Linear VSS Protocol. Starting from the distributed commitment functionality $\mathcal{F}_{\mathsf{DistCommit}}$ we construct a protocol that realizes $\mathcal{F}_{\mathsf{VSS}}$ with *perfect* security.

The task is to prevent a *corrupt* dealer from disrupting the opening of their commitment at a later time. A basic idea is to ask the dealer to robustly secret share the input s and send each share to a distinct party in a following layer. Each party can then commit to their share. This simple approach fails because the secret sharing provides no guarantees when the dealer is corrupted: the adversary can selectively abort the reconstruction by preventing the opening of different subsets of corrupted commitments.

An even bigger problem is that even with an honest dealer, corrupted parties can commit to arbitrary values. To tackle both these problems at once, we ask the dealer to commit to the randomness used in the sharing (in our case, a polynomial) and prove in ZK that that they are providing valid shares (with respect to this randomness) to the auxiliary layer. Then, parties in the auxiliary layer prove in ZK that they are committing to the values received from the dealer. Since we cannot rely on public randomness for these distributed ZK-proofs (we use the VSS protocol to implement our random beacon later) we resort to techniques based on bi-variate polynomials and leverage the honest majority in each layer.

The dealer produces a two-dimensional polynomial sharing of their input, and sends each share (now a univariate vertical projection of the bi-varate polynomial) to a distinct party in an auxiliary layer. The dealer also commits to their bi-variate polynomial. Each of the parties in the auxiliary layer now commits to the polynomial received from the dealer. The opening of commitments to the dealer's polynomial can fail if the dealer is corrupted, but the projections committed by honest parties in the auxiliary layer will open correctly, even if

those by dishonest parties might be inconsistent with the dealer's polynomial. To ensure consistency of all projections with the dealer's polynomial, every horizontal projection of the dealer's polynomial and the corresponding cross points with the vertical projections are opened towards distinct parties in another layer. If any inconsistency is detected, the conflict is then publicly resolved. Privacy is not an issue as there are no inconsistencies between an honest dealer's polynomial and honest parties' projections. The security of the protocol captured by Lemma 6 is proven in the full version of this paper, due to space constraints.

Linear VSS Protocol $\Pi_{\mathsf{VSS}}^{k_1,\ldots,k_w}$

Public Parameters. Dealer $\mathbf{D} \in \mathcal{L}_0$. Layers $\mathcal{L}_{k_1}, \ldots, \mathcal{L}_{k_w}$ receiving sharing states. Domain $\mathcal{S}$ of the secret. The domain $\mathbf{S}$ of each share. Finite field $\mathbb{F}$.

Secret Inputs. From $\mathbf{D}$ the secret s to share.

Resources. Functionality $\mathcal{F}_{\mathsf{DistCommit}}^{k_1,\ldots,k_w}$ (as the functionality is only invoked with this set of parameters throughout the protocol, in the description below we omit the parameters for legibility).

Layer $\mathcal{L}_0$: The dealer $\mathbf{D}$ does:

- Sample $f_{k,\ell} \leftarrow_{\$} \mathbb{F}$ for $k, \ell \in \{0, \ldots, t\}$ such that $(k, \ell) \neq (0, 0)$.
- Let $F(x, y) = \sum_{k,\ell=0}^{t} f_{k,\ell} x^k y^\ell$ where $f_{0,0} = s$.
- Let $h_i(x) = F(x, i) = \sum_{k=0}^{t} h_{i,k} x^k$.
- Let $v_i(y) = F(i, y) = \sum_{k=0}^{t} v_{i,k} y^k$.
- For all $k, \ell \in [0, t]$ input $f_{k,\ell}$ to $\mathcal{F}_{\mathsf{DistCommit}}[\mathbf{D}]$.[a]
- Send $(v_{i,0}, \ldots, v_{i,t})$ to P_i^1 via bilateral secure channels.

Layer $\mathcal{L}_1$: Each P_i^1 does:

- If $v_{i,k}$ was not received for some $k \in \{0, \ldots, t\}$ set $v_{i,k}$ to 0.
- Input $v_{i,k}$ to $\mathcal{F}_{\mathsf{DistCommit}}[i]$ for all $k \in \{0, \ldots, t\}$.

Layer $\mathcal{L}_{k_1}$: Each $P_s^{k_1}$ does:

- Input $\left(L = (j^0, \ldots, j^t), r = P_j^{k_2}\right)$ to $\mathcal{F}_{\mathsf{DistCommit}}[i]$ for $k \in \{0, \ldots, t\}$.[b]
- Input $\left(L = (i^0 j^0, \ldots, i^t j^t), r = \mathcal{L}_{k_4}\right)$ to $\mathcal{F}_{\mathsf{DistCommit}}[\mathbf{D}]$ for $i \in [n]$.

Layer $\mathcal{L}_{k_2}$: Each $P_j^{k_2}$ does:

- Receive value $\widehat{v_i(j)}$ from $\mathcal{F}_{\mathsf{DistCommit}}[i]$ for all $i \in [n]$.
- Receive values $\widehat{h_j(i)}$ from $\mathcal{F}_{\mathsf{DistCommit}}[\mathbf{D}]$ for all $i \in [n]$.
- If $\widehat{v_i(j)} = \perp$, or $\widehat{h_j(i)} = \perp$, or $\widehat{v_i(j)} \neq \widehat{h_j(i)}$, then broadcast $\mathsf{complain}_{i,j}$.

Layer $\mathcal{L}_{k_3}$: Each $P_s^{k_3}$ does:

- For all $i, j \in [n]$, if $\mathsf{complain}_{i,j}$ was broadcast by $P_j^{k_2}$ do:
  - Input $\big(L = (j^0, \ldots, j^t), r = \mathcal{L}_{k_4}\big)$ to $\mathcal{F}_{\mathsf{DistCommit}}[i]$.[c]
  - Input $\big(L = (i^0 j^0, \ldots, i^t j^t), r = \mathcal{L}_{k_4}\big)$ to $\mathcal{F}_{\mathsf{DistCommit}}[\mathbf{D}]$.[d]

Layer $\mathcal{L}_{k_4}$: Each $P_s^{k_4}$ does:

- For all $i, j \in [n]$, if $\mathsf{complain}_{i,j}$ was broadcast by $P_j^{k_2}$, do:
  - Receive value $\overline{v_i(j)}$ from $\mathcal{F}_{\mathsf{DistCommit}}[i]$.
  - Receive value $\overline{h_j(i)}$ from $\mathcal{F}_{\mathsf{DistCommit}}[\mathbf{D}]$.
  - If $\overline{h_j(i)} = \perp$ the dealer $\mathbf{D}$ is disqualified.
  - If $\overline{v_i(j)} = \perp$ or $\overline{v_i(j)} \neq \overline{h_j(i)}$, then add index i to a set $\mathcal{I}$ (if $|\mathcal{I}| > t$ then $\mathbf{D}$ is disqualified) and for all $j \in [n]$ do:
    - Input $\big(L = (j^0, \ldots, j^t), r = \mathcal{L}_{k_5}\big)$ to $\mathcal{F}_{\mathsf{DistCommit}}[i]$.[e]
    - Input $\big(L = (i^0 j^0, \ldots, i^t j^t), r = \mathcal{L}_{k_4}\big)$ to $\mathcal{F}_{\mathsf{DistCommit}}[\mathbf{D}]$.[f]

Layer $\mathcal{L}_{k_5}$: Each $P_s^{k_5}$ does:

- Receive values $\overline{v_i(j)}$ for all $j \in [n]$ from $\mathcal{F}_{\mathsf{DistCommit}}[i]$. 2
- Receive values $\overline{h_j(i)}$ for all $j \in [n]$ from $\mathcal{F}_{\mathsf{DistCommit}}[\mathbf{D}]$.
- If $\overline{h_j(i)} = \perp$ for any $j \in [n]$, then the dealer $\mathbf{D}$ is disqualified.

Sharing State VSS:

Public State. Set $\mathcal{I}$ and each $i \in \mathcal{I}$ polynomial $v_{i,\mathbf{D}}(y)$ through values $\overline{h_j(i)}$ for all $j \in [n]$ revealed in layer $\mathcal{L}_{k_5}$.
Private State. For all $i \notin \mathcal{I}$ the state of functionality $\mathcal{F}_{\mathsf{DistCommit}}[i]$.

Dealer With Multiple Inputs $(s_1, \ldots, s_\ell)$:

We described the protocol in the case where $\mathbf{D}$ only has one input s to avoid a notational blow-up. However, it is easy to generalize the construction to the case where $\mathbf{D}$ has multiple inputs $s_1, \ldots, s_\ell$. The protocol is simply executed ℓ times in parallel, but using the *same* instances of $\mathcal{F}_{\mathsf{DistCommit}}[i]$ for all $\iota \in [n]$ and $\mathcal{F}_{\mathsf{DistCommit}}[\mathbf{D}]$, which allow for an arbitrary number of inputs. This ensures homomorphism across the sender's inputs. Furthermore, in each parallel execution with input s_ι for $\iota \in [\ell]$ the set $\mathcal{I}^{(\iota)}$ might be different: we consider $\mathcal{I} = \bigcup_{\iota \in [\ell]} \mathcal{I}^{(\iota)}$ to be the union of all such sets. Again, if $|\mathcal{I}| > t$ then $\mathbf{D}$ is disqualified.

Revealing $L(s_1, \ldots, s_\ell)$ to $r \in \mathcal{L}'_k \cup \{\mathcal{L}_{k'}\}$:

Suppose $\mathbf{D}$ has inputs $s_1, \ldots, s_\ell$. If $\mathcal{D}$ was disqualified in any of the executions corresponding to any s_ι the honest party simply output 0.

Layer $\mathcal{L}_{k_r}$ for $r \geq 5$: Each $P_s^{k_i}$ does:

- For all $i \notin \mathcal{I}$ input (L', r) to $\mathcal{F}_{\mathsf{DistCommit}}[i]$, where L' is the linear function that comptutes $L(\widehat{v}_i^{(1)}(0), \ldots, \widehat{v}_i^{(n)}(0))$.
- For all $i \in \mathcal{I}$ input (π'_ι, r) to $\mathcal{F}_{\mathsf{DistCommit}}[i]$ for all $\iota \in [\ell]$.[g]

Layer $\mathcal{L}_{k'}$: Each $P_s^{k'}$ does:

- Receive output l_i from $\mathcal{F}_{\mathsf{DistCommit}}[i]$ for $i \notin \mathcal{I}$.
- Receive outputs $s_{i,\iota}$ from $\mathcal{F}_{\mathsf{DistCommit}}[i]$ for $i \in \mathcal{I}$ and $\iota \in [\ell]$.
- Let $l_i = L(s_{1,i}, \ldots, s_{\ell,i})$ for $i \in \mathcal{I}$.
- Interpolate the unique polynomial $\widehat{f}(x)$ through any of $t+1$ among points $(i, l_i)_{i \in [n]}$ and output $\widehat{f}(0)$.

Multiple Dealers $\mathbf{D}_1, \ldots, \mathbf{D}_n$:

We described the protocol in the case where there is only one dealer to avoid a notational blow-up. However, when invoked in parallel by different dealers the protocol provides linearity among values dealt by *different* dealers. Consider parallel executions with dealer $\mathbf{D}_\iota$ with input s_ι for $\iota \in [n]$. Suppose party P_i^1 commits to polynomials $\widehat{v}_i^{(\iota)}(y)$ in execution with $\mathbf{D}_\iota$. Revealing a linear combination of values $L'(s_1, \ldots, s_n)$ can be done by revealing the corresponding linear combination of the polynomials $\widehat{v}_i^{(\iota)}(y)$. Some care must be put to ensure privacy. Indeed, in executions with different dealers the sets $\mathcal{I}^{(\iota)}$ could be different. Suppose that $P_i^1 \in \mathcal{I}^{(1)}$ but $P_i^1 \notin \mathcal{I}^{(\iota)}$ for all other $\iota \neq 1$. Then, we cannot simply reconstruct all polynomials $\widehat{v}^{(i)}(0)$ for $\iota \neq 1$ and compute L' on public information, as this would violate the privacy of honest dealer's values. Therefore, we first compute the projection of L' on $\widehat{v}^{(i)}(0)$ for $\iota \neq 1$, and then add $\widehat{v}^{(1)}(0)$ afterwards. Details follow.

Revealing $L(s_1, \ldots, s_n)$ to $r \in \mathcal{L}'_k \cup \{\mathcal{L}_{k'}\}$:

Suppose dealer $\mathbf{D}_\iota$ has input s_ι.

Layer $\mathcal{L}_{k_r}$ for $r \geq 5$: Each $P_s^{k_i}$ does:

- If $L = (a_1, \ldots, a_n)$ then let $L' = (\widetilde{a}_1, \ldots, \widetilde{a_n})$ where $\widetilde{a}_\iota \leftarrow a_\iota$ if $i \notin \mathcal{I}^{(\iota)}$, and $\widetilde{a}_\iota \leftarrow 0$ if $i \in \mathcal{I}^{(\iota)}$.
- For all $i \in [n]$ input (L', r) to $\mathcal{F}_{\mathsf{DistCommit}}[i]$.

Layer $\mathcal{L}_{k'}$: Each $P_s^{k'}$ does:

- Receive output l'_i from $\mathcal{F}_{\mathsf{DistCommit}}[i]$ for $i \in [n]$.
- If $l'_i \neq \perp$ let $l_i = l'_i + \sum_{\iota \text{ such that } i \in \mathcal{I}^{(\iota)}} v_{i,\mathbf{D}_\iota}^{(\iota)}(0)$ for all $i \in [n]$.
- Interpolate the unique polynomial $\widehat{f}(x)$ through any of $t+1$ among points (i, l_i) and output $\widehat{f}(0)$.

[a] This notation is used to identify different parallel instances of $\mathcal{F}_{\mathsf{DistCommit}}$.
[b] This reveals $\widehat{v}_i(j)$ to $P_j^{k_2}$.
[c] This publicly reveal values $\widehat{v_i(j)}$ to all parties in $\mathcal{L}_{k_3}$.
[d] This publicly reveals $\widehat{F}(i,j) = \widehat{h}_j(i)$ to all parties in layer $\mathcal{L}_4$.
[e] This reveals values $\overline{h_j(i)}$ for all $j \in [n]$ to all parties in $\mathcal{L}_{k_5}$.
[f] This reveals values $\overline{v_i(j)}$ for all $j \in [n]$ to all parties in $\mathcal{L}_{k_5}$.
[g] We denote by π'_ι the linear projection map computing $\widehat{v}_i^{(\iota)}(0)$ for $\iota \in [\ell]$.

Lemma 6. *Assume that $t < n/2$. The (n, t, k')-layered protocol $\Pi_{\mathsf{VSS}}^{k_1,\ldots,k_w}$ realizes functionality $\mathcal{F}_{\mathsf{VSS}}^{k_1,\ldots,k_w}$ with $(0, t)$-statistical security (i.e. perfect security) in the $\mathcal{F}_{\mathsf{DistCommit}}^{k_1,\ldots,k_w}$-hybrid model.*

9 Circuit Evaluation

In this section, we explain how the layered components we developed so far can be used to securely evaluate a circuit C encoding a function f. Let $\mathbb{F}$ be a finite field, and let f be a function $f : \mathbb{F}^\ell \to \mathbb{F}^{\ell'}$. We denote by $\mathcal{F}_f$ the functionality that takes a set of inputs $(s_i)_{i \in \mathsf{inputs}_i}$ from all input clients $\mathcal{C}_i$ for $i \in [n]$ belonging to an initial layer $\mathcal{L}_0$, and delivers $f(s_1, \ldots, s_\ell)_{j \in \mathsf{outputs}_i}$ to the output clients $\mathcal{C}'_1, \ldots, \mathcal{C}'_n$. Due to space constraints, we describe the functionality $\mathcal{F}_f$, give a formal description of the protocol Π_{MPC}^C, and proof of the following theorem in the full version of this paper.

Theorem 2. *Let $t < n/2$. If C is a circuit with depth d computing f, the $(n, t, O(d))$ layered protocol Π_{MPC}^C realizes functionality $\mathcal{F}_f$ with $(\mathsf{negl}(\kappa), t)$-statistical security in the $(\mathcal{F}_{\mathsf{VSS}}, \mathcal{F}_{\mathsf{Beacon}})$-hybrid model.*

Protocol Invariant. Throughout the circuit evaluation, we maintain the following invariant: there is a layer of parties, say $\mathcal{L}_0$, holding a state encoding the input values a and b to every gate g in a certain level of the circuit C, and a layer $\mathcal{L}_k$ holding a state encoding the output $c = g(a, b)$ of g. It is clear that this invariant allows performing the computation of the whole circuit layer by layer. The state encoding an input value a to a gate g has three components:

1. A (t, n)-Shamir Sharing of a, where each party $P_i^0 \in \mathcal{L}_0$ holds share a_i.
2. A (t, n)-Shamir Sharing of a random value r (which is used to securely evaluate multiplication gates).
3. Commitments (in the strong sense, produced using $\mathcal{F}_{\mathsf{VSS}}$) to the coefficients of the polynomial $f_a(x)$ used for the Shamir Sharing of a. Observe that, because of the linear properties of $\mathcal{F}_{\mathsf{VSS}}$, this means that parties also hold commitments to *each share* a_i.

The need for such a cumbersome state is somewhat inherent to the $t \geq n/3$ honest majority setting. Unlike in the $t < n/3$ case, where robust linearly homomorphic sharings have a simple description (typically a polynomial sharing, for example Shamir sharing), in the honest majortiy setting combining robustness and linearity is trickier. In particular, the robust sharing scheme described above does *not* provide linearity across different dealers. We now descrive protocol Π^C_{MPC}, which comprises of four sub-protocols, two for receiving inputs and providing outputs to clients, and two for computing addition and multiplication gates.

Client Input Protocol. Clients must provide the necessary state encoding their input onto the layer tasked with the evaluation of the first level of the circuit. The protocol is simple because all the difficult guarantees (commitment, linearity among values input by different dealers) are derived from $\mathcal{F}_{\mathsf{VSS}}$, so that the only real task is to produce the polynomial sharing securely even when the client is dishonest. To achieve this, we simply require the client to VSS the coefficients of a polynomial which shares their input, and then we reconstruct each share towards the intended recipient exploiting the VSS linearity. This ensures that all shares lie on a polynomial of degree t even when the dealer is dishonest. We only describe the protocol for one client and one input, but a parallel version where every party in a layer has up to ℓ inputs can be obtained exactly as in Π_{VSS}.

Secure Multiplication. Layer $\mathcal{L}_0$ holds the states for the inputs a and b of the multiplication gate g. Each party P_i^0 locally multiplies their polynomial shares of a and b and computes $c_i = a_i \cdot b_i$. Then this party reproduces the input state but for the value c_i towards the layer (say $\mathcal{L}_k$) who is tasked to compute the following layer of the circuit. It does so by using the client input protocol Π_{Input}. Notice that $c = a \cdot b$ can be expressed as a linear combination of the c_i's for $i \in [n]$, because $(i, c_i)_{i\in[n]}$ are points on a polynomial $g(x)$ of degree $2t$ (the product of $f_a(x)$ and $f_b(x)$) such that $g(0) = c$, and both polynomial interpolation and polynomial evaluation at 0 are linear functions. Therefore, the state for c can be obtained from the states for each c_i locally.

However, a corrupt $P_i^0 \in \mathcal{I}_0$ who inputs a value $\widehat{c}_i \neq a_i \cdot b_i$ can easily disrupt the correctness of this procedure. We therefore require that each party proves, via a distributed ZK-proof, that the input $\widehat{c}_i$ they provided is indeed $\widehat{c}_i = a_i \cdot b_i$. To this end, each party P_i^0 produces new $\mathcal{F}_{\mathsf{VSS}}$ commitments to $\widehat{a}_i, \widehat{b}_i$ and $\widehat{c}_i$, and proves (to parties in some auxiliary layer $\mathcal{L}_{k'}$) that 1) $\widehat{a}_i = a_i$, 2) $\widehat{b}_i = b_i$, and 3) $\widehat{a}_i \cdot \widehat{b}_i = \widehat{c}_i$. If they fail to do so (honest parties never fail in these proofs) then parties in $\mathcal{L}_{k'}$ simply reveal values a_i and b_i to layer $\mathcal{L}_k$. Finally parties in $\mathcal{L}_k$ hold commitments to each c_i (the ones for which the proof fails are just taken as standard states of the reconstructed values) and with this to the product $c = a \cdot b$.

To finish, we just need to explain how parties perform the three required ZK-proofs. Let us first discuss the proof of equality for the commitments to values $\widehat{a}_i$ and a_i performed by P_i^0. To begin the proof of equality, party P_i^0 produces new commitments via $\mathcal{F}_{\mathsf{VSS}}$ to $\widehat{r}_i$ (his claimed version of r_i), towards all future layers until $\mathcal{L}_k$. Then, parties in some later layer (who also hold commitments

to a_i, r_i and $\widehat{a}_i$) receive a public random value ρ from functionality $\mathcal{F}_{\mathsf{Beacon}}$. The values $r_i + \rho a_i$ and $\widehat{r}_i + \rho \widehat{a}_i$ are opened publicly, and if they are different the proof fails. Intuitively, if $a_i \neq \widehat{a}_i$ or $r_i \neq \widehat{r}_i$, there is only one value ρ that satisfies the equality, and this can only be guessed with negligible probability if the space of random values sampled by the beacon is large enough.

The proof of correct multiplication to show $\widehat{a}_i \cdot \widehat{b}_i = \widehat{c}_i$ works as follows: party P_i^0 samples a random value β_i and commits (via $\mathcal{F}_{\mathsf{VSS}}$) to values β_i and $b_i\beta_i$. Now, parties in some later layer (who also hold commitments to $\widehat{a}_i, \widehat{b}_i$ and $\widehat{c}_i$) receive a random value ρ from functionality $\mathcal{F}_{\mathsf{Beacon}}$, and publicly open the value $\rho' = \rho a_i + \beta_i$. Finally, a later layer publicly opens the value $\rho' b_i - b_i\beta_i - \rho c_i$ and the proof succeeds if and only if this value is 0. Again, the intuition is that if $\widehat{a}_i \cdot \widehat{b}_i \neq \widehat{c}_i$, there is only one value ρ that satisfies the equality, and the proof succeeds with negligible probability if the value ρ comes from a large enough space. The fresh randomness to be used in the next layer of the circuit is generated in parallel to the multiplication protocol.

Secure Addition. Addition gates (and linear gates in general) could be evaluated locally by exploiting the linearity of $\mathcal{F}_{\mathsf{VSS}}$ and of polynomial sharings. However, since the linearity only holds for *parallel* executions of Π_{VSS}, to allow the next layer of parties to compute the next layer of the circuit, we need to "refresh" the state, so that the commitments to the inputs for the next layer are all generated in parallel executions of Π_{VSS}. This can be trivially achieved by multiplying by 1 after performing the addition locally. The state encoding the input of this dummy gate fixed to 1 can be computed as a default state of protocol Π_{Input}. We denote this procedure by Π_{Add}. Notice that this protocol preserves the invariant necessary for the circuit computation.

Client Output Protocol. The output protocol Π_{Output} is trivial: parties in the layer holding the state corresponding to output c of an output gate g can query $\mathcal{F}_{\mathsf{VSS}}[c]$ to reveal c to the intended recipient (or recipients).

References

1. Beerliová-Trubíniová, Z., Hirt, M.: Efficient multi-party computation with dispute control. In: Halevi, S., Rabin, T. (eds.) TCC 2006. LNCS, vol. 3876, pp. 305–328. Springer, Heidelberg (2006)
2. Ben-Or, M., Goldwasser, S., Wigderson, A.: Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In: 20th ACM STOC, pp. 1–10. ACM Press (1988)
3. Benhamouda, F., et al.: Can a public blockchain keep a secret? In: Pass, R., Pietrzak, K. (eds.) TCC 2020. Part I, volume 12550 of LNCS, pp. 260–290. Springer, Heidelberg (2020)
4. Canetti, R.: Security and composition of multiparty cryptographic protocols. J. Cryptol. **13**(1), 143–202 (2000)
5. Canetti, R.: Universally composable security: a new paradigm for cryptographic protocols. In: 42nd FOCS, pp. 136–145. IEEE Computer Society Press (2001)

6. Chaum, D., Crépeau, C., Damgård, I.: Multiparty unconditionally secure protocols (abstract) (informal contribution). In: Pomerance, C. (ed.) CRYPTO'87, volume 293 of LNCS, page 462. Springer, Heidelberg (1988)
7. Chaum, D., Crépeau, C., Damgard, I.: Multiparty unconditionally secure protocols (extended abstract). In: 20th ACM STOC, pp. 11–19. ACM Press (1988)
8. Choudhuri, A.R., Goel, A., Green, M., Jain, A., Kaptchuk, G.: Fluid MPC: secure multiparty computation with dynamic participants. In: Malkin, T., Peikert, C. (eds.) CRYPTO 2021. LNCS, vol. 12826, pp. 94–123. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-84245-1_4
9. Cramer, R., Damgård, I., Dziembowski, S., Hirt, M., Rabin, T.: Efficient multiparty computations secure against an adaptive adversary. In: Stern, J. (ed.) EUROCRYPT'99. LNCS, vol. 1592, pp. 311–326. Springer, Heidelberg (1999)
10. David, B., et al.: Perfect MPC over layered graphs. In: Handschuh, H., Lysyanskaya, A. (eds.) CRYPTO 2023. Part I, volume 14081 of LNCS, pp. 360–392. Springer, Heidelberg (2023)
11. Deligios, G., Goel, A., Liu-Zhang, C.D.: Maximally-fluid MPC with guaranteed output delivery. Cryptology ePrint Archive, Paper 2023/415 (2023). https://eprint.iacr.org/2023/415
12. Desmedt, Y., Wang, Y.: Perfectly secure message transmission revisited. In: Knudsen, L.R. (ed.) EUROCRYPT 2002. LNCS, vol. 2332, pp. 502–517. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-46035-7_33
13. Dolev, D., Dwork, C., Waarts, O., Yung, M.: Perfectly secure message transmission. J. ACM (JACM) **40**(1), 17–47 (1993)
14. Gennaro, R., Ishai, Y., Kushilevitz, E., Rabin, T.: The round complexity of verifiable secret sharing and secure multicast. In: 33rd ACM STOC, pp. 580–589. ACM Press (2001)
15. Gennaro, R., Rabin, M.O., Rabin, T.: Simplified VSS and fast-track multiparty computations with applications to threshold cryptography. In: Coan, B.A., Afek, Y., (eds.) 17th ACM PODC, pp. 101–111. ACM (1998)
16. Gentry, C., et al.: YOSO: you only speak once. In: Malkin, T., Peikert, C. (eds.) CRYPTO 2021. LNCS, vol. 12826, pp. 64–93. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-84245-1_3
17. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game or a completeness theorem for protocols with honest majority. In: Aho, A., (eds.) 19th ACM STOC, pp. 218–229. ACM Press (1987)
18. Hirt, M., Maurer, U.M.: Player simulation and general adversary structures in perfect multiparty computation. J. Cryptol. **13**(1), 31–60 (2000)
19. Hirt, M., Maurer, U.M., Przydatek, B.: Efficient secure multi-party computation. In: Okamoto, T. (ed.) ASIACRYPT 2000. LNCS, vol. 1976, pp. 143–161. Springer, Heidelberg (2000)
20. Hofheinz, D., Müller-Quade, J.: A synchronous model for multi-party computation and the incompleteness of oblivious transfer. In: Proceedings of FCS, pp. 117–130. Citeseer (2004)
21. Katz, J., Maurer, U., Tackmann, B., Zikas, V.: Universally composable synchronous computation. In: Sahai, A. (ed.) TCC 2013. LNCS, vol. 7785, pp. 477–498. Springer, Heidelberg (2013)
22. Lamport, L., Shostak, R., Pease, M.: The Byzantine Generals Problem, pp.203–226. Association for Computing Machinery, New York, NY, USA (2019)
23. Liu-Zhang, C.-D., Maurer, U.: Synchronous constructive cryptography. In: Pass, R., Pietrzak, K. (eds.) TCC 2020. Part II, volume 12551 of LNCS, pp. 439–472. Springer, Heidelberg (2020)

24. Manurangsi, P., Srinivasan, A., Vasudevan, P.N.: Nearly optimal robust secret sharing against rushing adversaries. In: Micciancio, D., Ristenpart, T. (eds.) CRYPTO 2020. Part III, volume 12172 of LNCS, pp. 156–185. Springer, Heidelberg (2020)
25. Nielsen, J.B.: On protocol security in the cryptographic model. BRICS (2003)
26. Ostrovsky, R., Yung, M.: How to withstand mobile virus attacks (extended abstract). In: Luigi L., (ed.) 10th ACM PODC, pp. 51–59. ACM (1991)
27. Rabin, T., Ben-Or, M.: Verifiable secret sharing and multiparty protocols with honest majority (extended abstract). In: 21st ACM STOC, pp. 73–85. ACM Press (1989)
28. Shamir, A.: How to share a secret. Commun. ACM **22**(11), 612–613 (1979)
29. Yao, A.C.C.: How to generate and exchange secrets (extended abstract). In: 27th FOCS, pp. 162–167. IEEE Computer Society Press (1986)

An Improvement Upon the Bounds for the Local Leakage Resilience of Shamir's Secret Sharing Scheme

Dustin Kasser(✉)

University of Georgia, 30602 Athens, GA, USA
dustin.kasser@uga.edu

Abstract. Shamir's Secret Sharing Scheme allows for the distribution of information amongst n parties so that any nt of them can combine their information to recover the secret, for some parameter $0 < t \leq 1$. By design, it is secure against the total corruption of $nt - 1$ parties, but open questions remain around its security against side-channel attacks, where an adversary may obtain a small amount of information about each of the n party's shares.

An initial result by Benhamouda, Degwekar, Ishai and Rabin showed that if n is sufficiently large and $t \geq 0.907$, then the scheme was secure under one bit of local leakage. These bounds continued to be improved in following works, and most recently Klein and Komargodski introduced a proof using a new analytical proxy that showed leakage resilience for $t \geq 0.69$.

In this paper we will use the analytic proxy of Klein and Komargodski to show leakage resilience for $t \geq 0.668$. We do this by introducing two new bounds on the proxy. The first uses a result from additive combinatorics to improve their original bound on the proxy. The second is an averaging argument that exploits the rarity of worst-case bounds occurring.

1 Introduction

In his 1979 paper "How to Share a Secret" [18], Shamir explains what is now aptly referred to as Shamir's Secret Sharing Scheme. This scheme allows for a secret to be divided among n parties so that if any tn of them work together the secret may be recovered, but a group of $tn - 1$ parties will gain no information about the secret. An example for $tn = 2$ would be to calculate a line $\ell(x)$ that passes through $(0, s)$, where s is the secret. Then each ordered pair $(i, \ell(i))$ could be distributed to each party.

Since Shamir's scheme is secure when at most $tn - 1$ parties are corrupted, it has been used as an important part of a variety of applications [2,5–11,17]. However, the hypothesis that all parties are entirely uncorrupted and pass no information to an adversary is very strong. In recent works it has been examined

E. Boyle and M. Mahmoody (Eds.): TCC 2024, LNCS 15367, pp. 395–422, 2025.
https://doi.org/10.1007/978-3-031-78023-3_13

whether Shamir's scheme is secure if each party leaks a small amount of information to an adversary. Before presenting the results on the security, it may be helpful to begin by defining the scheme and leakage resistance explicitly.

To begin the scheme, a prime $p > n$ is chosen and a secret $s \in \mathbb{F}_p$ is fixed. Then, a $tn - 1$ degree polynomial $\ell \in \mathbb{F}_p[x]$ with $\ell(0) = s$ is randomly chosen. Next, each ordered pair $(i, \ell(i))$ is distributed among the parties.

It is equivalent to consider the vector $x \in \mathbb{F}_p^{tn}$ as representing the coefficients of ℓ, and then to construct the vectors $\ell_i \in \mathbb{F}_p^{tn}$, where each $\ell_i = (i, i^2, i^3, ..., i^{tn})$ so that $\ell_i \cdot x = \ell(i)$. In this case, each $(i, \ell_i \cdot x)$ is distributed among the parties. It is this perspective that we shall use for the remainder of the paper.

The adversary is allowed to construct some leakage functions $f_i : \mathbb{F}_p^t \to \{-1, 1\}$ and obtains the pairs $(i, f_i(\ell_i \cdot x))$, from which they aim to reconstruct some information about the secret. The scheme is considered secure if the amount of information gained tends to zero as n grows to infinity. In Appendix B **X**, we formally state the definition of security, but throughout this paper we will focus on using the analytic proxy of Klein and Komargodski.

The scheme was originally shown to be secure when $t \geq 0.907$ by Benhamouda, Degwekar, Ishai, and Rabin [3], in which they presented an analytic proxy to bound the leakage. These results were independently improved to $t \geq 0.8675$ by Maji, Paskin-Cherniavski, Suad, and Wang [14] and to $t \geq 0.85$ by Benhamouda et al. [4], and then further to $t \geq 0.78$ by Maji, Ngyuen, Paskin-Cherniavski, and Wang [15]. A new analytic proxy was later introduced by Klein and Komargodski in [12], which improved the bound to $t \geq 0.688$ [12]. In a negative result, Nielsen and Simkin showed that Shamir's scheme is known to not be leakage resilient if $tn = O(n/\log(n))$ [16].

Before reading the proxy of Klein and Komargodski, it may be helpful to the reader to refer to Appendix B, where we formally define the Fourier transform. In this paper, we use a normalized Fourier transform so that for $f_i : \mathbb{F}_p \to \{-1, 1\}$, $\left\|\hat{f}_i\right\|_{\mathrm{L}^2} = 1$. Before continuing, we also define the function $f_S : \mathbb{F}_p^{tn}$ as

$$f_S(x) = \prod_{i \in S} f_i(x \cdot \ell_i)\,,$$

The proxy of Klein and Komargodski implies that if

$$2\left(p^3 \sum_{k \in \mathbb{F}_p \setminus 0} \sum_{S \subseteq [n]} \left|\hat{f}_S(k \cdot \ell_0)\right|^2\right)^{1/4} \tag{1}$$

decays to zero, then so too does the information available to an adversary. Because we have a term that is polynomial in p, we will need that $p = O(2^n)$ in order to gain our convergence results. They are able to obtain decay in (1) by proving that

$$\left|\hat{f}_S(k \cdot \ell_0)\right| \leq \left(\frac{2}{\pi}\right)^{2tn - |S|}\,, \tag{2}$$

and that when $|S| < tn$,

$$\left|\hat{f}_S(k \cdot \ell_0)\right| = 0$$

when $k \neq 0$. Then by a counting argument, it follows that (1) decays when $t \geq 0.688$.

In this paper, we will focus on showing that each

$$\sum_{\substack{S \subseteq [n] \\ |S| = (t+a)n}} \left|\hat{f}_S(\ell_0)\right|^2$$

decays to zero for $a \in \mathbb{N}$. We will use the slightly more general setting where the ℓ_i are any collection of vectors in $\mathbb{F}_p^{tn}$ so that every tn of them are linearly independent. The ℓ_i given by the secret sharing scheme satisfy this, but in this setting we may drop the k from $\hat{f}_S(k \cdot \ell_0)$ at the cost of a factor of at most p. We introduce two new bounds, described below, in order to give leakage resistance for $t \geq 0.668$.

Before illustrating our methods, we quickly examine in more detail how Klein and Komargodski obtained the bound in (2). We begin by assuming that $t > 0$. For each set $|S| = (t + a)n$ for some parameter $0 < a < 1 - t$, Klein and Komargodski show that they may bound $\left|\hat{f}_S(\ell_0)\right|$ by

$$\left|\hat{f}_S(\ell_0)\right| \leq \prod_{i \in A_1} \left\|\hat{f}_i\right\|_{\mathrm{L}^2} \prod_{j \in A_2} \left\|\hat{f}_j\right\|_{\mathrm{L}^2} \prod_{k \in B} \left\|\hat{f}_k\right\|_{\mathrm{L}^\infty} \tag{3}$$

where $|A_1| = |A_2| = an$ and $|B| = (t - a)n$. Using Plancherel, each $\left\|\hat{f}_i\right\|_{\mathrm{L}^2} = 1$. When the mean of f_i is small, $\left\|\hat{f}_i\right\|_{\mathrm{L}^\infty} \leq 2/\pi$. Their induction argument covered the cases when the mean of f_i is large replacing each term of $\left\|\hat{f}_i\right\|_{\mathrm{L}^\infty}$ with a $2/\pi$ to obtain the bound

$$\left|\hat{f}_S(\ell_0)\right| \leq \left(\frac{2}{\pi}\right)^{(t-a)n}, \tag{4}$$

which is equivalent to (2). The reader can find a similar proof of a slightly general version of this bound in Appendix A.

In Sect. 3 we are able to improve the bound to

$$\left|\hat{f}_S(\ell_0)\right| \leq \left(\frac{2}{\pi}\right)^{(t-0.66a)n}, \tag{5}$$

though we need the even more restrictive requirement that $a \leq t/4$. We instead use

$$\left|\hat{f}_S(\ell_0)\right| \leq \prod_{i \in A_1} \left\|\hat{f}_i\right\|_{\mathrm{L}^4} \prod_{i \in A_2} \left\|\hat{f}_i\right\|_{\mathrm{L}^4} \prod_{i \in A_3} \left\|\hat{f}_i\right\|_{\mathrm{L}^4} \prod_{i \in A_4} \left\|\hat{f}_i\right\|_{\mathrm{L}^4} \prod_{k \in C} \left\|\hat{f}_k\right\|_{\mathrm{L}^\infty}$$

and use a result of Lev [13] to obtain a bound on the L^4 norms. We then prove the bound using an induction argument similar to that of [12], as well as a counting argument to handle its failure cases.

Unfortunately, this L^4-style bound does not give much improvement, as the decay fails near $a = 0$ for both our bound and that of [12]. To obtain an improvement we also introduce an averaging argument. The core of the idea is that if two sets, S and T, share most elements, then if $\hat{f_S}(\ell_0)$ is large, $\hat{f_T}(\ell_0)$ should not be. In order to argue this more rigorously, we fix a set S' and choose another set T randomly, so that $S = S' \cup T$. For different choices of T, we would expect $\widehat{f_{S' \cup T}}(\ell_0)$ to not take consistent values. We can then average across all T to obtain the estimate that, if $|T| = (K + a)n$ for a parameter K fulfilling $a \leq K \leq t/2 - a$,

$$\sum_{T} |f_{S' \cup T}| \leq \left(\frac{2}{\pi}\right)^{2(t-K-3a)n}. \tag{6}$$

When we then sum across choices of S', we are over-counting our entries of $\left|\hat{f_S}(\ell_0)\right|$, and measuring this over-counting gives the bound

$$\sum_{|S|=(t+a)n} \left|\widehat{f_S}(\ell_0)\right|^2 \leq \binom{(t+a)n}{(t-K)n}^{-1} \cdot \binom{n}{(t-K)n} \cdot \left(\frac{2}{\pi}\right)^{2(t-K-3a)}. \tag{7}$$

Finally, we spend Sect. 5 showing that the bounds from (4) and (5), combined with the original bound in (2), indeed gives that Shamir's secret sharing scheme is secure for $t \geq 0.668$. We state this formally in the following theorem.

Theorem 1. *Let $n, t \in \mathbb{N}$ and $0 < t \leq 1$. Let $p = O(2^n)$. Then let $\{\ell_i\}_{i=0}^{n}$ be vectors in $\mathbb{F}_p^{tn}$ such that every tn of them are linearly independent. Let $\{f_i\}_{i=0}^{n}$ be some functions $f_i : \mathbb{F}_p \to \{-1, 1\}$. Let $f_S(x) = \prod f_i(\ell_i \cdot x)$. Then for $t \geq 0.668$,*

$$\sum_{S \subseteq [n]} \left|\hat{f_S}(k \cdot \ell_0)\right|^2 = 2^{-\Omega(n)} \tag{8}$$

As a final note on the structure of the paper, the reader may notice that we break from the notation of Klein and Komargodski, which would take $t \in \mathbb{N}$. We found that by taking $t, a \in [0, 1]$ it was easier to think of them as variables with which to graph various bounds; these graphs will appear throughout the paper, and may be accessed directly via links in Appendix C for the readers convenience.

2 Establishing a Bound on $\left\|\hat{\boldsymbol{f}}_{\boldsymbol{i}}\right\|_{\mathbf{L}^{\boldsymbol{q}}}$

We will begin by introducing a Theorem that follows from Corollary 2 in [13] by Lev.

Theorem 2. *Let $f : \mathbb{F}_p \to \{0,1\}$ be an arbitrary function. Let $g : \mathbb{F}_p \to \{0,1\}$ be a function such that $\|g\|_{L^1} = \|f\|_{L^1}$ and g is the indicator function for the set $[-a,b]$, where $|b-a| \leq 1$. Then for each $k \in \mathbb{N}$,*

$$\left\|\hat{f}\right\|_{L^{2k}} \leq \|\hat{g}\|_{L^{2k}} \ .$$

This allows us to move from examining arbitrary functions to indicators of intervals, where we may explicitly compute their L^{2k} norms, which we claim correspond to the following function.

Definition 1. *For $q > 2$, $q \in 2\mathbb{N}$, we define the function $L_q(\mu)$ as*

$$L_q(\mu) = 2\sum_{k=1}^{\sqrt{p}} \left| \frac{2}{\pi} \cdot \frac{\sin\left(\pi k \frac{\mu+1}{2}\right)}{k} \right|^q \tag{9}$$

While we will only use $L_4(\mu)$ in this paper, we state the results of this section in higher generality in case it is useful to others.

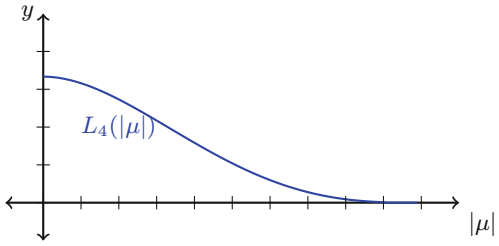


Fig. 1. A graph of $L_4(|\mu|)$ on $[0,1]$ with increments at 0.1 intervals.

Lemma 1. *If $f : \mathbb{F}_p \to \{-1,1\}$ with $0 < \mu = \mathbb{E}f$, then for all $q \in 2\mathbb{N}$, $q > 2$,*

$$\sum_{k \neq 0} \left|\hat{f}(k)\right|^q \leq L_q(\mu) + O\left(\frac{1}{\sqrt{p}}\right) ,$$

Proof. We will start by bounding f above by the indicator function of an interval of the form $[-a,a]$. We define the function $g : \mathbb{F}_p \to \{0,1\}$ as

$$g(x) = \frac{f(x)+1}{2}$$

so that $g : \mathbb{F}_p \to \{0,1\}$. Further,

$$\sum_{k \neq 0} \left|\hat{f}(k)\right|^q = 2\sum_{k \neq 0} |\hat{g}(k)|^q \ .$$

Since g is in the proper form to apply Theorem 2, we will focus on bounding it instead. Let $\nu = \|g\|_{\mathrm{L}^1}/p$ be its density of non-zero entries. Let $a, b \in \mathbb{Z}_{\geq 0}$ such that $|b-a| \leq 1$ and if $s(x)$ is the indicator function for $[-a, b]$, then $\|s\|_{\mathrm{L}^1} = \nu p$. From here, we may apply Theorem 2 to g and s so that

$$\sum_{k\neq 0} |\hat{g}(k)|^q \leq \sum_{k\neq 0} |\hat{s}(k)|^q \,.$$

Finally, we introduce the function $h : F_p \to \{0, 1\}$, defined as

$$h(x) = \begin{cases} 1 & : x \in \left[-\lfloor \frac{p\nu}{2} \rfloor, \lfloor \frac{p\nu}{\rfloor} \right] \\ 0 & : x \notin \left[-\lfloor \frac{p\nu}{2} \rfloor, \lfloor \frac{p\nu}{2} \rfloor\right] \end{cases}$$

The number of non-zero outputs for $h(x)$ and $s(x)$ differs by at most one, and so since our Fourier transform is normalized by $1/p$,

$$\sum_{k\neq 0} |\hat{s}(k)|^q \leq \sum_{k\neq 0} \left(\left|\hat{h(k)}\right| + 1/p\right)^q \,.$$

Factoring out the right-hand side, we obtain the inequality

$$\sum_{k\neq 0} |\hat{s}(k)|^q \leq \sum_{k\neq 0} \sum_{j=1}^{q} \left|\hat{h}(k)^j\right| \cdot (1/p)^{q-i} \cdot \binom{q}{i} .$$

We next exchange the order of the sums, to obtain

$$\sum_{k\neq 0} |\hat{s}(k)|^q \leq \sum_{i=0}^{q} \sum_{k\neq 0} \left|\hat{h}(k)\right|^i \cdot (1/p)^{q-i} \cdot \binom{q}{i} .$$

Since $\|h\|_{\mathrm{L}^r} \leq 1$ for $r \geq 1$, all terms with $q > i \geq 2$ are $O(1/p)$. When $i < 2$, $q - i > 1$, and so for $i = 1$ or $i = 0$,

$$\sum_{k\neq 0} \left|\hat{h}(k)\right|^i \cdot (1/p)^{q-i} \cdot \binom{q}{i} \leq \sum_{k\neq 0} 1 \cdot 1/p^2 \cdot q^2 \leq (p-1) \cdot q^2/p^2 = O(1/p) \,. \quad (10)$$

As we wish to examine the term when $i = q$, we reduce to the inequality

$$\sum_{k\neq 0} |\hat{s}(k)|^q \leq \left(\sum_{k\neq 0} \left|\hat{h}(k)\right|^q\right) + O(1/p) \,.$$

As

$$\sum_{k\neq 0} \left|\hat{f}(x)\right|^q \leq \sum_{k\neq 0} \left|2\hat{h}(k)\right|^q + O(1/p)$$

we shall proceed by obtaining estimates on each $\hat{h}(k)$. Notice that

$$\hat{h}(k) = \frac{1}{p} \sum_{x\in\mathbb{F}_p} h(x) e^{\frac{2\pi i}{p} kx} = \frac{1}{p} \sum_{\frac{-p\nu}{2} \leq x \leq \frac{p\nu}{2}} \left(\cos\left(\frac{2\pi}{p} xk\right) + i \sin\left(\frac{2\pi}{p} xk\right)\right)$$

As our sum is over x values that are symmetric about 0, our sine terms cancel. Then

$$\hat{h}(k) = \frac{1}{p} \sum_{\frac{-p\nu}{2} \le x \le \frac{p\nu}{2}} \cos\left(\frac{2\pi}{p} xk\right).$$

Note that for each $z \in \mathbb{Z}$,

$$\left| \int_z^{z+1} \cos\left(\frac{2\pi}{p} kx\right) dx - \cos\left(\frac{2\pi}{p} kn\right) \right| \le \frac{4\pi k}{p}.$$

Then when $|k| < \sqrt{p}$, we may write

$$\hat{h}(k) = \frac{1}{p} \int_{\frac{-p\nu}{2}}^{\frac{p\nu}{2}} \left(\cos\left(\frac{2\pi}{p} xk\right) + O\left(\frac{1}{\sqrt{p}}\right) \right) dx$$

Since $\nu \le 1$, pulling out the $O(1/\sqrt{p})$ leaves our original term unaffected, so that

$$\hat{h}(k) = O\left(\frac{1}{\sqrt{p}}\right) + \frac{1}{p} \int_{\frac{-p\nu}{2}}^{\frac{p\nu}{2}} \cos\left(\frac{2\pi}{p} xk\right) dx$$

Making a u substitution for $u = 2\pi xk/p$, we have that

$$\hat{h}(k) = O\left(\frac{1}{\sqrt{p}}\right) + \frac{1}{p} \int_{\frac{-2\pi k\nu}{2}}^{\frac{2\pi k\nu}{2}} \cos(u) \frac{p}{2\pi k} du = O\left(\frac{1}{\sqrt{p}}\right) + \frac{1}{\pi k} \int_0^{\pi k\nu} \cos(u)\, du$$

and so

$$\hat{h}(k) = O\left(\frac{1}{\sqrt{p}}\right) + \frac{\sin(\pi k\nu)}{\pi k}$$

We now only need to show that when $|k| > \sqrt{p}$, $\hat{h}(k) = O(1/\sqrt{p})$.

Let $J \in \mathbb{N}$ and $j = j(J) \in \mathbb{F}_p$ with $J \equiv j \mod p$. We choose J to be minimal such that $jk \in [-\sqrt{p}, \sqrt{p}]$. Note that $J \le \sqrt{p}$ by the pigeonhole principle.

Suppose that $\nu p \le J$; then $\nu \le \frac{1}{\sqrt{p}}$, and so

$$\left| \frac{1}{p} \sum_{\frac{-p\nu}{2} \le x \le \frac{p\nu}{2}} e^{\frac{2\pi i}{p} kx} \right| \le \frac{p\nu}{p} = O\left(\frac{1}{\sqrt{p}}\right).$$

It also follows that for any $|c|, |d| < 1/\sqrt{p}$, we may say that

$$\left| \frac{1}{p} \sum_{\frac{-p\nu}{2} - c \le x \le \frac{p\nu}{2} + d} e^{\frac{2\pi i}{p} kx} \right| = \left| \frac{1}{p} \sum_{\frac{-p\nu}{2} \le x \le \frac{p\nu}{2}} e^{\frac{2\pi i}{p} kx} \right| + O\left(\frac{1}{\sqrt{p}}\right).$$

Rather than examining the sum over all $x \in \mathbb{F}_p$, we will instead examine a partial sum, where the values range between $m+1$ and $m+J$ for a parameter m.

$$\sum_{x=m+1}^{m+J} e^{\frac{2\pi i}{p} kx}. \tag{11}$$

We may rewrite this as

$$\sum_{x=1}^{J} e^{\frac{2\pi i}{p}k(x+m)}\,.$$

We begin by noting that there must be some element $y = a/J \in \mathbb{Q}$ satisfying $y \in [-\sqrt{p}/J, \sqrt{p}/J]$ and $yJ \equiv kj \mod p$. Then it follows that $yJ \equiv kJ \mod p$.

$$\sum_{x=1}^{J} e^{\frac{2\pi i}{p}k(x+m)} = \sum_{x=1}^{J} e^{\frac{2\pi i}{p}(k(x+m)-yx)} + \left(e^{\frac{2\pi i}{p}k(x+m)} - e^{\frac{2\pi i}{p}(k(x+m)-yx)}\right)\,.$$

Notice that

$$\sum_{x=1}^{J} e^{\frac{2\pi i}{p}(k(x+m)-yx)} = e^{\frac{2\pi i}{p}km} \sum_{x=1}^{J} e^{\frac{2\pi i}{p}(k-y)x}\,.$$

Since $kJ - yJ \equiv 0 \mod p$, it follows that $k - y = zp/J$ for some $z \in \mathbb{Z}$. As $k > \sqrt{p} > y$, it follows that $z \neq 0$. Then

$$\sum_{x=1}^{J} e^{\frac{2\pi i}{p}(k(x+m)-yx)} = e^{\frac{2\pi i}{p}km} \sum_{x=1}^{J} e^{\frac{2\pi i}{J}zx}\,.$$

Since $e^{\frac{2\pi i}{J}zx}$ is a J-th root of unity, it follows that

$$\sum_{x=1}^{J} e^{\frac{2\pi i}{p}(k(x+m)-yx)} = 0.$$

Then we may rewrite (11) again as

$$\sum_{x=m+1}^{m+J} e^{\frac{2\pi i}{p}kx} = \sum_{x=1}^{J}\left(e^{\frac{2\pi i}{p}k(x+m)} - e^{\frac{2\pi i}{p}(k(x+m)-yx)}\right) = e^{\frac{2\pi ikm}{p}} \sum_{x=1}^{J}\left(e^{\frac{2\pi i}{p}kx} - e^{\frac{2\pi i}{p}(kx-yx)}\right)\,.$$

It is now useful to define the constant

$$\sum_{x=1}^{J}\left(e^{\frac{2\pi i}{p}kx} - e^{\frac{2\pi i}{p}(kx-yx)}\right) = \lambda_k\,,$$

so that

$$\sum_{x=1}^{j} e^{\frac{2\pi i}{p}k(x+m)} = \lambda_k \cdot e^{\frac{2\pi i}{p}km}\,.$$

Note that for $\theta, \phi \in \mathbb{R}$,

$$\left|e^{i\theta} - e^{i\phi}\right| \leq |\theta - \phi|\,,$$

and so

$$|\lambda_k| \leq \sum_{x=1}^{J}\left|\left(\frac{2\pi}{p}kx - \frac{2\pi}{p}\left(kx - yx\right)\right)\right| \leq \frac{2\pi}{p}\sum_{x=1}^{J}|yx| = \frac{2\pi}{p}\,|y|\,\frac{J(J+1)}{2}\,.$$

Since $|y| \leq \sqrt{p}/J$,

$$|\lambda_k| \leq \frac{2\pi}{p} \frac{\sqrt{p}}{J} \frac{J(J+1)}{2} \leq \frac{4\pi J}{\sqrt{p}} .$$

By our previous remark, we will rewrite $\hat{h}(k)$ as a sum over partial sums of length J, incurring at most $O(1/\sqrt{p})$ error. We state this formally below.

$$\left|\hat{h}(k)\right| = \left|\frac{1}{p} \sum_{\frac{-p\nu}{2} \leq x \leq \frac{p\nu}{2}} e^{\frac{2\pi i}{p} kx}\right| = \frac{1}{p} \sum_{m=-J\left\lfloor \frac{p\nu}{2J} \right\rfloor}^{m=J\left\lfloor \frac{p\nu}{2J} \right\rfloor} \sum_{x=m+1}^{m+J} e^{\frac{2\pi i}{p} kx} + O\left(\frac{1}{\sqrt{p}}\right) . \quad (12)$$

We use our bounds from above to write

$$\left|\hat{h}(k)\right| \leq \frac{1}{p} |\lambda_k| \, 3 \left\lfloor \frac{p\nu}{2J} \right\rfloor \leq \frac{1}{p} \frac{4\pi J}{\sqrt{p}} 3 \left\lfloor \frac{p\nu}{2J} \right\rfloor .$$

Simplifying, it follows that

$$\left|\hat{h}(k)\right| \leq \frac{6\pi}{\sqrt{p}} .$$

Then it follows that

$$\sum_{k \neq 0} \left|\hat{f}(k)\right|^q \leq \sum_{k \neq 0} \left|2\hat{h}(k)\right|^q + O(1/p) \leq \sum_{k \neq 0} \left|2 \cdot \frac{1}{\pi} \frac{\sin(\pi k\nu)}{k}\right|^q + O\left(\frac{1}{\sqrt{p}}\right) .$$

Notice that combining our positive and negative k,

$$\sum_{k \neq 0} \left|\hat{f}(k)\right|^q \leq 2 \sum_{k=1}^{\sqrt{p}} \left|\frac{2}{\pi} \cdot \frac{\sin(\pi k\nu)}{k}\right|^q + O\left(\frac{1}{\sqrt{p}}\right) .$$

Note that $\nu = (\mu + 1)/2$, and so

$$\sum_{k \neq 0} \left|\hat{f}(k)\right|^q \leq 2 \sum_{k=1}^{\sqrt{p}} \left|\frac{2}{\pi} \cdot \frac{\sin\left(\pi k \frac{\mu+1}{2}\right)}{k}\right|^q + O\left(\frac{1}{\sqrt{p}}\right) ,$$

which is as we claimed.

3 A New Bound on $\left|\hat{f}_S(\ell_0)\right|$

In this section we will be relying on Lemma 1. As we will eventually be forced to round certain terms up, we will suppress the error term of $O(1/\sqrt{p})$. We will simply ask that p be sufficiently large that the error caused by the $O(1/\sqrt{p})$ term fits under the error induced by rounding.

Theorem 3. *Let $S \subset \mathbb{F}_p$ such that $|S| = (t + a)n$ with $a \leq t/4$. Then for n sufficiently large,*

$$\left|\hat{f}_S(\ell_0)\right| \leq \left(\frac{2}{\pi}\right)^{(t-0.66a)n} .$$

Lemma 2. *Let $S \subset \mathbb{F}_p$ and $T \subset S$ such that $|S| = (t+a)n$ and $|T| = 4an$. Then*

$$\left|\hat{f}_S(\ell_0)\right| \leq \prod_{i\in T} \left\|\hat{f}_i\right\|_{L^4} \cdot \prod_{j\in S\setminus T} \left\|\hat{f}_j\right\|_{L^\infty} .$$

This is a similar result to Corollary 4.4 of [12]. We provide proofs for a similar result in Lemma 8 and Corollary 1 in Appendix A. For this reason, we will omit the proof of this Lemma.

Recall $L_q(|\mu|)$,

$$L_q(\mu) = 2\sum_{k=1}^{\sqrt{p}} \left| \frac{2}{\pi} \cdot \frac{\sin\left(\pi k \frac{\mu+1}{2}\right)}{k} \right|^q \tag{13}$$

as defined in Definition 1. Then we define $K : [0,1] \to [0,1]$ as

$$K(|\mu|) = \left(L_4(|\mu|) + |\mu|^4\right)^{\frac{1}{4}} .$$

Notice that for each i,

$$\left\|\hat{f}_i\right\|_{\mathrm{L}^4} \leq K(|\mu_i|) .$$

It is easy to verify through an approximation of K that for all $x \in [0, 0.75]$, $K(x) \leq K(0)$, and the K is increasing on $[0.75, 1]$. We state the following Lemma, which will have an inductive proof using techniques from [12].

Lemma 3. *Let $S \subset \mathbb{F}_p$ and $T \subset S$ such that $|S| = (t+a)n$, $|T| = 4an$. Let $B \subset T$ such that for each $i \in B$, $|\mu_i| > 0.836$. Let $G \subset T$ such that for each $i \in G$, $|\mu_i| \leq 2/\pi$. Further, let $|G| = 3\,|B|$. Then*

$$\left|\hat{f}_S(\ell_0)\right| \leq (K(0))^{4|B|} \prod_{i\in T\setminus(B\cup G)} \left\|\hat{f}_i\right\|_{L^4} \cdot \prod_{j\in S\setminus T} \left\|\hat{f}_j\right\|_{L^\infty} .$$

Proof. We will induct on $|B|$, beginning by noting that when $|B| = 0$ the lemma holds by Lemma 2. Suppose then that $|B| > 0$. Then choose some $b \in B$ and a set $G' \subset G$ with $|G'| = 3$. To ease our notational burden we will assume without loss of generality that $\mu_b \geq 0$. Then notice that we may rewrite

$$\hat{f}_S(\ell_0) = \mu_b \widehat{f_{S\setminus b}}(\ell_0) + \widehat{(f_b - \mu_b) f_{S\setminus b}}(\ell_0) . \tag{14}$$

We will bound the two terms separately. First, notice that by the induction hypothesis on $|B|$, choosing $\bar{T} = T\setminus(b \cup G')$, $\bar{B} = B\setminus b$, and $\bar{G} = G\setminus G'$,

$$\left|\mu_b \hat{f}_{S\setminus b}(\ell_0)\right| \leq \mu_b \cdot \prod_{i\in G'} \left\|\hat{f}_i\right\|_{\mathrm{L}^\infty} \cdot (K(0))^{4|B|-4} \cdot \prod_{i\in\bar{T}\setminus(\bar{B}\cup\bar{G})} \left\|\hat{f}_i\right\|_{\mathrm{L}^4} \cdot \prod_{j\in S\setminus(\bar{T}\cup b\cup G')} \left\|\hat{f}_j\right\|_{\mathrm{L}^\infty} .$$

Firstly, note that $\bar{T}\setminus(\bar{B}\cup\bar{G}) = T\setminus(B\cup G)$. Further, $S\setminus(\bar{T}\cup b\cup G') = S\setminus T$. Finally, by hypothesis on G, for each $i \in G'$,

$$\left\|\hat{f}_i\right\|_{\mathrm{L}^\infty} \leq \left(\frac{2}{\pi K(0)}\right) K(0) ,$$

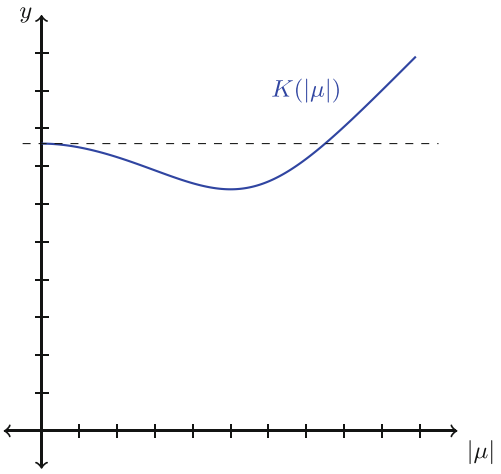


Fig. 2. A graph of $K(|\mu|)$ on $[0,1]$ with increments at 0.1 intervals.

and so we may conclude that

$$\left|\mu_b \hat{f}_{S\setminus b}(\ell_0)\right| \leq \mu_b \cdot \left(\frac{2}{\pi K(0)}\right)^3 \cdot (K(0))^{4|B|-1} \cdot \prod_{i\in T\setminus(B\cup G)} \left\|\hat{f}_i\right\|_{\mathrm{L}^4} \cdot \prod_{j\in S\setminus T} \left\|\hat{f}_j\right\|_{\mathrm{L}^\infty} .$$

Applying the induction hypothesis to the second term of (14), $\left|\widehat{(f_b - \mu_b) f_{S\setminus b}}\right|$ is bounded by

$$\left\|\widehat{f_b - \mu_b}\right\|_{\mathrm{L}^4} \cdot \prod_{i\in G'} \left\|\hat{f}_i\right\|_{\mathrm{L}^4} \cdot (K(0))^{4|B|-4} \prod_{i\in \bar{T}\setminus(\bar{B}\cup\bar{G})} \left\|\hat{f}_i\right\|_{\mathrm{L}^4} \cdot \prod_{j\in S\setminus(\bar{T}\cup b\cup G')} \left\|\hat{f}_j\right\|_{\mathrm{L}^\infty} .$$

We rewrite (3), using that for each $i \in G'$, $\left\|\hat{f}_i\right\|_{\mathrm{L}^4} \leq K(0)$, and so

$$\left|\widehat{(f_b - \mu_b) f_{S\setminus b}}\right| \leq (L_4(\mu_b))^{1/4} \cdot (K(0))^{4|B|-1} \prod_{i\in T\setminus(B\cup G)} \left\|\hat{f}_i\right\|_{\mathrm{L}^4} \cdot \prod_{j\in S\setminus T} \left\|\hat{f}_j\right\|_{\mathrm{L}^\infty} .$$

Then it suffices to show that

$$\mu_b \cdot \left(\frac{2}{\pi K(0)}\right)^3 + (L(\mu_b))^{1/4} \leq K(0) .$$

Unfortunately, this only holds for when $\mu_b \geq 0.836$, leading to the use of that bound, rather than the more desirable 0.75.

It is useful to note that

$$\mu_b \cdot \left(\frac{2}{\pi K(0)}\right)^3 + (L(\mu_b))^{1/4} = K(\mu_b)$$

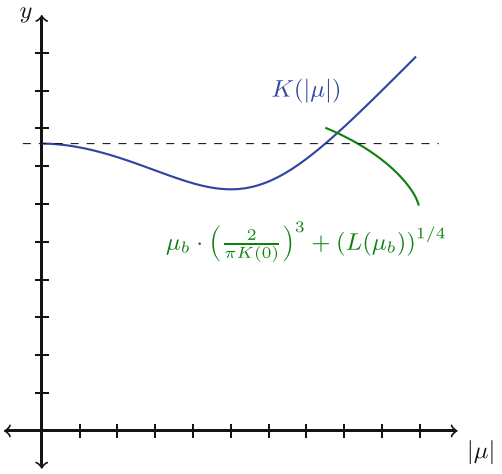


Fig. 3. A graph of displaying the induction bound compared against $K(|\mu|)$.

at approximately $\mu_b = 0.7817$. Since

$$\mu \cdot \left(\frac{2}{\pi K(0)}\right) + (L(\mu))^{1/4}$$

is decreasing in μ, we may run this argument again, but using a bound of $K(0.7818)$, by our remark above, to handle terms with $\mu \in (0.7817, 0.836]$ to gain the following Lemma.

Lemma 4. *Let $S \subset \mathbb{F}_p$ and $T \subset S$ such that $|S| = (t+a)n$, $|T| = 4an$. Let $B_1, B_2 \subset T$ such that for each $i \in B_1$, $|\mu_i| > 0.836$ and for each $j \in B_2$, $|\mu_j| \in (0.7817, 0.836]$. Let $G \subset T$ such that for each $i \in G$, $|\mu_i| \leq 2/\pi$. Further, let $|G| = 3\,|B_1 \cup B_2|$. Then*

$$\left|\hat{f}_S(\ell_0)\right| \leq (K(0))^{4|B_1|+3|B_2|} \cdot (K(0.7818))^{|B_2|} \prod_{i \in T \setminus (B_1 \cup B_2 \cup G)} \left\|\hat{f}_i\right\|_{L^4} \cdot \prod_{j \in S \setminus T} \left\|\hat{f}_j\right\|_{L^\infty}.$$

As an additional note, in the above Lemma our induction gives $K(0)^{|3|B_2||}$, which are the terms coming from our set G used to induct on B_2. The terms directly from B_2 provide the $K(0.7818)^{|B_2|}$.

From here we will restate the bound derived in [12], though we will preserve the extra terms

$$H(|\mu|) = \frac{2}{\pi}\,|\mu| + \cos\left(\frac{\pi}{2}\,|\mu|\right)$$

that come out of the induction argument. The reader may refer to a similar argument using L^2 bounds in the Appendix in Lemma 7, though one would modify it by taking $B = A$ and preserving the terms that come out due to induction. To avoid needless repetition of the argument, we omit the proof of Lemma 5.

Lemma 5. *Let $S \subset \mathbb{F}_p$ with $|S| = (t+a)n$, $a \leq 1-t$. Then let $B \subset T$ with $|B| \leq (t-a)n$ and for each $i \in B$, $|\mu_i| \geq 2/\pi$. Then*

$$\left|\hat{f}_S(\ell_0)\right| \leq \left(\frac{2}{\pi}\right)^{(t-a)n} \cdot \prod_{i \in B} H(|\mu_i|)\,.$$

To prove Theorem 3, we would like to get an extra saving of $(2/\pi)^{0.34an}$. This could happen if there are enough functions with large μ_i that the savings that we from $H(\mu)$ would give us this without a need for an L^4 argument. We proceed by examining when this occurs for some bounds on $|\mu|$.

We begin with the set

$$A = \{i \in S : |\mu_i| \in [0.836, 1]\}\ ,$$

which act as B_1 for Lemma 4. Then notice that

$$\prod_{i \in A} H(|\mu_i|) \leq (H(0.836))^{|A|} \leq \left(\frac{2}{\pi}\right)^{0.555|A|}\ .$$

Our second set is

$$B = \{i \in S : |\mu_i| \in [0.7817, 0.836)\}\ ,$$

which will act as B_2 for Lemma 4. Then we see that

$$\prod_{i \in B} H(|\mu_i|) \leq (H(0.7817))^{|B|} \leq \left(\frac{2}{\pi}\right)^{0.4|B|}\ .$$

Our third set is

$$C = \{i \in S : |\mu_i| \in [0.75, 0.7817)\}\ ,$$

for which we automatically obtain that $\left\|\hat{f}_i\right\|_{\mathrm{L}^4} \leq K(0.7817)$. We observe that

$$\prod_{i \in C} H(|\mu_i|) \leq (H(0.75))^{|C|} \leq \left(\frac{2}{\pi}\right)^{\frac{|C|}{3}}\ .$$

We now define our fourth set,

$$D = \{i \in S : |\mu_i| \in [2/\pi, 0.75)\}\ ,$$

which are the functions with $\left\|\hat{f}_i\right\|_{\mathrm{L}^4} \leq K(0)$, but $i \notin G$. Finally, we have that

$$\prod_{i \in E} H(|\mu_i|) \leq \left(H\left(\frac{2}{\pi}\right)\right)^{|D|} \leq \left(\frac{2}{\pi}\right)^{0.1238|D|}\ .$$

In the following proof, we will begin by arguing that if $|A \cup B \cup C \cup D|$ is large, then Lemma 5 gives us our result immediately. Then when $|A \cup B \cup C \cup D|$ is small, we may find a large enough set G to apply Lemma 4 and obtain our bound in this way.

Proof (Proof of Theorem 3). As $a < t/4$, it follows that $t - a > 3a$. Let our sets A, B, C, and D be defined as above. If $|A \cup B \cup C \cup D| > 3an$, we will remove elements from them until they are of size $3an$. Then notice that by Lemma 5,

$$\left|\hat{f}(\ell_0)\right| \leq \left(\frac{2}{\pi}\right)^{tn-an+0.555|A|+0.4|B|+\frac{|C|}{3}+0.1238|D|}.$$

It follows that if $0.555\,|A|+0.4\,|B|+\frac{|C|}{3}+0.1238\,|D| \geq 0.34an$, then we are done. We will assume then that

$$0.555\,|A| + 0.4\,|B| + \frac{|C|}{3} + 0.1238\,|D| \leq 0.34an\,, \tag{15}$$

Notice that for every $i \notin A \cup B \cup C \cup D$, $|\mu_i| \leq 2/\pi$. We want to construct a set G and a set T with $|T| = 4an$ and $G = 3(|A| + |B|$ so that $G, A, B, C, D \subseteq T$ in order to apply Lemma 4. We get this if we can obtain the bound that

$$3(|A| + |B|) \leq 4an - (|A| + |B| + |C| + |D|)\,. \tag{16}$$

We will begin by noticing that, by (15),

$$0.1238(|C| + |D|) \leq \frac{|C|}{3} + 0.1238\,|D| \leq$$
$$0.34an - 0.555\,|A| - 0.4\,|B| \leq 0.34an - 0.4\,(|A| + |B|)\,.$$

It follows that

$$|C| + |D| \leq \frac{0.34}{0.1238}an - \frac{0.4}{0.1238}\,(|A| + |B|)\,.$$

We now examine the right-hand side of (16), and see that

$$4an - |A| - |B| - |C| - |D| \geq 4an - |A| - |B| - \frac{0.34}{0.1238}an + \frac{0.4}{0.1238}(|A| + |B|)$$
$$\geq 1.25an + 2.232(|A| + |B|)\,.$$

Then (16) holds if

$$3(|A| + |B|) \leq 1.25an + 2.232(|A| + |B|)\,.$$

By moving over the $2.232(|A| + |B|)$ term and rounding aggressively, it suffices to prove that

$$|A| + |B| \leq an\,.$$

We immediately have this, as (15) gives that

$$0.555\,|A| + 0.4\,|B| \leq 0.34an\,,$$

and so we conclude that

$$3(|A| + |B|) \leq 4an - |A| - |B| - |C| - |D|\,.$$

Then let T be any set of size $4an$ with $A, B, C, D \subset T \subset S$. Let $G \subset T$ with $G \cap (A \cup B \cup C \cup D) = \emptyset$ and $|G| = 3(|A| + |B|)$. We let $B_1 = A$ and $B_2 = B$. Then we may apply Lemma 4 to obtain the bound

$$\left|\hat{f}_S(\ell_0)\right| \leq \left(\frac{2}{\pi}\right)^{(t-3a)n} (K(0))^{4an-|B|-|C|} \cdot (K(0.7818))^{|B|+|C|} .$$

It is clear that this bound is decreasing in $|B|+|C|$, but as we have from (15) that

$$|B| + |C| \leq 1.02an ,$$

it follows that

$$\left|\hat{f}_S(\ell_0)\right| \leq \left(\frac{2}{\pi}\right)^{t-3a} (K(0))^{2.98an} \cdot (K(0.7818))^{1.02an} .$$

From here, approximating $K(0)$ and $K(0.7818)$ gives that

$$\left|\hat{f}_S(\ell_0)\right| \leq \left(\frac{2}{\pi}\right)^{(t-3a)n} \cdot \left(\frac{2}{\pi}\right)^{2.35an} ,$$

which suffices for our bound.

Remark 1. This argument was run for $q = 4$, but it could plausibly be run for higher choices of q. It is worthwhile to note though, that

$$L_q(0) \geq \frac{2}{\pi} \cdot 2^{1/q} \tag{17}$$

for each q. Any application of Hölder along the lines of this argument will give

$$\left|\hat{f}_S(\ell_0)\right| \leq \prod_{i \in A} \left\|\hat{f}_i\right\|_{\mathrm{L}^q} \cdot \prod_{i \in B} \left\|\hat{f}_i\right\|_{\mathrm{L}^\infty} \tag{18}$$

where $|A| = qan$ and $|B| = (t + a - qa)n$. So, even with a successful induction argument, one should not expect these techniques to give bounds better than

$$\left|\hat{f}_S(\ell_0)\right| \leq 2^{an} \cdot \left(\frac{2}{\pi}\right)^{(t+a)n} \leq \left(\frac{2}{\pi}\right)^{(t-0.53a)n} . \tag{19}$$

4 A Bound via Subset Averaging

Theorem 4. *For each $0 \leq a \leq t/4$, and for every choice of K satisfying $a \leq K \leq t/2 - a$, the following bound holds.*

$$\sum_{|S|=(t+a)n} \left|\widehat{f_S}(\ell_0)\right|^2 \leq O\left(\binom{(t+a)n}{(t-K)n}^{-1} \cdot \binom{n}{(t-K)n} \cdot \left(\frac{2}{\pi}\right)^{2n(t-K-3a)}\right) .$$

Proof (Proof of Theorem 4).

Let $0 \leq a \leq 1-t$ such that $an \in \mathbb{N}$. We aim to give a bound on the size of

$$\sum_{|S|=n(t+a)} \left|\hat{f}_S\left(\ell_0\right)\right|^2 .$$

We start by choosing a parameter $K = K(a,t)$ which fulfills

$$a \leq K \leq t/2 - a\,.$$

We further require that $Kn \in \mathbb{N}$. Fix an $S' \subset [n]$ such that $|S'| = (t-K)n$. Next, select an $\tilde{S} \subseteq S'$ such that $\left|\tilde{S}\right| = (K+2a)n$. Finally, let $T \subset [n]$ be an arbitrary set such that $|T| = (K+a)n$ and $T \cap S' = \emptyset$.

Remark 2. In this argument, we will let $\tilde{S}$ be an arbitrary subset of S' of size $(K+2a)n$; however, we leave the choice of $\tilde{S}$ available in the argument in case others can find a way to exploit it.

Definition 2. *Let $\varphi \in \mathbb{F}_p^{S' \cup T}$. We say that (T, φ) is a* ***valid pair*** *if it satisfies*

$$\ell_0 = \sum_{i \in S'} \varphi_i \cdot \ell_i + \sum_{k \in T} \varphi_k \cdot \ell_k\,. \tag{20}$$

To clarify our notation, we consider $\mathbb{F}_p^{S' \cup T}$ to the space of $|S' \cup T|$-dimensional vectors with values in $\mathbb{F}_p$ and entries indexed by $S' \cup T$.

We next define $\lambda(T)$ to be a choice of vector such that $(T, \lambda(T))$ is a valid pair and

$$\prod_{i \in \tilde{S}} \left|\widehat{f_i}(\lambda_i(T))\right|$$

is maximized.

Let $\theta \in \mathbb{F}_p^{\tilde{S}}$ and define C_θ as the set of all valid pairs $(T, \lambda(T))$ such that $\forall i \in \tilde{S}$, $\lambda_i(T) = \theta_i$. Choose two valid pairs $(T, \lambda(T)), (T', \lambda(T')) \in C_\theta$. Notice that since each is a valid pair, we can subtract the two equations given by (20), to obtain

$$0 = \left(\sum_{k \in S' \setminus \tilde{S}} (\lambda_k(T) - \lambda_k(T')) \cdot \ell_k\right) + \left(\sum_{i \in T} \lambda_i(T) \cdot \ell_i\right) + \left(\sum_{j \in T'} \lambda_j(T') \cdot \ell_j\right) .$$

Then 0 is expressed as the sum of some vectors. We notice that the number of distinct vectors is at most

$$|T| + |T'| + \left|S' \setminus \tilde{S}\right| = n(K+a) + n(K+a) + n(t-K) - n(K+2a) = nt$$

vectors. But as any tn vectors are linearly independent, it follows that either $(T, \lambda(T)) = (T', \lambda(T'))$ or $\lambda(T')_i = 0$ for all $i \in (S' \setminus \tilde{S}) \cup T$. The second option would imply that

$$\ell_0 = \sum_{i \in \tilde{S}} \theta_i \cdot \ell_i\,,$$

which cannot happen as $\tilde{S}$ contains less than tn vectors. Then we conclude that $|C_\theta| \leq 1$.

Lemma 6. *For fixed S', $\tilde{S}$, and T, where $|T| \geq 2an$, we have that*

$$\left|\widehat{f}_{S'\cup T}\right| \leq O\left(\prod_{i\in\tilde{S}}\left|\widehat{f}_i\left(\lambda_i(T)\right)\right| \cdot \left(\frac{2}{\pi}\right)^{n(t-K-3a)}\right). \tag{21}$$

This Lemma uses an argument modified from that of Lemma 4.2 in [12]. The reader may find our proof in Appendix A.

We may reindex our summation so that

$$\sum_T \left|\widehat{f}_{S'\cup T}(\ell_0)\right|^2 = \sum_{\theta\in\mathbb{F}_p^{\tilde{S}}}\left(\sum_{T\in C_\theta}\left|\widehat{f}_{S'\cup T}(\ell_0)\right|^2\right).$$

Applying Lemma 6, it follows that

$$\sum_T \left|\widehat{f}_{S'\cup T}(\ell_0)\right|^2 \leq \sum_{\theta\in\mathbb{F}_p^{\tilde{S}}}\left(\sum_{T\in C_\theta} O\left(\prod_{i\in\tilde{S}}\left|\widehat{f}_i\left(\lambda_i(T)\right)\right|^2 \cdot \left(\frac{2}{\pi}\right)^{2n(t-K-3a)}\right)\right).$$

As each C_θ contains at most one element, we may rewrite this bound as

$$\sum_T \left|\widehat{f}_{S'\cup T}(\ell_0)\right|^2 = O\left(\left(\frac{2}{\pi}\right)^{2n(t-K-3a)} \cdot \sum_{\theta\in\mathbb{F}_p^{\tilde{S}}}\left(\prod_{i\in\tilde{S}}\left|\widehat{f}_i\left(\theta_i\right)\right|^2\right)\right).$$

We may re-order our sum in θ again so that

$$\prod_{i\in\tilde{S}}\left|\widehat{f}_i\left(\theta_i\right)\right|^2 = \prod_{i\in\tilde{S}}\sum_{k\in\mathbb{F}_p}\left|\hat{f}_i(k\right|^2 = \prod_{i\in\tilde{s}}\left\|\hat{f}_i\right\|_{\mathrm{L}^2}.$$

By Plancherel,

$$\sum_T \left|\widehat{f}_{S'\cup T}(\ell_0)\right|^2 = O\left(\left(\frac{2}{\pi}\right)^{2n(t-K-3a)}\right).$$

Notice that we may represent any $|S| = (t+a)n$ as $S'\cup T$ in $\binom{(t+a)n}{(t-k)n}$ different ways. Then

$$\sum_{|S|=(t+a)n}\left|\widehat{f}_S(\ell_0)\right|^2 = \binom{(t+a)n}{(t-k)n}^{-1} \cdot \sum_{|S'|=(t-k)n}\sum_{|T|=(k+a)n}\left|\widehat{f}_{S'\cup T}(\ell_0)\right|^2.$$

Using our previous bounds, we see that

$$\sum_{|S|=(t+a)n}\left|\widehat{f}_S(\ell_0)\right|^2 \leq O\left(\binom{(t+a)n}{(t-k)n}^{-1} \cdot \sum_{|S'|=(t-k)n}\left(\frac{2}{\pi}\right)^{2n(t-k-3a)}\right).$$

Counting our choices of S', we finally obtain that

$$\sum_{|S|=(t+a)n} \left|\widehat{f}_S(\ell_0)\right|^2 \leq O\left(\binom{(t+a)n}{(t-k)n}^{-1} \cdot \binom{n}{(t-k)n} \cdot \left(\frac{2}{\pi}\right)^{2n(t-k-3a)}\right).$$

5 Convergence When $t \geq 0.67$

What remains is to show that the results come together to give our promised result. We'll begin by examining the range of a over which our averaging argument, Theorem 4, gives decay. Recall that it says that, taking $K = a$,

$$\sum_{|S|=(t+a)n} \left|\widehat{f}_S(\ell_0)\right|^2 \leq \binom{(t+a)n}{(t-a)n}^{-1} \cdot \binom{n}{(t-a)n} \cdot \left(\frac{2}{\pi}\right)^{2(t-4a)n}.$$

Using Stirling's Approximation, we may rewrite this as, for n large,

$$\sum_{|S|=t+a} \left|\widehat{f}_S(\ell_0)\right|^2 \leq 2 \cdot \frac{\sqrt{4\pi an}(2an)^{2an} \cdot \sqrt{2\pi(t-a)n}((t-a)n)^{(t-a)n}}{\sqrt{2\pi(t+a)n}\,(n(t+a))^{n(t+a)}}$$
$$\cdot \frac{\sqrt{2\pi n} n^n}{\sqrt{2\pi(t-a)n}((t-a)n)^{(t-a)n} \cdot \sqrt{2\pi(n-nt+na)}(n-nt+na)^{n-nt+na}}$$
$$\cdot \left(\frac{2}{\pi}\right)^{2(t-4a)n}.$$

Simplifying, this reduces to

$$\sum_{|S|=t+a} \left|\widehat{f}_S(\ell_0)\right|^2 \leq 2\sqrt{\frac{2a}{(t+a)(1-t+a)}}$$
$$\cdot \left(\frac{(2a)^{2a}}{(t+a)^{t+a} \cdot (1-t+a)^{1-t+a}} \cdot \left(\frac{2}{\pi}\right)^{2(t-4a)}\right)^n.$$

Then it suffices to examine when

$$E_1(t,a) = \frac{(2a)^{2a}}{(t+a)^{t+a} \cdot (1-t+a)^{1-t+a}} \cdot \left(\frac{2}{\pi}\right)^{2(t-4a)} < 1. \tag{22}$$

Here we may plug in $t = 0.668$ from Theorem 1, and we see that (22) reaches 1 at approximately $a = 0.0054$ and $a = 0.0991$, and so we gain exponential decay for $x \in (0.0054, 0.0991)$. Applying logarithms and differentiating, we find that (22) is decreasing in t if

$$\frac{1-t+a}{t+a} \cdot \frac{4}{\pi^2} < 1, \tag{23}$$

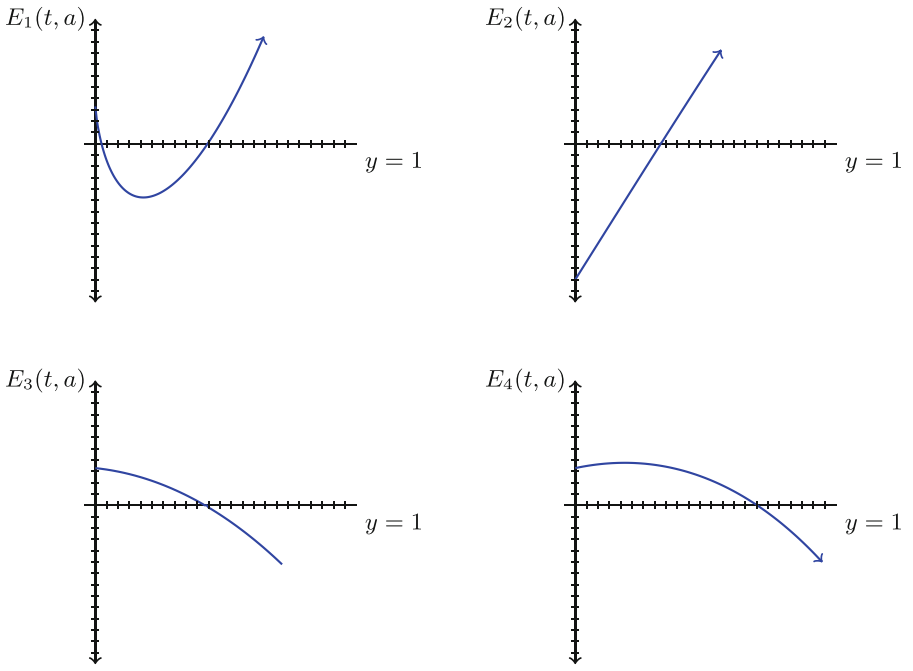


Fig. 4. Truncated graphs of each of the estimation functions E_i with $t = 0.688$. Each tick represents a length of 0.01. Except for E_3, which ends its domain at $a = t/4$, the other functions have extended domains that are truncated here for space. Rather than the a-axis, the line is drawn at $y = 1$, to make it clear when the E_i cross the threshold.

which holds for $t > 0.668$ as $1 - t + a < t + a$ for such values.

To cover the gap where $a \leq 0.0054$, we repeat this process with $K = t/2 - a$, so that

$$\sum_{|S|=(t+a)n} \left|\widehat{f}_S(\ell_0)\right|^2 \leq \binom{(t+a)n}{(t/2+a)n}^{-1} \cdot \binom{n}{(t/2+a)} \cdot \left(\frac{2}{\pi}\right)^{2(t/2-2a)n},$$

and so by Stirling's Approximation, it suffices to examine when

$$E_2(t,a) = \frac{(t/2)^{(t/2)}}{(t+a)^{t+a} \cdot (1-t/2-a)^{1-t/2-a}} \cdot \left(\frac{2}{\pi}\right)^{2(t/2-2a)} < 1.$$

Once again taking $t = 0.668$, we compute that the value is 0.8801 when $a = 0$, and the next time that it reaches 1 when $a = 0.07557$. Similarly applying logarithms and differentiating one may see that this is decreasing in t at fixed values of a.

We now may examine the efficacy of Theorem 3. This gives us that

$$\sum_{|S|=(t+a)n} \left|\widehat{f}_S(\ell_0)\right|^2 \leq \binom{n}{(t+a)n} \cdot \left(\frac{2}{\pi}\right)^{(t-0.66a)2n}$$

Once again, we may apply Stirling's Approximation, and it suffices to examine when

$$E_3(t,a) = \frac{1}{(t+a)^{t+a} \cdot (1-t-a)^{1-t-a}} \left(\frac{2}{\pi}\right)^{2(t-0.66a)} < 1\,,$$

and taking $t = 0.668$, we see that it holds when $a > 0.0936$. However, as our theorem requires that $a < t/4$, this only applies when $a < 0.167$. Once again, one may take logarithms and differentiate to show that this bound decreases in t.

We will finally apply the bound from [12], that

$$\sum_{|S|=(t+a)n} \left|\widehat{f}_S(\ell_0)\right|^2 \leq \binom{n}{(t+a)n} \cdot \left(\frac{2}{\pi}\right)^{(t-a)2n}\,.$$

By Stirling's approximation, we may instead examine when

$$E_4(t,a) = \frac{1}{(t+a)^{t+a} \cdot (1-t-a)^{1-t-a}} \left(\frac{2}{\pi}\right)^{2(t-a)} < 1\,.$$

This holds when $a > 0.1604$, Applying logarithms and derivatives, we see that these are decreasing in t for fixed a as well, and so our result holds, as we have covered all values of $0 \leq a < 1-t$ for $t \geq 0.668$.

6 Discussion

In a 2019 paper [1], Balister et al. proved a result implies the existence of functions $f_i : \mathbb{F}_p \to \{-1, 1\}$ so that for each $k \in \mathbb{F}_p$, $\left|\hat{f}_i(k)\right| \geq \delta/\sqrt{p}$, for p sufficiently large and some small fixed $\delta > 0$. This unfortunately means that one should not expect purely functional-analytic methods to work when $t \leq 1/2$, as the triangle inequality used one the convolution behaves as we highlight below. Recall that, for $|S| = n$,

$$\left|\hat{f}_S(\ell_0)\right| \leq \sum_{\substack{\lambda_1,\ldots\lambda_n \\ \sum_{i=1}^n \lambda_i \ell_i = \ell_0}} \prod_{i=1}^n \left|\hat{f_i}(\lambda_i\right| \tag{24}$$

is an inequality that is taken in obtaining our bounds. But for functions as generated by [1], and using our argument that $n - tn$ of the ℓ_i determine the convolution,

$$\sum_{\substack{\lambda_1,\ldots\lambda_n \\ \sum_{i=1}^n \lambda_i \ell_i = \ell_0}} \prod_{i=1}^n \left|\hat{f_i}(\lambda_i\right| \geq \sum_{\substack{\lambda_1,\ldots\lambda_n \\ \sum_{i=1}^n \lambda_i \ell_i = \ell_0}} \prod_{i=1}^n \frac{\delta}{\sqrt{p}} = p^{n-tn} \cdot \frac{\delta^n}{p^{n/2}} = p^{n/2-tn}\delta^n\,. \tag{25}$$

Then if $tn < n/2$, clearly the sum after taking the triangle inequality is large.

We would like to note that this does not necessarily mean that the analytic proxy fails in these cases, or that functional analytic techniques are useless, only that some additional argument that does not use these techniques will be necessary to handle functions that have this lower bound on their Fourier transforms.

We'll now attempt a more direct commentary on generalizing the techniques presented here. We suspect that one cannot push the bounds in Sect. 2 and Sect. 3 much further; as we remarked, the bounds on $\|f_i\|_{\mathrm{L}^q}$ can't do better than $2^{1/q} \cdot 2/\pi$. That said, these results may be useful for others, or in cases where a functional-analytic bound is useful after some additional cancellation argument has been made. We are hopeful that if an improved bound on $\left|\hat{f}_S(\ell_0\right|$ is found, then the argument made in Sect. 4 may be adapted to that setting. Most of the properties of the bound in [12] are not used directly, so it may be possible to use this argument to boost other bounds in future papers.

Acknowledgements. Thank you to Dr Simkin for introducing me to this problem. An additional thank you to Dr Magyar, Dr. Petridis, and Ms LaRue for helping refine the proof and writing. Thank you for the thoughtful comments of the reviewers to help improve this paper. This material is based upon work supported by the National Science Foundation under Grant No. 2054214.

A Appendix A

We will re-state Lemma 6 in slightly more general terms here that will be easier to remember throughout this proof. We begin by defining the set V, which we will use throughout this proof.

Definition 3. *Let V be the set of all $\varphi \in \mathbb{F}_p^{(t+a)n}$ such that*

$$\ell_0 = \sum_{i=1}^{(t+a)n} \varphi_i \cdot \ell_i \,. \tag{26}$$

Lemma 7. *For each $S \subset [n]$, $|S| = (t+a)n$ for $0 \leq t \leq 1$ and $0 \leq a \leq t$, let $B \subseteq S$ where $|B| = (t-a)n$ and let $A \subseteq B$. Then*

$$\left|\widehat{f_{S' \cup T}}(\ell_0)\right| \leq O\left(\left(\sup_{\varphi \in V} \prod_{i \in B \setminus A} \left|\hat{f}_i(\varphi_i\right|\right)\left(\frac{2}{\pi}\right)^{|A|}\right)$$

We define $\pi_1(\varphi) : \mathbb{F}_p^{tn+an} \to \mathbb{F}_p^{tn}$ as

$$\pi_1(\varphi) = \sum_{i=1}^{an} \varphi_i \ell_i \,.$$

Similarly, we define

$$\pi_2(\varphi) = \sum_{i=tn+1}^{tn+an} \varphi_i \ell_i$$

and

$$\pi_3(\varphi) = \sum_{i=an+1}^{tn} \varphi_i \ell_i \,.$$

We can now state and prove a variant of Lemma 4.3 from [12], page 17.

Lemma 8. *Let $0 < t \leq 1$, $nt \in \mathbb{N}$, $0 \leq a \leq 1-t$, and $\{\ell_i\}_{i=0}^{(t+a)n}$ be vectors in $\mathbb{F}_p^{tn}$ such that every tn of them are linearly independent. Let $A, B : \mathbb{F}_p^{an} \to \{-1, 1\}$ and $C : \mathbb{F}_p^{(t-a)n} \to \{-1, 1\}$ be any functions. We write*

$$F(x) = A(\ell_1 \cdot x, ..., \ell_{an} \cdot x) \cdot C(\ell_{an+1} \cdot x, ..., \ell_{tn} \cdot x) \cdot B(\ell_{tn+1} \cdot x, ..., \ell_{(t+a)n} \cdot x) \,. \quad (27)$$

Then

$$\left|\widehat{F}(\ell_0)\right| \leq \|A\|_{L^2} \cdot \|B\|_{L^2} \cdot \sup_{\varphi \in V} \left|\widehat{C}(\pi_3(\varphi))\right| \quad (28)$$

Proof. We write $A'(x)$, $B'(x)$, and $C'(x)$ to suppress the use of the ℓ_i such that $F(x) = A'(x) \cdot B'(x) \cdot C'(x)$. As the Fourier transformation of a product is a convolution of Fourier transformations, we may write

$$\widehat{F}(\ell_0) = \left(\widehat{A' \cdot B' \cdot C'}\right)(\ell_0) = \sum_{\beta+\gamma+\delta=\ell_0} \widehat{A'}(\beta)\widehat{B'}(\gamma)\widehat{C'}(\delta) \quad (29)$$

We claim that we may rewrite this again to have that

$$\hat{F}(\ell_0) = \sum_{\varphi \in V} \widehat{A'}(\pi_1(\varphi))\widehat{B'}(\pi_2(\varphi))\widehat{C'}(\pi_3(\varphi)) \,. \quad (30)$$

To prove this, consider $\hat{A}'(\beta)$ for some β linearly independent of $\{\ell_i\}_{i=1}^{an}$. Then choose some vector ℓ^* that is orthogonal to the ℓ_i so that for some k_i, k^*,

$$\beta = k^*\ell^* + \sum_{i=1}^{an} k_i \ell_i \,.$$

Let $\perp \ell^*$ be the set of vectors in $\mathbb{F}_p^{tn}$ perpendicular to ℓ^*. Recall that

$$\hat{A}'(\beta) = \frac{1}{p^t} \sum_{x \in \perp \ell^*} \sum_{k \in \mathbb{F}_p} A'(x + k\ell^*) e^{\frac{2\pi i}{p}(x+k\ell^*)\cdot\beta}$$

Since each $\ell_i \cdot \ell^* = 0$, it follows that

$$\hat{A}'(\beta) = \frac{1}{p^t} \sum_{x \in \perp \ell^*} A'(x) \sum_{k \in \mathbb{F}_p} e^{\frac{2\pi i}{p}(x+k\ell^*)\cdot\beta} = 0 \,.$$

Therefore, we will only sum over precisely those β such that $\beta = \pi_1(\varphi)$, $\varphi \in \mathbb{F}_p^{tn+an}$. We may do an identical argument for B' and C'. Then for such a φ, it is necessary that $\pi_1(\varphi) + \pi_2(\varphi) + \pi_3(\varphi) = \ell_0$, and so $\varphi \in V$, which proves our claim.

Then we may write that

$$\left|\widehat{F}(\ell_0)\right| \leq \sup_{\varphi \in V} \left|\widehat{C'}(\pi_3(\varphi))\right| \cdot \sum_{\varphi \in V} \left|\widehat{A'}(\pi_1(\varphi))\widehat{B'}(\pi_2(\varphi))\right| .$$

If we fix $\pi_1(\varphi)$, notice that

$$\ell_0 - \sum_{i=1}^{an} \varphi_i \ell_i = \sum_{i=an+1}^{an+tn} \varphi_i \ell_i .$$

There are exactly tn vectors on the right-hand side of the equation, and thus we conclude that $\pi_2(\varphi)$ and $\pi_3(\varphi)$ are determined by $\pi_1(\varphi)$ when $\varphi \in V$. Then if W is the span of $\{\ell_i\}_{i=1}^{an}$, we may define $\pi_1^{-1} : W \to V$ in the natural way. Thus,

$$\left|\widehat{F}(\ell_0)\right| \leq \sup_{\varphi \in V} \left|\widehat{C'}(\pi_3(\varphi))\right| \cdot \sum_{\phi \in W} \left|\widehat{A'}(\phi)\widehat{B'}(\pi_2\left(\pi_1^{-1}(\phi)\right))\right| .$$

It follows that our previous logic that $\pi_2\left(\pi_1^{-1}(\phi)\right)$ is a bijection onto the span of ℓ_i for $i \in [tn+1, tn+an]$, and so we may apply Cauchy-Schwartz and Plancherel to obtain our result.

We now begin by stating an analogue of Corollary 4.4 from page 17 of [12].

Corollary 1. *Let $B \subset S$ with $|B| = (t-a)n$. Then*

$$|f_S(\ell_0)| \leq \sup_{\varphi \in V} \left(\prod_{i \in B} \left\| \widehat{f_i} \right\|_{L^\infty} \right) . \tag{31}$$

Proof. Choose two sets, T_1, T_2 with $|T_1| = |T_2| = an$ and $T_1 \sqcup T_2 \sqcup B = S$. Then we define

$$A' = \prod_{i \in T_1} f_i(\ell_i \cdot x) \tag{32}$$

$$B' = \prod_{i \in T_2} f_i(\ell_i \cdot x) \tag{33}$$

$$C' = \prod_{i \in B} f_i(\ell_i \cdot x) . \tag{34}$$

We can apply Lemma 8 in the natural way to, we have that

$$\left|\widehat{f_S}(\ell_0)\right| \leq \left(\prod_{i \in T_1} \left\| \widehat{f_j} \right\|_{\mathrm{L}^2} \right) \cdot \left(\prod_{j \in T_2} \left\| \widehat{f_j} \right\|_{\mathrm{L}^2} \right) \cdot \left(\sup_{\varphi \in V} \prod_{B} \left| \widehat{f_i}(\varphi_i) \right| \right) \tag{35}$$

As each f_i maps to either 1 or -1, it follows that for all i, $\left\| \hat{f}_i \right\|_{\mathrm{L}^2} = 1$

Before proceeding with our proof of Lemma 6, we recall Claim 4.5 from [12], which we will use directly.

Lemma 9. *Let $f : \mathbb{F}_p \to [-1, 1]$ be a function with $\mathbb{E}[f] = \mu$. Then for all $k \neq 0$ we have*

$$\left|\widehat{f}(k)\right| \leq \frac{2}{\pi} \cos\left(\frac{\pi}{2}\mu\right) + O(1/p^2)\,. \tag{36}$$

Proof (Proof of Lemma 7). Notice that if $B = S' \cup T'$ and $A = (S' \cup T')\backslash \tilde{S}$, our statement implies Lemma 6.

We will follow with the convention in [12] by suppressing the extra $O(1/p^2)$ term that comes from Lemma 9.

Let $|S| = tn + an$. Note that the bound trivially holds when $an < 0$, as $\hat{f}_S(\ell_0) = 0$. Then we will begin by inducting on an for $an \geq 0$. By Corollary 1, the bound holds whenever $|A| = 0$, so we will also induct on $|A|$, assuming that the bound holds for all $|A'| < |A|$.

We will begin by selecting an arbitrary index $I \in A$. Notice that if $\left\|\widehat{f}_I\right\|_{\mathrm{L}^\infty} \leq \frac{2}{\pi} + O(1/p^2)$, then we apply the induction hypothesis on $A\backslash\{I\}$ so that

$$\left|\widehat{f_S}(\ell_0)\right| \leq \left(\frac{2}{\pi}\right)^{|A|-1} \cdot \sup_{\varphi \in V} \left(\prod_{i \in B\backslash(A\backslash\{I\})} \left|\widehat{f}_i(\varphi_i)\right| \right)\,. \tag{37}$$

We may extract a term of $\left\|\hat{f}_I\right\|_{\mathrm{L}^\infty}$ from the second term in the product, and this gives us the extra factor of $2/\pi$ that we need to conclude the proof.

Then consider the case when $\left\|\widehat{f}_I\right\|_{\mathrm{L}^\infty} \geq 2/\pi$; if this is the case, then $|\mathbb{E}[f_i]| = |\mu_i| \geq 2/\pi$, as Lemma 9 tells us that the largest value of $\widehat{f_I}$ away from 0 cannot be larger that $2/\pi$. We will now define the balance function $g = f_I - \mu$. Applying Lemma 9 to f_I, along with the fact that $\widehat{\mu}(k) = 0$ for $k \neq 0$, it follows that $\|g\|_{\mathrm{L}^\infty} \leq 2/\pi \cos(\pi\mu/2)$.

Notice that we may write

$$\widehat{f_S}(\ell_0) = \mu \cdot \widehat{f}_{S\backslash\{I\}}(\ell_0) + \widehat{\left(g \cdot f_{S\backslash\{I\}}\right)}(\ell_0) \tag{38}$$

We will bound the two terms in Equation (38) separately using the induction hypothesis, and this will give us our result.

To bound $\mu \cdot \widehat{f}_{S\backslash\{I\}}(\ell_0)$, we begin by noticing that if $an - 1 < 0$, we are done, as $\widehat{f}_{S\backslash\{I\}}(\ell_0) = 0$ in that case. We assume then that $an > 0$.

Choose two indices $J, K \in S\backslash B$. We define the sets $A' = (A\backslash\{I\}) \cup \{J, K\}$, $B' = (B\backslash\{I\}) \cup \{J, K\}$, and $S' = S\backslash\{I\}$. We may apply the induction hypothesis for $|S'| = tn + an - 1$ to see that

$$\left|\mu \cdot \widehat{f}_{S\backslash\{I\}}(\ell_0)\right| \leq |\mu| \cdot \left(\frac{2}{\pi}\right)^{|A'|} \cdot \sup_{\varphi \in V} \left(\prod_{i \in B'\backslash A'} \left|\widehat{f}_i(\varphi_i)\right| \right)\,. \tag{39}$$

First notice that $B'\backslash A' = B\backslash A$. Further, we may compute that $|A'| = |A| + 1$, and so

$$\left|\mu \cdot \widehat{f}_{S\backslash\{I\}}(\ell_0)\right| \leq |\mu| \cdot \left(\frac{2}{\pi}\right)^{|A|+1} \cdot \sup_{\varphi \in V} \left(\prod_{i \in B\backslash A} \left|\widehat{f}_i(\varphi_i)\right| \right)\,. \tag{40}$$

Bounding the second term of (38) is simpler, as we will simply choose $A' = A \setminus \{I\}$ and apply the induction hypothesis for $|A| - 1$ to see that

$$\left|\widehat{(g \cdot f_{S\setminus\{I\}})}(\ell_0)\right| \leq \left(\frac{2}{\pi}\right)^{|A'|} \cdot \sup_{\varphi \in V} \left(|\hat{g}(\varphi_I)| \cdot \prod_{i \in B \setminus A} \left|\widehat{f}_i(\varphi_i)\right| \right). \tag{41}$$

We now apply that $\|\hat{g}\|_{\mathrm{L}^\infty} \leq 2/\pi \cos(\pi/2\mu)$ to say that

$$\left|\widehat{(g \cdot f_{S\setminus\{I\}})}(\ell_0)\right| \leq \left(\frac{2}{\pi}\right)^{|A|-1} \cdot \left|\frac{2}{\pi} \cos\left(\frac{\pi}{2}\mu\right)\right| \cdot \sup_{\varphi \in V} \left(\prod_{i \in B \setminus A} \left|\widehat{f}_i(\varphi_i)\right| \right). \tag{42}$$

Then it suffices to show that

$$\begin{aligned} |\mu| \cdot \left(\frac{2}{\pi}\right)^{|A|+1} \cdot \sup_{\varphi \in V} \left(\prod_{i \in B \setminus A} \left|\widehat{f}_i(\varphi_i)\right| \right) \\ + \left(\frac{2}{\pi}\right)^{|A|-1} \cdot \left|\frac{2}{\pi} \cos\left(\frac{\pi}{2}\mu\right)\right| \cdot \sup_{\varphi \in V} \left(\prod_{i \in B \setminus A} \left|\widehat{f}_i(\varphi_i)\right| \right) \\ \leq \left(\frac{2}{\pi}\right)^{|A|} \cdot \sup_{\varphi \in V} \left(\prod_{i \in B \setminus A} \left|\widehat{f}_i(\varphi_i)\right| \right). \end{aligned} \tag{43}$$

We may divide through by the common terms, and so it is sufficient to show that

$$\frac{2}{\pi} |\mu| + cos\left(\frac{\pi}{2}\mu\right) \leq 1. \tag{44}$$

As we are working under the restriction that $2/\pi \leq |\mu| \leq 1$, this holds for all $|\mu|$ in the range, concluding our proof.

B Appendix B

In this Appendix we will briefly state our definitions of the Fourier transform, as well as some standard results on them.

We define the characters of the group $\mathbb{F}_p$ as $\chi_k : \mathbb{F}_p \to \mathbb{C}$ as

$$\chi_k(x) = e^{-\frac{2\pi i}{p} kx}$$

and the characters of $\mathbb{F}_p^{tn}$, $\chi_\varphi : \mathbb{F}_p^t \to \mathbb{C}$, as

$$\chi_\varphi(x) = e^{\frac{2\pi i}{p} \phi \cdot x} \tag{45}$$

Then for $f : \mathbb{F}_p \to \mathbb{C}$, $k \in \mathbb{F}_p$, we define

$$\hat{f}(k) = \frac{1}{p} \sum_{x \in \mathbb{F}_p} f(x) \chi_k(x) = \frac{1}{p} \sum_{x \in \mathbb{F}_p} f(x) e^{\frac{2\pi i}{p} k \cdot x}$$

and for $g \in \mathbb{F}_p^{tn}$, $\varphi \in \mathbb{F}_p^{tn}$,

$$\hat{g}(\varphi) = \frac{1}{p^{tn}} \sum_{x \in \mathbb{F}_p} g(x)\chi_\varphi(x) = \frac{1}{p^{tn}} \sum_{x \in \mathbb{F}_p} g(x) e^{\frac{2\pi i}{p}\varphi \cdot x}.$$

One of the Fourier-analytic theorems that we use in this paper are Plancherel's theorem, which states that, for $f_i : \mathbb{F}_p \to \mathbb{C}$,

$$\|f_i\|_{\mathrm{L}^2} = p \left\| \hat{f}_i \right\|_{\mathrm{L}^2},$$

and similarly for $g_i : \mathbb{F}_p^{tn} \to \mathbb{C}$, we have that $\|g_i\|_{\mathrm{L}^2} = p^{tn} \|\hat{g}_i\|_{\mathrm{L}^2}$. We also use the fact that for two functions $f, g : \mathbb{F}_p \to \mathbb{C}$,

$$\widehat{fg}(\varphi) = \sum_{\alpha+\beta=\varphi} \hat{f}(\alpha) \cdot \hat{g}(\beta) = (\hat{f} * \hat{g})(\varphi).$$

We also will take the time here to formally define leakage resilience. We begin by defining the function $leak : \mathbb{F}_p^{tn} \to \mathbb{F}_p^n$ as $leak(x) = (f_1(\ell_1 \cdot x), ..., f_n(\ell_n \cdot x))$, representing the leaked bits that the adversary sees. Then the adversary can expect that one secret s is more likely than another s', given a fixed $leak(x) = A$, if

$$|P\left(leak(x) = A | \ell_0 \cdot x = s\right) - P\left(leak(x) = A | \ell_0 \cdot x = s'\right)|$$

is large. Here we use probability and conditional probability in the standard ways. We say that the scheme is leakage resilient if the expected amount of information gained,

$$\sum_{A \in \mathbb{F}_p^n} |P\left(leak(x) = A | \ell_0 \cdot x = s\right) - P\left(leak(x) = A | \ell_0 \cdot x = s'\right)|,$$

tends to zero for each s, s' as n tends to infinity.

C Appendix C

Desmos Link for Figure 1

Desmos Link for Figure 2

Desmos Link for Figure 3

Desmos Link for Figure 4

References

1. Balister, P., et al.: Flat littlewood polynomials exist. Ann. Math. **192**(3)977–1004 (2020). https://doi.org/10.4007/annals.2020.192.3.6
2. Ben-Or, M., Goldwasser, S., Wigderson, A.: Completeness theorems for non-cryptographic fault-tolerant distributed computation. In: Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing. STOC '88. Chicago, Illinois, USA: Association for Computing Machinery, 1988, pp. 1–10. ISBN: 0897912640. https://doi.org/10.1145/62212.62213
3. Benhamouda, F., et al.: On the Local Leakage Resilience of Linear Secret Sharing Schemes. In: J. Cryptology **34** (2018). https://api.semanticscholar.org/CorpusID:206716311
4. Benhamouda, F., et al.: On the local leakage resilience of linear secret sharing schemes. In: J. Cryptology **34**(2) (2021). Publisher Copyright: 2021, The Author(s), under exclusive licence to International Association for Cryptologic Research. ISSN: 0933-2790. https://doi.org/10.1007/s00145-021-09375-2
5. Chaum, D., Crépeau, C., Damgard, I.: Multiparty unconditionally secure protocols. In: Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing. STOC '88. Chicago, Illinois, USA: Association for Computing Machinery, 1988, pp. 11–19 (1988). ISBN: 0897912640. https://doi.org/10.1145/62212.62214.
6. De Santis, A., et al.: How to share a function securely. In: Proceedings of the Twenty-Sixth Annual ACM Symposium on Theory of Computing. STOC '94. Montreal, Quebec, Canada: Association for Computing Machinery, 1994, pp. 522–533 (1994). ISBN: 0897916638. https://doi.org/10.1145/195058.195405.
7. Desmedt, Y., Frankel, Y.: Threshold cryptosystems. In: Advances in Cryptology - CRYPTO' 89 Proceedings. Ed. by Gilles Brassard. New York, NY: Springer New York, 1990, pp. 307–315 (1990). ISBN: 978-0-387-34805-6
8. Faust, S., et al.: Protecting circuits from leakage: the computationally- bounded and noisy cases. In: Advances in Cryptology - EUROCRYPT 2010. Ed. by Henri Gilbert. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 135–156 (2010). ISBN: 978-3-642-13190-5
9. Frankel, Y.: A practical protocol for large group oriented networks. In: Advances in Cryptology - EUROCRYPT '89. Ed. by Jean-Jacques Quisquater and Joos Vandewalle. Berlin, Heidelberg: Springer Berlin Heidelberg, 1990, pp. 56–61 (1990). ISBN: 978-3-540-46885-1

10. Goldreich, O., Micali, S., Wigderson, A.: How to play ANY mental game. In: Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing. STOC '87. New York, New York, USA: Association for Computing Machinery, 1987, pp. 218–229 (1987). ISBN: 0897912217. https://doi.org/10.1145/28395.28420.
11. Ishai, Y., Sahai, A., Wagner, D.: Private circuits: securing hardware against probing attacks. In: Advances in Cryptology - CRYPTO 2003. Ed. by Dan Boneh. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 463–481 (2003). ISBN: 978-3-540-45146-4
12. Klein, O., Komargodski, I.: New bounds on the local leakage resilience of Shamir's secret sharing scheme. Cryptology ePrint Archive, Paper 2023/805 (2023). https://eprint.iacr.org/2023/805
13. Lev, V.F.: Linear equations over Fp and moments of exponential sums. Duke Math. J. **107**(2), 239–263 (2001). https://doi.org/10.1215/S0012-7094-01-10722-9
14. Maji, H.K., et al.: Constructing locally leakage-resilient linear secret-sharing schemes. In: Advances in Cryptology - CRYPTO 2021 - 41st Annual International Cryptology Conference, CRYPTO 2021, Proceedings. Ed. by Tal Malkin and Chris Peikert. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). Publisher Copyright: 2021, International Association for Cryptologic Research.; 41st Annual International Cryptology Conference, CRYPTO 2021 ; Conference date: 16-08- 2021 Through 20-08-2021. Springer Science and Business Media Deutschland GmbH, 2021, pp. 779–808 (2021). ISBN: 9783030842512. https://doi.org/10.1007/978-3-030-84252-9_26
15. Maji, H.K., et al.: Improved bound on the local leakage-resilience of shamir's secret sharing. In: 2022 IEEE International Symposium on Information Theory (ISIT). Espoo, Finland: IEEE Press, 2022, pp. 2678– 2683. https://doi.org/10.1109/ISIT50566.2022.9834695
16. Nielsen, J.B., Simkin, M.: Lower bounds for leakage-resilient secret sharing. In: Advances in Cryptology - EUROCRYPT (2020). https://eprint.iacr.org/2019/181
17. Rothblum, G.N.: How to compute under AC0 leakage without secure hardware. In: Advances in Cryptology - CRYPTO 2012. Ed. by Reihaneh Safavi-Naini and Ran Canetti. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 552–569 (2012). ISBN: 978-3-642-32009-5
18. Shamir, A.: How to share a secret. In: Commun. ACM **22**(11), 612–613 (1979).ISSN: 0001-0782. https://doi.org/10.1145/359168.359176

Information-Theoretic Multi-server Private Information Retrieval with Client Preprocessing

Jaspal Singh[1,2(✉)], Yu Wei[2], and Vassilis Zikas[2]

[1] Purdue University, West Lafayette, USA
sing1361@purdue.edu
[2] Georgia Institute of Technology, Atlanta, USA
{ywei368,vzikas}@gatech.edu

Abstract. A private information retrieval (PIR) protocol allows a client to fetch any entry from single or multiple servers who hold a public database (of size n) while ensuring no server learns any information about the client's query. Initial works on PIR were focused on reducing the communication complexity of PIR schemes. However, standard PIR protocols are often impractical to use in applications involving large databases, due to its inherent large server-side computation complexity, that's at least linear in the database size. Hence, a line of research has focused on considering alternative PIR models that can achieve improved server complexity.

The model of private information retrieval with client prepossessing has received a lot of interest beginning with the work due to Corrigan-Gibbs and Kogan (Eurocrypt 2020). In this model, the client interacts with two servers in an offline phase and it stores a local state, which it uses in the online phase to perform PIR queries. Constructions in this model achieve online client/server computation and bandwidth that's sublinear in the database size, at the cost of a one-time expensive offline phase. Till date all known constructions in this model are based on symmetric key primitives or on stronger public key assumptions like Decisional Diffie-Hellman (DDH) and Learning with Error (LWE). This work initiates the study of unconditional PIR with client prepossessing - where we avoid using any cryptographic assumptions. We present a new PIR protocol for $2t$ servers (where $t \in [2, \log_2 n/2]$) with threshold 1, where client and server online computation is $\widetilde{\mathcal{O}}(\sqrt{n})$[1] - matching the computation costs of other works based on cryptographic assumptions. The client storage and online communication complexity are $\widetilde{\mathcal{O}}(n^{0.5+1/2t})$ and $\widetilde{\mathcal{O}}(n^{1/2})$ respectively. Compared to previous works our PIR with client preprocessing protocol also has a very concretely efficient client/server online computation phase - which is dominated by xor operations, compared to cryptographic operations that are orders of magnitude slower. As a building block for our construction, we introduce a new information-theoretic primitive called *privately multi-puncturable*

Work done while the authors were at Purdue University.

E. Boyle and M. Mahmoody (Eds.): TCC 2024, LNCS 15367, pp. 423–450, 2025.
https://doi.org/10.1007/978-3-031-78023-3_14

random set (PMPRS), which might be of independent interest. This new primitive can be viewed as a generalization of privately puncturable pseudo-random set, which is the key cryptographic building block used in previous works on PIR with client preprocessing.([1] the $\widetilde{\mathcal{O}}(.)$ notation hides poly log factors)

1 Introduction

First introduced by Chor et al. [9], a private information retrieval (PIR) protocol allows a client to fetch any entry of a public database held by a single or multiple non-colluding servers. A line of works beginning with Chor et al. [9] have focused on reducing the communication complexity of PIR in the single and multiple server cases under various cryptographic assumptions [3,5,8,14,15,33]. PIR has been employed as a useful building block for many cryptographic applications, including private contact discovery [12,21], anonymous communication [29], and safe browsing [24].

Standard PIR protocols however are generally inefficient when they are used for applications involving very large databases. One major factor contributing to the inefficiency of all known PIR schemes is the linear server computation complexity per query. This inefficiency is inherent in the standard PIR model in both the single and multi-server case and both the statistical and computational setting [6]. Hence, a line of research has focused on considering alternative PIR models, that allow for sublinear server complexity per query, either in the worst case or in an amortized sense. These includes models focused on batch PIR queries [1,2,6,19,22,27,32] and PIR with pre-processing [6,10,20,26,31,34]. Specifically, the PIR with client pre-processing model has garnered a lot of attention beginning with a work by Corrigan-Gibbs and Kogan [11].

PIR with Client Preprocessing. In this 2-server PIR model introduced by Corrigan-Gibbs and Kogan [11], the client interacts with two non-colluding servers during an offline phase where the servers receive as input a database of size n. At the end of this phase the client maintains some sublinear sized state and there's no state stored on the server side. This offline phase is often computationally expensive - with each server doing linear computation in the database size. In the online phase, the client can make an unbounded number of PIR queries using its stored state - such that both online communication and online client/server computation are sublinear in the database size! In [11] the authors were able to construct a PIR protocol in this model with $\widetilde{\mathcal{O}}(n^{1/2})$ online client/server computation and $\widetilde{\mathcal{O}}\ (n^{1/2})$ client state size. Furthermore, in the online phase, the client query size is $\widetilde{\mathcal{O}}\ (n^{1/2})$ and the server response is $\mathcal{O}(1)$ - leading to online communication complexity of $\widetilde{\mathcal{O}}(n^{1/2})$. Their original 2-server construction is based on one-way functions (OWF), but since then its also been extended to the single server model and its been improved using other cryptographic primitives [10,16,17,20,26,30,31,34]. The key building cryptographic building block used in [11] and follow-up works is some variant of *privately puncturable pseudo-random set*, which we describe in greater detail next. We refer to

all PIR preprocessing protocols that are based on this primitive to be designed in the *Corrigan-Gibbs and Kogan (CGK) paradigm.*

Corrigan-Gibbs and Kogan (CGK) Paradigm (Based on Privately Puncturable Pseudo-Random Sets.) A privately puncturable pseudo-random set consists of four algorithms $(\mathsf{Gen}, \mathsf{Set}, \mathsf{Test}, \mathsf{Punc})$. Gen is a randomized function that outputs a short key k corresponding to a pseudo-random set. Function $\mathsf{Set}(k)$ outputs the corresponding pseudo-random set, which has distribution computationally indistinguishable from a random set of size $\sqrt{n}$ from domain $[n] = \{0, 1, \ldots, n-1\}$. Function $\mathsf{Test}(k, x)$ outputs a bit checking whether $x \stackrel{?}{\in} \mathsf{Set}(k)$. $\mathsf{Punc}(k, x)$ outputs a punctured key k', such that $\mathsf{Set}(k') = \mathsf{Set}(k) \setminus \{x\}$ and the key k' hides x. The first construction for this primitive in [11] was based on pseudo-random permutations - where the key and punctured key have sizes κ and $\widetilde{\mathcal{O}}(\sqrt{n})$ respectively, where κ is the private key security parameter. The computation complexity of Test and Set algorithms are $\widetilde{\mathcal{O}}(1)$ and $\widetilde{\mathcal{O}}(\sqrt{n})$ respectively.

A very rough sketch of the PIR with client preprocesisng scheme of [11] is as follows: the client generates $T = \widetilde{\mathcal{O}}(\sqrt{n})$ privately puncturable pseudo-random set keys $(k_1, k_2, \ldots, k_T)$ and it sends them to the first server in the offline phase. This server responds back with hint bits $h_i = \oplus_{j \in \mathsf{Set}(k_i)} DB[j]$ where DB is a database of size n held by both parties. The client stores these T keys and the corresponding hint bits as its client state. In the online phase, the client receives as input some queries $x \in [n]$ and it finds a key k_i from its state such that, $\mathsf{Test}(k_i, x) = \text{TRUE}$. It then sends the punctured key $k' \leftarrow \mathsf{Punc}(k_i, x)$ to the second server - which responds with $r \leftarrow \oplus_{j \in \mathsf{Set}(k')} DB[j]$, which we refer to as the 'database xor bit' with respect to the key k'. The client can now compute $DB[x] = r \oplus h_i$, which is the expected output of the PIR online phase.

This is a simplified version of the original protocol in [11], and the original construction has a few more features. Firstly, in the online phase, the client also interacts with the first server to replenish the key and hint bit that it used to compute $DB[x]$, and this ensure that the client state always contains T privately puncturable pseudo-random keys before each online PIR query. Secondly, note that the right server in the above simplified protocol always views a key of a set of size $(\sqrt{n} - 1)$ punctured at x - and hence its view is **not** independent of x. The author further use privacy amplification techniques to avoid this kind of leakage.

All follow up PIR with preprocessing works in this CGK paradigm use cryptographic assumptions and they focus on designing a more efficient privately puncturable pseudo-random set - where the keys has short description size, while they allow for efficient set membership testing, set enumeration and puncturing [17,26,31]. In all these works, the client and server online computation is dominated with $\widetilde{\mathcal{O}}(\sqrt{n})$ cryptographic operations (either based on OWF or public-key primitives) and their bandwidth and client storage complexity have a multiplicative factor of cryptographic key length as well.

Information-Theoretic Setting. In this work we focus our attention on the feasibility of designing PIR with client preprocessing schemes with sublinear client state, online computation and communication in the information theoretic setting i.e. with no cryptographic assumption. In particular, in the Corrigan-Gibbs and Kogan paradigm we investigate the feasibility of designing an information theoretic analog of privately puncturable pseudo-random sets. A major challenge here is to represent a pseudo-random set of size $\sqrt{n}$ with a key of size $o(\sqrt{n})$, while still allowing for efficient set membership, set enumeration and private puncturing.

The key observation that helps in the design of this primitive is that the correctness of the PIR scheme only requires that for any $x \in [n]$, $Pr(x \in \mathsf{Set}(k)) = 1/\sqrt{n}$, where $k \leftarrow \mathsf{Gen}()$, and its not required that $\mathsf{Set}(k)$ has the same distribution as a random set of size $\sqrt{n}$ (which is a stricter requirement). We exploit this observation in our design of a privately puncturable random set, where the keys simultaneously have sufficient randomness and structure to ensure sublinear size, while allowing for puncturing that hide the punctured element.

Outside theoretical interest, PIR protocols in the information-theoretic setting would be of attractive from a practical viewpoint as well. The computation complexity in information-theoretic protocols is dominated by simpler algorithmic operations (like bit shift, xor, etc.), which are generally faster than cryptographic operations in both private-key and public-key regime.

1.1 Our Contribution

Information-Theoretic PIR with Client Preprocesing. We initiate the study of the PIR with client preprocessing model in the information theoretic setting based on the CGK paradigm [11]. We propose a $2t$ server PIR with preprocessing protocol with corruption threshold 1[1], where the client maintains a state of size $\widetilde{\mathcal{O}}(n^{1/2+1/2t})$, with online client computation $\widetilde{\mathcal{O}}(\sqrt{n})$, online per server computation $\widetilde{\mathcal{O}}(\sqrt{n})$ and online communication $\widetilde{\mathcal{O}}(\sqrt{n})$.

In particular, setting $t = 2$, we get a 4-server PIR with preprocessing protocols with client storage $\widetilde{\mathcal{O}}(n^{3/4})$, client/server online computation/bandwidth $\widetilde{\mathcal{O}}(n^{1/2})$. Setting $t = \log(n)/2$, we get a $\log(n)$-server PIR with preprocessing protocols with $\widetilde{\mathcal{O}}(n^{1/2})$ client storage, client/server online computation and online bandwidth $\widetilde{\mathcal{O}}(n^{1/2})$ - where these cost match the original 2-server PIR with preprocessing construction of Corrigan-Gibbs and Kogan [11] based on OWFs.

All the client/server computation costs reported here are in number of bit operations, and unlike other PIR protocols it does not have a multiplicative $O(\mathsf{poly}\log(\kappa))$ factor, where κ is the security parameter. The online communication of our scheme has no security parameter multiplicative factor either, which is the case for all previous PIR with preprocessing constructions in the CGK model.

[1] i.e. all but one server are honest.

Improving PIR Communication Complexity. In Sect. 4.1 we slightly modify the above construction to reduce the online bandwidth, and the online server response bandwidth at the cost of doubling the number of servers. We achieve a $4t$ server PIR with preprocessing protocol with threshold 1 with online bandwidth $n^{1/2t+o(1)}$, where the online computation asymptotic complexities of the client/server stay sublinear in the database size.

New Information-Theoretic Primitive. The key building block in our PIR construction is a (t, n)-*privately multi-puncturable random set* (PMPRS), which has five algorithms (Gen, Set, Test, Punc, $\mathsf{DotProdEval}$). Similar to the analogous cryptographic primitive, $\mathsf{Gen}()$ outputs a key k, where $\mathsf{Set}(k)$ outputs a set of size $\sqrt{n}$ with domain $[n]$ and each element from the domain is contained in the set with probability $1/\sqrt{n}$. Function $\mathsf{Test}(k, x)$ checks if $x \stackrel{?}{\in} \mathsf{Set}(k)$. The multi-puncturing function $\mathsf{Punc}(k, x)$ outputs t tuples of the form $(k_i, S_i, \mathsf{ind}_i)$ for $i \in [t]$, where each punctured key k_i corresponds to set S_i, and correctness requires that sets $S_0, \ldots, S_{t-1}$ are pairwise disjoint, and their union equals $\mathsf{Set}(k) \setminus \{x\}$. We call this primitive a 'multi-puncturable' random set, since the partitioned set $S \setminus \{x\}$ is divided into t disjoint sets. Privacy of this scheme requires that each of these punctured keys are simulatable given just the parameters t, n, which implies that they hide the punctured element x. This is a generalization of the traditional privately puncturable set primitive in [11], where Punc function outputs a single punctured key. The function $\mathsf{DotProdEval}(k_i, i, DB)$ exactly captures the server computation - which involves generating the partial punctured set and compute database xor bit wrt the input punctured key. However, instead of outputting a single bit, this algorithm outputs a vector $\vec{v_i}$, such that the idx_i^{th} bit has the expected result i.e. $\vec{v_i}[\mathsf{idx}] = \oplus_{j \in S_i} DB[j]$. This kind of correctness requirement in the CGK model was also first considered in TreePIR [26] - which is a 2-server PIR with preprocessing construction based on DDH assumption.

We propose an information theoretic construction for (t, n)-PMPRS (for when $n^{1/2t}$ is an integer) where the key and each punctured key have sizes $\widetilde{\mathcal{O}}(tn^{1/2t})$. The running time of $\mathsf{Gen}, \mathsf{Test}, \mathsf{Set}, \mathsf{DotProdEval}$ are $\widetilde{\mathcal{O}}(tn^{1/2t})$, $\widetilde{\mathcal{O}}(1), \widetilde{\mathcal{O}}(\sqrt{n})$ and $\widetilde{\mathcal{O}}(\sqrt{n})$ respectively.

1.2 Technical Overview

We divide our technical overview in two parts, first we highlight the key ideas behind our PMPRS construction, and next we show how this primitive can be used to construct a multi-server PIR with client preprocessing.

$((\log_2 n)/2, n)$-PMPRS Construction. To illustrate some of the key ideas in our construction, in this subsection we depict a $((\log_2 n)/2, n)$-PMPRS construction where n is an even power of 2. The general (t, n) construction and its formal proof of security are presented in Sect. 3.

Our scheme generates PMPRS keys that correspond to *well-partitioned sets*, which are sets that contains a single element from each chunk of the domain $[n]$,

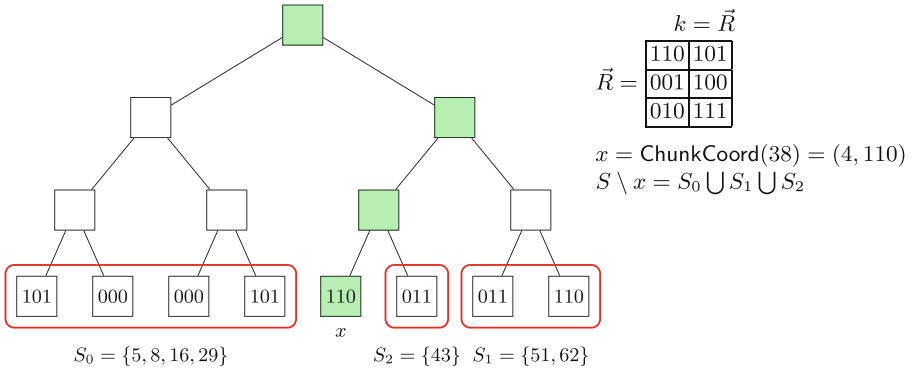


Fig. 1. An example of $(t, n) - \mathsf{PMPRS}$ with $n = 64, t = 3$. The leaf nodes contain the offset values within the specific chunks, which can be computed using the vector $\vec{R}$. The sets S_0, S_1, S_2 represent the multi-puncturing obtained if the PMPRS set is punctured at $x = 38$ that's contained in the set. The leaf nodes corresponding to each set S_i and the path to the punctured element x are highlighted using the red boxes and the green filled boxes respectively

where the i^{th} chunk is defined as $\{i\sqrt{n}, i\sqrt{n}+1, \ldots, i\sqrt{n}+(\sqrt{n}-1)\}$. Hence, any well-partitioned set contains $\sqrt{n}$ elements - one for each chunk. We use the bijective map $(c_x, \delta_x) \leftarrow \mathsf{ChunkCoord}(x)$ to map any element $x \in [n]$ to its corresponding chunk $c_x = (\lfloor x/\sqrt{n} \rfloor) \in [\sqrt{n}]$ and the offset within the chunk $\delta_x =$ bit decomposition of $(x \mod \sqrt{n})$. We sometimes use the integer modulo $\sqrt{n}$ representation of δ_x as well, but it'll always be clear from the context. Sets with this structure were first used in a single server PIR with preprocessing construction PIANO [34], where the privately puncturable pseudo-random key is constructed using a pseudo-random function (PRF). The description of each algorithm in our PMPRS scheme is as follows:

- **Gen**(): outputs a matrix $\vec{R}$ of dimension $t \times 2$, where $t = (\log_2 n)/2$ and each element is sampled randomly from the domain $\{0,1\}^t$. We use t as a shorthand for $\log_2(n)/2$ throughout the description of this construction.
- **Set**$(k = \vec{R})$: outputs a well-partitioned set, where the offset of the element in the i^{th} chunk is given by $\oplus_{j=0}^{t-1} \vec{R}[j][i^j]$, where $(i^0, \ldots, i^{t-1}) \leftarrow \mathsf{bit\text{-}decomp}_2(i)$ is the bit decomposition of i. Since, $i \in [\sqrt{n}]$, the bit decomposition of i has $\log_2(\sqrt{n}) = t$ bits.
- **Test**$(\vec{R}, x)$: first compute $(c_x, \delta_x) \leftarrow \mathsf{ChunkCoord}(x)$, and then check if the offset of the element in $\mathsf{Set}(\vec{R})$ in the c_x^{th} chunk is δ_x as follows: $\oplus_{j=0}^{t-1} \vec{R}[j][c_x^j] \stackrel{?}{=} \delta_x$, where $(c_x^0, \ldots, c_x^{t-1}) \leftarrow \mathsf{bit\text{-}decomp}_2(c_x)$.

We can visualize this well-partitioned set using a full binary tree $T_{2,n}$ with depth $t = \log_2(n)/2$ (and hence it has $\sqrt{n}$ leaves) as shows in Figure 1. We associate random values $(R[i][0], R[i][1])$ with depth i and we associate the i^{th} leaf in the tree with the i^{th} chunk. For any path from root to a leaf, we can

xor one of the random strings at each depth, corresponding to whether the path travels along the left or the right child at that depth. Hence, the value computed at the i^{th} leaf equals exactly the offset of the element in the i^{th} chunk, as was computed in $\mathsf{Set}(\vec{R})$. We will use this tree based interpretation in the description of the following two algorithms of our PMPRS scheme:

- $((S_0, k_0, \mathsf{ind}_0), \ldots, (S_{t-1}, k_{t-1}, \mathsf{ind}_{t-1}) \leftarrow \textbf{Punc}(k, x)$: To puncture the set $\mathsf{Set}(k)$ at element x, we can partition the tree $T_{2,n}$ after removing the path from root to the chunk containing x into t disjoint trees, where the i^{th} tree T_i (for $i \in [t]$) has root at depth $i+1$, and it has 2^{t-i-1} leaves - which corresponds to the i^{th} partitioned set S_i. Hence, we have $\cup_{i=0}^{t-1} S_i = S \setminus \{x\}$. An example of these t sets forming a disjoint union of the punctured set is also highlighted in Fig. 1. However, note that each set S_i cannot be part of the key k_i since it leaks some information about x - particularly it leaks that this set doesn't contain the element of puncturing. To ensure privacy, while satisfying a correctness definition, we define the i^{th} key k_i such that it contains sufficient information to compute offsets of all elements in S_i, but it contains no information about the chunks that correspond to those offsets in S_i. This decoupling of the offsets and the chunks is critical for making the scheme secure. Concretely, the key k_i has three components: a matrix $\vec{R}_i = \vec{R}[i+1:][:]$ i.e. R_i contains all rows $\geq i+1$ of R, it also contains a correction $\mathsf{corr} = \oplus_{j=0}^{i-1} \vec{R}[j][c_x^j]$ where $(c_x, \delta_x) \leftarrow \mathsf{ChunkCoord}(x)$ and $(c_x^0, \ldots, c_x^{t-1}) \leftarrow \mathsf{bit\text{-}decomp}_2(c_x)$, and finally it contains $R[i][1 - c_x^i]$. The first component of the key contains information from R from depth $i+1$ and higher, the second component contains partial information of R from depth 0 to $i-1$, and in particular it contains the xor of bits in R from these lower depths corresponding to the bit decomposition of c_x, and the third and final component contains one of the two random strings of R associated with depth i that is **not** used in computing δ_x (the offset of the punctured element). In the next function description we elaborate on how the indexes idx_i are computed and how the key k_i is used to construct S_i to satify the correctness definition. Its easy to see that key k_i hides the element x since it contains no information about $R[i][c_x^i]$ - masking the value of δ_x defined as $\oplus_{j=0}^{t-1} R[j][c_x^j]$, and it contains no information about the chunk containing the punctured element.
- $\vec{v_i} \leftarrow \textbf{DotProdEval}(k_i, i, DB)$: Given $k_i = (\vec{R}_i, \mathsf{corr}, r)$, we first compute an offset vector $\vec{\delta}$ of length 2^{t-i-1} - such that this vector contains the offsets in the leaf nodes of tree T_i in increasing order of chunk indexes. This vector can be computed using $\vec{R}_i$ in a similar fashion to how the offset vector is computed in Set algorithm, and further each element of this is xored with $\mathsf{corr} \oplus r$. By construction, vector $\vec{\delta}$ contains offsets of all elements in S_i in order. However, note that k_i hides the exact chunk indexes (which depend on the item being punctured) that these offsets correspond to. Here, we make the key observation that there are exactly 2^{i+1} possible trees which could be T_i - one for each tree rooted at depth $(i+1)$ in $T_{2,n}$. Hence, for each of those 2^{i+1} trees in order we compute the following: consider the sequence of chunks (represented by a vector $\vec{c}$) corresponding to its leaf nodes in the tree,

and compute a set $S' = \{\mathsf{ChunkCoord}^{-1}(\vec{c}[j], \vec{\delta}[j]) | j \in [2^{t-j-1}]\}$. Append to the vector v_i (which is initialized as null vector) with the bit $\oplus_{j \in S'} DB[j]$. Exactly one of these 2^{i+1} trees would be T_i, and let it be the idx^{th} tree in the sequence. Then by construction we have $\vec{v}_i[\mathsf{idx}_i] = \oplus_{j \in S_i} DB[j]$ - satisfying the correctness definition of PMPRS. It takes $O(2^{t-i-1})$ time to compute each bit of $\vec{v}_i$ and hence the running time of DotProdEval is $2^{i+1}.O(2^{t-i-1}) = O(2^t) = O(\sqrt{n})$.

This construction gives us an information theoretic PMPRS construction with key size $O(\log^2(n))$ for a $\sqrt{n}$ sized random set - such that it supports efficient set membership, set enumeration and t-puncturing! We extend the above construction in Sect. 3 in a couple ways. Firstly, we ensure that Gen can generate a set containing a specific element Δ (which is needed in the PIR construction), and secondly we give non-trivial PMPRS constructions for smaller t values. In the general construction we define $d = n^{1/2t}$ (which must be an integer) and we consider d-ary full tree $T_{n,d}$ (of depth $t = \log_d n$) over the domain $[n]$ instead of a binary tree. Here to puncture at a leaf node, we can partition the remaining tree into disjoint union of t "punctured trees" - which is defined as a tree with one of the root's children subtrees being removed. A major challenge in the general construction was to ensure that the DotProdEval has $O(\sqrt{n})$ complexity, as the trivial approach of considering all possible punctured subtrees at depth i lead to computation complexity $O(d\sqrt{n})$, which can be $\omega(\sqrt{n})$ for very small t or large d. We discuss this issue and the proposed solution in detail in Sect. 3.

$2t$-Server PIR with Client Preprocessing. Our PIR protocol follows the CGK paradigm. In the **offline phase** the client generates $T = \widetilde{\mathcal{O}}(\sqrt{n})$ (t, n)-PMPRS keys and sends them to server 0. The server responds back with the hint bits for each of these keys, which is computed as follows for a given key k: $\oplus_{j \in \mathsf{Set}(k)} DB[j]$. The client stores the keys and the hint bits as its state.

In the **online phase**, the client inputs an index $x \in [n]$ and it finds a PMPRS key such that $\mathsf{Test}(k, x) = \text{TRUE}$. It computes $((k_0, \mathsf{idx}_0), \ldots, (k_{t-1}, \mathsf{idx}_{t-1})) \leftarrow \mathsf{Punc}(k, x)$[2], and sends k_i to server $(t + i)$. The server responds back with the vector $\vec{v_i} \leftarrow \mathsf{DotProdEval}(k_i, i, DB)$. By the correctness of the PMPRS primitive, we have $\oplus_{i=0}^{t-1} \vec{v_i}[\mathsf{idx}_i] = \oplus_{j \in \mathsf{Set}(S) \setminus \{x\}} DB[j]$. And hence if h is the hint bit corresponding to the key k, then the client can compute $h \oplus (\oplus_{i=0}^{t-1} \vec{v_i}[\mathsf{idx}_i]) = DB[x]$ - which is the desired output.

The above construction ends up using the pair (k, h) - and hence we replenish the state with a new key-hint pair to maintain the same client state. For this, the client samples a new key k' such that $x \in \mathsf{Set}(k')$. It punctures this key k' at x and it sends its t components to servers $0, 1, \ldots, (t-1)$ in the online phase. Each server responds back with vector output of DotProdEval algorithm. Using these vectors and the database bit $DB[x]$ the client can compute the hint bit $h' = \oplus_{j \in \mathsf{Set}(k')} DB[j]$.

[2] Here we ignore the sets S_i output by Punc algorithm since they are not used in the PIR construction.

The privacy of the scheme is ensured by the fact that in each online query each server only views a single punctured PMPRS key - which is simulatable by definition. And the correctness of this scheme follows from the definition of the PMPRS construction as described above. We defer the details of the security proof and the complexity analysis to Sect. 4.

1.3 Related Work

A trivial approach to solve the PIR problem would be for the client to download the entire database from the server, and store just the element of interest. However, this leads to linear bandwidth cost. Hence, a line of work starting with Chor et al. [9] have focused on reducing the bandwidth cost in the single server [8,13,20,25,28] and multi-server setting [3,5,7,14,15,18,33]. However, all these works have linear computation complexity for each server - which is inherent in the standard PIR model as proven by Beimel et al. [6]. To overcome this barrier, broadly speaking two models were introduced - PIR with batch queries and PIR with preprocessing.

In a PIR scheme with batch queries a client takes as input a sequence of k indexes, for which it privately queries the server(s). Here the goal is to amortize the server computation cost across the k queries. A number of works study this model of batch queries [1,2,6,19,22,27,32], and in particular the work due to Ishai, Kushilevitz, Ostrovsky and Sahai [22] achieve the optimal amortized per query server complexity of $\widetilde{\mathcal{O}}(n/k)$.

PIR with pre-processing was first proposed by Beimel, Ishai and Malkin [6]. In their scheme, in the offline phase the two non-colluding servers do a one-time computation to store a new encoding of the database with super-linear size. In the online phase the server can support an unbounded number of client queries, where the client stores no state from the preprocessing phase. They introduce two information theoretic protocols, one where each server stores a state of size $O(n^2)$ with online computation $O(n/\log^2 n)$, and bandwidth $O(n^{1/3})$. Their second scheme achieves online computation and bandwidth $O(n^{0.5+\epsilon})$ for any ϵ, where the server storage is $\omega(n)$ and it exponentially increases with decrease in ϵ. Compared to this information theoretic preprocessing scheme, our construction has no super linear server state after preprocessing and it enjoys a lower client/server computation and bandwidth complexity at the price of a higher number of servers with corruption threshold 1.

PIR with client-side preprocessing was first introduced by Kogan and Corrigan-Gibbs [24] in the 2-server model, which achieve client state, online client/server computation and bandwidth $\widetilde{\mathcal{O}}(\sqrt{n})$. This scheme was later improved in future works, where the focus is either to improve the asymptotic or concrete online bandwidth [17,26,31]. Recently a number of single server PIR with preprocessing protocols were also proposed in the CGK paradigm [10,16,20,30,34] - where a single server can perform both the offline and online phase while satisfying the privacy requirement. All these previous works on PIR with client preprocessing are either in the OWF regime or they used some public key assumption like ϕ-hiding, DDH and LWE.

Concurrent Work. A recent work due to Ishai et al. [23] also study the problem of PIR with client preprocessing in the information-theoretic setting, but in the single and two server case. They provide a set of protocols with various performance tradeoffs between server computation, client space and bandwidth. In particular, all the proposed constructions with sublinear online server time in [23] have client space, server computation and online bandwidth $\Omega(n^{2/3}), \Omega(n^{2/3})$ and $\Omega(n^{1/3})$ respectively. We improve on all these three performance metric in the case of 4 or more servers in the information theoretic and client preprocessing PIR setting.

2 Preliminaries

2.1 Algorithmic Notation

A function $f : \mathbb{N} \to \mathbb{R}$ is called negligible it shrinks faster than any inverse polynomial i.e. for any polynomial $p()$, there exist an $N \in \mathbb{N}$, such that $f(n) < 1/p(n)$ for every $n \geq N$. We use the notation $\mathsf{negl}.(n)$ to represent any arbitrary negligible function in n. We use shorthand notation $S = \dot{\bigcup}_{i=0}^{m-1} S_i$ to represent that the m sets $S_0, \ldots, S_{m-1}$ are pairwise disjoint and their union equals set S.

Notation with an overset arrow (e.g. $\vec{v}, \vec{M}$) is used to represent vectors and matrices, where capitalized letters are used specifically for matrices. Notation $\leftarrow_{\$} R$ signifies sampling a random element from set R. For domain $[n] = \{0, 1, \ldots, n-1\}$, we define the i^{th} chunk as the set $\{\sqrt{n}i, \sqrt{n}i+1, \ldots, \sqrt{n}i+(\sqrt{n}-1)\}$. Hence, we can view the domain $[n]$ as a disjoint union of $\sqrt{n}$ chunks. Define bijection $\mathsf{ChunkCoord}(x) = (c_x, \delta_x) \in [\sqrt{n}] \times \{0,1\}^{\log n/2}$, where $c_x = \lfloor x/\sqrt{n} \rfloor$ is the chunk that contains x and $\delta_x =$ bit decomposition of $(x \mod \sqrt{n})$ is the offset signifying which specific element in the chunk corresponds to x. We refer to c_x as the chunk coordinate of x. Sometimes in the paper we refer to the mod $\sqrt{n}$ representation of δ_x interchangeably with its bit decomposition - but it will always be clear from the context. A set S from domain $[n] = \{0, 1, \ldots, n-1\}$ is called *well partitioned* if it contains exactly one element from each chunk. Particularly, note that the description of a well partitioned set can be given by just a vector of offsets of size $[\sqrt{n}]$, which corresponds to offsets of the elements in each chunk.

Function $\vec{v} \leftarrow \mathsf{trim}(\vec{u}, i)$ takes as input a vector u (let say of size n) and an index $i \in [n]$, then it outputs a trimmed version of the input vector with the i^{th} element removed. Hence, $\vec{v}[j] = \vec{u}[j]$ for $0 \leq j < i$ and $\vec{v}[j] = \vec{u}[j+1]$ for $j \in [i, n-1]$.

For any n, t where $d = n^{1/2t}$ is an integer, we use notation $T_{d,n}$ to represent a full d-ary tree of depth t $(= 1/2 \log_d n)$ - where the i^{th} leaf node correspond to the i^{th} chunk of the domain $[n]$. Note $T_{d,n}$ has exactly $\sqrt{n}$ leaf nodes. We use the notation $\mathsf{chunks}_{d,n}(v)$ to represent increasing sequence of chunk indexes contained in subtree rooted at node v in tree $T_{d,n}$. Additionally, $\mathsf{chunks}_{d,n}(v, u)$ outputs the vector of increasing chunk indexes in sub-tree rooted at v in $T_{d,n}$ excluding the chunks/leaf nodes in the subtree rooted at the u^{th} child of v,

where $u \in [d]$. Hence, vector $\mathsf{chunks}_{d,n}(v,u)$ doesn't contain any chunk indexes contained in the subtree rooted at the u^{th} child node of v.

2.2 Multi-server PIR with Client Preprocessing (with Threshold 1)

We adapt 2-server PIR with client preprocessing syntax and adaptive security definitions from Corrigan-Gibbs and Kogan [11] and Shi et al. [31] to the multi-server and the information theoretic setting here.

An l server protocol contains $(l+1)$ parties: a single Client and l non-colluding servers Server_0, Server_1, $\ldots$, Server_{l-1}. All parties receive as input the statistical security parameter λ and the database size n. The protocol proceeds as follows:

- **Offline phase**: All the servers receive as input a database $DB \in \{0,1\}^n$. The client sends a single message to each server, which responds back with a single message to the client. The client uses these l responses to compute some state that it stores as output of this offline phase.
- **Online phase**: The servers can serve an unbounded queries of the following form: client receives as input an index $x \in [n]$, following which the client sends a single message to each of the servers as a function of its state and index x. Each server responds back to the client with a single message, which allows the client to compute an output bit $y \in \{0,1\}$.

Correctness. For any database $DB \in \{0,1\}^n$ and an arbitrary sequence of queries $(x_1, x_2, \ldots)$, the client outputs $DB[x_i]$ at the end of the i^{th} online query phase with probability at least $1 - \mathsf{negl}.(\lambda)$.

Privacy. The PIR scheme is said to be private with threshold 1, if there exists a probabilistic polynomial time simulator $\mathsf{Sim}(1^\lambda, 1^n)$ such that for an adversary acting as the j^{th} server (for any $j \in [l]$), polynomially bounded (in λ) parameters n and q and $DB \in \{0,1\}^n$, the view of the adversary $\mathcal{A}$ in the following two experiments is statistically indistinguishable:

- **Real**: An honest Client interacts with $\mathcal{A}(1^\lambda, 1^n, DB)$ who acts as Server_j and it may actively deviate from the prescribed PIR protocol. At the start of each online phase $i \in [q]$, $\mathcal{A}$ adaptively picks a query $x_i \in [n]$ which is the input of the Client in the same phase.
- **Ideal**: The simulator Sim acts as a Client and it interacts with $\mathcal{A}(1^\lambda, 1^n, DB)$ who acts as Server_j and which may actively deviate from the prescribed PIR protocol. At the start of each online phase $i \in [q]$, $\mathcal{A}$ adaptively picks a query $x_i \in [n]$ for the client, which is **not** input to Sim.

3 Privately Multi-puncturable Random Set (PMPRS)

In this section we present formal syntax and security definition of our newly introduced PMPRS primitive. Following which we present our PMPRS construction, which is based on random sets with some structure imposed by d-ary trees (for $d = n^{1/2t}$) using a minimal amount of randomness.

Definition 1 (PMPRS syntax). *A (t, n)-PMPRS scheme with input domain $[n] = \{0, \ldots, n-1\}$ consists of five algorithms (Gen, Set, Test, Punc, $\mathsf{DotProdEval}$) with the following syntax:*

- $k \leftarrow \mathsf{Gen}(\Delta, 1^t, 1^n)$*: outputs a short key $k \in \{0,1\}^*$ corresponding to a random set containing element $\Delta \in [n]$. The parameter Δ is an optional input to this algorithm*
- $S \leftarrow \mathsf{Set}(k)$*: takes as input a key k, and it outputs a random well partitioned set from domain $[n]$*
- $b \leftarrow \mathsf{Test}(k, x)$*: takes as input a key k, an element $x \in [n]$, and it outputs a boolean value* TRUE *or* FALSE*- corresponding to whether element x is contained in the set represented by k*
- $((S_0, k_0, \mathsf{ind}_0), (S_1, k_1, \mathsf{ind}_1), \ldots, (S_{t-1}, k_{t-1}, \mathsf{ind}_{t-1})) \leftarrow \mathsf{Punc}(k, x)$*: outputs t punctured keys $k_0, \ldots, k_{t-1}$, with corresponding integer indexes $\mathsf{idx}_0, \ldots, \mathsf{idx}_{t-1}$ and sets $S_0, \ldots, S_{t-1}$, such that the t sets form a disjoint union of punctured set $\mathsf{Set}(k) \setminus \{x\}$.*
- $\vec{v}_i \leftarrow \mathsf{DotProdEval}(i, k_i, DB)$*: is a deterministic function that takes in a punctured key k_i, a vector $DB \in \{0,1\}^n$ and it outputs a vector $\vec{v}_i$, such that its ind_i^{th} bit corresponds to $\oplus_{j \in S_i} DB[j]$ - the database xor bit for one of the partitioned sets*

Definition 2 (PMPRS security). *A (t, n)-PMPRS scheme (Gen, Set, Test, Punc, $\mathsf{DotProdEval}$) for domain $[n]$ is λ-**secure** if is satisfies the following conditions:*

- ***Correctness:*** *For any $\Delta, x \in [n]$ and $DB \in \{0,1\}^n$, let*

$$k \leftarrow \mathsf{Gen}(\Delta, 1^t, 1^n),\ S \leftarrow \mathsf{Set}(k)$$
$$((S_0, k_0, \mathsf{ind}_0), \ldots, (S_{t-1}, k_{t-1}, \mathsf{ind}_{t-1})) \leftarrow \mathsf{Punc}(k, x)$$

  then the following holds:
  - $\Delta \in \mathsf{Set}(k)$
  - $S \setminus \{x\} = \dot{\bigcup} S_i$
  - *for $i \in [t]$, $\vec{v}_i[\mathsf{ind}_i] = \oplus_{j \in S_i} DB[j]$*

  The second and third correctness requirements mentioned above also hold when the optional Δ parameter is not input to Gen algorithm
- ***Privacy:*** *There exists a simulator Sim such that for all $x \in [n], i \in [t]$, the following distributions are statistically indistinguishable in λ:*

$$\boxed{\begin{array}{l} k \leftarrow \mathsf{Gen}(x) \\ ((S_0, k_0, \mathsf{ind}_0), \ldots, (S_{t-1}, k_{t-1}, \mathsf{ind}_{t-1})) \leftarrow \mathsf{Punc}(k, x) \\ \text{return } k_i \end{array}} \approx_\lambda \mathsf{Sim}(1^t, 1^n, i)$$

  Each punctured key k_i can be simulated using just the parameters t, n, i, or in other words, it hides the punctured element x. It should also be noted that the vectors $\vec{v}_i$ can be deterministically computed using the $\mathsf{DotProdEval}$ algorithm with input key k_i (which is simulatable) and vector DB. Hence, $\vec{v}_i$ hides element x as well, even if one of its bits correspond to the correct database xor bit on one of the punctured sets.

– ***Randomness:*** *The set output by* $\mathsf{Set}(\mathsf{Gen}(1^t, 1^n))$ *contains any element* $x \in [n]$ *with probability* $1/\sqrt{n}$*, where the probability is taken over the randomness of* Gen *algorithm. Additionally,* $\mathsf{Set}(\mathsf{Gen}(\Delta, 1^t, 1^n))$ *contains any element* x *not in the same chunk as* Δ *with probability* $1/\sqrt{n}$

Our PMPRS construction satisfies a stronger security guarantee which we define next.

Definition 3. *A* λ-***secure*** *PMPRS scheme with* $\lambda = 0$ *is called perfectly secure.*

Efficiency Requirements. We measure the efficiency of any PMPRS scheme in terms of the size of the keys and the punctured keys - which would contribute to the communication potocol of our PIR scheme. We also measure the computation complexity of the Gen, Test, Set, Punc and DotProdEval algorithms, which would contribute to the computation complexity of the client and the servers in our PIR scheme.

3.1 Proposed PMPRS Construction

We follow the blueprints of the PMPRS construction described in Sect. 1.2, but extend it to random sets generated using a d-ary tree structure instead of a binary tree. The formal description of our generic $(t, n)-$PMPRS construction, where $d = n^{1/2t}$ is an integer is given in Fig. 3. We give a high level description of all the algorithms in this construction next.

The Gen function takes as input an additional Δ parameter, which should be contained in the random set corresponding to the output key k. The PMPRS key output of Gen consists of a matrix $\vec{R}$ of dimension $t \times d$ and an additional element corr. The value of corr is picked such that the well-partitioned set generated by k contains x.

The algorithm Set on input $(\vec{R}, \mathsf{corr})$ outputs a well partitioned set of size $\sqrt{n}$, where the element in chunk c with base d bit decomposition $(c^0, \ldots, c^{t-1})$ is given by $\mathsf{corr} \oplus \left(\oplus_{i=0}^{t-1} \vec{R}[i][c^i]\right)$. Intuitively, this refers to the xor of corr with the random strings in $\vec{R}$ corresponding to the path in tree $T_{d,n}$ from root to leaf c.

Hence, the algorithm Test on input x such that $(c_x, \delta_x) \leftarrow \mathsf{ChunkCoord}(x)$, just checks if $\mathsf{Set}(k)$ has offset δ_x in chunk c_x. Note, that this doesn't require enumerating the entire well partitioned set, and it can be performed in time linear in the depth of the tree $T_{d,n}$.

Function Punc takes as input a PMPRS key k and the index of puncturing x, such that $x \in \mathsf{Set}(k)$. At a high level, this function outputs t punctured keys $k_0, \ldots, k_{t-1}$ and corresponding sets $S_0, \ldots, S_{t-1}$ such that the t sets form a disjoint union of punctured set $S \setminus \{x\}$. Removing the path from root to the leaf/chunk containing x in $T_{d,n}$ partitions the remaining tree intro t "punctured trees", where the i^{th} punctured tree (lets call it T'_i for $i \in [t]$) contains a subtree with root at depth i after *removing* the subtree rooted at exactly one of its

children nodes. This structure is also depicted in Fig. 2. The set S_i contains elements of S with chunk indexes in exactly in the leaf nodes of sub-tree T'_i. Each key k_i is constructed such that it contains exactly the information needed to compute the offset (in order) of all elements corresponding to the leaf nodes in T'_i.

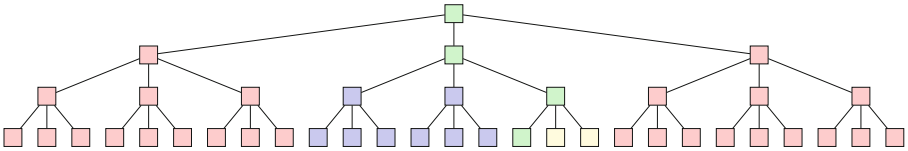


Fig. 2. Example tree $T_{d,n}$ associated with $(t,n)-\mathsf{PMPRS}$ for parameters $n = 27^2, t = 3$, implying $d = 3$. The green path corresponds to the punctured element. Then punctured trees corresponding to sets S_0, S_1, S_2 output of Punc are colored red, blue and yellow respectively except for their roots, which are on the green path. Particularly note each of these "punctured trees" has root at a unique depth, and exactly one of their children subtrees missing

The function $\mathsf{DotProdEval}$ captures the computation performed by each server in our PIR scheme based on PMPRS. On input i, k_i, DB the goal of this algorithm is to compute the database xor bit of set S_i (i.e. $\oplus_{j \in S_i} DB[j]$). We can view this expected output as the dot product between two vectors: the database DB and the indicator vector $\vec{I}_{S_i}$ of set $S_i \subset [n]$[3]. However, our PMPRS scheme allows for a correctness notion - where $\mathsf{DotProdEval}$ outputs a vector $\vec{v}_i$ such that its idx^{th} bit (which was output of Punc) is the correct expected output. At a high level, this algorithm works in two stages:

- Given the punctured key k_i we can first compute an offset vector $\vec{\delta}$ - which contains the offsets of all elements in S_i in order from left to right chunk.
- Secondly, the algorithm computes the chunk vector $\vec{c}$ for every possible "punctured subtree" at depth i in $T_{d,n}$ - where exactly one of them is T'_i. For each of these possible punctured subtrees, we can compute the punctured set (given offsets $\vec{\delta}$ and corresponding chunk indexes $\vec{c}$). We use notation S_{uw} to represent corresponding to a tree rooted at node u with subtree at child node w punctured. The algorithm computes database xor bit for S_{uw} and it appends it to the output vector v_i. If idx_i refers to the index of chunk sequence for T'_i, then by construction $\vec{v}_i[\mathsf{idx}_i] = \oplus_{j \in S_i} DB[j]$ - proving the PMPRS scheme is correct. The privacy follows from the observation that the offset of the punctured element $x = \mathsf{ChunkCoord}^{-1}(c_x, \delta_x)$ is given by $\delta_x = \mathsf{corr} \oplus \left(\left(\oplus_{i=0}^{t-1} \vec{R}[i][c_x^i]\right)\right)$ where $(c_0^x, \ldots, c_{t-1}^x)$ is the base-d bit decomposition of c_x, and the fact that key k_i contains no information about $R[i][c_x^i]$ - which is one of the randomly sampled elements in Gen corresponding to depth i in tree $T_{d,n}$.

[3] The indicator vector $\vec{I}_S$ of a set S from domain [n] is a bit vector of size n such that $\vec{I}_S[i] = 1 \iff i \in S$.

The trickiest part is to prove that $\mathsf{DotProdEval}$ has run time $\widetilde{\mathcal{O}}(\sqrt{n})$ on arbitrary input i, d_i, DB. Note that there are d^{i+1} punctured subtrees at depth i or sets S_{uw} that might correspond to the set S_i, since there are d^i nodes at depth i, where any of its d children subtrees could be punctured. Each of these sets S_{uw} has size $d^{t-i} - d^{t-i-1}$. Hence, trivially computing the database xor bit for each of these sets would lead to complexity $\widetilde{\mathcal{O}}(d^{t-i-1}(d-1)d^{i+1}) = \widetilde{\mathcal{O}}((d-1).d^t) = \mathcal{O}(d\sqrt{n})$, which can be $\omega(\sqrt{n})$ when $d = \omega(1)$. To reduce the computation complexity, we make the key observation that for any node u in $T_{d,n}$ at depth i and two adjacent children nodes w, w' of u, the sets S_{uw} and $S_{uw'}$ only differ in $2.d^{t-i-1}$ elements, and otherwise they overlap. Hence, given the database xor bit for set S_{uw}, we can compute the database xor bit for set $S_{uw'}$ in time $\widetilde{\mathcal{O}}(d^{t-i-1})$ instead of $\widetilde{\mathcal{O}}(d^{t-i})$ time that it takes to compute it trivially. This gives us the needed factor $O(d)$ improvement in the runtime - making the complexity of $\mathsf{DotProdEval}$ $\widetilde{\mathcal{O}}(\sqrt{n})$.

Theorem 1. *Let F be a (t, n)-PMPRS construction shown in Fig. 3. Then F is perfectly secure.*

Proof. By Lemma 1, we know that F satisfies the correctness property defined in Definition 2. By Lemma 2, we know that F satisfies the randomness property defined in Definition 2. By Lemma 3, we can construct the simulator Sim that satisfies the following condition for $\lambda = 0$:

$$\boxed{\begin{array}{l} k \leftarrow \mathsf{Gen}(x) \\ ((S_0, k_0, \mathsf{ind}_0), \ldots, (S_{t-1}, k_{t-1}, \mathsf{ind}_{t-1})) \leftarrow \mathsf{Punc}(Gen(\Delta, 1^t, 1^n), x) \\ \text{return } k_i \end{array}} \approx_\lambda \mathsf{Sim}(1^t, 1^n, i)$$

Our simulator Construction Works as Follows: (on input t, n, i)

- Initialize $(d-1)$-length vector $\vec{r}_i$ where each element is uniformly distributed over $[\sqrt{n}]$.
- Initialize $(t - i - 1) \times d$ random matrix $\vec{R}_i$ where each element is uniformly distributed over $[\sqrt{n}]$.
- Initialize corr sampled from uniform distribution over $[\sqrt{n}]$.
- Return $k_i \leftarrow (\mathsf{corr}_i, \vec{r}_i, \vec{R}_i)$.

Lemma 1 (Correctness). *The (t, n)-PMPRS construction shown in Fig. 3 satisfies the **correctness** definition given in Definition 2.*

Proof. We first consider the case that the parameter Δ is given as input. First we check $\Delta \in \mathsf{Set}(\mathsf{Gen}(\Delta, 1^t, 1^n))$. This is by construction and we could show it passes the membership test $F.\mathrm{Test}(k, \Delta)$. Let $(c_\Delta, \delta_\Delta) \leftarrow \mathsf{ChunkCoord}(\Delta)$ and $(c_\Delta^0, c_\Delta^1, \ldots, c_\Delta^{t-1}) \leftarrow \mathsf{bit\text{-}decomp}_d(c_\Delta)$. We can verify that

$$\mathsf{corr} \oplus \left(\oplus_{j=0}^{t-1} R[j][c_\Delta^j]\right) = \delta_\Delta \oplus \left(\oplus_{j=0}^{t-1} R[j][c_\Delta^j]\right) \oplus \left(\oplus_{j=0}^{t-1} R[j][c_\Delta^j]\right) = \delta_\Delta.$$

Then we check $S \setminus \{x\} = \dot{\bigcup} S_i$. We first show that for distinct $i, j \in [t]$, S_i and S_j are disjoint. Let C_i and C_j be the corresponding set of chunk coordinates of

Let $d = n^{1/2t}$

$\underline{\mathsf{Gen}\ (1^t, 1^n, \Delta)}$:

$(c_\Delta, \delta_\Delta) \leftarrow \mathsf{ChunkCoord}(\Delta)$

$(c^0_\Delta, c^1_\Delta, \ldots, c^{t-1}_\Delta) \leftarrow \mathsf{bit\text{-}decomp}_d(c_\Delta)$

Pick $t \times d$ random matrix $\vec{R}$ where each element is sampled $\leftarrow_\$ \{0,1\}^m$

If Δ parameter is input, set $\mathsf{corr} \leftarrow \delta_\Delta \oplus (\oplus_{i=0}^{t-1} \vec{R}[i][c^i_\Delta])$, else set $\mathsf{corr} \leftarrow_\$ \{0,1\}^m$

return $(\vec{R}, \mathsf{corr})$

$\underline{\mathsf{Set}(k = (\vec{R}, \mathsf{corr})) :}$

Initialize $S \leftarrow \{\}$

For each $c \in [\sqrt{n}]$:

$(c^0, c^1, \ldots, c^{t-1}) \leftarrow \mathsf{bit\text{-}decomp}_d(c)$

update $S \leftarrow S \cup \left\{ \mathsf{ChunkCoord}^{-1}\left(c, \mathsf{corr} \oplus \left(\oplus_{i=0}^{t-1} \vec{R}[i][c^i]\right)\right)\right\}$

return S

$\underline{\mathsf{Test}(k = (\vec{R}, \mathsf{corr}), x) :}$

$(c_x, \delta_x) \leftarrow \mathsf{ChunkCoord}(x)$

$(c^0_x, c^1_x, \ldots, c^{t-1}_x) \leftarrow \mathsf{bit\text{-}decomp}_d(c_x)$

return $\mathsf{corr} \oplus \left(\oplus_{i=0}^{t-1} \vec{R}[i][c^i_x]\right) \stackrel{?}{=} \delta_x$

$\underline{\mathsf{Punc}(k = (\vec{R}, \mathsf{corr}), x)}$:

$(c_x, \delta_x) \leftarrow \mathsf{ChunkCoord}(x)$

$(c^0_x, c^1_x, \ldots, c^{t-1}_x) \leftarrow \mathsf{bit\text{-}decomp}_d(c_x)$

For $i \in [t]$:

Set $\vec{R}_i \leftarrow \vec{R}[i+1:][:]$

Set $\vec{r}_i \leftarrow \mathsf{trim}(\vec{R}[i], c^i_x)$

Set $\mathsf{corr}_i \leftarrow (\oplus_{j=0}^{i-1} \vec{R}[j][c^j_x]) \oplus \mathsf{corr}$

$k_i \leftarrow (\mathsf{corr}_i, \vec{r}_i, \vec{R}_i)$

Set $z \leftarrow \mathsf{bit\text{-}decomp}_d^{-1}(c^0_x, \ldots, c^{i-1}_x)$

$\mathsf{ind}_i \leftarrow zd + c^i_x$

$u \leftarrow$ node in $T_{d,n}$ at depth i on path from root to c_x-th leaf node

$S_i \leftarrow$ subset of $\mathsf{Set}(k) \setminus \{x\}$ with chunk coordinates in $\mathsf{chunks}_{d,n}(u, c^i_x)$

return $((S_0, k_0, \mathsf{ind}_0), \ldots, (S_{t-1}, k_{t-1}, \mathsf{ind}_{t-1}))$

$\underline{\mathsf{DotProdEval}(i, k_i, DB)}$:

Parse k_i as $(\mathsf{corr}_i, \vec{r}_i, \vec{R}_i)$

Initialize vectors $\vec{v}_i, \vec{\delta}$ of size d^{i+1} and $(d-1) \cdot d^{t-i-1}$ respectively

For $j \in [d-1], j' \in [d^{t-i-1}]$:

$(c^0, c^1, \ldots, c^{t-i-1}) \leftarrow \mathsf{bit\text{-}decomp}_d(j')$

$\vec{\delta}[j \cdot d^{t-i-1} + j'] \leftarrow \mathsf{corr} \oplus \vec{r}_i[j] \oplus \left(\oplus_{i'=0}^{i'=t-i-1} \vec{R}_i[i'][c^{i'}]\right)$

Initialize an iterator $j' \leftarrow 0$

For each depth i node u in $T_{d,n}$, its child node $w \in [d]$ (in left-to-right order):

$\vec{c} \leftarrow \mathsf{chunks}_{d,n}(u, w)$

Initialize set $S_{uw} \leftarrow \{\mathsf{ChunkCoord}^{-1}(\vec{c}[j], \vec{\delta}[j] |\ j \in [|\vec{\delta}|])\}$

Set $\vec{v}_i[j'] \leftarrow \oplus_{j \in S_{uw}} DB[j]$

Update $j' \leftarrow j' + 1$

Return $\vec{v}_i$

Fig. 3. Proposed (t, n)-PMPRS construction (where $n^{1/2t}$ is an integer)

S_i and S_j. We show S_i and S_j are disjoint by showing C_i and C_j are disjoint, this is simply because if two elements are in different chunks, they cannot be the same. WLOG, we consider the case $i < j$. Let $(c_x, \delta_x) \leftarrow \mathsf{ChunkCoord}(x)$ and let p be the internal node in $T_{d,n}$ at depth i on path from root to c_x-th leaf node. By definition $C_i = \mathsf{chunks}_{d,n}(p, c_x^i)$, which is the set of chunks excluding the chunks/leaf nodes in the sub-tree rooted at c_x^i-th child of p and C_j is the subset of chunks in this excluded sub-tree, so C_i and C_j are disjoint. Then we show $S \setminus \{x\} = \bigcup S_i$. This is because, by construction, $\bigcup C_i = [\sqrt{n}] \setminus c_x$ includes all the leaf nodes of the tree $T_{d,n}$ except the c_x-th leaf, which represents the set $S \setminus \{x\}$.

We next show that for all $i \in [t]$, $v_i[\mathsf{ind}_i] = \oplus_{j \in S_i} DB[j]$. WLOG, we fixed an arbitrary $i \in [t]$. By ind_i's definition, we find the set S_{uw} where w is the c_x^i-th children node of u and u be the internal node in $T_{d,n}$ at depth i on the path from root to c_x-th leaf node. By checking the definition of S_{uw}, we can see it exactly equals to S_i. So we have

$$\vec{v}_i[\mathsf{ind}_i] = \oplus_{j \in S_{uw}} DB[j] = \oplus_{j \in S_i} DB[j].$$

Lastly, We consider the case that the optional parameter Δ is not given as input. We no longer have the requirement that $\Delta \in \mathsf{Set}(\mathsf{Gen}(\Delta, 1^t, 1^n))$. For the statement $S \setminus \{x\} = \dot{\bigcup} S_i$ and $v_i[\mathsf{ind}_i] = \oplus_{j \in S_i} DB[j]$, since our above proof doesn't rely on Δ, it still holds when Δ is not input.

Lemma 2 (Randomness). *Let F be a (t, n)-PMPRS construction shown in Fig. 3. Then, for any $x \in [n]$,*

$$\Pr\left[x \in F.\mathsf{Set}(\mathsf{Gen}(1^t, 1^n))\right] = \frac{1}{\sqrt{n}}.$$

Additionally, for any $\Delta \in [n]$, and any $x \in [n]$ not in the same chunk as Δ,

$$\Pr\left[x \in F.\mathsf{Set}(\mathsf{Gen}(\Delta, 1^t, 1^n))\right] = \frac{1}{\sqrt{n}}.$$

Proof. Let $(c_x, \delta_x) \leftarrow \mathsf{ChunkCoord}(x)$ and $(c_x^0, c_x^1, \ldots, c_x^{t-1}) \leftarrow \mathsf{bit\text{-}decomp}_d(c_x)$. Let X be the c_x-th element being added into the set $F.\mathsf{Set}(\mathsf{Gen}(1^t, 1^n))$. Then, we have

$$\begin{aligned}
\Pr\left[x \in F.\mathsf{Set}(\mathsf{Gen}(1^t, 1^n))\right] &= \Pr\left[X = \delta_x\right] \\
&= \Pr\left[\mathsf{corr} \oplus \left(\oplus_{j=0}^{t-1} \vec{R}[j][c_x^j]\right) = \delta_x\right] \\
&= \frac{1}{\sqrt{n}}.
\end{aligned}$$

The second last step is by definition of Fig. 3. The last step is because $\vec{R}[j][c_x^j]$ and corr are mutually independent and uniformly distributed over $[\sqrt{n}]$, so does the sum of them $\mathsf{corr} \oplus \left(\oplus_{j=0}^{t-1} \vec{R}[j][c_x^j]\right)$.

Similarly, for any $\Delta \in [n]$, and any $x \in [n]$ not in the same chunk as Δ, let $(c_x, \delta_x) \leftarrow \mathsf{ChunkCoord}(x)$ and $(c_x^0, c_x^1, \ldots, c_x^{t-1}) \leftarrow \mathsf{bit\text{-}decomp}_d(c_x)$. Let X be the c_x-th element being added into the set $F.\mathsf{Set}(\mathsf{Gen}(1^t, 1^n)$. let $(c_\Delta, \delta_\Delta) \leftarrow \mathsf{ChunkCoord}(\Delta)$ and $(c_\Delta^0, c_\Delta^1, \ldots, c_\Delta^{t-1}) \leftarrow \mathsf{bit\text{-}decomp}_d(c_\Delta)$. We have

$$\begin{aligned}
\Pr\left[x \in F.\mathsf{Set}(\mathsf{Gen}(\Delta, 1^t, 1^n))\right] &= \Pr\left[X = \delta_x\right] \\
&= \Pr\left[\mathsf{corr} \oplus \left(\oplus_{j=0}^{t-1} \vec{R}[j][c_x^j]\right) = \delta_x\right] \\
&= \Pr\left[\delta_\Delta \oplus \left(\oplus_{j=0}^{t-1} \vec{R}[j][c_\Delta^j]\right) \oplus \left(\oplus_{j=0}^{t-1} \vec{R}[j][c_x^j]\right) = \delta_x\right] \\
&= \Pr\left[\left(\oplus_{j=0}^{t-1} \vec{R}[j][c_\Delta^j]\right) \oplus \left(\oplus_{j=0}^{t-1} \vec{R}[j][c_x^j]\right) = \delta_x \oplus \delta_\Delta\right] \\
&= \frac{1}{\sqrt{n}}.
\end{aligned}$$

The last step is because every $\vec{R}[j][c_\Delta^j]$ and $\vec{R}[j][c_x^j]$ are mutually independent and uniformly distributed over $[\sqrt{n}]$ since $c_x \neq c_\Delta$ by definition. Therefore, the sum of them $\left(\oplus_{j=0}^{t-1} \vec{R}[j][c_\Delta^j]\right) \oplus \left(\oplus_{j=0}^{t-1} \vec{R}[j][c_x^j]\right)$ is uniformly distributed over $[\sqrt{n}]$.

Lemma 3 (Privacy). *Let F be a (t, n)-PMPRS construction shown in Fig. 3, $x \in [n]$, and $((S_0, k_0, \mathsf{ind}_0), \ldots, (S_{t-1}, k_{t-1}, \mathsf{ind}_{t-1})) \leftarrow F.\mathsf{Punc}(F.\mathsf{Gen}(x), x)$. For any $x \in [n], i \in [t]$, $k_i = (\mathsf{corr}_i, \vec{r}_i, \vec{R}_i)$ follows a joint distribution with size $(t-i) \times d$ where each component is independently and uniformly distributed over $[\sqrt{n}]$.*

Proof. We first show that every element in $\mathsf{corr}_i, \vec{r}_i, \vec{R}_i$ follows a uniform distribution over $[m]$, and then we will show that elements in $\mathsf{corr}_i, \vec{r}_i, \vec{R}_i$ are mutually independent.

Recall that $\vec{R}$ is a $t \times d$ random matrix where each element is uniformly distributed over $[\sqrt{n}]$. Let $(c_x, \delta_x) \leftarrow \mathsf{ChunkCoord}(x)$ and $(c_x^0, c_x^1, \ldots, c_x^{t-1}) \leftarrow \mathsf{bit\text{-}decomp}_d(c_x)$. By corr's definition, $\mathsf{corr} = \delta_x \oplus (\oplus_{j=0}^{t-1} \vec{R}[j][c_x^j])$ is uniformly distributed over $[\sqrt{n}]$. So $\mathsf{corr}_i \leftarrow (\oplus_{j=0}^{i-1} \vec{R}[j][c_x^j]) \oplus \mathsf{corr}$ is uniformly distributed over $[\sqrt{n}]$. Since all elements in $\vec{r}_i$ are defined as xor of elements in $\vec{R}$, so they are uniformly distributed over $[\sqrt{n}]$. Lastly, by $\vec{R}_i$'s definition, every element in $\vec{R}_i$ is a copy of an element in $\vec{R}$, so all elements in $\vec{R}_i$ are uniformly distributed over $[\sqrt{n}]$.

We now show elements in $\mathsf{corr}_i, \vec{r}_i, \vec{R}_i$ are mutual independent.

- We initialize an empty set S and add all elements in $\vec{R}_i$ and $\vec{r}_i$ into S. We know that elements in S are mutual independent since, by definition, every element in $\vec{R}_i$ or $\vec{r}_i$ is a copy of an distinct element in $\vec{R}$ and so is independently sampled.
- By corr_i's definition, we know corr_i is independent of S, since $(\oplus_{j=0}^{i-1} \vec{R}[j][c_x^j])$ is independent of S.
- Update $S \leftarrow S \cup \{\mathsf{corr}_i\}$, we know all elements in S are mutual independent.

Since S equals the union of elements in $\mathsf{corr}_i, \vec{r}_i, \vec{R}_i$, we conclude that elements in $\mathsf{corr}_i, \vec{r}_i, \vec{R}_i$ are mutual independent.

Theorem 2. *Let F be a (t, n)-PMPRS construction shown in Fig. 3. Then*

- *The time complexity of $F.\mathsf{Test}(k, x)$ for any valid k, x is $O(\mathsf{poly}.\log(n))$.*
- *The time complexity of $F.\mathsf{Set}(k)$ for any valid k is $\widetilde{\mathcal{O}}(\sqrt{n})$.*
- *The time complexity of $F.\mathsf{Punc}(k, x)$ for any valid k, x is $\widetilde{\mathcal{O}}(\sqrt{n} + t^2 n^{1/2t})$. Additionally, for $t \in [2, \frac{1}{2}\log(n)]$, $F.\mathsf{Punc}(k, x)$ runs in $\widetilde{\mathcal{O}}(\sqrt{n})$.*
- *The time complexity of $F.\mathsf{DotProdEval}(i, k_i, DB)$ for any valid i, k_i, DB is $\widetilde{\mathcal{O}}(\sqrt{n})$.*
- *The key k and punctured key k_i have size $\widetilde{\mathcal{O}}(tn^{1/2t})$ for every $i \in [t]$.*

Proof. We first note that for any input from $[n]$, all the $\mathsf{ChunkCoord}(\cdot)$, $\mathsf{ChunkCoord}^{-1}(\cdot, \cdot)$, $\mathsf{bit\text{-}decomp}(\cdot)$ and $\oplus$ operations can be done in $O(\mathsf{poly}.\log(n))$.

By $F.\mathsf{Test}(k, x)$'s definition, we can verify that it runs in $O(\mathsf{poly}.\log(n))$ for any valid k, x.

By $F.\mathsf{Set}(k)$'s definition, it runs a *for* loop $O(\sqrt{n})$ times and each loop can be done in $O(\mathsf{poly}.\log(n))$. Therefore, $F.\mathsf{Set}(k)$ runs in $O(\mathsf{poly}.\log(n) \cdot \sqrt{n}) = \widetilde{\mathcal{O}}(\sqrt{n})$ for any valid k.

By $F.\mathsf{Punc}(k, x)$'s definition, it computes $(S_i, k_i, \mathsf{ind}_i)$ for each server $i \in [t]$. k_i can be computed in $O(id \cdot \mathsf{poly}.\log(n))$, ind_i can be computed in $O(\mathsf{poly}.\log(n))$ and S_i can be computed in $O(d^{t-i}\mathsf{poly}.\log(n))$. Summing up together, $F.\mathsf{Punc}(k, x)$ runs in $O(\mathsf{poly}.\log(n) \cdot (\sqrt{n} + t^2 n^{1/2t}))$. Additionally, if we choose $t \in [2, \frac{1}{2}\log(n)]$, $F.\mathsf{Punc}(k, x)$ runs in $O(\mathsf{poly}.\log(n) \cdot \sqrt{n}) = \widetilde{\mathcal{O}}(\sqrt{n})$.

We now compute the time complexity of $F.\mathsf{DotProdEval}(i, k_i, DB)$. To construct the offset vector $\vec{\delta}$, $\mathsf{DotProdEval}$ runs a *for* loop $O(d^{t-i})$ times and each loop can be done in $O(\mathsf{poly}.\log(n))$.

To compute a single element in $\vec{v}_i$, since the corresponding set S_{uw} is with size $O(d^{t-i})$, so the total computation time is $O(d^{t-i}\mathsf{poly}.\log(n))$. However, we note that, for adjacent element in $\vec{v}_i$, their corresponding set S_{uw} and $S_{uw'}$ are only different in $2 \cdot d^{t-i-1}$ elements. This is because w and w' are sibling nodes and the chunks/leaf nodes in the sub-tree rooted at w and w' are the only elements that differentiate S_{uw} and $S_{uw'}$. This observation says that we only need to do a full computation on $\vec{v}_i[0]$, and all the rest elements of $\vec{v}_i$ can be computed in $O(d^{t-i-1}\mathsf{poly}.\log(n))$ time. Since $\vec{v}_i$ has d^{i+1} nodes, the total computation of $\vec{v}_i$ is $O(d^t\mathsf{poly}.\log(n))$.

Summing up the complexity of computing both vectors $\vec{\delta}$ and $\vec{v}_i$, we conclude that $F.\mathsf{DotProdEval}(i, k_i, DB)$ runs in $O(\mathsf{poly}.\log(n) \cdot d^t) = O(\mathsf{poly}.\log(n) \cdot \sqrt{n}) = \widetilde{\mathcal{O}}(\sqrt{n})$ for any valid i, k_i, DB.

Lastly, to see the size of $k = (\vec{R}, \mathsf{corr})$, since $\vec{R}$ is a $t \times d$ matrix where every entry has size $\frac{1}{2}\log_2 n$ and corr is a $\frac{1}{2}\log_2 n$-bits string. Therefore, k has size $O(td \cdot \mathsf{poly}.\log(n)) = \widetilde{\mathcal{O}}(tn^{1/2t})$. Similarly, by definition, punctured key $k_i = (\mathsf{corr}_i, \vec{r}_i, \vec{R}_i)$ has size $O(td \cdot \mathsf{poly}.\log(n)) = \widetilde{\mathcal{O}}(tn^{1/2t})$.

4 Proposed Scheme for PIR with Client Preprocessing

As described in the technical overview, our PIR construction is in the CGK paradigm, where instead of 2 servers, our construction assumes $2t$ servers for $t \geq 2$. Next we reiterate a high level outline for the PIR protocol, which we present formally in Fig. 4.

Let n represent the size of the database DB, and $2t$ is the number of servers. In the *offline phase* the client generates PMPRS keys: $k_i \leftarrow \mathsf{Gen}()$ for $i \in [T] = \{1, 2, , \ldots, T\}$, where $T = \lambda\sqrt{n}$ and λ is the statistical security parameter. The client sends these PMPRS keys to Server 0. Let $\vec{k}$ be a vector of keys such that $\vec{k}[i] = k_i$.

Server 0 interprets each of these PMPRS keys as a partitioned pseudo-random set (using Set function) each of size $\sqrt{n}$ - allowing it to compute the hint bit $h_i = \oplus_{j \in \mathsf{Set}(k_i)} DB[j]$. The Server 0 sends back the vector $\vec{h}$ with $\vec{h}[i] = h_i$ to the Client. The Client stores state $(\vec{k}, \vec{h})$ as output of the offline phase.

In the *online phase*, the client on input PIR index $x \in [n]$ first searches for a key k_i in $\vec{k}$ such that $\mathsf{Test}(k_i, x) = \text{TRUE}$. With probability $(1 - \mathsf{negl}.(\lambda))$ such a key would exist. This follows from the randomness property of the PMPRS primitive:

$$Pr(x \notin \mathsf{Set}(\mathsf{Gen}(1^t, 1^n))) = \left(1 - 1/\sqrt{n}\right)$$

$$\implies Pr\left(x \notin \mathsf{Set}(k_0) \wedge x \notin \mathsf{Set}(k_1) \ldots \wedge x \notin \mathsf{Set}(k_{T-1})\right) = (1 - /\sqrt{n})^{\lambda\sqrt{n}} \leq e^{-\lambda}$$

Next, the Client computes the punctured keys as $((S_0, k_0, \mathsf{ind}_0), (S_1, k_1, \mathsf{ind}_1), \ldots, (S_{t-1}, k_{t-1}, \mathsf{ind}_{t-1})) \leftarrow \mathsf{Punc}(k, x)$ and sends to Server $(t + i)$ the punctured key k_i. Each of these servers respond back with vectors $\vec{v}_i$ as output by $\mathsf{DotProdEval}(k_i, i, DB)$. The client can now compute the PIR output $DB[x]$ as $h_i \oplus \left(\oplus_{j=0}^{t-1} \vec{v}_i[\mathsf{idx}_i]\right)$.

The client ends up consuming a PMPRS key and corresponding hint bit (k_i, h_i). To replenish the same, the Client generates a new PMPRS key k' containing x and it computes its hint bit by sharing punctured keys of $\mathsf{Punc}(k', x)$ with Servers $0, 1, \ldots t - 1$.

Theorem 3. *Suppose that F is ϵ_λ-secure (t, n)-PMPRS, then the $2t$-server PIR scheme (shown in Fig. 4) that supports $\mathsf{poly}(\lambda)$ queries is ϵ_λ-private.*

Proof. The proof is a direct combination of Lemma 4 and Lemma 5.

Lemma 4. *Suppose that F is ϵ_λ-secure (t, n)-PMPRS, then for any polynomial function $p(\cdot)$, and any adversary $\mathcal{A}$ that acts on behalf of Server $i \in \{t, \cdots, 2t-1\}$ and adaptively makes $p(\lambda)$ queries, there exists a PPT simulator $\mathsf{Sim}(1^\lambda, 1^n)$, where the polynomial is in terms of n, λ, such that*

$$\text{view}_{Real} \approx_{\epsilon_\lambda} \text{view}_{\mathsf{Sim}},$$

where $\text{view}_{Real}, \text{view}_{\mathsf{Sim}}$ are the distributions of $\mathcal{A}$'s views interacting with a real client in Fig. 4 and Sim, respectively.

Given a (t, n)-PMPRS F
Let $T = \lambda\sqrt{n}$.

Offline Phase:
- Client computes $k_i \leftarrow F.\mathsf{Gen}(1^t, 1^n)$ for every $i \in [T]$
- Client initializes vector $\vec{k}$ such that $\vec{k}[i] = k_i$
- Client sends $\vec{k}$ to Server 0
- Server 0 computes $h_i = \oplus_{j \in F.\mathsf{Set}(k_i)} DB[j]$ for every $i \in [T]$
- Server 0 initializes vector $\vec{h}$ such that $\vec{h}[i] = h_i$ and send it back to the Client
- Client stores $\vec{h}$ and $\vec{k}$ as its local state

Online Phase (Client inputs $x \in [n]$):
- Client computes the following:
  - Find a $i \in [T]$ such that $F.\mathsf{Test}(\vec{k}[i], x) = \text{TRUE}$ and set $k \leftarrow \vec{k}[i]$ and $h \leftarrow \vec{h}[i]$
  - $((S_0, k_0, \mathsf{ind}_0), (S_1, k_1, \mathsf{ind}_1), \ldots, (S_{t-1}, k_{t-1}, \mathsf{ind}_{t-1})) \leftarrow F.\mathsf{Punc}(k, x)$
- Client sends k_i to Server $t + i$ for each $i \in [t]$
- Each server $t + i$ (for $i \in [t]$) computes the following:
  - $\vec{v}_i \leftarrow F.\mathsf{DotProdEval}(i, k_i, DB)$
  - Send $\vec{v}_i$ back to Client
- Client on receiving $\vec{v}_i$ from each Server $t + i$ (for $i \in [t]$) computes:
  - $DB[x] \leftarrow \vec{v}_0[\mathsf{ind}_0] \oplus \cdots \oplus \vec{v}_{t-1}[\mathsf{ind}_{t-1}] \oplus h$
  - $k' \leftarrow F.\mathsf{Gen}(1^t, 1^n, x)$
  - $((S'_0, k'_0, \mathsf{ind}'_0), (S'_1, k'_1, \mathsf{ind}'_1), \ldots, (S'_{t-1}, k'_{t-1}, \mathsf{ind}'_{t-1})) \leftarrow F.\mathsf{Punc}(k', x)$
- Client sends k'_i to Server i for each $i \in [t]$
- Each server i (for $i \in [t]$) computes the following:
  - $\vec{v}'_i \leftarrow F.\mathsf{DotProdEval}(i, k'_i, DB)$
  - Send $\vec{v}'_i$ back to Client
- Client on receiving $\vec{v}'_i$ from each Server i (for $i \in [t]$) computes:
  - $h' \leftarrow \vec{v}'_0[\mathsf{ind}'_0] \oplus \cdots \oplus \vec{v}'_{t-1}[\mathsf{ind}'_{t-1}] \oplus DB[x]$
  - Update the used (k, h) pair from vectors $\vec{k}, \vec{h}$ with (k', h')

Fig. 4. Proposed $2t$ server PIR with pre-processing protocol for database of size n given a (t, n)-PMPRS

Proof. We first construct the following simulator Sim. Note that $\text{view}_{\mathsf{Sim}}$ follows the distribution $(k_i^1, \cdots, k_i^{p(\lambda)})$.

Simulator Construction
Upon receiving the q-th query index $idx \in [n]$, if $q > p(\lambda)$ then aborts; otherwise computes the following:

- Ignores idx and samples a new index $y \leftarrow_\$ [n]$.
- $k^q \leftarrow F.\mathsf{Gen}(1^t, 1^n, y)$.
- Computes
  $((S_0, k_0^q, \mathsf{ind}_0), (S_1, k_1^q, \mathsf{ind}_1), \ldots, (S_{t-1}, k_{t-1}^q, \mathsf{ind}_{t-1})) \leftarrow F.Punc(k^q, y)$.
- Sends k_i^q to $\mathcal{A}$.

Indistinguishability of view_{Real} **and** view_{SIM}**.**

To prove $\text{view}_{Real} \approx_{\epsilon_\lambda} \text{view}_{\mathsf{Sim}}$, we follow a standard hybrid argument. We first construct Experiment Hyb1 described below. From the privacy property of of the underlying PMPRS scheme F, we have $\text{view}_{SIM} \approx_{\epsilon_\lambda} \text{view}_{Hyb1}$. We highlight the difference between Sim and Experiment Hyb1 with a shaded background.

Experiment Hyb1. Upon receiving the q-th query index $idx \in [n]$, if $q > p(\lambda)$ then aborts; otherwise proceeds the following:

- $k_i^q \leftarrow \mathsf{Sim}_F(1^t, 1^n, i)$, where Sim_F is a simulator for F defined in Definition 2.
- Sends k_i^q to $\mathcal{A}$.

We again follow the Privacy property of the underlying PMPRS scheme F, and have $\text{view}_{Hyb1} \approx_{\epsilon_\lambda} \text{view}_{Hyb2}$.

Experiment Hyb2. Upon receiving the q-th query index $idx \in [n]$, if $q > p(\lambda)$ then aborts; otherwise proceeds the following:

- Computes $k^q \leftarrow F.\mathsf{Gen}(1^t, 1^n, idx)$.
- Computes $((S_0, k_0^q, \mathsf{ind}_0), (S_1, k_1^q, \mathsf{ind}_1), \ldots, (S_{t-1}, k_{t-1}^q, \mathsf{ind}_{t-1})) \leftarrow F.Punc(k^q, idx)$.
- Sends k_i^q to $\mathcal{A}$.

We highlight the difference between the Realworld Construction and the Experiment Hyb2 with a shaded background. Note that view_{Real} follows the distribution $(k_i^1, \cdots, k_i^{p(\lambda)})$ in the Realworld Construction below. The difference between the Realworld Construction and the Experiment Hyb2 is that the puncturable random sets in *Real* is generated offline and there is a negligible probability of it not being able to find a random set containing idx (by the guarantee of choosing parameter T). Since the adversary didn't participate in the offline phase, it has no chance to see the puncturable random set generated in offline phase, so $\text{view}_{Hyb2} \approx_{\epsilon_\lambda} \text{view}_{Real}$.

Realworld Construction. Upon receiving the q-th query index $idx \in [n]$, if $q > p(\lambda)$ then aborts; otherwise proceeds the following:

- Find a $k \in \vec{k}$ such that $F.\mathsf{Test}(k, x) = \text{TRUE}$
- Set $k^q \leftarrow k$
- Computes $((S_0, k_0^q, \mathsf{ind}_0), (S_1, k_1^q, \mathsf{ind}_1), \ldots, (S_{t-1}, k_{t-1}^q, \mathsf{ind}_{t-1})) \leftarrow F.Punc(k^q, idx)$.
- Sends k_i^q to $\mathcal{A}$.

By the standard hybrid argument, we conclude that $\text{view}_{Real} \approx_{\epsilon_\lambda} \text{view}_{\mathsf{Sim}}$.

Lemma 5. *Suppose that F is ϵ_λ-secure (t, n)-PMPRS, for any polynomial function $p(\cdot)$, and any adversary $\mathcal{A}$ that acts on behalf of Server $i \in [t]$ and adaptively makes $p(\lambda)$ queries, there exists a PPT simulator $\mathsf{Sim}(1^\lambda, 1^n)$, where the polynomial is in terms of n, λ, such that*

$$\text{view}_{Real} \approx_{\epsilon_\lambda} \text{view}_{\mathsf{Sim}},$$

where $\text{view}_{Real}, \text{view}_{\mathsf{Sim}}$ are the distributions of $\mathcal{A}$'s views interacting with a real client in Fig. 4 and Sim, respectively.

Proof. We first construct the following simulator Sim for any adversary $\mathcal{A}$ that acts on behalf of Server $i \in \{1, \cdots, t-1\}$, and prove $\text{view}_{Real} \approx_{\epsilon_\lambda} \text{view}_{\mathsf{Sim}}$. Then, we will show how to extend the simulator and the proof to $\mathcal{A}$ that acts on behalf of Server 0, which also participates the offline phase. Note that $\text{view}_{\mathsf{Sim}}$ follows the distribution $(k_i^1, \cdots, k_i^{p(\lambda)})$.

Simulator Construction
Upon receiving the q-th query index $idx \in [n]$, if $q > p(\lambda)$ then aborts; otherwise proceeds the following:

- Ignores idx and samples a new index $y \leftarrow_\$ [n]$.
- Computes $k^q \leftarrow F.\mathsf{Gen}(1^t, 1^n, y)$.
- Computes $((S_0, k_0^q, \mathsf{ind}_0), (S_1, k_1^q, \mathsf{ind}_1), \ldots, (S_{t-1}, k_{t-1}^q, \mathsf{ind}_{t-1})) \leftarrow F.Punc(k^q, y)$.
- Sends k_i^q to $\mathcal{A}$.

Indistinguishability of view_{Real} and view_{SIM}. To prove $\text{view}_{Real} \approx_{\epsilon_\lambda} \text{view}_{\mathsf{Sim}}$, we follow a standard hybrid argument. We construct Experiment Hyb1 in the below. Directly following the Privacy property of the underlying PMPRS scheme F, we have $\text{view}_{SIM} \approx_{\epsilon_\lambda} \text{view}_{Hyb1}$. We highlight the difference between Sim and Experiment Hyb1 with a shaded background.

Experiment Hyb1. Upon receiving the q-th query index $idx \in [n]$, if $q > p(\lambda)$ then aborts; otherwise proceeds the following:

- $k_i^q \leftarrow \mathsf{Sim}_F(1^t, 1^n, i)$, where Sim_F is a simulator for F defined in Definition 2.
- Sends k_i^q to $\mathcal{A}$.

We again follow the Privacy property of the underlying PMPRS scheme F, and have $\text{view}_{Hyb1} \approx_{\epsilon_\lambda} \text{view}_{Real}$.

Realworld Construction. Upon receiving the q-th query index $idx \in [n]$, if $q > p(\lambda)$ then aborts; otherwise proceeds the following:

- Computes $k^q \leftarrow F.\mathsf{Gen}(1^t, 1^n, idx)$.
- Computes $((S_0, k_0^q, \mathsf{ind}_0), (S_1, k_1^q, \mathsf{ind}_1), \ldots, (S_{t-1}, k_{t-1}^q, \mathsf{ind}_{t-1})) \leftarrow F.Punc(k^q, idx)$.
- Sends k_i^q to $\mathcal{A}$.

We then construct the simulator Sim_0 for any adversary $\mathcal{A}$ that acts on behalf of Server 0. The proof of $\text{view}_{Real} \approx_{\epsilon_\lambda} \text{view}_{\mathsf{Sim}_0}$ follows exactly the same flow in the above.

Simulator Construction
In the offline phase,

- For $i = 1$ to T: computes $k_i \leftarrow F.\mathsf{Gen}(1^t, 1^n)$.
- Sends $k_0 \ldots, k_{T-1}$ to $\mathcal{A}$.

In the online phase, upon receiving the q-th query index $idx \in [n]$, if $q > p(\lambda)$ then aborts; otherwise proceeds the following:

- Ignores idx and samples a new index $y \leftarrow_{\$} [n]$.
- Computes $k^q \leftarrow F.\mathsf{Gen}(1^t, 1^n, y)$.
- Computes $((S_0, k_0^q, \mathsf{ind}_0), (S_1, k_1^q, \mathsf{ind}_1), \ldots, (S_{t-1}, k_{t-1}^q, \mathsf{ind}_{t-1})) \leftarrow F.Punc(k^q, y)$.
- Sends k_i^q to $\mathcal{A}$.

Theorem 4. *The $2t$-server PIR with client preprocessing protocol (in Fig. 4) instantiated with the $(t, n) - \mathsf{PMPRS}$ F (in Fig. 3) has the following complexity:*

- $\widetilde{\mathcal{O}}(\lambda\sqrt{n}t^2 n^{\frac{1}{2t}})$ *client storage. If* $t \in [2, \log_2(n)]$, $\widetilde{\mathcal{O}}(\lambda\sqrt{n})$ *client storage;*
- *No additional server storage after offline phase;*
- *Offline Phase:*
  - $\widetilde{\mathcal{O}}(\lambda n)$ *server time and* $\widetilde{\mathcal{O}}(\lambda\sqrt{n}tn^{\frac{1}{2n}})$ *client time; if* $t \in [2, \log_2(n)]$, $\widetilde{\mathcal{O}}(\lambda\sqrt{n})$ *client time;*
  - $\widetilde{\mathcal{O}}(\lambda n^{1/2+1/2t})$ *communication;*
- *Online Phase:*
  - $\widetilde{\mathcal{O}}(\sqrt{n})$ *server time and* $\widetilde{\mathcal{O}}(\sqrt{n} + t^2 n^{1/2t})$ *client time; if* $t \in [2, \log_2(n)]$, $\widetilde{\mathcal{O}}(\sqrt{n})$ *client time;*
  - $\widetilde{\mathcal{O}}(\sqrt{n}t)$ *communication; if* $t \in [2, \log_2(n)]$, $\widetilde{\mathcal{O}}(\sqrt{n})$ *communication.*

Therefore, the amortized communication per query is $\widetilde{\mathcal{O}}(\sqrt{n})$, and the amortized server computation and client computation per query is $\widetilde{\mathcal{O}}(\sqrt{n})$ if we choose $t \in [2, \log_2(n)]$.

Proof. On the client side, it stores the hint vector $\vec{h}$ and the key vector $\vec{k}$ and also needs a buffer to store $F.\mathsf{Punc}$'s output. Recall $\vec{h}$ and $\vec{k}$ both have size $T = O(\lambda\sqrt{n})$ and each element requires $O(1)$ and $\widetilde{\mathcal{O}}(tn^{\frac{1}{2t}})$ storage, separately. $F.\mathsf{Punc}$'s output requires $O(t\log(n))$, $\widetilde{\mathcal{O}}(t^2 n^{\frac{1}{2t}})$, $O(\sqrt{n}\log(n))$ storage for S, k, ind, separately. Summing up together, client needs storage $\widetilde{\mathcal{O}}(\lambda\sqrt{n}t^2 n^{\frac{1}{2t}})$. If we choose $t \in [2, \log_2(n)]$, client-side storage is $\widetilde{\mathcal{O}}(\lambda\sqrt{n})$.

During the offline phase, Server 0 computes $F.\mathsf{Set}()$ function T times, so its computation is bounded by $\widetilde{\mathcal{O}}(\lambda n)$. Client computes $F.\mathsf{Gen}()$ function T times, so its computation is bounded by $\widetilde{\mathcal{O}}(\lambda\sqrt{n}tn^{\frac{1}{2n}})$. If we choose $t \in [2, \log_2(n)]$, client's computation is $\widetilde{\mathcal{O}}(\lambda\sqrt{n})$. For the communication, Server 0 and Client communicate $\vec{h}$ and $\vec{k}$, the size of which are $\widetilde{\mathcal{O}}(\lambda\sqrt{n}t^2 n^{\frac{1}{2t}})$. If we choose $t \in [2, \log_2(n)]$, the communication overhead is $\widetilde{\mathcal{O}}(\lambda\sqrt{n})$.

During the online phase, each Server i (for $i \in [2t]$) computes $F.\mathsf{DotProdEval}$ per query in $\widetilde{\mathcal{O}}(\sqrt{n})$. Client computes $F.\mathsf{Punc}$ twice per query in $\widetilde{\mathcal{O}}(\sqrt{n}+t^2 n^{1/2t})$.

If we choose $t \in [2, \log_2(n)/2]$, the client-side computation is $\widetilde{\mathcal{O}}(\sqrt{n})$ per query. The communication between servers and Client is bounded by $\widetilde{\mathcal{O}}(\sqrt{n}t)$. Since Client receives $\vec{v}_i$ from each server $i \in [2t]$ and the size of $\vec{v}_i$ is bounded by $\sqrt{n}$. If we choose $t \in [2, \log_2(n)/2]$, the communication overhead is $\widetilde{\mathcal{O}}(\lambda\sqrt{n})$.

The correctness proof of our PIR scheme is pretty straightforward and it follows the same blueprints as other PIR correctness proofs in CGK paradigm [11,26]. At a high level, we prove the client always maintains a state containing T random PMPRS keys. In each online phase the client finds a key k containing its query x with probability $1 - \mathsf{negl}.(\lambda)$. Using the key k and its hint bit the client retrieves the correct database bit $DB[x]$ and it replenishes the used key and hint bit, where the correctness of our PMPRS scheme ensures the correctness of the online phase of our construction.

Remark 1 (Extending our PIR scheme for arbitrary n). The proposed $2t$ server PIR scheme with client preprocessing assumes a (t, n)-PMPRS as building block. However, our PMPRS scheme gives us a construction only for parameters t, n such that $n^{1/2t}$ is an integer. To get a PIR scheme for arbitrary $n \in \mathbb{N}$ and $2t$ servers, we can find the smallest integer m greater than or equal to n such that $m^{1/2t}$ is an integer, then we have $m = O(tn)$. Now we can pad the database of size n with $m - n$ dummy elements and then use our PIR scheme based on (t, m)-PMPRS to query the modified database. For $t \in [2, \log(n)/2]$ the asymptotic complexity of offline phase and online phase of this modified protocol would remain unchanged up to polylogarithmic factors in n.

4.1 Improving PIR Communication Complexity

In our proposed PIR with client preprocessing scheme, the online phase communication is dominated by the cost of server responses - which include vectors $\vec{v}_i$ output by the DotProdEval algorithm. However, note that the Client is interested in learning just the $\mathsf{idx}_0, \mathsf{idx}_1, \ldots, \mathsf{idx}_{t-1}^{th}$ bits of the vectors $\vec{v}_0, \vec{v}_1, \ldots, \ldots, \vec{v}_{t-1}$ respectively, since these specific bits allow the client to compute the database xor bit of the punctured set. In our constructions these vectors $\vec{v}_0, \vec{v}_1, \ldots, \ldots, \vec{v}_{t-1}$ are of length $d, d^2, \ldots, d^t = \sqrt{n}$ respectively (where $d = n^{1/2t}$ is an integer). Hence, a natural approach to reduce communication would be to use a PIR scheme where the database on the server side are the vectors $\vec{v}_0, \vec{v}_1, \ldots, \ldots, \vec{v}_{t-1}$ with client query indexes $\mathsf{idx}_0, \mathsf{idx}_1, \ldots, \mathsf{idx}_{t-1}^{th}$, instead of downloading the entire vectors on to the client. However, there exist no non-trivial information theoretic PIR schemes in the single server model [4].

Hence, our next approach would be to consider $4t$ servers instead of $2t$ servers - two servers for each server in the original PIR scheme. For every server and its copy the client sends the same online query - and hence these pair of servers compute the same vector $\vec{v}_i$ as output of DotProdEval in the online phase. And now the Client can use a 2-server PIR scheme to retrieve just the bit of interest $\vec{v}_i[\mathsf{idx}_i]$ in sublinear communication and linear computation in the database size. Instantiating the 2-server PIR primitive with the most communication efficient information-theoretic PIR due to Dvir and Gopi [14] gives us the following result:

Theorem 5. *There exists a 4t-server PIR with client preprocessing protocol with threshold 1 with* $\widetilde{\mathcal{O}}(\lambda\sqrt{n})$ *client storage;* $\widetilde{\mathcal{O}}(\lambda\sqrt{n})$ *online client complexity;* $2^{0.5(\log(n)+O(\sqrt{\log n}))}$ *online server complexity and* $n^{1/2d+o(1)}$ *online bandwidth per query.*

Acknowledgements. We thank all anonymous reviewers for their valuable comments. This work was done while all authors Jaspal Singh, Yu Wei and Vassilis Zikas were at Purdue University. Yu Wei and Vassilis Zikas were funded in part by NSF grant No. 2055599, AnalytiXIN, and Sunday Group, Inc. Jaspal Singh was funded in part by AnalytiXIN, and Sunday Group, Inc.

References

1. Angel, S., Chen, H., Laine, K., Setty, S.T.V.: PIR with compressed queries and amortized query processing. In: 2018 IEEE Symposium on Security and Privacy, pp. 962–979. IEEE Computer Society Press, May 2018
2. Angel, S., Setty, S.: Unobservable communication over fully untrusted infrastructure. In: 2th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16), pp. 551–569 (2016)
3. Beimel, A., Ishai, Y.: Information-theoretic private information retrieval: a unified construction. In: Orejas, F., Spirakis, P.G., van Leeuwen, J. (eds.) ICALP 2001. LNCS, vol. 2076, pp. 912–926. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-48224-5_74
4. Beimel, A., Ishai, Y., Kushilevitz, E., Malkin, T.: One-way functions are essential for single-server private information retrieval. In: 31st ACM STOC, pp. 89–98. ACM Press, May 1999
5. Beimel, A., Ishai, Y., Kushilevitz, E., Raymond, J.-F.: Breaking the $O(n^{1/(2k-1)})$ barrier for information-theoretic private information retrieval. In: 43rd FOCS, pp. 261–270. IEEE Computer Society Press, November 2002
6. Beimel, A., Ishai, Y., Malkin, T.: Reducing the servers computation in private information retrieval: PIR with preprocessing. In: Bellare, M. (ed.) CRYPTO 2000. LNCS, vol. 1880, pp. 55–73. Springer, Heidelberg (2000). https://doi.org/10.1007/3-540-44598-6_4
7. Boyle, E., Gilboa, N., Ishai, Y.: Function secret sharing. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015, Part II. LNCS, vol. 9057, pp. 337–367. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46803-6_12
8. Cachin, C., Micali, S., Stadler, M.: Computationally private information retrieval with polylogarithmic communication. In: Stern, J. (ed.) EUROCRYPT 1999. LNCS, vol. 1592, pp. 402–414. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48910-X_28
9. Chor, B., Kushilevitz, E., Goldreich, O., Sudan, M.: Private information retrieval. J. ACM (JACM) **45**(6), 965–981 (1998)
10. Corrigan-Gibbs, H., Henzinger, A., Kogan, D.: Single-server private information retrieval with sublinear amortized time. In: Dunkelman, O., Dziembowski, S. (eds.) EUROCRYPT 2022. LNCS, vol. 13276, pp. 3–33 . Springer, Cham (2022). https://doi.org/10.1007/978-3-031-07085-3_1
11. Corrigan-Gibbs, H., Kogan, D.: Private information retrieval with sublinear online time. In: Canteaut, A., Ishai, Y. (eds.) EUROCRYPT 2020. LNCS, vol. 12105, pp. 44–75. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-45721-1_3

12. Demmler, D., Rindal, P., Rosulek, M., Trieu, N.: PIR-PSI: scaling private contact discovery. Cryptology ePrint Archive (2018)
13. Döttling, N., Garg, S., Ishai, Y., Malavolta, G., Mour, T., Ostrovsky, R.: Trapdoor hash functions and their applications. In: Boldyreva, A., Micciancio, D. (eds.) CRYPTO 2019, Part III. LNCS, vol. 11694, pp. 3–32. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-26954-8_1
14. Dvir, Z., Gopi, S.: 2-server PIR with subpolynomial communication. J. ACM (JACM) **63**(4), 1–15 (2016)
15. Efremenko, K.: 3-query locally decodable codes of subexponential length. In: Mitzenmacher, M. (ed.) 41st ACM STOC, pp. 39–44. ACM Press, May/June (2009)
16. Fisch, B., Lazzaretti, A., Liu, Z., Papamanthou, C.: Single server PIR via homomorphic thorp shuffles. Cryptology ePrint Archive (2024)
17. Ghoshal, A., Zhou, M., Shi, E.: Efficient pre-processing PIR without public-key cryptography. In: Joye, M., Leander, G. (eds.) EUROCRYPT 2024. LNCS, vol. 14656, pp. 210–240. Springer, Cham (2024). https://doi.org/10.1007/978-3-031-58751-1_8
18. Gilboa, N., Ishai, Y.: Distributed point functions and their applications. In: Nguyen, P.Q., Oswald, E. (eds.) EUROCRYPT 2014. LNCS, vol. 8441, pp. 640–658. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-642-55220-5_35
19. Henry, R., Huang, Y., Goldberg, I.: One (block) size fits all: PIR and SPIR with variable-length records via multi-block queries. In: NDSS (2013)
20. Henzinger, A., Hong, M.M., Corrigan-Gibbs, H., Meiklejohn, S., Vaikuntanathan, V.: One server for the price of two: simple and fast {single-server} private information retrieval. In: 32nd USENIX Security Symposium (USENIX Security 23), pp. 3889–3905 (2023)
21. Hetz, L., Schneider, T., Weinert, C.: Scaling mobile private contact discovery to billions of users. Cryptology ePrint Archive (2023)
22. Ishai, Y., Kushilevitz, E., Ostrovsky, R., Sahai, A.: Batch codes and their applications. In: Babai, L. (ed.) 36th ACM STOC, pp. 262–271. ACM Press, June 2004
23. Ishai, Y., Shi, E., Wichs, D.: PIR with client-side preprocessing: information-theoretic constructions and lower bounds. In: Reyzin, L., Stebila, D. (eds.) CRYPTO 2024. LNCS, vol. 14928. Springer, Cham (2024). https://doi.org/10.1007/978-3-031-68400-5_5
24. Kogan, D., Corrigan-Gibbs, H.: Private blocklist lookups with checklist. In: 30th USENIX Security Symposium (USENIX Security 21), pp. 875–892 (2021)
25. Kushilevitz, E., Ostrovsky, R.: Replication is not needed: single database, computationally-private information retrieval. In: Proceedings 38th Annual Symposium on Foundations of Computer Science, pp. 364–373. IEEE (1997)
26. Lazzaretti, A., Papamanthou, C.: TreePIR: sublinear-time and polylog-bandwidth private information retrieval from DDH. Cryptology ePrint Archive (2023)
27. Lueks, W., Goldberg, I.: Sublinear scaling for multi-client private information retrieval. In: Böhme, R., Okamoto, T. (eds.) FC 2015. LNCS, vol. 8975, pp. 168–186. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-47854-7_10
28. Menon, S.J., Wu, D.J.: SPIRAL: fast, high-rate single-server PIR via FHE composition. In: 2022 IEEE Symposium on Security and Privacy, pp. 930–947. IEEE Computer Society Press, May 2022
29. Mittal, P., Olumofin, F., Troncoso, C., Borisov, N., Goldberg. I.: {PIR-Tor}: scalable anonymous communication using private information retrieval. In: 20th USENIX Security Symposium (USENIX Security 11) (2011)

30. Mughees, M.H., Sun, I., Ren, L.: Simple and practical amortized sublinear private information retrieval. Cryptology ePrint Archive (2023)
31. Shi, E., Aqeel, W., Chandrasekaran, B., Maggs, B.: Puncturable pseudorandom sets and private information retrieval with near-optimal online bandwidth and time. In: Malkin, T., Peikert, C. (eds.) CRYPTO 2021, Part IV. LNCS, vol. 12828, pp. 641–669. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-84259-8_22
32. Stinson, D., Wei, R., Paterson, M.B.: Combinatorial batch codes. Adv. Math. Commun. **3**(1), 13–27 (2009)
33. Yekhanin, S.: Towards 3-query locally decodable codes of subexponential length. J. ACM (JACM) **55**(1), 1–16 (2008)
34. Zhou, M., Park, A., Shi, E., Zheng, W.: Piano: extremely simple, single-server PIR with sublinear server computation. Cryptology ePrint Archive (2023)

Asynchronous Agreement on a Core Set in Constant Expected Time and More Efficient Asynchronous VSS and MPC

Ittai Abraham[1], Gilad Ashsarov[2], Arpita Patra[3], and Gilad Stern[4(✉)]

[1] Intel Labs, Petah Tikvah, USA
ittai.abraham@intel.com
[2] Bar-Ilan University, Ramat Gan, Israel
[3] Indian Institute of Science, Bengaluru, India
[4] Tel Aviv University, Tel Aviv-Yafo, Israel
giladstern@tauex.tau.ac.il

Abstract. A major challenge of any *asynchronous* MPC protocol is the need to reach an agreement on the set of private inputs to be used as input for the MPC functionality. Ben-Or, Canetti and Goldreich [STOC 93] call this problem *Agreement on a Core Set* (ACS) and solve it by running n parallel instances of asynchronous binary Byzantine agreements. To the best of our knowledge, all results in the perfect and statistical security setting used this same paradigm for solving ACS. Using all known asynchronous binary Byzantine agreement protocols, this type of ACS has $\Omega(\log n)$ expected round complexity, which results in such a bound on the round complexity of MPC protocols as well (even for constant depth circuits).

We provide a new solution for Agreement on a Core Set that runs in expected $\mathcal{O}(1)$ rounds. Our perfectly secure variant is optimally resilient ($t < n/4$) and requires just $\mathcal{O}(n^4 \log n)$ expected communication complexity. We show a similar result with statistical security for $t < n/3$. Our ACS is based on a new notion of *Asynchronously Validated Asynchronous Byzantine Agreement* (AVABA) and new information-theoretic analogs to techniques used in the authenticated model. Along the way, we also construct a new perfectly secure packed asynchronous verifiable secret sharing (AVSS) protocol with just $\mathcal{O}(n^3 \log n)$ communication complexity, improving the state of the art by a factor of $\mathcal{O}(n)$. This leads to a more efficient asynchronous MPC that matches the state-of-the-art synchronous MPC.

1 Introduction

Broadly, there are two main network conditions where secure multiparty computation protocols were studied. The first is the synchronous setting, where all messages sent between honest parties arrive after some known bounded delay.

For a full version of this paper, see [4].

E. Boyle and M. Mahmoody (Eds.): TCC 2024, LNCS 15367, pp. 451–482, 2025.
https://doi.org/10.1007/978-3-031-78023-3_15

The choice of this delay bound is critical: setting a large delay causes the protocol to be inefficient and slow, while setting a small delay might lead to non-termination. The second category is the asynchronous model, where each message sent between honest parties arrives after some finite delay. This model allows protocols to dynamically adjust to adversarial network conditions and terminate even when the adversary can adaptively manipulate the delays.

One of the core challenges for MPC protocols in the *asynchronous* setting is that they must reach *agreement* on which private inputs to use as input for the circuit. Ben-Or, Canetti and Goldreich (BCG) [11] call this problem Agreement on a Core Set (ACS). In this paper, we consider protocols with *optimal resilience in the asynchronous model against computationally unbounded adversaries*. From the lower bound of [5,11,13], *perfect security* for MPC implies that the number of corruptions in this setting is at most $t < n/4$, so optimal resilience is when $n = 4t + 1$. This is in contrast to the optimal resilience of $n = 3t + 1$ in the synchronous perfect security setting and the asynchronous statistical security setting. The seminal result of [11,15] is the first work to obtain perfect security with optimal resilience in the asynchronous model.

Before proceeding, we refine the problem definition. Our main motivating application for ACS is in asynchronous secure computation. Each party shares its input at the beginning of the protocol using asynchronous verifiable secret sharing (AVSS). When a dealer is honest, then all honest parties will eventually receive valid shares. If the dealer is corrupted and one honest party successfully completes the AVSS, then all honest parties will eventually also receive valid shares. However, some instance of corrupted dealers might never terminate, and some instances of honest dealers might be very slow (due to adversarial dealys). The parties then wish to agree on a common core set of $n - t$ parties whose AVSS has been successfully completed or will eventually terminate. Reaching an agreement is crucial for the sequel of the secure protocol. Using an ACS protocol, parties agree on some set of $n-t$ parties ("core") whose AVSS has terminated or will eventually terminate for all parties. The difficulty is that due to asynchrony, some of the inputs of honest parties (which instances terminated) might arrive dynamically, and the corrupted parties might input identities of instances that will never terminate.

In terms of round complexity, the best one can hope for is reaching agreement in constant expectation [25]. However, to the best of our knowledge, all results in the asynchronous information-theoretic setting run $\mathcal{O}(n)$ parallel asynchronous binary Byzantine agreement instances to agree on a core set. All known asynchronous binary agreement protocols follow a geometric distribution, and composing n such protocols in parallel, means that the expectation of the maximum is $\Omega(\log n)$. So for over 30 years, the best expected round complexity for asynchronous MPC has $\Omega(\log n)$ overhead (even for constant depth circuits)[1]. A natural question remained open:

[1] As opposed to some claims in the literature, the work of [12] does not provide an $O(1)$ expected time ACS; see Sect. 1.2.

Is there an asynchronous MPC with ***constant*** *expected running time overhead? Or is there an inherent $\Omega(\log n)$ lower bound for ACS due to asynchrony?*

1.1 Our Contributions

Our main contributions are (1) a novel protocol for agreement on a core set in constant expected time via a new multi-valued agreement protocol with *asynchronous validation*; (2) Efficiency improvements in the communication complexity of asynchronous verifiable secret sharing. Our new ACS and AVSS together significantly improve the communication complexity and round complexity of asynchronous MPC.

Asynchronously Validated Asynchronous Byzantine Agreement (AVABA). We achieve ACS via a new notion that we introduce, called "AVABA". This is an information-theoretic version of Validated Asynchronous Byzantine Agreement (VABA [7]), where the external validity function is replaced with asynchronous validation. Our AVABA protocol is perfectly secure and resilient to $t < n/4$ corruptions. For inputs of size $\mathcal{O}(n)$ bits, it runs in $\mathcal{O}(1)$ expected time and requires $\mathcal{O}(n^4 \log n)$ expected communication complexity. Parties are guaranteed to reach an agreement on an input of one of the parties, and the value is guaranteed to pass an asynchronous validation. In the MPC setting, this validation checks that the input contains $n - t$ parties who verifiably completed the input-sharing phase. To the best of our knowledge, the most efficient agreement protocols [10,24] with constant expected rounds and $t < n/4$ currently require $\mathcal{O}(n^6 \log n)$ bits to be sent in expectation. Furthermore, these protocols are binary agreement protocols. Our protocol improves the efficiency of those protocols and allows for multi-valued agreement.

Theorem 1.1 (Asynchronously Validated Asynchronous Byzantine Agreement (informal))**.** *There exists a perfectly secure protocol for asynchronous Byzantine agreement with asynchronous validation (AVABA) that is resilient to $t < n/4$ Byzantine corruptions. Each party has a valid input of size $\mathcal{O}(n \log n)$ bits. The protocol runs in constant expected time and $\mathcal{O}(n^4 \log n)$ expected communication complexity.*

Agreement on a Core Set (ACS). Using this AVABA protocol and an asynchronous validation checking which parties shared their inputs, we implement a perfectly secure, $t < n/4$ resilient constant expected time protocol for Agreement on a Core Set (ACS) with an expected $\mathcal{O}(n^4 \log n)$ communication complexity. To the best of our knowledge, this is the first time ACS is solved via a multi-valued agreement in the information-theoretic setting without using any binary agreement building blocks. See Sect. 1.2 for more details.

Corollary 1.2. *There exists a perfectly secure protocol for asynchronous agreement on a core set (ACS) with an asynchronous validation resilient to $t < n/4$ Byzantine corruptions. The protocol runs in constant expected time and $\mathcal{O}(n^4 \log n)$ expected communication complexity.*

As we elaborate in the related work section, the communication cost of ACS from [15] is $\mathcal{O}(n^7 \log n)$.

Extensions for $t < n/3$ Corruptions and Statistical Security. We also extend our results to the statistical settings, and derive resilience to $t < n/3$ corruptions. Set Sect. 2.3 for further details.

Asynchronous Verifiable Secret Sharing. Our second main contribution is a new asynchronous verifiable secret sharing (AVSS):

Theorem 1.3. *There exists a perfectly secure protocol for asynchronous verifiable secret sharing resilient to $t < n/4$ malicious corruptions. For sharing X secrets (of $\mathcal{O}(\log n)$ bits each), the total communication complexity is $\mathcal{O}(nX + n^3 \log n)$.*

This means that we get $\mathcal{O}(n)$ overhead for sharing $\Omega(n^2)$ secrets. Prior to our work, the AVSS protocol of [11] achieves total communication complexity of $\mathcal{O}(n^4 \log n)$ for sharing *one* secret, and the one by [19,29] obtains a total communication complexity of $\mathcal{O}(nX + n^4 \log n)$ bits for X secrets. That is, for obtaining $\mathcal{O}(n)$ overhead the dealer has to share $\Omega(n^3)$ secrets. In our ACS protocol, the dealer has to share just $\mathcal{O}(n)$ secrets, in which case our AVSS requires $\mathcal{O}(n^3 \log n)$ as opposed to $\mathcal{O}(n^4 \log n)$ by [19,29], improving the communication by a factor of $\mathcal{O}(n)$.

Conclusion: Asynchronous MPC. When plugging our new AVSS and new ACS in the recent asynchronous MPC protocol of [3], we obtain the following corollary:

Corollary 1.4. *For a circuit with C multiplication gates and depth D, there exists a perfectly secure, optimally-resilient asynchronous MPC protocol with $\mathcal{O}((Cn + Dn^2 + n^4) \log n)$ communication complexity and $\mathcal{O}(D)$ expected runtime.*

Without our work, when combining the protocol of [3] with the ACS protocol of [15], together with the AVSS protocol of [19,29], the cost of the entire MPC is $\mathcal{O}((Cn + Dn^2 + n^7) \log n)$ and $\mathcal{O}(D + \log n)$ expected-time. Our work improves both the communication and round complexities. See Sect. 2.3.

Synchronous vs. Asynchronous MPC. We conclude this section by noting an accepted claim regarding the relationship between synchronous and asynchronous MPC is that (see, e.g., Nielsen [28]):

> *"Synchronous MPC has higher security and requires less communication than asynchronous MPC."*

The first is part of the sentence is due to the different optimal bounds: Optimal resilient in perfect MPC in the synchronous setting is $t < n/3$ whereas in the asynchronous setting is $t < n/4$. The second part, as claimed in [28] is due to the fact that "Another advantage of synchronous MPC is that we know how to construct them with much less communication in terms of bits sent than the asynchronous ones".

Our work shows that the second part of the claim is *false*, at least, for perfect MPC. Specifically, the current most efficient perfect *synchronous* MPC [2] protocol achieves the **exact same complexity** as ours: $\mathcal{O}((Cn + Dn^2 + n^4) \log n)$ with $\mathcal{O}(D)$ expected rounds.[2] Note that while one could run asynchronous protocols in synchronous networks, asynchronous MPC protocols are designed for lower corruption thresholds and only take $n - t$ inputs into consideration. This even gives rise to the hope that perhaps, one can even construct a more efficient asynchronous MPC protocol than a synchronous protocol, and our understanding of the relationship between synchronous and asynchronous protocols is still lacking.

1.2 Related Work

Agreement on a Core Set via n Parallel Binary Agreements. In the asynchronous setting, an MPC protocol cannot wait for input from all parties. One important task of any MPC protocol in the asynchronous setting is reaching *agreement* on the set of parties whose private input is used as the input for the MPC circuit. To solve this, Ben-Or, Canetti and Goldreich [11] suggest a protocol called Agreement on a Core Set (ACS). To the best of our knowledge, all previous asynchronous MPC protocols (in the perfect and statistical security setting) use the same ACS protocol suggested by [11]. This ACS is based on running n parallel binary agreements. Roughly speaking, parties enter input 1 to the ith binary agreement when they see that party P_i has completed secret sharing its input and enter all remaining instances with the value 0 once they see at least $n - t$ agreements terminate with a decision of 1.

On the positive side, this elegant solution requires just simple binary agreement as a building block. On the negative side, each binary agreement instance has an independent constant probability of terminating in each round, so in isolation, each instance has a constant expectation. However, the expectation of the maximum of n such independent instances is $\Omega(\log n)$. Therefore, this approach of running separate binary instances seems to have a natural barrier for obtaining $\mathcal{O}(1)$ expected round complexity. Lastly, the best known binary agreement protocols [10,24] in this setting require the expected total of $\mathcal{O}(n^7 \log n)$ communication bits for the ACS.

On Ben-Or and El-Yaniv's Work. The work of [11] claims that a result of Ben-Or and El-Yaniv solves ACS in constant expected rounds. Here we explain why this is not the case. The work by Ben-Or and El-Yaniv [12] (published 10 years after [11] cites them) deals with executing n concurrent instances of Byzantine Agreement. The first part of [12] is for the synchronous model and we believe it can be used to agree on a common subset in synchrony. The second part claims that these techniques can be extended to solve some variant of multi-sender agreement in constant expected rounds in the asynchronous model.

[2] A somewhat incomparable result [26] removes the $\mathcal{O}(Dn^2)$ in the communication complexity: $\mathcal{O}((Cn + n^5) \log n)$ communication with $\mathcal{O}(D + n)$ expected rounds. We are not aware of a comparable result in the asynchronous case.

The situation for the asynchronous model is different. First, we note that [12] explicitly do **not** mention that they can solve ACS in the asynchronous model. Indeed, the stated results of [12] and the techniques of [12] do **not** provide a way to solve ACS (as needed for asynchronous MPC) in constant expected rounds. They only solve an easier problem in which the input of each party exists at the beginning of the protocol (unlike ACS, where due to asynchrony some of the inputs may arrive dynamically over time).

The Work of Ben-Or, Kelmer and Rabin [13]**.** In [13]'s ACS protocol, parties first invoke the BA instances with input 1 for parties who are deemed valid according to a validity condition (in the case of MPC, dealers whose VSS instances have been completed). Parties input 0 to the remaining instances only after seeing $n-t$ instances with output 1. Trying to naively apply the techniques of [12] does not work because they require starting **all** BA instances at the same time and **synchronizing** them using Select (the Select protocol in round r waits for all $n \log n$ BA instances to reach round $r+1$). It is possible that a less naive approach may work, synchronizing some of the BA instances using Select, and then initiating the rest. This seems to require a much more subtle approach since parties are required to wait for the agreed output for each party to be 1 before proceeding (while dealing with $\log(n)$ BA instances per party), and possibly using Select several times.

A possible alternative approach is having each party set all inputs to the BA instances at once, after seeing that at least $n-t$ of those inputs are 1. Using this approach, it is possible that no party has the unanimous support of all honest parties, meaning that each party has at least one honest party input 0 to its BA instance. In this case, parties can output 0 in all instances and thus output an empty set as the agreed core. Even protocols that strengthen the validity conditions are likely to fail because it is possible that most BA instances have many 0 and 1 inputs, resulting in small cores (for example, of size $t+1$ as opposed to $n-t$). We believe that obtaining a less naive protocol could potentially be an interesting follow-up work.

Recent and Concurrent Work. Two recent and concurrent works deal with tasks related to information-theoretic agreement on a core set with constant round complexity. Duan *et al.* [23] claims to have an information-theoretic construction of an ACS in constant expected rounds. However, their construction uses cryptographic hash functions and a threshold PRF, instantiated by a threshold signature scheme. So while their work solves ACS with constant round complexity it is not perfectly secure or statistically secure (it does not obtain $\mathcal{O}(1)$ expected rounds against an unbounded adversary). Moreover, our core consensus protocol obtains the same efficiency while requiring significantly weaker primitives. Our core consensus protocol obtains the same asymptotic $\mathcal{O}(n^3 \log n)$ bit complexity but requires just a weak leader election (which can be implemented information theoretically via AVSS, as we show). On the other hand, [23] requires a strong leader election, which, to the best of our knowledge, requires a DKG and a computationally bounded adversary for the required complexity. An additional recent work in the cryptographic setting is that of Das *et al.* [22]. Their work

only relies on a cryptographic hash function without additional setup assumptions and achieves $\mathcal{O}(\lambda n^3)$ bit complexity and $\mathcal{O}(1)$ time complexity (Table 1).

Cohen *et al.* [21] construct a constant expected round protocol for a linear number of binary Byzantine agreement protocols (where all inputs arrive at the beginning of the protocol). The first version of [21] offhandedly remarks that this primitive can be used to construct a constant round ACS protocol. However, as stated in our discussion of Ben-Or and El-Yaniv's work, these protocols require parties to know their inputs to all instances of binary agreement at the same time. The currently known reductions from binary agreement to ACS in the *asynchronus* setting rely on this not being the case.

A newer version of [21] was made public after an earlier version of our work appeared online [4]. In this newer version, the authors use the information theoretic Gather protocol defined in our paper in order to solve ACS (the gather protocol in [6] relied on cryptographic assumptions). The resulting ACS of [21] uses our techniques, and requires at least $\Omega(n^5\kappa^2 + n^6)$ bits of communication and is statistically secure, where κ is a statistical security parameter. This bound stems from each party having to share n secrets. Using the packed secret sharing protocol of [20], each such sharing requires $\Omega(n^4\kappa^2 + n^5)$ bits of communication. Constructing a statistical ACS protocol using our techniques will also have the same bottleneck of n packed sharings, resulting in the same communication complexity. In comparison, our perfect ACS protocol can be used to solve ACS in just $\mathcal{O}(n^4 \log n)$ bits of communication with perfect security.

Table 1. A comparison of ACS schemes, in terms of: (1) the total number of bits sent; (2) the expected number of rounds; (3) the type of security provided by the protocol (either perfect security, statistical security, or computational security reliant on cryptographic assumptions). All of the works are optimally resilient, i.e. assume $n > 4t$ for perfectly secure schemes and $n > 3t$ for statistical and computational schemes. κ and λ are statistical and cryptographic security parameters respectively. Both [21] and our statistical protocol are evaluated using the packed ACSS protocol of [20] for n instances of packed secret sharing, assuming sharing $\mathcal{O}(n)$ secrets costs $\mathcal{O}(n^4\kappa^2 + n^5)$.

Protocol	Bit Complexity	Expected Time	Security
	Complexity	Complexity	
BCG [11]	$\mathcal{O}(n^7 \log n)$	$\mathcal{O}(\log n)$	perfect
BKR [13]	$\Omega(n^{13}k^2)$	$\mathcal{O}(\log n)$	statistical
FIN [23]	$\mathcal{O}(\lambda n^3)$	$\mathcal{O}(1)$	computational with DKG
Cohen et al. [21]	$\mathcal{O}(n^5\kappa^2 + n^6)$	$\mathcal{O}(1)$	statistical
Das et al. [22]	$\mathcal{O}(\lambda n^3)$	$\mathcal{O}(1)$	computational
This work (statistical)	$\mathcal{O}(n^5\kappa^2 + n^6)$	$\mathcal{O}(1)$	statistical
This work (perfect)	$\mathcal{O}(n^4 \log n)$	$\mathcal{O}(1)$	perfect

2 Technical Overview

We provide a high level overview of our main techniques. In Sect. 2.1 we provide a brief overview of our AVSS protocol, which might be of an independent interest. In Sect. 2.2 we provide the overall structure of the ACS protocol, and our AVABA protocol. We also provide some extensions to our result in Sect. 2.3.

The Model. Before we start, let us first introduce the model. We assume asynchronous communication, which means that the adversary can arbitrarily delay messages sent between honest parties, while it cannot necessarily see their content. Messages between honest parties can be delayed but must be delivered eventually. Honest parties, therefore, cannot distinguish between the case where a message (from a corrupted party to an honest party) has never been sent or whether a message (from an honest party to an honest party) is delayed. Thus, protocols must make sure that parties do terminate and parties cannot wait to receive messages from all parties. Parties can wait to receive messages from more than $n - t$ parties only when they are certain that not all messages from honest parties have been received.

Notation. To describe the ith party, we use i or P_i interchangeably. In Sect. 2.1 we overview our improvement in the communication complexity of AVSS. In Sect. 2.2 we overview our new asynchronously validated asynchronous Byzantine Agreement.

2.1 Packed Asynchronous Verifiable Secret Sharing

In this section, we describe an information-theoretic packed AVSS protocol that requires $\mathcal{O}(nX + n^3 \log n)$ for sharing X secrets.

A Quick Overview of the AVSS Protocol of [11]. We start with a quick overview of the AVSS protocol of Ben-Or, Canetti and Goldreich [11]. As a first step, we present an inefficient version where the dealer runs in exponential time.

1. The dealer chooses a random bivariate polynomial $S(X, Y)$ of degree-t in both variables such that $S(0, 0) = s$. It then gives each party P_i the shares $f_i(X) = S(X, i)$, $g_i(Y) = S(i, Y)$.
2. After receiving their f_i and g_i polynomials, which we call their shares, and seeing that they are of the correct degrees, every party P_i forwards the values $f_i(j), g_i(j)$ to every P_j.
3. When a party P_i receives a forwarded pair of values $f_j(i), g_j(i)$ from some party P_j, it verifies that these values are consistent with the values it has received from the dealer. Namely, that $f_i(j) = g_j(i)$ $(= S(j, i))$ and $g_i(j) = f_j(i)$ $(= S(i, j))$. If this is the case, then P_i broadcasts $\langle \texttt{ok}, i, j \rangle$, signifying that P_i agrees with P_j.
4. The dealer initiates a graph G with $V = [n]$ and $E = \emptyset$. Then, upon receiving broadcasted messages $\langle \texttt{ok}, i, j \rangle$ and $\langle \texttt{ok}, j, i \rangle$ it adds the edge (i, j) to E. It then looks for a clique $K \subseteq [n]$ in G. If found, it broadcasts $\langle \texttt{clique}, K \rangle$. Otherwise, it continues to listen to more `ok` messages, updates its graph, and repeats.

5. Each party also initiates a graph as the dealer and adds edges in a similar manner. Once the dealer broadcasts $\langle \texttt{clique}, K \rangle$, the party verifies that K is a clique in its respective graph. If so, it terminates. Otherwise, it continues to listen to more edges.

It is easy to see that if one honest party P_j terminates, all honest parties will eventually terminate. This is because P_j terminates only after the dealer has broadcasted a clique and P_j has verified the same clique in its respective graph. Since all the messages that P_j considers are broadcasted, each other honest party will eventually see the same clique in its graph. Moreover, if the dealer is honest, then the dealer will eventually see the clique of all honest parties in its graph, will broadcast it, and all honest parties will eventually verify it as well.

To show binding (i.e., there is a well-defined secret at the end of the sharing phase), assume that the dealer has broadcasted some clique and an honest party has verified that clique. Since the clique contains at least $3t+1$ parties, it contains at least $2t + 1$ honest parties. Since each pair of honest parties has verified that their shares agree, they all lie on the same bivariate polynomial $S(x, y)$. Moreover, parties only consider members of the clique during reconstruction. This implies that in the reconstruction phase, we will have at least $2t+1$ correct shares and at most t errors, and therefore reconstruction is guaranteed.

The Star Algorithm. To make the dealer computationally efficient, Canetti [15] defines the FindStar algorithm that finds a large "star" [15] in a graph, which can be thought of as a relaxation of a clique. It receives an undirected graph $G = (V, E)$ as input and outputs a pair of sets $C, D \subseteq V$ such that $C \subseteq D$ and there exists an edge $(u, v) \in E$ for every $u \in C, v \in D$, and such that $|C| \geq n - 2t$ $(= 2t + 1)$ and $|D| \geq n - t$ $(= 3t + 1)$. The algorithm might also output "no star was found". In addition, Canetti [15] showed a polynomial time algorithm that finds such a STAR if there exists a clique of size $n - t$. The dealer then looks for a STAR in the graph instead of a clique, and once found, it broadcasts $\langle \texttt{star}, C, D \rangle$. Each party verifies that (C, D) is a STAR in its graph, and terminates if so.

This guarantees validity and binding: **Validity:** If the dealer is honest, then the protocol should terminate, and reconstruction should be the secret that the dealer shared. When the dealer is honest, eventually, there will be a clique in the graph of size $n - t$. The STAR algorithm then outputs a star (C, D), and all honest parties will eventually verify that (C, D) is a STAR. All honest parties receive shares that are consistent with the dealer's polynomial. **Binding:** Once an honest party terminates (regardless of whether or not the dealer is honest), the set C contains at least $t + 1$ honest parties, and all their shares must agree. Therefore, the set C defines a unique bivariate polynomial $S(X, Y)$, and the shares of all honest parties in C lie on that polynomial. Moreover, the set D contains at least $2t + 1$ honest parties, and their shares agree with all parties in C and, therefore, must also agree with S. We obtain that there are at least $2t+1$ honest parties with valid shares, and therefore reconstruction is guaranteed even if t errors are introduced during the reconstruction phase, by utilizing Reed Solomon decoding.

Cost. When considering the costs of broadcasting ok messages, the above protocol requires $\mathcal{O}(n^4 \log n)$ bits to be transmitted over the point-to-point channels. This is because each message of size L being broadcasted requires $\mathcal{O}(n^2 L)$ bit sent over the point-to-point channels. Since each party P_i broadcasts $(\texttt{ok}, i, j)$ we have $\mathcal{O}(n^2)$ messages being broadcasted. This implies that overall, we have $\mathcal{O}(n^4 \log n)$ overhead for sharing a single secret.

Reducing the Cost. To reduce the cost, the protocol of [29] utilizes the following two tricks:

- **Packing:** Instead of having a bivariate polynomial of degree t in both variables, the dealer can embed $t + 1$ secrets in one bivariate polynomial. That is, the dealer holds secrets $s_0, \ldots, s_t$, and uniformly samples a bivariate polynomial $S(X, Y)$ of degree $2t$ in X and degree t in Y such that $S(-k, 0) = s_k$ for every $k \in \{0, \ldots, t\}$.
- **Batching:** Instead of having one instance of AVSS, we can run $\mathcal{O}(n^2)$ instances in parallel while re-using the broadcast messages across all instances. That is, each P_i broadcasts $\langle \texttt{ok}, i, j \rangle$ only after it verified the shares of j across all $\mathcal{O}(n)$ instances.

Those two ideas together lead to a protocol in which parties send $\mathcal{O}(nX + n^4 \log n)$ bits point-to-point for sharing X secrets (of $\mathcal{O}(\log n)$ bits each). This yeilds an overhead of $\mathcal{O}(n)$ per secret, starting from $\Omega(n^3)$ secrets. However, if the dealer has to distribute only $\mathcal{O}(n)$ secrets (as in our ACS protocol), we get an overhead of $\mathcal{O}(n^3)$.

The difficulty is that there are n^2 "short messages" to be broadcasted ($\langle \texttt{ok}, i, j \rangle$), and the overhead of each broadcast is $\mathcal{O}(n^2)$. One might try to amortize these costs by having parties send $\mathcal{O}(n)$ of these ok messages at the same time (in which case, a broadcast with an overhead of $\mathcal{O}(n)$ can be used). However, parties do not even know how many ok they might broadcast. For instance, in the case of an honest dealer, parties know that they will eventually broadcast oks for the $n - t$ honest parties, but they do not know how many corrupted parties would send them correct sub-shares. So they can wait and broadcast one large $n - t$ ok message, but then they will still have to broadcast the remaining (even up to $t = \mathcal{O}(n)$) messages one-by-one, as they arrive one-by-one. Moreover, all parties must hear all the edges between honest parties, to verify that the graph has a star. Thus, total communication of $\Omega(n^4)$ looks like a natural barrier.

Breaking the $\Omega(n^4)$-Barrier. We achieve $\mathcal{O}(n)$-overhead for $o(n^3)$ secrets. To reduce the overhead when the dealer has to share $o(n^3)$ secrets, we further improve the protocol and add one more optimization to packing and batching:

- **No broadcast:** We completely eliminate any broadcast message in the protocol.

To achieve this property, first, consider trying to replace any broadcast message with multicast (the sender simply sends the message to all parties). Edges (i, j)

between pairs of honest parties will appear in all graphs and will be consistent. On the other hand, edges between corrupted parties or between an honest party and a corrupted party might not be consistent in the different graphs.

To overcome this difficulty, we instruct each party P_i to look for its own STAR (C_i, D_i). Moreover, in addition to those two sets, we look for an extended star (see, e.g., [29]) (C_i, D_i, E_i, F_i) which satisfies the following properties:

- C_i: a clique of size (at least) $n - 2t$ (i.e., $2t + 1$), as before.
- D_i: a set of size (at least) $n - t$ that agrees with all C_i (i.e., for all $d \in D_i$ and $c \in C_i$ there exists an edge – (c, d)). This is again as before.
- F_i: a set of size $n - t$ of all vertices that have at least $n - 2t$ edges to C_i.
- E_i: a set of size $n - t$ of all vertices that have at least $n - t$ edges to F_i.

Each party finds an extended star in its graph. Then, the challenge is that different parties might have different graphs. Nevertheless, we claim that the following holds: (1) Validity: If the dealer is honest, then all honest parties will eventually find extended stars in their respective graphs; (2) Binding: Any pair of extended stars found by honest parties define the exact same bivariate polynomial, even they do not necessarily have the same graphs.

- **Validity:** It is also easy to see that if the dealer is honest, then all honest parties will eventually find such (C_i, D_i, E_i, F_i): The clique of all honest parties will eventually appear in the respective graphs of each honest party. An extended star will then always be found.
- **Binding:** We claim that for every honest party P_j, the honest parties in the sets that P_j has found, i.e., the honest parties in the sets (C_j, D_j, E_j, F_j), define a unique bivariate polynomial. Specifically:
  1. The set C_j contains at least $t + 1$ honest parties; take an arbitrary subset $C'_j \subseteq C_j$ of cardinality $t + 1$; their f-shares (each is of degree-$2t$) define a unique bivariate polynomial $S_j(X, Y)$ of degree $(2t, t)$;
  2. The set D_j contains at least $2t + 1$ honest parties, and each such party agrees with all parties in C_j; As such, each such party must hold a g-share that lies on $S_j(X, Y)$. Since D_j contains at least $2t + 1$ honest parties, it also implies that all the other honest parties (if exist) in $C_j \setminus C'_j$ hold f-shares that lie on S_j.
  3. The set F_j contains at least $2t + 1$ honest parties; each such honest party agrees with at least $n - 2t$ (i.e., $\geq 2t + 1$) parties in C_j, i.e., with at least $t + 1$ honest parties in C_j. Thus, all the g-shares of parties in F_j lie on S_j.
  4. The set E_j contains at least $2t + 1$ honest parties; each such party agrees with at least $3t + 1$ parties in F_j, i.e., with at least $2t + 1$ honest parties. As such, all the f-shares of parties in E_j lie on S_j.

  Moreover, we claim that for two honest parties P_j and P_k that might have distinct extended stars (C_j, D_j, E_j, F_j) and (C_k, D_k, E_k, F_k) that define the bivariate polynomials S_j and S_k respectively, $S_j = S_k$. This is because E_j and E_k are both sets of size $3t+1$ and, therefore, must have an intersection of size at least $2t+1$ and thus have at least $t+1$ honest parties in their intersection.

The f-shares of those parties uniquely define S_j and S_k, respectively, and thus it must hold that $S_k = S_j$.

The Rest of the Protocol. In the rest of the protocol, parties that do have shares help the other parties reconstruct their shares. Since the shares of all honest parties that have an extended star must define the same bivariate polynomial, we get that, eventually, all parties hold shares on that polynomial. Therefore, we get a *complete secret sharing*: all honest parties have shares at the end of the protocol. This makes the reconstruction phase almost trivial. Parties just send their shares to one another, and use Reed Solomon decoding to eliminate errors. We refer to Sect. 5 for full specification and proofs.

2.2 ACS and AVABA

We now describe how we construct the ACS protocol, and our new notion called "AVABA", which is an information-theoretic version of Validated Asynchronous Byzantine Agreement (VABA [7]). This is a multivalued agreement protocol that allows parties to agree on a value which is "validated". The notion of validated will become clearer soon.

Our ACS. Recall that the main goal of ACS is to agree on a common core set of $n-t$ parties whose AVSS successfully terminated. Parties can asynchronously validate which parties can be considered: A party P_i validates P_j when the AVSS of P_j as a dealer terminates. When a party P_i validates a set of $n-t$ parties, it broadcasts its set S_i containing those $n-t$ parties; However, it continue to store and update S_i as more parties are being validated (i.e., their AVSS has terminated). The parties now run an instance of AVABA – and try to reach an agreement on the sets S_i; the main difference is that this set S_i is dynamic, and as P_i validates additional parties, say, some P_k, it allows the agreed output to contain such additional parties that were dynamically added – such as P_k. The AVABA protocol guarantees that all honest parties will reach an agreement on the set, and whoever appears in the output was validated by an honest party. See Sect. 8.

AVABA. The construction of an AVABA protocol follows the ideas and construction of the No Waitin' HotStuff (NWH) protocol of [6], and replaces the cryptographic validation function with an asynchronous validation notion, as described above. Seeing as the NWH protocol is designed in the authenticated setting and uses a signature scheme, our work adapts these ideas to the information-theoretic setting, removing the need for cryptography.

Our AVABA protocol proceeds in iterations called "views". Parties start each view by exchanging suggestions for possible outputs from the protocol. These values are either derived from messages they saw in previous views, or simply their inputs if no suitable previous values exist. Every party then chooses the suggestion from the most recent view, and broadcasts it as its proposal for the current view. After parties broadcast their proposals, one of them is chosen

retroactively and obliviously using a Verifiable Leader Election (VLE) protocol. Following that, parties check whether the proposal can be safely output from the protocol without contradicting values output by parties previously. If that is the case, they do so. We emphasize that when an honest leader is elected, this is always the case. Otherwise, they proceed to the next view, while ensuring other parties can proceed as well. We now elaborate on how the parties pick the leader.

Verifiable Leader Election. The first challenge is constructing a verifiable leader election (VLE) protocol. In ordinary leader election, the goal is for all parties to agree on the identity of an honest leader with some constant probability. In our setting, the chosen leader might be validated by some asynchronous validation process (specifically, in our AVABA, a party is validated once it broadcasted a proposal for an output). Our protocol is inspired by the synchronous leader election protocol of [27], its efficiency improvements [1], and the asynchronous authenticated proposal election protocol in the computational setting of [6].

The main idea of leader election is to assign to each party a random rank c_i, and then pick the party with the maximal rank as the leader. Each party cannot assign a random rank to itself, as corrupted parties will not choose their values uniformly at random. Instead, each party P_j contributes a sub-rank $c_{j\to i}$ to each P_i, and we define (for now) the rank of P_i to be $c_i = \sum_{j=1}^{n} c_{j\to i}$. We call each $c_{j\to i}$ the contribution of P_j to P_i. We cannot just let parties contribute random sub-ranks, as the corrupted parties will wait to see the sub-ranks that the honest parties contributed and then pick their own sub-ranks so that a corrupted leader will be elected. Instead, the parties first "commit" to the sub-ranks and later "reveal" them. The commitment is performed using AVSS.

We borrow ideas from [1,6] and instead of having $\mathcal{O}(n^2)$ AVSS instances (i.e., $c_{i\to j}$ for every i, j), we use $\mathcal{O}(n)$ "packed" AVSS instances in which each dealer can share $\mathcal{O}(n)$ secrets at once. This reduces the number of instances to $\mathcal{O}(n)$. As mentioned, we improve the cost of packed AVSS by a factor of $\mathcal{O}(n)$, leading to a more efficient VLE.

Verifiable Party Gather. Since the model is asynchronous, the above protocol suffers from a similar problem to our starting point: how can the parties know and agree on which AVSS instances terminated successfully and can be considered as contributions? Parties do not know whether to wait until a particular AVSS instance terminates, as it might never terminate. On the other hand, agreeing on which AVSS instances were terminated is exactly the ACS problem!

Luckily, we do not have to reach an agreement fully. We avoid strong agreement using two tools. First, we let each party P_i choose a set of $t+1$ dealers that have successfully shared secrets. The value c_i of P_i is defined to be the sum of their secrets. Since it is a sum of $t+1$ parties, it must include at least one honest dealer, which means that c_i is uniformly distributed. Parties then broadcast their choice of dealers, and wait to receive at least $n-t$ such broadcasts.

However, if some broadcasts are delayed, we again run into a similar problem to ACS: different parties might not consider the same set of parties as potential

leaders, and as such parties might not agree on the chosen leader. The parties have to agree on which broadcasts to consider. We now employ our second tool to "roughly agree" on which broadcasts were received: the verifiable party gather protocol. Verifiable party gather is a relaxation in which the parties might output distinct sets, say $C_1, \ldots, C_n$, but with the following two guarantees: (1) All parties in all sets have been validated by at least one honest party (which means that eventually, they will all be validated); (2) The different sets are distinct, but are all supersets of some large "core" of $n - t$ parties.

Since all of the c_i ranks are uniform, each party has the same probability of having the maximal value. If we are lucky and the party with the maximal random value is an honest party in the shared core, all parties will see its c_i rank and elect it as a leader. Luckily, since the core is large, there are many honest parties in the core, and this event happens with a large probability. The core is of size $n - t$ in our case, and thus it contains $n - 2t$ honest parties, which yields a success probability of $\frac{n-2t}{n} \geq \frac{1}{2}$. See more in Sect. 6.

Our verifiable party gather protocol is inspired by [6]. Unlike [6], which relies on signatures and authentication, we implement an information theoretic version of gather whose inputs comply with asynchronous validation. Moreover, our gather protocol is unique in that it outputs a set of parties, while their values are inferred via asynchronous validation. See more on the gather protocol in Sect. 4.

Verifiable Leader Election $\Longrightarrow$ AVABA. We now slightly elaborate on how we move from VLE to AVABA. Here the main challenge is working with asynchronously validated inputs and maintaining both safety (that all honest parties output the same value) and liveness (that the protocol terminates) over the different iterations (views). Our protocol is an information-theoretic adaptation of [6].

For safety, we use a common approach in authenticated protocols [18,30] of using lock certificates and adapt them to the information-theoretic setting. Some of these techniques are inspired by the approach of [8] that adapts the protocol of [30] to partial synchrony. Here we show how to obtain liveness under fully asynchronous network conditions.

For liveness in asynchrony, there are two major challenges. The first is guaranteeing that all honest parties will reach agreement on the leader's proposal if a unique honest leader is elected. For this, we use the key certificate approach of [7,30] and adapt it to the information-theoretic setting. The second, more challenging problem is guaranteeing that honest parties eventually proceed to the next iteration (view) if the current iteration does not lead to agreement. As in [6], we observe that there are two triggers to changing views (i.e., giving up on the current iteration/leader):

- The first is when two different honest parties decided on different leaders (failure of the VLE); or
- The second is when the leader sends a proposal whose key is lower than a lock held by some party (blame event): parties check whether the leader proposed

values that are "acceptable" based on their current state, and "blame" the leader publicly if that is not the case.

In [6] these two events can be verified cryptographically, so any honest party that observes this event simply forwards it to all parties. In our setting, we adapt validate the correctness of messages asynchronously instead of using cryptographic tools. Roughly speaking, when the VLE fails or a blame message is sent, parties record it and wait for it to be asynchronously validated. We refer the reader to Sect. 7 for further details.

Structure. Figure 1 shows the structure of our ACS protocol, including the different building blocks and showing their complexity.

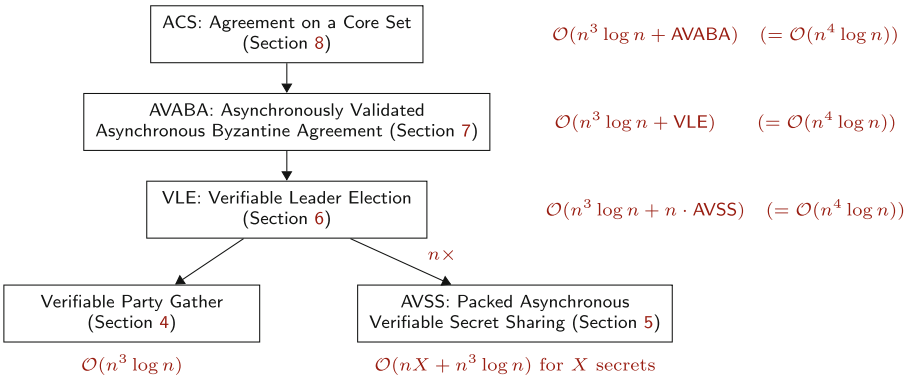


Fig. 1. The structure of our ACS protocol; In red: The communication complexity of each primitive.

2.3 Extensions

We conclude the overview and the preliminary part of the paper by discussing two extensions: The statistical settings (Sect. 2.3), and the ramifications of our ACS and AVSS to asynchronous MPC (Sect. 2.3).

The Statistical Settings. Our AVABA protocol requires $\mathcal{O}(n)$ secrets (each of size $\mathcal{O}(\log n)$ bits) to be shared per party per round and can be generalized to a protocol resilient to $t < n/3$ corruptions. Our ACS protocol uses packed AVSS to generate randomness.

As proven in [5,13], when $n < 4t$, any AVSS protocol must have some non-zero probability of non-termination. The work of [17] constructs such an AVSS protocol with an adjustable security parameter ϵ, allowing the protocol to fail or not terminate with ϵ probability. It is possible to use such an AVSS protocol in our construction, resulting in an AVABA protocol with a similar probability of non-termination, as described in the following:

Theorem 2.1 (General Asynchronously Validated Asynchronous Byzantine Agreement (informal)). *Let $c \in [3,4]$. Given a $n > ct$ resilient protocol for asynchronous verifiable secret sharing that has $S(n,\epsilon)$ communication complexity, $\epsilon \geq 0$ error, and $1-\epsilon$ probability of termination, there exists an agreement protocol that is resilient to t corruptions as long as $n > ct$. Moreover, the protocol is $\tilde{\mathcal{O}}(\epsilon)$ secure (and in particular for $\epsilon = 0$ is perfectly-secure). With probability $1-\tilde{\mathcal{O}}(\epsilon)$, the protocol runs in constant expected time (and in particular for $\epsilon = 0$ is almost-surely terminating) and has $\mathcal{O}(n^3 \log n + n^2 S(n,\epsilon))$ expected communication complexity.*

In the above theorem, setting $c = 3$, we get the first statistical ACS protocol for any $n > 3t$ parties that terminates in constant expected time, conditioned upon the success of the protocol.

Corollary 2.2. *There exists a statistically secure protocol for asynchronous agreement on a core set with asynchronous validation resilient to $t < n/3$ Byzantine corruptions. Conditioned upon the protocol succeeding with probability $1-\epsilon$, the protocol runs in constant expected time.*

It is also possible to directly construct our AVABA protocol using a single call to a verifiable leader election protocol per round instead. This means that any construction of such a protocol will immediately yield an AVABA protocol and consequently an ACS protocol as well. More precisely:

Theorem 2.3 (General Asynchronously Validated Asynchronous Byzantine Agreement (informal)). *Let $c \in [3,4]$. Given a $n > ct$ resilient protocol for verifiable leader election that has $LE(n,\epsilon)$ communication complexity, $\epsilon \geq 0$ error, and $1-\epsilon$ probability of termination, there exists an agreement protocol that is resilient to t corruptions as long as $n > ct$. Moreover, the protocol is $\tilde{\mathcal{O}}(\epsilon)$ secure (and in particular for $\epsilon = 0$ is perfectly-secure). With probability $1-\tilde{\mathcal{O}}(\epsilon)$, the protocol runs in constant expected time (and in particular for $\epsilon = 0$ is almost-surely terminating) and has $\mathcal{O}(n^3 \log n + LE(n,\epsilon))$ expected communication complexity.*

Asynchronous Secure Computation. Plugging our new AVSS and the ACS protocols in the recent asynchronous MPC protocol of [3] leads to an efficiency improvement. Specifically, instead of MPC with $\mathcal{O}((Cn + Dn^2 + n^7)\log n)$ communication and $\mathcal{O}(D + \log n)$ expected-time using the ACS of [15] and the AVSS of [19], we obtain $\mathcal{O}((Cn + Dn^2 + n^4)\log n)$ communication and $\mathcal{O}(D)$ expected-time.

The MPC protocol of [3] has the following structure:

Offline: Beaver Triplets Generation. The goal is to distribute (Shamir, univariate degree-t) shares of random secret values a, b, and c, such that $c = ab$. This is performed as follows:

1. **Triplets with and without a dealer.** Each party first distributes secrets a_i, b_i, c_i such that $c_i = a_i \cdot b_i$. If the computation requires C multiplications in

total, each dealer has to generate C/n such triplets. Using the previous best AVSS, this step requires $\mathcal{O}(n^4 \log n + C \log n)$ communication for each dealer, i.e., a total of $\mathcal{O}(n^5 \log n + Cn \log n)$ for all parties combined. Using our AVSS protocol, this step is automatically reduced to $\mathcal{O}(n^4 \log n + Cn \log n)$. Both protocols are constant expected number of rounds.

2. **Agreeing on a core set (ACS):** The parties then have to agree on a core set of parties whose beaver triplets generation terminated and will be considered in the sequel of the computation. The communication cost of the ACS from [15] is $\mathcal{O}(n^7 \log n)$ with $\mathcal{O}(\log n)$ rounds, which we reduce to $\mathcal{O}(n^4 \log n)$ and expected constant time.
3. **Triplets with no dealer:** Once agreed on the core, there is a way to extract $\mathcal{O}(n)$ triplets with no dealer (i.e., when no party knows the secrets a, b and c) from $\mathcal{O}(n)$ triplets with a dealer (where the dealer knows the secrets a, b and c). This step costs $\mathcal{O}(n^2 \log n + Cn \log n)$.

To conclude, generating C multiplication triplets costs a total of $\mathcal{O}(n^4 \log n + Cn \log n)$.

Online. The second step follows the standard structure where each party shares its input (using AVSS), and the parties evaluate the circuit gate-by-gate while consuming the multiplication triplets they have generated, using the method of [19]. Using our AVSS, the sharing phase is reduced from $\mathcal{O}(n^5 \log n)$ to $\mathcal{O}(n^4 \log n)$. The computation of the circuit using the multiplication triplets remains $\mathcal{O}((Cn + Dn^2) \log n)$ with an $\mathcal{O}(D)$ time requirement.

In total, using our ACS and AVSS, we obtain a protocol that requires $\mathcal{O}((Cn + Dn^2 + n^4) \log n)$ communication and $\mathcal{O}(D)$ time.

3 Definitions and Assumptions

3.1 Network and Threat Model

This work deals with protocols for n parties with point-to-point communication channels. The network is assumed to be asynchronous, which means that there is no bound on message delay, but all messages must arrive in a finite time. The protocols below are designed to be secure against a computationally unbounded malicious (Byzantine) adversary. The AVSS protocol is secure when the adversary controls $t < \frac{n}{4}$ parties, whereas the other protocols are secure even if the adversary controls $t < \frac{n}{3}$ parties (assuming an AVSS protocol). Furthermore, for simplicity, our modeling of the functionalities and the simulation proofs assume a static adversary, but we believe that our construction can be extended to the adaptive adversary. In the full version [4], we also provide proofs for property-based definitions that are secure against adaptive corruption.

3.2 Asynchronous Secure Computation and SUC

We model our protocols in the simplified universally composable setting (SUC), formalized by Canetti, Cohen, and Lindell [16], which implies UC security. We

briefly overview the model here, where this overview is taken almost verbatim from [3].

We consider an asynchronous network where the parties are $\{P_1, \ldots, P_n\}$. The parties are connected via pairwise ideal private channels. To model asynchrony, messages sent on a channel can be arbitrarily delayed, however, they are guaranteed to be eventually received after some finite number of activations of the adversary. In general, the order in which messages are received might be different from the order in which they were sent. Yet, to simplify notation and improve readability, we assume that the messages that a party receives from a channel are guaranteed to be delivered in the order they were sent. This can be achieved using standard techniques – counters, and acknowledgements, and so we just make this simplification assumption.

Main Difference from SUC. The SUC model allows the adversary to also drop messages, and the adversary is not limited to eventually delivering all messages. To model "eventual delivery" (which is the essence of the asynchronous model), we limit the capabilities of the adversary and quantify over adversaries that eventually transmit each message in the network (i.e., they do not drop messages). Formally, any message sent must be delivered after some finite number of activations of the adversary.

As in SUC, the parties are modeled as interactive Turing machines, with code tapes, input tapes, outputs tapes, incoming communication tapes, outgoing communication tape, random tape and work tape.

Communication. In each execution there is an *environment* $\mathcal{Z}$, an *adversary* $\mathcal{A}$, participating *parties* $P_1, \ldots, P_n$, and possibly an *ideal functionality* $\mathcal{F}$ and a *simulator* $\mathcal{S}$. The parties, adversary and ideal functionality are connected in a star configuration, where all communication is via an additional *router machine* that takes instructions from the adversary. That is, each entity has one outgoing channel to the router and one incoming channel. When P_i sends a message to P_j, it sends it to the router, and the message is stored by the router. The router delivers general information about the message to the adversary (i.e., "a header" but not the "content". That is, the adversary can know the type of the message and its size, but cannot see its content). When the adversary allows the delivery of the message, the router delivers the message to P_j. As mentioned, we quantify only over all adversaries that eventually deliver all messages. In particular, even in an execution with an ideal functionality, communication between the parties and this functionality is done via the router machine and is subject to (finite) delivery delays imposed by the adversary.

Note that the router machine is also part of the ideal model. When the functionality gives for instance, some output to party P_j, then this is performed via the router, and the simulator is notified. Thus, if the adversary, for instance, delays the delivery of the output of some party P_j, we do not explicitly mention that in the functionality (e.g., "wait to receive OK_j from the adversary and then deliver the output to P_j"), yet it is captured by the model.

Finally, the environment $\mathcal{Z}$ communicates with the adversary directly and not via the router. In particular, the environment can communicate with the

adversary (and it cannot communicate even with the ideal functionality $\mathcal{F}$). In addition, $\mathcal{Z}$ can *write* inputs to the honest parties' input tapes and can *read* their output tapes.

Execution in the Ideal Model. In the ideal model we consider an execution of the environment $\mathcal{Z}$, dummy parties $P_1, \ldots, P_n$, the router, a functionality $\mathcal{F}$ and a simulator $\mathcal{S}$. In the ideal model with a functionality $\mathcal{F}$ the parties follow a fixed ideal-model protocol. The execution is as follows:

1. The environment is first activated with some input z.
2. The environment delivers the inputs to the dummy honest parties, which forward the inputs to the functionality (recall that this is done via the router, which then gives some leakage about the message header to $\mathcal{S}$, which can adaptively delay the delivery by any finite amount). Moreover, $\mathcal{Z}$ can also give some initial inputs to the corrupted parties via $\mathcal{S}$.
3. At a later stage, where the dummy parties receive output from the functionality $\mathcal{F}$, they just write the outputs on their output tapes (and $\mathcal{Z}$ can read those outputs). Again, these messages go through the router, and the simulator can delay them.
4. At the end of the interaction, $\mathcal{Z}$ outputs some bit b.

The simulator $\mathcal{S}$ can send messages to $\mathcal{Z}$ and to the functionality $\mathcal{F}$. The simulator cannot directly communicate with the participating parties. We stress that in the ideal model, the simulator $\mathcal{S}$ interacts with $\mathcal{Z}$ in an online way, and the environment can essentially read the outputs of the honest parties and query the simulator (i.e., can ask to receive a simulated transcript of the adversary's view) at any point of the execution. We denote by $\mathsf{ideal}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(z)$ an execution of this ideal model of the functionality $\mathcal{F}$ with a simulator $\mathcal{S}$ and environment $\mathcal{Z}$, which starts with an input z.

Execution in the Real Model with Protocol π. In the real model, there is no ideal functionality and the participating parties are $\mathcal{Z}$, the parties $P_1, \ldots, P_n$, the router and the real-world adversary $\mathcal{A}$.

1. The environment is first activated with some input z, and it can give inputs to the honest parties, as well as some initial inputs to the corrupted parties controlled by the adversary $\mathcal{A}$.
2. The parties run the protocol π as specified, while the corrupted parties are controlled by $\mathcal{A}$. The environment can see at any point the outputs of the honest parties, and communicate directly with the adversary $\mathcal{A}$ (and see, without loss of generality, its partial view).
3. All messages go through the router and the adversary gets notified. The adversary can decide when to deliver each message.
4. At the end of the execution, the environment outputs some bit b.

We denote by $\mathsf{real}_{\pi,\mathcal{A},\mathcal{Z}}(z)$ an execution of this real model with the protocol π, the real-world adversary $\mathcal{A}$ and the environment $\mathcal{Z}$, which starts with some input z.

Definition 3.1. *We say that an adversary $\mathcal{A}$ is an* asynchronous adversary *if for any message that it receives from the router, it allows its delivery within some finite number of activations of $\mathcal{A}$.*

Definition 3.2. *Let π be a protocol and let $\mathcal{F}$ be an ideal functionality. We say that π securely computes $\mathcal{F}$ in the asynchronous setting if for every real-model asynchronous adversary $\mathcal{A}$ there exists an ideal-world adversary $\mathcal{S}$ that runs in polynomial time in $\mathcal{A}$'s running time, such that for every $\mathcal{Z}$:*

$$\{\mathsf{ideal}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(z)\}_z \equiv \{\mathsf{real}_{\pi,\mathcal{A},\mathcal{Z}}(z)\}_z$$

3.3 Cooperative Adversaries

Most of the functionalities discussed in this paper do not involve private inputs. Specifically, in many of these functionalities, the adversary is aware of all parties' inputs and outputs. The adversary's primary capability is influencing these outputs to some degree. Such functionalities often include a command like setOutput, allowing the adversary to set the output for certain honest parties under specific conditions verified by the functionality. This can lead to a scenario where the adversary might choose not to cooperate, failing to send any inputs and forcing the functionality to "get stuck".

To simplify matters, we model the adversary is always "cooperative". This is done for simplicity of exposition, to enhance readability and provide a more concise description of the functionalities. In this model, the adversary eventually responds and cooperates with the functionality.

We justify this modeling by noting that any functionality assuming a cooperative adversary can be transformed to handle a non-cooperative adversary. In such a case, the functionality would specify default outputs for honest parties. This involves the functionality calling the setOutput command that the ideal adversary would typically invoke. This message is routed to itself via the router, and the adversary is notified. The adversary can decide to rush and reply with its own setOutput command, or, it must eventually pass the first command to the functionality (recall that we assume eventual delivery), which then sends the output to the honest parties. In the case of two setOutput commands, the second one will be ignored. When describing a functionality that works for every adversary, not necessarily a cooperative one, the eventual output will be the one that it sent by the functionality to itself.

In other words, by modeling functionalities with cooperative adversary, we effectively "shift" some of the functionality's description to the simulator, resulting in more concise functionalities that precisely describe the adversary's capabilities. For instance, in F_{Gather} (Sect. 4), the adversary might choose different outputs for different parties, but the functionality ensures that all outputs share a significant common subset. We just define that the adversary is capable of choosing different outputs to different parties, without necessarily describing what the outputs are when the adversary does not cooperate. This means that a simulator can use its access to define outputs, or give up this right by being uncooperative and allow the functionality to define outputs instead.

3.4 Reliable Broadcast

We assume the existence of a Reliable Broadcast protocol. A *Reliable Broadcast* protocol is an asynchronous protocol with a designated *sender*. The sender has some *input value* M from some known domain $\mathcal{M}$ and each party may *output* a value in $\mathcal{M}$. A Reliable Broadcast protocol has the following properties assuming all honest parties participate in the protocol:

- **Agreement.** If two honest parties output some value, then it's the same value.
- **Validity.** If the dealer is honest, then every honest party that completes the protocol outputs the dealer's input value, M.
- **Termination.** If the dealer is honest, then all honest parties complete the protocol and output a value. Furthermore, if some honest party completes the protocol, every honest party completes the protocol.

Concretely, we use the reliable broadcast protocol of [9] in which parties send $\mathcal{O}(n^2 \log n + n \cdot m)$ bits when broadcasting a message of size $\mathcal{O}(m)$.

4 Verifiable Party Gather

In the leader election protocol, we wish to agree on a large set of parties that actively participated and elect a leader among them. However, since some instances might terminate earlier than others for some parties, exactly agreeing on the set is non-trivial and potentially expensive. We know, however, that if an instance has terminated for one honest party then it eventually terminates for all parties. The gather functionality comes to "synchronize" the parties. Exactly agreeing on all terminating instances is rather expensive (in fact, this is the end goal of "agreement on a core set" – of Sect. 8). At this point, we slightly relax our requirements, the parties might output different sets, but under the following condition:

> **CORE:** there exists some core C^* of size $n - t$ or greater such that the output of every honest party contains C^*.

The parties also give their outputs to one another, and whenever P_j receives the output of P_i it verifies that the output was computed correctly. As we will see, all the messages sent are public, and therefore the view of the different parties should be the same. The only difference is the different scheduling of messages. Thus, it should be possible to eventually verify an honestly generated output, at least eventually.

4.1 Property-Based Definition

To aid understanding, we first give some property based definition of this primitive. We later describe its functionality (Functionality 4.1). (Our proof refers to the functionality, the property-based definition is given just for completeness.)

- **Syntax:** Each party eventually P_j eventually sets its output to be $(\mathsf{output}, j, C_j)$; Moreover, it receives the sets of the other parties and verifies them (and $(\mathsf{output}, k, C_k)$ will also be part of its output for other parties P_k).
- **Core:** Once the first honest party outputs a value from the protocol, there exists a core set C^* such that $|C^*| \geq n - t$. If an honest party P_j outputs a set $(\mathsf{output}, j, C_j)$ then it holds that $C^* \subseteq C_j$.
- **Completeness:** For every honest P_j, if P_j outputs $(\mathsf{output}, j, C_j)$ then all honest parties would verify its set and have $(\mathsf{output}, j, C_j)$ as part of their output.
- **Agreement on verification:** For a corrupted P_i, if some honest party outputs $(\mathsf{output}, i, C_i)$, then all honest parties eventually output $(\mathsf{output}, i, C_i)$.
- **Validation of verification:** For a corrupted P_i, if some honest party P_j outputs $(\mathsf{output}, i, C_i)$, then it holds that $C^* \subseteq C_i$ and each element in C_i has been validated by P_j.

4.2 Gather Functionality

We now describe the gather functionality. As described in Sect. 3.3, we describe the functionality with a cooperative adversary - namely, we assume that it always replies to the functionality.

Functionality 4.1: Gather Functionality

The functionality is reactive. Initialize $C_1, \ldots, C_n = C^* = \emptyset$.

- $\mathsf{validate}(i, j)$: Whenever the functionality received this command from some party P_i (via the router), forward $(\mathsf{validate}, i, j)$ to the ideal adversary and record the message.
- $\mathsf{setCore}(C)$: Whenever the ideal adversary sends this command, verify that $|C| \geq n - t$ and that for every $x \in C$, there exists a recorded $\mathsf{validate}(\ell, x)$ for an honest P_ℓ. If so, store $C^* = C$; Otherwise, ignore the command.
- $\mathsf{setOutput}(i, C'_i)$: Whenever the ideal adversary sends this command, verify that $|C'_i| \geq n - t$, $C^* \subseteq C'_i$ and for every $x \in C'_i$, there exists a recorded $\mathsf{validate}(\ell, x)$ for an honest P_ℓ. If $C_i = \emptyset$ replace it with $C_i = C'_i$, and send $(\mathsf{output}, j, C_j)$ to all parties. Otherwise ($C_i \neq \emptyset$), ignore the command.

Input Assumption 4.2. *We prove that the protocol implements the functionality for* restricted *environments. In particular, we assume the following:*

1. *For every pair of honest parties (j, k), the environment issues* $\mathsf{validate}(j, k)$.
2. *If the functionality issues* $\mathsf{validate}(j, i)$ *from an honest P_j and corrupted P_i, for any other honest P_k, the environment will also issue* $\mathsf{validate}(k, i)$.

A construction of the Gather protocol, a proof of its correctness and a complexity analysis are provided in the full version of the paper.

5 Packed AVSS

5.1 The Functionality

The protocol we present implements a complete secret sharing, where all parties receive output. The dealer inputs a bivariate polynomial. If the polynomial is of the appropriate degree, all parties receive shares on that polynomial and the functionality can terminate. If the polynomial is not from the expected degree, the functionality does not terminate.

Functionality 5.1: Sharing phase of AVSS

1. The dealer sends the functionality a bivariate polynomial $S(X, Y)$.
2. The functionality verifies that $S(X, Y)$ is of degree at most $2t$ in X and degree at most t in Y. If not - the functionality does not terminate.
3. Otherwise, it sends all the shares $S(X, i), S(i, Y)$ for each corrupted party P_i to the ideal adversary. Moreover, it sends the shares $S(X, j), S(j, Y)$ to all honest parties $j \notin I$. Recall that the adversary has the ability to delay those messages.

5.2 Reconstruction

Functionality 5.2: Reconstruction Functionality

1. Upon receiving $(\mathsf{point}, i, k, p_i(-k))$ from all honest parties with the same index $k \in \{0, \ldots, -t\}$, forward the message to the ideal adversary .
2. After receiving these messages from $t + 1$ parties, reconstruct the unique degree-t univariate polynomial $q(Y)$ satisfying $q(j) = p_j(-k)$ for each honest P_j received.
3. Send $q(Y)$ to all parties.

Input Assumption 5.3. *It is assumed that all the shares of the honest parties lie on a unique degree-t univariate polynomial $g(Y)$.*

5.3 Putting It All Together

We now show a reactive functionality that combines the share and reconstruct phases.

Functionality 5.4: Reactive AVSS - F_{AVSS}

The functionality is parameterized by the identify of the dealer.

– $\mathsf{Share}(s_0, \ldots, s_t)$: When the dealer transmits this message to the functionality with input $(s_0, \ldots, s_t)$, forward Share to the ideal adversary, and record $(s_0, \ldots, s_t)$. Reply to all parties with the message shared.

– Reconstruct(k) with $k \in \{0, \ldots, -t\}$: Whenever this message is received from some party P_i, record that message. Once $t + 1$ honest parties sent Reconstruct(k), reply with (k, s_k) to all parties that sent that command and all following commands.

Protocols realizing these functionalities, proofs of their correctness and analysis of their complexity are provided in the full version of the paper.

6 Verifiable Leader Election

A perfect leader election would allow all parties to output one common randomly elected party. *Verifiable Leader Election* (VLE) is an asynchronous protocol that tries to capture this spirit but obtains weaker properties. Intuitively, there is only a constant probability that all parties elect the same honest party. In the remaining cases, the adversary can control the output and even cause different parties to have different outputs. However, even in these cases, all parties eventually output some value. We proceed to define the VLE functionality in Sect. 6.1.

6.1 The Functionality

Following Sect. 3.3, we define the functionality for cooperative adversaries. Once again, there is an external validity command that can be invoked, with the same input assumption as Input Assumption 4.2. The functionality samples a random rank for each party. Ideally, we would like all parties to consider all parties as possible candidates, and to choose the one with the highest rank as leader. As we will see, by delaying the delivery of messages, the adversary can make some parties not consider other parties as possible candidates. However, there is a large core of parties that is considered by all parties. After describing the functionality, we prove that this suffices in order to elect an agreed leader with a constant probability.

Functionality 6.1: F_{VLE} **– The Verifiable Leader Election Functionality**

1. validate(i, j): Whenever the command is received from some party P_i, forward (validate$, i, j$) to the ideal adversary and record the message.
2. setCore(C^*): Upon receiving this command from the ideal adversary, with $C^* \subseteq n$, and $|C^*| \geq n - t$, verify that for every $k \in C^*$ there exists an honest j such that validate(j, k) was recorded. Then, the functionality stores C^* and chooses a random rank to every party $r_1, \ldots, r_n$.
3. setCandidates(i, C_i): Upon receiving this command from the ideal adversary, verify that that $C^* \subseteq C_i$, and that for every $k \in C_i$ there exists an honest j such that validate(j, k) was recorded. The functionality sets $\ell_i = argmax\{r_k \mid k \in C_i\}$. Add (output$, i, \ell_i$) to all parties, and send $(r_k)_{k \in C_i}$ to the ideal adversary.
   It is assumed that the adversary sends setCandidates(j, C_j) for every honest party P_j, and it might also send such commands for some corrupted parties, according to its choice.

We assume the exact same input assumption as in Input Assumption 4.2.

Properties of the Ideal Functionality. We show that the functionality satisfies the following properties. In particular, we show that with a probability of at least 1/3, all honest parties agree on an honest leader, and the output of the corrupted parties (if it exists) also defines the same leader.

Claim 6.2. *The following properties hold:*

1. *With probability at least* $1/3$*, there exists an index of an honest party* ℓ^**, such that* $\ell_i = \ell^*$ *for all* i *for which* $(\mathsf{output}, i, \ell_i)$ *has been sent to all parties (no matter whether* P_i *is honest or corrupted). Furthermore,* ℓ^* *is defined at the time that the first* $(\mathsf{output}, i, \ell_i)$ *has been sent by the functionality.*
2. *For every* $(\mathsf{output}, i, \ell_i)$ *that has been transmitted by the functionality to all parties, there exists an honest* j *such that the command* $\mathsf{validate}(j, \ell_i)$ *has been received.*

Proof. From the properties of the functionality, at the first time the functionality sent $(\mathsf{output}, j, C_j)$, there exists a core C^* of at least $n-t$ indices, such that every $C_i \subseteq C^*$. Note that the ranks are chosen only after the core C^* is fixed. Moreover, the adversary cannot see the rank r_i of some party P_i, unless i is included in some C_j in some $\mathsf{setCandidates}(j, C_j)$ command.

Each party $i \in C^*$ has probability $1/n$ to have the maximal rank[3]. among all $[n]$, i.e., including all supersets of C^*. Since $|C^*| \geq n - t$, we have at least $n - 2t$ honest parties in C^*, and therefore the probability that some honest party ℓ^* in C^* has the maximal rank in $[n]$ is at least $(n - 2t)/n \geq 1/3$. Since each C_i must include C^*, all output messages have ℓ^* as the leader. This is true even though the adversary chooses C_i adaptively, as C_i is chosen after C^* and all the ranks are determined.

For the second part of the claim, whenever the functionality receives some $\mathsf{setCandidates}(i, C_i)$ message it verifies that all parties in C_i has been validated (i.e., for each $k \in C_i$, some honest P_j has sent $\mathsf{validate}(j, k)$). The chosen ℓ_i is some index in C_i, and therefore ℓ_i must have been validated.

The VLE protocol, a proof of its correctness and a complexity analysis can be found in the full version of the paper.

7 Asynchronously Validated Asynchronous Byzantine Agreement

This section deals with constructing our AVABA protocol, which is built upon ideas in [6] and [8] and adapts them to the asynchronous information-theoretic setting.

[3] We ignore a negligible probability of two parties having the same rank. This can be accounted for by sampling from a large enough $\mathbb{F}$ and noting that $\frac{n-2t}{n} \geq \frac{n}{3} + \frac{1}{n}$.

7.1 The Functionality

Assuming once again a cooperative adversary (see Sect. 3.3), the functionality is relatively simple to describe: There is an external validity command, where each party might validate some value $v \in \mathcal{V}$. At some point, the adversary decides on a value x and sends it to the functionality. If x has been validated by some honest party, then this value is accepted and is sent to all parties as output.

Functionality 7.1: F_{AVABA}

The functionality is parameterized with a domain $\mathcal{V}$, and support the following commands:

$\mathsf{validate}(i, x)$: Upon receiving this command from party P_i with a value $v \in \mathcal{V}$, forward the command to the adversary and store this command.

$\mathsf{setInput}(j, x_j)$ Upon receiving this command from an honest P_j with $x_j \in \mathcal{V}$, forward the command to the adversary and store this command.

$\mathsf{setOutput}(x)$: Upon receiving this command from the ideal adversary with $x \in \mathcal{V}$, verify that there exists an honest j such that $\mathsf{validate}(j, x)$ or $\mathsf{setInput}(j, x_j)$ is recorded, send (output, x) to all parties and terminate.

Input Assumption 7.2. *We prove that the protocol implements the functionality for restricted environments that follow the following assumptions:*

1. *If for some honest party P_j the environment issued* $\mathsf{setInput}(j, x_j)$ *or* $\mathsf{validate}(j, x_j)$ *then for every other honest party P_k a command* $\mathsf{setInput}(k, x_k)$ *or* $\mathsf{validate}(k, x_k)$ *will be issued.*

We remark that our protocol actually satisfies a stronger property, named "α-quality": With probability α, all parties output the input x_j of a party P_j that was honest when starting the protocol. This is not reflected in our functionality, and we will prove this property as a property-based definition. This property is unnecessary for achieving the final ACS in Sect. 8, and we prove it for completeness.

7.2 The Protocol

The protocol of [6] heavily relies on cryptographic primitives (signatures) to obtain externally valid outputs. Here we use asynchronous external validity instead. This requires redefining and adapting new information-theoretic variants of verifiable gather (party gather) and verifiable leader election. The protocol of [8] modifies the cryptographic protocol of [30] to the information-theoretic setting in partial synchrony. Here we show how to extend this to full asynchronous network conditions, which in turn requires a new information-theoretic view change protocol and consistency checks for sent values.

In the AVABA protocol, parties proceed in "views", which are just an iteration number (asynchronous "phases"). In each view, parties propose values to agree

upon and then try to choose an honest leader using the VLE protocol. Recall that the VLE protocol might fail: either by not agreeing on the chosen leader, or by electing a corrupt leader. However, once an honest leader is chosen, its value will be adopted and the parties will terminate.

AVABA uses the "Key-Lock-Commit" paradigm used in previous HotStuff protocols (VABA, IT-HS and NWH) to maintain safety and liveness. Given a value, a party first puts a "key" on it (this is just an indicator on the view number), then a "lock", and finally "commits" to it. Intuitively, locks help guarantee safety by forcing parties to ignore old values, and ensuring that parties can commit. Keys are used to ensure progress by convincing parties to accept proposed values in spite of their locks if no commitment was made. In more detail:

- **Commit:** When a party commits to a value, it knows that this value will be the output, and it pushes toward termination. It sends to other parties that it is ready to terminate.
- **Lock:** A lock consists of two values: lock, which is a view number, and lock_val which is the value seen when setting the lock (which is the potential output). A party might lock a value, but this lock can still be later removed. However, it indicates that this is a value that the parties should agree on, and it tells it to all other parties. Once it hears $n - t$ locks on the same value from other parties, it changes its status to "commit".
- **Key:** This indicates that a party is witness to a current value; If it hears enough "key" messages on the same value – it moves to "lock". Just like a lock, a key consists of two values: key, which is a view number, and key_val which is the suggested value.

Overview. The parties proceed in 5 (asynchronous) rounds in each view. The general idea is that parties first confirm that they all agree on the leader elected in the VLE protocol, set a key, set a lock to the elected leader's proposal, confirm that they are all locked, commit to the lock, and terminate.

If at any point they see that the VLE failed, they move onto a new view and announce that they are doing so. Recall that in VLE, eventually each party sees the output of the other parties, and therefore if two leaders are elected, then all parties will see that eventually.

In the NWH protocol, parties provided cryptographic proofs for their keys and locks in the form of signatures on `echo` and `key` messages respectively. These signatures are inherently transferable since they can be sent to any party who can verify those signatures on their own. We cannot use signatures. To allow the "transfer" of such proofs, parties broadcast their `echo` and `key` messages. This allows a party that formed a key or a lock to know that any other party will eventually hear the same `echo` and `key` messages and believe that it could have formed that key or lock. Similar techniques are employed when providing `blame` messages, which are used to inform parties of a failed VLE session.

In each view (iteration), the parties run two protocols in parallel:

1. viewChange: The parties send each other suggestions for the current view. Parties then compute their proposal for the current view. Every P_i broadcasts

this proposal, which is later used if it is chosen as a leader. Initially, this is the input of P_i, but later in the protocol, P_i might adopt some other value based on other parties' suggestions.
2. processMessages: This is the key-lock-commit mechanism. The leader is chosen, and the parties proceed to see if they can agree on the proposal of the leader.

Round 1: The first round in each view begins with a viewChange protocol. In viewChange parties choose their proposals and broadcast them. They send their current key to all other parties in a `suggest` message. Before accepting a key, parties make sure that it could have been achieved in the relevant view by waiting to receive the broadcasted messages required to form a key (`echo` messages to be explained later). Upon accepting $n-t$ keys, parties choose the key and value from the most recent view and broadcast the chosen key and value in a `proposal` message. Following that, they call the VLE protocol to choose a leader for the current view, while P_i validates P_j in VLE only if P_j has broadcasted a valid proposal. This guarantees that any chosen leader has broadcasted a proposal.

Round 2: In the second round, parties check whether the VLE was successful or not. If it was successful they continue in the view, but if it was not, they inform each other and proceed to the next view.

- Upon electing a leader using the VLE protocol, if the leader's proposed value is correct, i.e. it contains a key and value pair that could have been set in a view later than the current lock, then send an `echo` message to all other parties.
- If the leader's proposed value is "incorrect", send a `blame` message and proceed to the next view. In this context, by an "incorrect proposal" we mean that its key is not high enough to open the receiving party's current lock. Since a party puts a lock on a value based on public messages, every party can check that the purported lock could have been set in a later view by waiting to receive the same broadcasted `key` messages required to set that lock, and verify whether the `blame` message is valid (note, however, that we cannot verify that a particular `blame` message is false). Upon sending or receiving a valid `blame` message, reject the leader, and proceed to the next view.
- Since each party P_i can see the outputs of all other parties from the VLE protocol, it can verify if two different parties elected different leaders. In that case, the leader election fails, and the party proceeds to the next view.

Round 3: Parties proceed to this round if they have received many `echo` messages without seeing an error in the form of a `blame` or that of two different elected leaders. This also means that no other value was committed to in an earlier view, meaning that a key can be formed. Upon receiving $n-t$ `echo` messages, update the key and key_val fields before sending a `key` message to all parties.

Round 4: Upon receiving $n-t$ `key` messages, update the lock and lock_val fields before sending a `lock` message to all parties. Before setting a lock, every party

makes sure that at least $t+1$ honest parties set their keys to the current value. By doing that, every party guarantees that when choosing which value and key to input to the VLE protocol, all honest parties will hear of the current value and will be capable of opening any older lock an honest party might have.

Round 5: Finally, upon receiving $n-t$ correct `lock` messages, parties send `commit` messages with the same value. Before committing to a value, every party makes sure that at least $t+1$ honest parties have set their lock in the current view. These parties will not echo any message about any other value in subsequent views unless an adequate key is provided. Since forming a key requires a message from one of those parties, we can reason inductively that no correct key will be formed for a differing value in any subsequent view.

Output: In order to allow parties to terminate, a termination gadget is also run outside of any specific view. Similarly to Bracha broadcast [14], every party echoes a `commit` message if it sees $t+1$ such messages with the same value. Finally, parties terminate after seeing $n-t$ such messages.

8 Agreement on a Core Set (ACS)

We now turn to the main functionality: agreement on a core set, which is a simple corollary of Sect. 7. Recall the main application for MPC: Each party secret shares its input. If the dealer is honest, it is guaranteed that the sharing phase will terminate and all honest parties will receive their shares. If the dealer is corrupted, and the sharing phase terminated for one honest party, then it will eventually terminate for all other honest parties. The goal of ACS is to agree on a set of parties whose sharing phase has terminated.

The input assumption is the same as in Assumption 4.2: All honest parties validate each other, and if some honest party validates a corrupted party P_i, then eventually all honest parties will validate P_i. The functionality has a $\mathsf{validate}(i,j)$ command ("P_i sees that the sharing phase of P_j has terminated"). The adversary eventually chooses a set C of $n-t$ parties, and all parties receive C as output. The guarantee is that for every $k \in C$, there is some honest party that validated k. Again, we assume a cooperative adversary; otherwise, the functionality is slightly more complicated. In that case, core set C is set to be the indices k for which there is some honest party that validated k, and the parties receive that output when $n-t$ honest parties validated $n-t$ parties each. We now proceed to the definition of the functionality and the protocol.

Functionality 8.1: $\mathcal{F}_{\mathsf{ACS}}$ – Agreement on a Core Set Functionality

- $\mathsf{validate}(i,j)$: Whenever this functionality receives this command from some party P_i (via the router), forward the command to the ideal adversary and record the command.
- $\mathsf{setOutput}(C)$: Whenever the ideal adversary sends this command, verify that $|C| \geq n-t$ and that for every $x \in C$, there exists a recorded $\mathsf{validate}(\ell, x)$ for an honest P_ℓ. Send (output, C) to all parties and terminate.

A construction of the protocol, proofs of its correctness and a complexity analysis are provided in the full version of the paper.

Acknowledgements. We would like to thank Victor Shoup for valuable discussions and feedback. G. Asharov was supported by the Israel Science Foundation (grant No. 2439/20), and by JPM Faculty Research Award. G. Stern was supported in part by ISF 2338/23, AFOSR Award FA9550-23-1-0387, AFOSR Award FA9550-23-1-0312, and an Algorand Foundation grant. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the United States Government, AFOSR or the Algorand Foundation.

References

1. Abraham, I., Asharov, G., Patil, S., Patra, A.: Asymptotically free broadcast in constant expected time via packed VSS. In: Kiltz, E., Vaikuntanathan, V. (eds.) Theory of Cryptography, TCC 2022, Part I. LNCS, vol. 13747, pp. 384–414. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-22318-1_14
2. Abraham, I., Asharov, G., Patil, S., Patra, A.: Detect, pack and batch: perfectly-secure MPC with linear communication and constant expected time. In: Hazay, C., Stam, M. (eds.) EUROCRYPT 2023, Part II. LNCS, vol. 14005, pp. 251–281. Springer, Cham (2023). https://doi.org/10.1007/978-3-031-30617-4_9
3. Abraham, I., Asharov, G., Patil, S., Patra, A.: Perfect asynchronous MPC with linear communication overhead. In: Joye, M., Leander, G. (eds.) EUROCRYPT 2024, Part V. LNCS, vol. 14655, pp. 280–309. Springer, Cham (2024). https://doi.org/10.1007/978-3-031-58740-5_10
4. Abraham, I., Asharov, G., Patra, A., Stern, G.: Asynchronous agreement on a core set in constant expected time and more efficient asynchronous VSS and MPC. Cryptology ePrint Archive, Paper 2023/1130 (2023). https://eprint.iacr.org/2023/1130
5. Abraham, I., Dolev, D., Stern, G.: Revisiting asynchronous fault tolerant computation with optimal resilience. Distrib. Comput. **35**(4), 333–355 (2022)
6. Abraham, I., Jovanovic, P., Maller, M., Meiklejohn, S., Stern, G., Tomescu, A.: Reaching consensus for asynchronous distributed key generation. In: Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing, pp. 363–373 (2021)
7. Abraham, I., Malkhi, D., Spiegelman, A.: Asymptotically optimal validated asynchronous byzantine agreement. In: Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing, pp. 337–346. ACM, New York, July 2019. https://doi.org/10.1145/3293611.3331612
8. Abraham, I., Stern, G.: Information theoretic hotstuff. In: OPODIS. LIPIcs, vol. 184, pp. 11:1–11:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Dagstuhl, Germany (2020)
9. Alhaddad, N., Das, S., Duan, S., Ren, L., Varia, M., Xiang, Z., Zhang, H.: Balanced byzantine reliable broadcast with near-optimal communication and improved computation. In: Proceedings of the 2022 ACM Symposium on Principles of Distributed Computing, PODC 2022, pp. 399–417. Association for Computing Machinery, New York. (2022). https://doi.org/10.1145/3519270.3538475
10. Bangalore, L., Choudhury, A., Patra, A.: Almost-surely terminating asynchronous byzantine agreement revisited. In: Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing, pp. 295–304 (2018)

11. Ben-Or, M., Canetti, R., Goldreich, O.: Asynchronous secure computation. In: Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing, STOC 1993, pp. 52–61. Association for Computing Machinery, New York (1993). https://doi.org/10.1145/167088.167109
12. Ben-Or, M., El-Yaniv, R.: Resilient-optimal interactive consistency in constant time. Distrib. Comput. **16**(4), 249–262 (2003). https://doi.org/10.1007/s00446-002-0083-3
13. Ben-Or, M., Kelmer, B., Rabin, T.: Asynchronous secure computations with optimal resilience (extended abstract). In: Proceedings of the Thirteenth Annual ACM Symposium on Principles of Distributed Computing, PODC 1994, pp. 183–192. Association for Computing Machinery, New York (1994). https://doi.org/10.1145/197917.198088
14. Bracha, G.: Asynchronous byzantine agreement protocols. Inf. Comput. **75**(2), 130–143 (1987)
15. Canetti, R.: Studies in secure multiparty computation and applications. Ph.D. thesis, Citeseer (1996)
16. Canetti, R., Cohen, A., Lindell, Y.: A simpler variant of universally composable security for standard multiparty computation. In: Gennaro, R., Robshaw, M. (eds.) CRYPTO 2015, Part II. LNCS, vol. 9216, pp. 3–22. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48000-7_1
17. Canetti, R., Rabin, T.: Fast asynchronous byzantine agreement with optimal resilience. In: Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing, STOC 1993, pp. 42–51. Association for Computing Machinery, New York (1993). https://doi.org/10.1145/167088.167105
18. Castro, M., Liskov, B.: Practical byzantine fault tolerance. In: Seltzer, M.I., Leach, P.J. (eds.) Proceedings of the Third USENIX Symposium on Operating Systems Design and Implementation (OSDI), New Orleans, Louisiana, USA, 22–25 February 1999, pp. 173–186. USENIX Association (1999). https://dl.acm.org/citation.cfm?id=296824
19. Choudhury, A., Patra, A.: An efficient framework for unconditionally secure multiparty computation. IEEE Trans. Inf. Theory (2016)
20. Choudhury, A., Patra, A.: On the communication efficiency of statistically-secure asynchronous MPC with optimal resilience. Cryptology ePrint Archive, Paper 2022/913 (2022). https://eprint.iacr.org/2022/913
21. Cohen, R., Forghani, P., Garay, J., Patel, R., Zikas, V.: Concurrent asynchronous byzantine agreement in expected-constant rounds, revisited. Cryptology ePrint Archive (2023)
22. Das, S., Duan, S., Liu, S., Momose, A., Ren, L., Shoup, V.: Asynchronous consensus without trusted setup or public-key cryptography. Cryptology ePrint Archive, Paper 2024/677 (2024). https://doi.org/10.1145/3658644.3670327. https://eprint.iacr.org/2024/677
23. Duan, S., Wang, X., Zhang, H.: Practical signature-free asynchronous common subset in constant time. IACR Cryptol. ePrint Arch. p. 154 (2023). https://eprint.iacr.org/2023/154
24. Feldman, P.N.: Optimal algorithms for Byzantine agreement. Ph.D. thesis, Massachusetts Institute of Technology (1988)
25. Fischer, M.J., Lynch, N.A., Paterson, M.S.: Impossibility of distributed consensus with one faulty process. J. ACM **32**(2), 374–382 (1985). https://doi.org/10.1145/3149.214121

26. Goyal, V., Liu, Y., Song, Y.: Communication-efficient unconditional MPC with guaranteed output delivery. In: Boldyreva, A., Micciancio, D. (eds.) CRYPTO 2019, Part II. LNCS, vol. 11693, pp. 85–114. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-26951-7_4
27. Katz, J., Koo, C.Y.: On expected constant-round protocols for byzantine agreement. J. Comput. Syst. Sci. **75**(2), 91–112 (2009)
28. Nielsen, J.B.: MPC techniques series, part 4: Beaver's trick (2021). https://medium.com/partisia-blockchain/beavers-trick-e275e79839cc
29. Patra, A., Choudhury, A., Rangan, C.P.: Efficient asynchronous verifiable secret sharing and multiparty computation. J. Cryptol. **28**(1), 49–109 (2015). https://doi.org/10.1007/s00145-013-9172-7
30. Yin, M., Malkhi, D., Reiter, M.K., Gueta, G.G., Abraham, I.: HotStuff: BFT consensus with linearity and responsiveness. In: Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing, PODC 2019, pp. 347–356. Association for Computing Machinery, New York (2019).https://doi.org/10.1145/3293611.3331591

Secret Sharing

Distributing Keys and Random Secrets with Constant Complexity

Benny Applebaum[1(✉)] and Benny Pinkas[2]

[1] Tel Aviv University, Tel Aviv, Israel
bennyap@post.tau.ac.il
[2] Aptos Labs and Bar Ilan University, Ramat Gan, Israel
benny@pinkas.net

Abstract. In the *Distributed Secret Sharing Generation* (DSG) problem n parties wish to obliviously sample a secret-sharing of a random value s taken from some finite field, without letting any of the parties learn s. *Distributed Key Generation* (DKG) is a closely related variant of the problem in which, in addition to their private shares, the parties also generate a public "commitment" g^s to the secret. Both DSG and DKG are central primitives in the domain of secure multiparty computation and threshold cryptography.

In this paper, we study the communication complexity of DSG and DKG. Motivated by large-scale cryptocurrency and blockchain applications, we ask whether it is possible to obtain protocols in which the communication per party is a constant that does not grow with the number of parties. We answer this question to the affirmative in a model where broadcast communication is implemented via a public bulletin board (e.g., a ledger). Specifically, we present a constant-round DSG/DKG protocol in which the number of bits that each party sends/receives from the public bulletin board is a constant that depends only on the security parameter and the field size but does not grow with the number of parties n. In contrast, in all existing solutions at least some of the parties send $\Omega(n)$ bits.

Our protocol works in the near-threshold setting. Given arbitrary privacy/correctness parameters $0 < \tau_p < \tau_c < 1$, the protocol tolerates up to $\tau_p n$ actively corrupted parties and delivers shares of a random secret according to some $\tau_p n$-private $\tau_c n$-correct secret sharing scheme, such that the adversary cannot bias the secret or learn anything about it. The protocol is based on non-interactive zero-knowledge proofs, non-interactive commitments and a novel secret-sharing scheme with special robustness properties that is based on Low-Density Parity-Check codes. As a secondary contribution, we extend the formal MPC-based treatment of DKG/DSG, and study new aspects of Affine Secret Sharing Schemes.

Keywords: Secret Sharing · Distributed Key Generation · Secure Computation

B. A. is supported by ISF grant no. 2805/21 and by the European Union (ERC-2022-ADG) under grant agreement no.101097959 NFITSC.

E. Boyle and M. Mahmoody (Eds.): TCC 2024, LNCS 15367, pp. 485–516, 2025.
https://doi.org/10.1007/978-3-031-78023-3_16

1 Introduction

Consider the following secure multiparty computation problem: n parties wish to obliviously sample a secret sharing of a random value s taken from some finite field $\mathbb{F}$ without letting any of the parties learn the value of s. Roughly speaking, given a privacy threshold and a correctness threshold $t_{\mathsf{p}} < t_{\mathsf{c}}$, the protocol must ensure that any adversary $\mathcal{A}$ that actively (aka maliciously) corrupts up to t_{p} parties learns nothing about the secret s and cannot bias it, and that any set of t_{c} honest parties can recover the secret even in the presence of $\mathcal{A}$.

This task, hereafter referred to as *Distributed Secret Sharing Generation* (DSG), can be viewed as a natural extension of Verifiable Secret Sharing (VSS), with the difference being that in DSG the secret is obliviously sampled, whereas in VSS it is chosen by a designated dealer. DSG plays an important role in many secure multiparty computation protocols (MPC), especially in the online/offline setting. It is also closely related to the problem of *Distributed Key Generation* (DKG), in which, at the end of the sharing phase, the protocol publishes a commitment to the secret (e.g., the value g^s where g is a generator of a cyclic group of appropriate order).[1] (See, e.g. [8,10,16,18,19,21,23,26,29,33,38].) DKG protocols are typically employed to distribute a private key s and publish a corresponding public key for a threshold signature scheme or cryptosystem (e.g., ElGamal, ECDSA, Schnorr, and BLS). The rise of digital currencies and proof-of-stake blockchains have lead to the deployment of DSG protocols for large scale systems with hundreds and even thousands of users [12,22]. As a result, an extensive body of research is currently devoted to the study of DSG protocols and their complexity. There are also many real-world implementations of DKG, e.g. [4,12–14,32,35,36,39].

The communication Complexity of DSG and DKG. In this work we study the communication complexity of DSG and DKG.Motivated by recent applications, we assume that the vast majority of the communication is performed via a public ledger. That is, in the distribution protocol parties write messages on a public bulletin board (BB) and read messages from the board, but cannot erase anything. The communication to the BB is non-anonymous and authenticated (e.g., via digital signatures and PKI). We also allow parties to send messages via private authenticated channels though such channels can be emulated over the public board assuming public-key encryption. Qualitatively, the BB can be simply viewed as an implementation of a broadcast channel. However it allows for a refined communication complexity measure. Specifically, we define the *upstream complexity* of a party as the number of bits that the party sends either to the BB or via private channels, the *downstream complexity* as the number of bits that a party reads from the BB or receives via private channels, and refer to the sum

[1] There are several definitions and variants of DSG and DKG. In particular, sometimes the protocol is required to generate public commitments to the private shares, and, typically, one should be able to securely recover the secret in the exponent of a given group element h. To simplify the exposition, we postpone the formal definition, but mention for now that our protocols support these features.

of the upstream complexity and downstream complexity as the *communication complexity* of the party. The *total communication complexity* is the sum of the communication complexity of all parties. For example, if Alice publishes 10 bits on the BB and Bob reads only the first 2 bits then Alice's communication complexity is 10 but Bob's communication complexity is only 2. When measuring the communication complexity of DSG and DKG we will focus on the distribution phase and ignore the communication complexity of the task of recovering the secret. Indeed, in all standard protocols (including the ones in this paper) one can release the secret s (or a committed version of it) to a receiver R via a single-round protocol with a linear communication complexity. If only a few parties need to recover the secret, the reduction in the communication is effective. For example, in the setting of threshold signatures, where a single client receives a signature on a document, only the client has to read $O(n)$ symbols from the BB whereas the n servers only have constant communication.

Existing Solutions. The most common approach is to reduce the DSG problem to n parallel calls to verifiable secret sharing (VSS), where in each call a different party P_j deals shares of a random secret according to some linear secret sharing scheme (e.g., Shamir [37]), and where the final shares are defined by locally summing-up the received shares. One can optimize the protocol a little bit by using only $t_{\mathsf{p}} + 1$ dealers. Assuming that $t_{\mathsf{p}}, t_{\mathsf{c}}$ are both linear in n (which will be taken to be our default setting), the total communication complexity of this protocol is about $\Omega(n^2)$ field elements since $\Omega(n)$ parties must each communicate at least $\Omega(n)$ field elements (even if the protocol is only passively secure). General MPC solutions for computing the randomized sharing functionality also lead to a similar communication cost. One can slightly improve the communication by running the protocol over a small super-logarithmic size sub-committee that is chosen at random via collective random coin-tossing. Ignoring the cost of the coin-tossing protocol, this reduces the total communication to $\omega(n \log n)$ field elements although the communication overhead is unbalanced and some parties still have a communication complexity of $\Theta(n)$.[2] Finally, we mention that if many instances of DSG are needed then one can amortize the communication cost to $O(1)$ per instance (e.g., via the use of hyper-invertible matrices [5]). This approach is typically useful for MPC applications, but is less useful in the DKG setting when the protocol is being used to set-up a single private key. Our main goal is to understand whether a constant cost can be established in a non-amortized setting. That is, we ask:

> How does the communication complexity of DSG and DKG protocols scale with the number of parties n? Specifically, is it possible to design a protocol

[2] In the standard model where there is no external source of randomness (e.g., random beacon), coin-tossing protocols suffer from a quadratic downstream communication cost (since $\Omega(n)$ parties contribute randomness to the process and each party has to read these contributions). Also, in the standard model, this solution is restricted to non-adaptive adversaries that select the corrupted parties *before* the committee is established.

where the communication complexity of each party is a constant that is independent of n?

The Information Bottleneck. All existing protocols suffer from the following "information bottleneck" which affects their communication overhead: There are some (typically, $\Omega(n)$) parties whose input influences the outputs of $\Omega(n)$ parties. However, in a constant-round protocol with a constant communication overhead per party each party can only affect the output of a constant number of parties.

2 Our Results

We show that n parties can obliviously sample a secret sharing for a random secret with constant communication complexity per party. Formally, we prove the following theorem.

Theorem 1 (main Theorem). *Assuming the existence of NIZKs the following holds. For any constants $0 < \tau_{\mathsf{p}} < \tau_{\mathsf{c}} < 1$ and every field $\mathbb{F}$ of size super-polynomial in the number of parties, there exists a constant-round n-party DSG (resp., DKG) protocol over $\mathbb{F}$ with privacy threshold of $\tau_{\mathsf{p}} n$ and recovery threshold of $\tau_{\mathsf{c}} n$ such that each party sends and receives only a constant number of field elements and a constant number of commitments and NIZK proofs for constant-size relations. Moreover, each party computes only a constant number of arithmetic operations and cryptographic operations.*

In contrast, existing protocols fail to achieve constant communication even in relaxed models, e.g., if the downstream complexity is ignored and broadcasting a bit is counted as a unit cost operation or if the upstream complexity is ignored and only downstream complexity is counted. We proceed with some comments.

1. (About the thresholds) The protocol guarantees that even if the adversary actively corrupts up to $t_{\mathsf{p}} = \tau_{\mathsf{p}} n$ parties, at the end of the protocol the (honest) parties hold shares of a random secret according to t_{p}-private t_{c}-correct secret-sharing scheme for $t_{\mathsf{c}} = \tau_{\mathsf{c}} n$. For example, we can assume that every uncorrupted party will be participating in the reconstruction procedure and so we can take $t_{\mathsf{p}} = n - t_{\mathsf{c}}$ (e.g., $t_{\mathsf{p}} = n/3$ and $t_{\mathsf{c}} = 2n/3$). We note that the theorem is still meaningful even when $t_{\mathsf{p}} + t_{\mathsf{c}} \neq n$. (For example, to support the case of some honest parties being offline and not participating in the reconstruction of the secret, the parameters can be set such that $t_{\mathsf{p}} + t_{\mathsf{c}} < n$, e.g., $t_{\mathsf{p}} = n/3$ and $t_{\mathsf{c}} = 0.5n$ to support up to $n/6$ honest parties being offline. As another example, to support the case of parties that are passively dishonest and thus leak their shares while still participating in the reconstruction, the parameters can set such that $t_{\mathsf{p}} + t_{\mathsf{c}} > n$, e.g., $t_{\mathsf{p}} = 0.6n$ and $t_{\mathsf{c}} = 0.7n$.)[3] It should be emphasized that, unlike Shamir-based schemes, we do not get an exact threshold of $t_{\mathsf{c}} = t_{\mathsf{p}} + 1$, rather we only get *near-threshold* secret sharing.

[3] Formally, we capture this property via the use of a *mixed* adversary [15] that applies different types of corruptions. See Sect. 5.

2. (The field size) The limitation on the field size being super-polynomial in n can be completely waived at the expense of allowing a large constant gap between the parameters τ_{c} and τ_{p}. That is, for any finite field (including the binary field) the theorem can be proved for some constants $\tau_{\mathsf{c}} < \tau_{\mathsf{p}}$. (See Remark 6.) We focus on large fields since this is the natural setting for DKG applications (e.g., when the secret is taken to be the private key of a DLOG-based system).
3. (Formalizing security) Despite the popularity of DKG, there is no single canonical definition for its security. Building on Katz [27] and Gennaro et al. [19], we formalize security via an MPC framework by presenting abstract DSG/DKG functionalities that are independent of any concrete secret sharing scheme (similarly to the abstract definition of a commitment functionality). We assume that the network is synchronous and consider computationally-bounded, rushing, non-adaptive adversaries. Simple variants of our protocol achieve adaptive security assuming secure erasures and perfect commitments. Our simulators are straight-line black-box, so given UC-secure building blocks the protocol can be proved to be UC-secure.
4. (Round complexity) The number of rounds is a constant that grows linearly with the privacy-to-correctness gap $\tau_{\mathsf{c}} - \tau_{\mathsf{p}}$. We can optimize the round complexity and get 3 rounds if we allow a larger correctness-to-privacy gap (i.e., settle on some universal constants $\tau_{\mathsf{p}} < \tau_{\mathsf{c}}$). For DSG, we can even reduce the round complexity to 2, assuming a public-key infrastructure. This two-round solution can be also applied to DKG at the expense of a slight relaxation of the functionality (see the full version [3]).
5. (Concrete communication) Using DLOG-based primitives (e.g., ElGamal commitments and RO-based NIZK), each party communicates a constant number of elements from $\mathbb{F}$ and the underlying group $\mathbb{G}$, where the constant is determined by information-theoretic objects (the sparsity of some low-density parity-check matrices). A concrete instantiation is described in the full version [3]. It therefore seems likely that one can get competitive practical results, at least for some range of parameters. We leave this direction for future work.
6. (The cost of recovering of the secret) Our DSG protocol generates $O(n)$ public elements on the BB. To recover the secret (either directly or taken to the power of some public group element h) one has to read these values from the BB and receive $O(1)$ values from each of at least t_{c} honest parties. Other existing protocols typically suffer from a similar cost as they have to read some certificates for the validity of the submitted shares (e.g., commitments). However, unlike other protocols, in our setting such access to the BB is necessary even if the adversary remains silent and only valid honestly-generated shares are being used. Put differently, our protocol suffers from the non-standard caveat that the local shares of authorized coalitions of size t_{c} have no information about the secret, and recovery is possible only when these shares are accompanied by the public values that are published on the BB. Similarly, in the context of DKG, computing the "public key" g^s requires reading $\Omega(n)$ public values that are available on the BB. Of course, such a computation can

be done once and for all, and can be verified later by anyone based on public values. So in terms of usability, this property does not seem very limiting. Interestingly, it turns out that this non-standard property is *necessary* for bypassing the aforementioned *information bottleneck*, as we prove in the full version [3, Appendix A].

2.1 Technical Overview

To prove Theorem 1, we will try to design a special-purpose secret-sharing scheme (SS) that natively supports distributed sampling. This requirement is satisfied if the shares are independently distributed. On the other hand, the correctness requirement implies that shares must be highly correlated. To bypass this problem, we design a scheme in which the shares are sampled independently at random and the correlation is achieved by publishing global public information that depends on all the shares. For example, think of the following variant of Shamir's scheme where each party i locally samples a random field element y_i as its share. We can think of these shares as defining a polynomial f of degree $n-1$ for which $f(i) = y_i, \forall i \in [n]$. To add redundancy, the parties securely evaluate the polynomial f in additional $m = n - t$ points $n+1, ..., n+m$ and publish the resulting vector $y = (y_{n+1}, \ldots, y_{n+m})$ on the BB as a "public header". Given this information, every set of t parties can recover the secret $f(0)$. Now, our goal is to securely compute a function that takes a single field element from each party and publishes $O(n)$ field elements on the BB. At least in terms of information (ignoring secrecy), this may be doable by using $O(1)$ communication per party. While it is not clear how to do it securely, at least we do not face the previous information bottleneck.

The AFS Abstraction. Let us abstract the above idea. Our goal is to design a secret-sharing with public header y that is available to all parties such that (1) a random sharing $x = (x_1, \ldots, x_n)$ can be sampled by letting each party sample her share x_i uniformly at random, and (2) the header y can be securely computed based on x. We note that any linear secret sharing (LSS) Σ can be brought to this form. To see this, observe that a random sharing $x = (x_1, \ldots, x_n) \in \mathbb{F}^n$ according to Σ is a random vector in the Kernel of some $m \times n$ "constraint" matrix M (i.e. $M \cdot x = 0$), and the secret s associated with x can always be written as some linear combination $v \in \mathbb{F}^n$ of the shares, i.e., $s = \sum_i v_i x_i$. Consider a new secret sharing scheme in which $x = (x_1, \ldots, x_n)$ are sampled uniformly at random, rather than be sampled subject to $M \cdot x = 0$, and where the public header $y \in \mathbb{F}^m$ is taken to be the "syndrome" $M \cdot x$. It is not hard to see that a set T of parties can reconstruct the secret in the new scheme (given y) if and only if it can reconstruct the secret in the original scheme Σ. Thus, any LSS, specified by M and v, gives rise to an *affine secret sharing scheme* (AFS) with similar privacy and correctness thresholds. More generally, the scheme remains secure for any fixing of the public header y when the vector of shares x is sampled uniformly subject to $Mx = y$. (See Sect. 4 for formal definitions and statements.)

Computing the Public Header Efficiently. Our goal now is to find an MPC-friendly AFS such that the mapping $\mathcal{F} : x \mapsto y$ given by $y = Mx$ can be securely computed with low communication. (The output of $\mathcal{F}$ should also consist of commitments to the private shares x_i, but let us ignore this for simplicity.) The functionality $\mathcal{F}$ takes a single field element from each party and publishes $m = \Omega(n)$ elements, and generic protocols for this task consume $\Omega(n^2)$ communication even in the presence of a broadcast channel. To cope with this problem, we employ a concrete secret-sharing scheme in which the constraint matrix M is *sparse*, i.e., each row and column have only a constant number of ones. Such a scheme was recently suggested by [2] based on Low-Density-Parity-Check Codes (LDPC). We extend their construction and show that such secret-sharing schemes can achieve near-threshold parameters.[4] Since the matrix is sparse, each output y_i depends on a constant number of inputs, and each party i affects a constant number of outputs. In the passive (aka semi-honest) setting, this immediately leads to a highly efficient protocol for computing the public header y. For instance, to compute an output $y_i = x_1 + \ldots + x_d$, the d relevant parties collectively generate an additive sharing of zero, with shares $r_1, \ldots, r_d$ given to the d parties, and post to the BB the values $x_i + r_i$ that sum-up to y_i. (To generate a sharing of zero we let each relevant party additively share the value zero and take the sum of these shares.)

One can handle active (malicious) adversaries by applying the GMW compiler (or cheaper variants of it). That is, we let the parties publicly commit to their inputs and randomness, send private messages publicly via the use of public-key encryption, and use NIZK to prove the consistency of their messages with the committed values and the previous rounds. The communication per party remains constant. One may worry that the adversary chooses its shares in a non-uniform way, however, it is not hard to show that such an attack does not violate the security of the secret. (The adversary still has no control or knowledge about the secret.) A more serious problem arises when the adversary *aborts* some of the outputs. Indeed, if a corrupt party aborts then it is impossible to compute any output y_i that depends on the input of that party.

Handling Aborts. Assuming an honest majority, a naive solution for aborts is to force parties to share their inputs at the beginning of the protocol, and later when a party aborts have the other parties reveal the corresponding input. This solution has a linear communication cost per party and is therefore not applicable in our context. Alternatively, since the aborts in our case are *identifiable* [25] (i.e., we can identify a corrupted party that misbehaves) we can *repeat* the computation for an aborted output y_i without the corrupted parties. It is possible to implement this solution with low communication. However, it can be shown that the adversary can force a linear number of rounds by corrupting only a single

[4] Along the way, we prove that, over large fields, LDPC codes can approach the singleton bound – a result that may be of independent interest. See Remark 1.

party in each "correction round".[5] To derive a constant round solution, we take a different route and require the underlying AFS to be *robust* against a bounded number of erasures of the *public header*. (This notion of robustness should not be confused with the standard notion of robust secret sharing tolerating faulty shares.)

Robust AFS. Roughly speaking, in robust AFS, we want the secret to be recoverable even if the adversary erases some subset $B \subset [m]$ of the entries of $y = (y_1, \ldots, y_m)$. (Think of $|B|$ as a small constant fraction of m.) Intuitively, this means that a subset of t_{c} honest parties T that holds the shares $(x_i)_{i \in T}$ should be able to recover the secret $s = \sum_i v_i x_i$ given only some of the public shares $(y_i)_{i \notin B}$. Unfortunately, when the matrix is sparse such a strong level of robustness cannot be achieved since the adversary can erase all the $O(1)$ equations in which, say, the first honest party participates. This means that an honest coalition that does not contain the first honest party cannot recover x_1 and thus cannot reconstruct the secret $s = \sum_i v_i x_i$. Indeed, erasures can effectively remove all the information about the shares of some of the (possibly honest) parties. We solve the problem by compromising on the following weaker notion of robustness: After the removal of B it should be possible to efficiently locate a set A of parties such that after their removal, the *residual scheme* (M', y', v') obtained by removing (more precisely zero-ing) the B entries of y, the A entries of v, and the $B \times A$ submatrix of M, still supports recovery for a sufficiently large correctness threshold t_{c}. This means that we can "sacrifice" the B entries of y and still recover the newly defined secret $s' = \sum_{i \notin A} v_i x_i$. Observe that the adversary effectively shifts the secret to s', moreover, the adversary (which is rushing) can choose which subset B to abort after seeing the entire vector of pubic shares y. The robustness definition takes this into account and guarantees that, even under such an attack, security holds (i.e., the secret remains private and independent of the adversary's attack). To make this approach work, we show that sparse matrices can be used to derive robust AFS. We also need to carefully define ideal functionalities that capture the adversary's behavior and show that, when instantiated with robust-AFS, they realize the abstract DSG and DKG functionalities.

Achieving Constant Rounds, Constant Communication and Near-Threshold. With the help of robust-AFS, we can run the GMW-based protocol for computing the headers $y = Mx$ and simply give up on the "erased" header $(y_i), i \in B$. This immediately yields a three-round protocol with low communication assuming that the adversary can erase up to b entries where b is the robustness parameter of the AFS. This limitation induces a very small (yet linear) bound on the privacy threshold t_{p}. In order to improve this and derive near-threshold

[5] Such a protocol has an "optimistic" constant round complexity (when there are no aborts), and a "pessimistic" linear round complexity. Moreover, if the adversary delays the protocol by $r \leq t_{\mathsf{p}}$ rounds it must publicly reveal $\Omega(r)$ corrupted parties. Assuming some penalty mechanism, such a protocol may be acceptable in practice.

scheme, we apply the robustness property in a less aggressive way. Specifically, the robustness parameter b is taken to be some small (linear in n) value, and we apply robustness only if there are less than b erasures. If the number of erasures is larger, we remove the parties that were publicly identified as cheaters and recompute the missing entries. This is done repeatedly, and it can be shown that after a constant number of rounds (that depends on the $t_{\mathsf{p}}, t_{\mathsf{c}}$ and b), the number of erasures is sufficiently small and robustness can be applied. This strategy is non-trivial to implement with constant downstream communication (since we do not even have enough bandwidth to publish the number of missing entries), nevertheless we realize this approach with constant communication by carefully postponing some of the computation to a post-processing public-decoding phase that is invoked after the sharing phase as part of the reconstruction.

Organization. Following some preliminaries in Sect. 3, we devote Sect. 4 to the study of AFS including definitions, properties, and sparse constructions. In Sect. 5 we formalize DSG and DKG protocols in an MPC framework and show how to realize these notions based on appropriate protocols for robust-AFS. Communication-efficient protocols for distributing a secret according to a sparse robust-AFS are presented in Sect. 6. A concrete instantiation of this protocol appears in the full version [3].

3 Preliminaries

General Notation. We let $[n]$ denote the set of integers $\{1, \ldots, n\}$. For an $m \times n$ matrix $M = (M_{j,i})_{j \in [m], i \in [n]}$ and sets $R \subset [m]$ and $C \subset [n]$, we let $M[R; C]$ denote the $m \times n$ matrix whose (j,i)th entry is $M_{j,i}$ if $(j,i) \in R \times C$ and zero otherwise. We also let $M[; C] := M[[m]; C]$ be the matrix that agrees with M on the columns in C and takes the value zero in all other columns. The complement of a set $T \subset [n]$ is denoted by $\bar{T}$. For random variables X and Y, we write $X \equiv Y$ to denote that X and Y are identically distributed.

Cryptographic Definitions and Primitives. We use the standard notion of a *non-interactive commitment scheme* $\mathsf{Com}_{\mathsf{crscm}}(x; k)$ where crscm is a random reference string crscm, x is a message and k is a random commitment key k. (See the full version [3] for a definition.) To simplify notation, we typically omit the reference string crscm from the description of the commitment algorithm. Such commitments can be constructed based on one-way functions [24,31].

We employ *Non-interactive zero-knowledge proofs of knowledge* (NIZK). Specifically, following [27], we rely on ID-based simulation-sound NIZK proof system (see also [28,34]). Syntactically, this means that proofs are generated with respect to an identifier. Roughly, zero-knowledge requires that simulated proofs are indistinguishable from real proofs even for adaptively chosen statements. Simulation soundness requires that if an adversary who is given an access to simulated proofs with respect to a set of identities H, can generate a valid proof with respect to any identity outside H, then a valid witness can be extracted. The formal definition appears in the full version [3].

We assume familiarity with standard MPC definitions (see, e.g., [7,20]). Throughout the paper we let C denote the set of corrupted parties and H denote the set of honest parties.

The BB Model. We assume that parties have an access to a public bulletin board (BB) that is abstracted as an array or dictionary with random access. The array is partitioned to sections, and each party has an exclusive write-once permission for her section, i.e., only party i can write an element to the ith sub-array and once an element was written to cell number j, this value remains unchanged forever, and so parties who read these cell will always *agree* on its value. We view the elements on the BB as publicly available to all the parties, that is, all the parties have read permission to all sections. Our protocols naturally define for every message an address (or a key) in which it is stored, and instruct each party which addresses to read from the BB in each step.[6] (Malicious parties can, of course, read everything.) For the sake of clarity, when describing a protocol, we typically treat the BB as a broadcast channel (keeping the mapping between messages and their addresses implicit), and only later analyze the fine-grained communication and see how many elements a party reads/writes during the protocol.

4 Secret Sharing

4.1 Definitions and Basic Facts

Through the paper, we assume that $\mathbb{F}$ is a finite field or a family $\mathbb{F} = \{\mathbb{F}_n\}$ of finite fields whose size grows with the security parameter or the number of parties. In the latter case, we assume that field operations can be implemented in polynomial time, and keep the dependency in n implicit.

We use a slightly non-standard variant of the notion of Linear Secret Sharing schemes. Roughly, (1) we assume that the share of each party is a single field element and (2) we replace linearity with affineness. (See Remark 2 for an explanation about the usage of affineness.) In addition, for convenience, our definition is centered around the process of sampling a random secret sharing vector that corresponds to a random secret, as opposed to sharing a given secret. (This difference is mainly syntactic and one can easily move between these two variants.)

Definition 1 (AFS: Semantics). An n-party $(t_{\mathsf{p}}, t_{\mathsf{c}})$ *Affine Secret Sharing* Scheme (AFS) over a finite field $\mathbb{F}$ is a pair (Σ, Rec) where Σ is a probability distribution of sampling shares over an affine subspace of $\mathbb{F}^n$ and Rec is a recovery algorithm that takes a subset $T \subset [n]$ and a vector of shares $x[T] = (x_i)_{i \in T} \in \mathbb{F}^{|T|}$ and outputs a secret $s \in \mathbb{F}$ with the following properties:

[6] We note that this convention is aligned with modern blockchains (e.g., Ethereum, Solana, Aptos) that implement storage as a key-value store (RocksDB in the latter two), and support direct retrieval of data using keys. Thus the download channel of reading values from the BB is cheap and straightforward.

– t_{c}-Correctness: For every subset $T \subset [n]$ of size at least t_{c} (hereafter referred to as "authorized") it holds that

$$\Pr_{x \overset{R}{\leftarrow} \Sigma}[\mathsf{Rec}(T, x[T]) = s(x)] = 1,$$

where $s(x) = \mathsf{Rec}([n], x)$ is referred to as the secret associated with the vector of sharing x. Furthermore, for every fixing of T the mapping $\mathsf{Rec}(T, \cdot) : \mathbb{F}^{|T|} \to \mathbb{F}$ is a linear mapping.

– t_{p}-Privacy: For every set $T \subset [n]$ of size at most t_{p} (hereafter referred to as "unauthorized"), we have

$$(x[T], s(x)) \equiv (x[T], s'),$$

where $x \overset{R}{\leftarrow} \Sigma$, and $s' \overset{R}{\leftarrow} \mathbb{F}$ is chosen independently and uniformly.

Standard Representation. By default, we assume that the AFS works as follows:

– The AFS is specified by an $m \times n$ *constraint matrix* M, a column *offset vector* $y \in \mathbb{F}^m$, and a row vector $v \in \mathbb{F}^n$ referred to as the *extraction vector*.
– The sampling algorithm $\Sigma_{M,y,v}$ samples a uniform solution $x \in \mathbb{F}^n$ to the set of equations $Mx = y$. When y is the all zero vector the scheme is *linear* as opposed to affine.
– The underlying secret $s(x) = \sum_i x_i v_i$ is taken to be the inner-product between the vector of shares x and the extraction vector v. (The terminology is borrowed from the notion of *randomness extractors*, i.e., we extract the secret from the randomness of the parties).
– The recovery algorithm expresses the missing shares as a linear combination of the existing shares, and outputs the multiplication of v by the vector of shares. More precisely, the recovery algorithm $\mathsf{Rec}_{M,y,v}(T, x[T])$ finds a row vector $\alpha \in \mathbb{F}^m$ such that $\alpha \cdot M[\ ;\bar{T}] = v[\bar{T}]$, and outputs $\sum_{i \in T} v_i x_i + \alpha \cdot (y - M[\ ;T] \cdot x[T])$.
If there is no such vector α, i.e., $v[\bar{T}]$ is not in the row-span of $M[\ ;\bar{T}]$, the recovery algorithm fails. Note that both Σ and Rec can be computed efficiently by making $\mathrm{poly}(n)$ number of arithmetic operations over $\mathbb{F}$.

The following simple fact characterizes the correctness and privacy in linear algebraic terms. (This is a straightforward generalization of the well-known linear algebraic characterization of linear secert sharing to the affine setting).

Fact 2 (linear-algebraic characterization of privacy and correctness). *Let $M \in \mathbb{F}^{m \times n}$, $y \in \mathbb{F}^m$ and $v \in \mathbb{F}^n$, and assume that the offset vector y is a vector in the image of the constraint matrix M. Let x be a uniformly chosen solution to the system $Mx = y$ and let $s(x) = \sum_i x_i v_i$ denote the random variable induced by the choice of x. For every set $T \subset [n]$, if $v[\bar{T}] \in \mathrm{rowspan}(M[\ ;\bar{T}])$ then*

$$\Pr_x[\mathsf{Rec}_{M,y,z}(T, x[T]) = s(x)] = 1,$$

and otherwise,

$$(x[T], s(x)) \equiv (x[T], s')$$

where s' is uniform over $\mathbb{F}$. Consequently, $(\Sigma_{M,y,v}, \mathsf{Rec}_{M,y,v})$ is t_c-correct (resp., t_p-private) if and only if for every set $T \subset [n]$ of size t_c (resp., t_p) the vector $v[\bar{T}]$ is spanned (resp., not spanned) by $M[\ ;\bar{T}]$.

We say that (M, y, v) is t_c-*correct* (resp., t_p-*private*) if the AFS given by $(\Sigma_{M,y,v}, \mathsf{Rec}_{M,y,v})$ is t_c-correct (resp., t_p-private). By Fact 2, the offset vector y plays no role in the privacy/correctness of the scheme as long as it is in the image of M.We will always assume that the offset vector y is in the image of the constraint matrix M, and accordingly refer to (M, v) as t_c-*correct* (resp., t_p-*private*) if (M, y, v) is t_c-correct (resp., t_p-private) for every y is in the image of M.

From Codes to an Affine Secret Sharing Scheme (AFS). It is not hard to see that the correctness property can be based solely on the error correction properties of the constraint matrix M, regardless of the choice of v. Formally, define the *dual distance* of M, denoted by $\mathsf{dd}(M)$, to be the smallest number of linearly *dependent* columns of M (over $\mathbb{F}$). Note that this means that for every subset $T \subset [n]$ of size lesser than $\mathsf{dd}(M)$ the matrix $M[\ ;T]$ is of rank at least $|T|$ and so $v[\bar{T}] \in \text{rowspan}(M[\ ;\bar{T}])$ for every vector $v \in \mathbb{F}^n$. Therefore, the tuple (M, y, v) is $(n - \mathsf{dd}(M) + 1)$-correct no matter how the extraction vector v is chosen. Privacy now boils down to the selection of the extraction vector. We say that the extraction vector v is t_p-*private* for M (over $\mathbb{F}$) if for every t_p-subset $T \subset [n]$, it holds that $v[\bar{T}] \notin \text{rowspan}(M[\ ;\bar{T}])$. Then, we have the following immediate claim (whose proof is implicit in [2] and is closely related to the general transformation of [11]).

Claim 3. *For every $m \times n$ matrix M, vector $y \in \mathbb{F}^m$ in the image of M and extraction vector $v \in \mathbb{F}^n$ which is t_p-private for M, the tuple (M, y, v) is $(n - \mathsf{dd}(M) + 1)$-correct and t_p-private. Moreover, except with probability $|\mathbb{F}|^{-(n-m-t_p)}\binom{n}{t_p}$, a randomly chosen vector $v \xleftarrow{R} \mathbb{F}^n$ is t_p-private for M.*

Proof. The first part follows from the above discussion and Fact 2. The "Moreover" part, follows by noting that for any fixed t_p-subset $T \subset [n]$, the rank of $M[\ ;\bar{T}]$ is m, and therefore, the probability that $v[\bar{T}] \in \text{rowspan}(M[\ ;\bar{T}])$ is at most $|\mathbb{F}|^m/|\mathbb{F}|^{n-t_p}$. By applying a union-bound over all possible t_p-subsets, we get a failure probability of $|\mathbb{F}|^{-(n-t_p-m)}\binom{n}{t_p}$, as required. □

Remark 1 (Near-Threshold AFS). Assuming that the field size grows asymptotically with the number of parties (e.g., $|\mathbb{F}| > \omega(1)$) we can take $t_p = (1-\varepsilon)(n-m)$ for an arbitrary small constant $\varepsilon > 0$, and still get a negligible failure probability of $2^{-\Omega(n)}$.[7] If the distance of the code approaches the singleton bound, i.e.,

[7] If the field is exponentially large (which is a reasonable scenario in the context of threshold cryptography), we can even take $t_p = (n - m)$.

$\mathsf{dd}(M) > (1-\varepsilon)m$, then $t_{\mathsf{c}} = (1+\varepsilon)(n-m)$. Altogether, we get an almost-tight privacy-to-correctness gap $t_{\mathsf{c}} - t_{\mathsf{p}} \leq 2\varepsilon(n-m)$.

For small fields (including the case of the binary field), we cannot hope to get arbitrarily small gap [6]. Still for a code with constant relative distance and constant rate, we still get, except with negligible probability, some non-trivial constants $0 < t_{\mathsf{p}} < t_{\mathsf{c}} < 1$ that are bounded-away from zero and one.

Collections of AFS. A $(\tau_{\mathsf{p}}, \tau_{\mathsf{c}})$-AFS collection with error $\varepsilon(\cdot)$ is specified by a probabilistic polynomial-time algorithm Z that given 1^n samples an index $z = (M, y, v)$ such that, except with probability $\varepsilon(n)$, the pair $(\Sigma_z, \mathsf{Rec}_z)$ forms an n-party $(\tau_{\mathsf{p}} n, \tau_{\mathsf{c}} n)$-AFS. By default, we assume that the error parameter ε is negligible in n. We may also assume that Z samples only the constraint matrix M and the extraction vector v since any y in the image of M can be used. We say that an AFS collection is *sparse* if the number of non-zero elements in every row and column of the constraint matrix is bounded by a fixed constant that does not grow with n.

Remark 2 (Why Should We Use Affine Schemes?). By Fact 2, privacy and correctness depend only on the constraint matrix M and the extraction vector v, and any offset vector y (in the column span of M) can be used. For this reason, the standard choice in the literature is to focus on LSS (as opposed to AFS) and assume that y is the all-zero vector. Still, for computational efficiency, it will be beneficial to employ a non-zero y since, in some cases sampling x conditioned on $Mx = \mathbf{0}$ is more expensive than sampling a uniform x and setting $y = Mx$. In particular, in a distributed setting, each party can sample its own share x_i independently at random, and then the parties reveal y via MPC. This approach will be used in our DSG and DKG constructions. Getting back to the information bottleneck mentioned in the introduction, the use of a non-zero vector y is in fact necessary for achieving our results.

Remark 3 (From Affine to Linear). In many applications of secret sharing affineness provides a sufficiently good substitute for linearity. Moreover, if this is not the case then one can easily turn an affine sharing x of a random secret s under the AFS $z = (M, y, v)$ into a linear sharing of a random secret s' under the linear secret sharing scheme $(M, \mathbf{0})$. This can be done by letting each party i locally redefine its share to $x_i - x'_i$ where $x' \in \mathbb{F}^n$ is some canonical vector for which $Mx' = y$. It is not hard to verify that the resulting sharing vector $x - x'$ is a random sharing under the scheme $z' = (M, \mathbf{0}, v)$ of the shifted secret $s - s'$ where $s' = \sum_i x'_i v_i$ is the secret associated with x' under $z = (M, y, v)$. Moreover, if (M, y, v) is $(t_{\mathsf{p}}, t_{\mathsf{c}})$-AFS then the scheme $(M, \mathbf{0}, v)$ is an $(t_{\mathsf{p}}, t_{\mathsf{c}})$-LSS.

4.2 AFS from Expanders

In this section we define a certain expansion property for matrices, use existing techniques (Fact 4) to sample matrices with this property, and prove (Theorem 5) that such expanders can be used to construct a near-threshold sparse-AFS.

Matrices, Sparsity, and Expansion. Let $M = (M_{j,i})_{j\in[m],i\in[n]}$ be an $m\times n$ matrix. We say that M is *d-sparse* if every column of M has at most d non-zero elements, and say that it is *(d,r)-sparse* if, in addition, every row of M has at most r elements. We say that M is (ℓ, e) *column-expanding* (or just expanding) if for every set S of at most ℓ columns, the submatrix $M[\ ;S]$ has at least $e\cdot|S|$ non-zero rows. Let $\eta_e(M)$ denote the largest ℓ for which M is (ℓ,e) expanding. Note that $\eta_e(M)$ is monotonically decreasing with e and, for d-sparse matrices, $\eta_e(M)=0$ for any $e>d$. It is well known (see, e.g., [40, Problem 5.5.]) that for (d,r)-sparse matrices and every $a>d/2$,

$$\eta_a(M) \le \mathsf{dd}(M) - 1 \le \eta_1(M), \tag{1}$$

where the equation holds regardless of the choice of the finite field $\mathbb{F}$ over which the dual-distance is computed. That is, expansion beyond half-the-column-sparsity, $d/2$, guarantees good distance, whereas non-shrinkage (expansion of at least 1) is necessary for good distance. Jumping ahead, we note that for large fields and properly chosen matrices, non-shrinkage is also sufficient for good distance.

Collections of Matrices. For constants $\mu\in(0,1)$ and $d,r\in\mathbb{N}$, a *collection of (μ,d,r)-matrices* is defined by a (possibly randomized) polynomial-time algorithm $\mathcal{M}$ that given 1^n outputs a (d,r)-sparse $\mu n\times n$ matrix over $\mathbb{F}$. We say that the collection is (λ,e) *expanding with error $\epsilon(n)$* (resp., *has distance* dd *with error $\epsilon(n)$*) if the resulting matrix is $(\lambda n,e)$ expanding (resp., has distance $\mathsf{dd}n$) except with probability $\epsilon(n)$. By default, we assume that ϵ is a negligible function. The following constructions are based on [1,9].

Fact 4 (Expanding Collections). *For every constant $\mu\in(0,1)$, constant $\varepsilon>0$, and constant $\lambda<\mu/(1+\varepsilon)$ there exist constants d,r, and a collection of (μ,d,r) binary matrices that are $(\lambda,1+\varepsilon)$ expanding with a negligible error probability.*

Also, for every constant $\mu\in(0,1)$, there exist constants d,r,λ and a collection of (μ,d,r) binary matrices that are $(\lambda,0.9d)$ expanding with zero error probability.

We note that the constant 0.9 in the second part of the fact was chosen arbitrarily, and could be replaced by any constant larger than 0.5.

Proof. Observe that it suffices to prove the statement without worrying about the row sparsity. Indeed, the average row sparsity must be d/μ and so, by Markov's inequality, for every $\alpha>0$, all but α-fraction of the rows have weight at most $r=d/(\mu\alpha)$. By removing these heavy rows we get (d,r)-sparsity at the expense of a small constant degradation in the parameter λ. The second part of the theorem now follows immediately from the celebrated result of [9].

To prove the first part we rely on [1]. Since the statement in the original paper refers to a slightly different regime of parameters, we sketch the argument here. Consider a random $\mu n\times n$ binary matrix R that each of its columns is sampled

independently at random so that each column contains d ones. Let p_ℓ denote the probability that there exists a *non-expanding set* of exactly ℓ columns, i.e., a set that fails to expand by a factor of $1+\varepsilon$. A standard calculation shows that

$$p_\ell \leq \left[c_{\varepsilon,\mu} \left(\frac{(1+\varepsilon)\ell}{\mu n} \right)^{d-2-\varepsilon} \right]^\ell,$$

where $c_{\varepsilon,\mu}$ is a constant that depends on ε and μ but is independent of d. By taking d to be a sufficiently large constant, we can guarantee that p_ℓ is negligible for every $\omega(1) < \ell < \mu n/(1+\varepsilon)$. However, for constant size ℓ's we get an inverse polynomial failure probability, which means that the overall failure probability $\sum_{\ell \leq \mu n/(1+\varepsilon)} p_\ell$ is inverse polynomial in n. To reduce the error to be negligible, we use the construction from [1] that samples a sparse matrix M' such that, except with negligible probability $\gamma(n)$, there are no non-expanding sets of size smaller than ℓ_0 for some super-constant parameter $\ell_0 = \omega(1)$. Let M denote the matrix obtained by taking the union of M' with a random sparse matrix R (i.e., $M_{i,j} = M'_{i,j} \vee R_{i,j}$ for each i,j). Then M is a sparse matrix that does not have a non-expanding set of size smaller than $\mu n/(1+\varepsilon)$ except with probability $\gamma(n) + \sum_{\ell_0 < \ell \leq \mu n/(1+\varepsilon)} p_\ell$ which is negligible in n. □

Theorem 5 (near-threshold sparse-AFS from expanders). *For every constants $\tau_\mathsf{p} < \tau_\mathsf{c}$ there exists constants d, r such that for every field $\mathbb{F}$ of size super-polynomial $n^{\omega(1)}$, there exists a $(\tau_\mathsf{p}, \tau_\mathsf{c})$-AFS over $\mathbb{F}$ whose constraint matrix is (d,r)-sparse. Furthermore, except with negligible probability the dual distance of the constraint matrix is $(1-\tau_\mathsf{c})n$.*

Proof. Let $\mu \in (1-\tau_\mathsf{c}, 1-\tau_\mathsf{p})$ be a constant. Given 1^n, we sample a tuple (M, y, v) as follows. (1) Use the first part of Fact 4 to sample a sparse $\mu n \times n$ binary matrix which is $((1-\tau_\mathsf{c})\mu, 1+\varepsilon)$ expanding for some constant $\varepsilon > 0$. Next, replace each non-zero position by a uniformly chosen field element and let M denote the resulting matrix. It is shown in [41, Lemma 3.9] that, except with negligible probability $|\mathbb{F}|^{-1}$, the dual distance of M over $\mathbb{F}$, is at least as large as the expansion parameter $(1-\tau_\mathsf{c})n$. Sample a random reconstruction vector $v \xleftarrow{R} \mathbb{F}^n$ and take y to be an arbitrary vector in the image of M. By Remark 1, except with exponentially small probability, we get a $(\tau_\mathsf{c} n, \tau_\mathsf{p} n)$-AFS (since $\tau_\mathsf{p} < 1-\mu$ and $\tau_\mathsf{c} = 1 - \mathsf{dd}(M)/n$), as required. □

Remark 4 (LDPC Codes that Almost Achieve the Singleton Bound). Our proof implicitly shows that when the field $\mathbb{F}$ is sufficiently large (say super-polynomial in n), for every $\varepsilon > 0$ there are d_ε-sparse $m \times n$ parity-check matrices whose distance Δ approaches the singleton bound, i.e., $\Delta > (1-\varepsilon)n$. Moreover, such codes can be efficiently sampled with negligible error probability. To the best of our knowledge, this result does not appear in the literature. For comparison, the work of [30, Thm. 2.14] shows that such codes can achieve the Gilbert-Varshamov bound when the sparsity grows with the field size.

Remark 5 (Sparse-AFS Over Small Fields). One can efficiently construct matrices that achieve a constant rate and a constant distance even under constant size fields [17]. In fact, by using the second part of Fact 4 and the connection between expansion beyond half-the-column-sparsity $d/2$ in Eq. (1), one can get binary matrices that achieve constant distance over any finite field. By sampling a random extraction vector as in Claim 3, we get a $(\tau_{\mathsf{p}}, \tau_{\mathsf{c}})$-AFS, for some non-trivial constants $0 < \tau_{\mathsf{p}} < \tau_{\mathsf{c}} < 0$, that works over small fields whose constraint matrix is a sparse binary matrix. (In fact, by using the techniques of [2], we can get a single scheme that works universally over all finite fields that also enjoys several efficiency features in reconstruction.)

Example 1 (Sparse-AFS Over Large Fields: Concrete Numbers). Say that the field is of size at least 2^{100} *(in threshold systems the size is typically larger, e.g.,* $\approx 2^{255}$ *for Schnorr's signature). Consider the following examples:*

1. *Say that we have* $n \geq 10$ *parties and take an AFS matrix with* $\mu \cdot n$ *rows where* $\mu = 0.5$*. By standard expansion calculations, there are sparse matrices with* $d = 8$ *and* $r \approx 16$*, that achieve* $\tau_{\mathsf{c}} = 0.66$*.*[8] *By choosing a random extraction vector, we get* $\tau_{\mathsf{p}} = 0.4$ *except with failure probability* 2^{-100}*. So we get a* $(8, 16)$*-sparse* $(0.4, 0.66)$*-AFS. (This favorably compares to the canonical setting of* $1/3$ *corrupt vs* $2/3$ *honest that is used in many scenarios.)*
2. *As another data point, assume that the field* $\mathbb{F}$ *is of size at* 2^{255} *and that the number of parties* $n > 50$*. Then, by taking* $\mu = 0.6$*, we can get a* $(4, 10)$*-sparse matrix with* $\tau_{\mathsf{p}} = 0.39$*,* $\tau_{\mathsf{c}} = 0.9$*.*

4.3 Robust AFS

Motivation. When a DKG is run, some participants might behave maliciously and corrupt some of the shares that are needed for reconstructing the newly distributed key. We need the AFS to be robust to such attempts. For concreteness, consider the following scenario. Given an AFS $z = (M, y, v)$, we distribute a vector of random shares $x \in \mathbb{F}^n$ by sampling a uniform solution to the system $Mx = y$. Then, an adversary who controls a t_{p}-subset $T \subset [n]$ of the parties gets his shares $x[T]$ and is allowed to corrupt the index z by erasing a small subset B of the entries of the vector y. (Think of B as a small constant fraction of n.) Intuitively, we want the secret to still be recoverable given only $(M[\bar{B};\ ], y[\bar{B}], v)$. Unfortunately, when the matrix is sparse this is impossible since the adversary can, for example, include in B and erase all the $O(1)$ equations in which the first honest party participates. In this case, an honest coalition that does not contain the first honest party has no information on x_1 and cannot reconstruct the secret $s = \sum_i v_i x_i$.

[8] The calculation here is based on the probabilistic method (i.e., we bound the probability that a random sparse matrix fails to expand well). We ignore here the issue of finding explicit expanding matrices and note that this can be done via several existing techniques, e.g., [1].

Indeed, erasures in z effectively remove all the information about the shares of some of the (possibly honest) parties. The key idea is to make sure that these "lost" parties will not affect the secret by zero-ing the corresponding entries of the extraction vector. This way the secret remains recoverable even without the missing shares. In more details, we compromise on the following weaker notion of robustness: After the removal of B, it should be possible to locate a set A of parties such that even if their shares are lost, the *residual scheme* $z_2 = (M[\bar{B}; \bar{A}], y[\bar{B}], v[\bar{A}])$, namely the n-party scheme containing all the original shares except those of A that are effectively taken to be zero, still supports recovery for a sufficiently large correctness threshold t_{c}. That is, for $z_2 = (M[\bar{B}; \bar{A}], y[\bar{B}], v[\bar{A}])$, the recovery algorithm Rec_{z_2} can t_{c}-recover the secret $s' = \sum_{i \in \bar{A}} x_i v_i$ even when the shares x are sampled according to Σ_z (i.e., as a uniform solution to $Mx = y$). Note that the secret associated with x is changed to s' since we use the restricted extraction vector $v[\bar{A}]$. So privacy now means that the secret s' should remain information-theoretically hidden given $x[T]$. That is, the restricted extraction vector $v[\bar{A}]$ should be t_{p}-private for the *original* matrix M. Jumping ahead, note that by erasing B, the adversary effectively shifts the shared secret from s to s'. Still this does not bias the output since s' is still uniform and since our DSG/DKG protocols will ensure that the choice of B is independent of the secret. We continue with a formal definition of robust AFS.

Definition 2 (Robust AFS). Let M be an $m \times n$ matrix, y be a vector in the column span of M, and $v \in \mathbb{F}^n$. We say that the tuple $z = (M, y, v)$ is *b-robust* $(t_{\mathsf{p}}, t_{\mathsf{c}})$*-AFS* if for every b-subset $B \subset [m]$ there exists a set $A = A(B) \subset [n]$, referred to as the *sacrificed set of* B, such that for

$$z_1 = (M, y, v[\bar{A}]) \quad \text{and} \quad z_2 = (M[\bar{B}; \bar{A}], y[\bar{B}], v[\bar{A}])$$

the pair $(\Sigma_{z_1}, \mathsf{Rec}_{z_2})$ forms a $(t_{\mathsf{p}}, t_{\mathsf{c}})$-AFS. (We use this pair of algorithms since the secret is shared with Σ_{z_1} and recovered with Rec_{z_2}.) We further assume that if B is an empty set then A must be an empty set as well, and therefore every b-robust $(t_{\mathsf{p}}, t_{\mathsf{c}})$-AFS, for $b \geq 0$, is also $(t_{\mathsf{p}}, t_{\mathsf{c}})$-AFS, and a 0-robust $(t_{\mathsf{p}}, t_{\mathsf{c}})$-AFS is simply a $(t_{\mathsf{p}}, t_{\mathsf{c}})$-AFS.

An AFS ensemble with an index sampler Z is *β-robust* $(\tau_{\mathsf{p}}, \tau_{\mathsf{c}})$*-AFS* if for all but negligible probability over $(M, y, v) \xleftarrow{R} Z(1^n)$, the AFS (M, y, v) is βn-robust $(\tau_{\mathsf{p}} n, \tau_{\mathsf{c}} n)$-AFS. We also require that the set A should be efficiently computable given (M, B).

Importantly, the residual scheme is defined over n parties, and the definition guarantees that even after the erasure, every subset of t_{c} parties (possibly including parties in A) can recover the secret.

Lemma 1 (Sparse AFS are Robust). *Suppose that (M, y, v) is a $(t_{\mathsf{p}}, t_{\mathsf{c}})$-AFS and that M is a (d, r)-sparse matrix whose dual distance is $\Delta = n - t_{\mathsf{c}} + 1$. Then (M, y, v) is a b-robust $(t_{\mathsf{p}} - b \cdot r, t_{\mathsf{c}})$-AFS for every b. Furthermore, the set $A(B)$ is taken to be the columns whose support intersects with B, i.e., $A(B) = \{i : \exists j \in B, M[j, i] \neq 0\}$.*

Note that the lemma keeps the correctness parameter t_{c} unchanged, and the extra robustness property only affects the privacy thershold.

Proof (of Lemma 1). Fix a b-subset $B \subset [m]$ and let $A = A(B)$ as defined above. We begin by claiming that ($\star$) the $\bar{B} \times \bar{A}$ sub-matrix L of M has distance of at least $\Delta = n - t_{\mathsf{c}} + 1$. For this it suffices to show that any set of $\Delta - 1$ columns $\{w_i\}_{i \in S}$ in L are linearly independent. To see this, recall that each column vector w_i is obtained from a column $\hat{w}_i$ of M via the projection $w_i = \hat{w}_i[\bar{B}]$ where the B-coordinates of w_i are known to be zero (otherwise $i \in A$ and w_i is not a column of L). This means that $\{w_i\}_{i \in S}$ is linearly independent if the M-vectors $\{\hat{w}_i\}_{i \in S}$ are linearly dependent, which is the case by the assumption on the distance of M.

Let $t'_{\mathsf{p}} = t_{\mathsf{p}} - b \cdot r$. We will now prove that $(\Sigma_{z_1}, \mathsf{Rec}_{z_2})$ forms a $(t'_{\mathsf{p}}, t_{\mathsf{c}})$-AFS where z_1, z_2 are defined as in Definition 2. Let $x \in \mathbb{F}^n$ be a random solution to the system $Mx = y$. Then, $x' = x[\bar{A}]$ is also a solution to the system defined by $M' = M[\bar{B}; \bar{A}]$ and $y' = y[\bar{B}]$). Let $v' = v[\bar{A}]$. Fix a set $T \subset [n]$ of size at least t_{c}. Given $x[T]$, the recovery algorithm Rec_{z_2} recovers the secret $s' = \sum_i v'_i x'_i$ if and only if $v'[\bar{T}]$ is spanned by the rows of $M'[\ ; \bar{T}]$. This condition is equivalent to the condition that $v'[\bar{T} \cap \bar{A}]$ is spanned by the rows of $M'[\ ; \bar{T} \cap \bar{A}]$ (since the A entries/columns are set to zero). This is indeed the case, since the set $\bar{T} \cap \bar{A}$ is of size at most $n - t_{\mathsf{c}}$ which is smaller than the distance, Δ, of the $\bar{A} \times \bar{B}$ sub-matrix L of M, as shown in ($\star$).

Fix a t'_{p}-subset $T \subset [n]$. To show that s' is distributed independently of $x[T]$, it suffices to show that

$$v'[\bar{T}] = v[\bar{A} \cap \bar{T}] \text{ is not in } \operatorname{colspan}(M[\ ; \bar{T}]). \tag{2}$$

Taking $S := A \cup T$, it holds that $\bar{S} = \bar{A} \cap \bar{T}$, and so (2) holds if $v[\bar{S}] \notin \operatorname{colspan}(M[\ ; \bar{T}])$ which must be the case since $|S| = (t'_{\mathsf{p}} + |A|) \leq t'_{\mathsf{p}} + br = t_{\mathsf{p}}$ and since v is t_{p}-private for M by assumption. (The inequality $|A| \leq b \cdot r$ follows by the sparsity condition on the matrix). □

By combining Lemma 1 with Theorem 5 we derive the following corollary.

Corollary 1 (Near-Threshold Robust Sparse-AFS). *For every constants $\tau_{\mathsf{p}} < \tau_{\mathsf{c}}$ there exists constants d, r such that for every field $\mathbb{F}$ of size super-polynomial $n^{\omega(1)}$, there exists a (d, r)-sparse AFS collection which is β-robust $(\tau_{\mathsf{p}} - r\beta, \tau_{\mathsf{c}})$-AFS over $\mathbb{F}$ for every $\beta \geq 0$.*

Remark 6 (Robust Sparse Binary AFS). For constant-size fields (e.g., the binary field), a similar corollary can be obtained for some constants $\tau_{\mathsf{p}} < \tau_{\mathsf{c}}$, by combining Lemma 1 with Remark 5.

5 Distributed Secret-Sharing Generation

Following Katz [27] we define *distributed key generation in the discrete-logarithm setting* (hereafter referred to as DKG), and *distributed secret-sharing generation*

DSG, via an MPC framework. While Katz's definitions are tailored to Shamir-based DKG, we will need slightly more general definitions that are compatible with general collections of AFS schemes. We begin with an abstract version that captures the desired security properties and move on to more concrete variants, formally captured by *canonical* protocols, that provide additional efficiency features.

5.1 DSG and DKG: Abstract Version

Syntactically, a DSG is a two-stage n-party protocol where the parties hold no input. The first phase, Share, distributes to each party a private share and generates some public information. At the second phase, Rec, the parties recover the secret $s \in \mathbb{F}$, where the field $\mathbb{F} = \mathbb{F}_p$ is implicitly specified as part of the parameters of the scheme. For technical reasons, it will be convenient to add a special party, a "client", whose role is to invoke the two stages of the protocol. The syntax of DKG is similar, except that after the sharing phase, the protocol reveals also the public key g^s as part of the public information where g generates a cyclic group $\mathbb{G}$ of order p that is given implicitly as part of the parameters of the scheme. In its most abstract form, the scheme should realize the following reactive ideal functionality (Functionality 6). The term "broadcasts" should be interpreted as writing a message on the public BB.

Functionality 6 ($\mathcal{F}_{\mathsf{dsg}}$ and $\mathcal{F}_{\mathsf{dkg}}$). *The functionality has two phases that are invoked by the client:*

1. *Share phase: the functionality samples a secret $s \in \mathbb{F}$ and broadcasts the message "shared", and, in the case of $\mathcal{F}_{\mathsf{dkg}}$, also the value g^s.*
2. *Recovery phase: the functionality broadcasts the secret s.*
   (For $\mathcal{F}_{\mathsf{dkg}}$, we can consider a variant in which the client specifies a public group element h, and the functionality broadcasts the pair (h, h^s).)

We will say that a protocol $(t_{\mathsf{p}}, t_{\mathsf{c}})$-*realizes* $\mathcal{F}_{\mathsf{dsg}}$ (resp., $(t_{\mathsf{p}}, t_{\mathsf{c}})$-*realizes* $\mathcal{F}_{\mathsf{dkg}}$) if it realizes $\mathcal{F}_{\mathsf{dsg}}$ (resp., $\mathcal{F}_{\mathsf{dkg}}$) in the presence of a mixed adversary that controls the client, and corrupts up to t_{p} parties with an arbitrary mix of t_1 passive and t_2 active corruptions as long as $t_1 + t_2 \leq t_{\mathsf{p}}$. In addition, at the reconstruction phase, the adversary is allowed to abort (i.e., "crash") additional t_3 honest parties as long as $n - (t_1 + t_3) \geq t_{\mathsf{c}}$, i.e., at least t_{c} parties honestly participate in the reconstruction. This definition implies that any set of t_{c} honest/passively corrupted parties can recover the secret even when the adversary submits faulty shares on behalf of the actively corrupted parties. Such a protocol is also private in the sense that during the sharing phase, an adversary controlling up to t_{p} parties cannot bias the distribution of the secret and cannot learn anything about s (except for what follows from g^s in the case of DKG). In particular, the above MPC-based definition implies the property-based definition of DKG from [19] in its strongest form. We always assume that $t_{\mathsf{p}} < t_{\mathsf{c}}$ and note that the definition is meaningful even when $t_{\mathsf{c}} + t_{\mathsf{p}} \neq n$ (due the use of a mixed adversary).

Remark 7 (Relaxation). For completeness, we present here a relaxed variant of the definition, which is not used in our work. In some scenarios, it makes sense to relax the definition by requiring simulatability only for the sharing phase. Formally, we say that a two-phase protocol Π *weakly realizes* $\mathcal{F}_{\mathsf{dsg}}$ if for every adversary $\mathcal{A}$ there exists an efficient simulator Sim such that the random variable $(\mathsf{View}^{\mathsf{Share}}_{\mathcal{A},\Pi}, \mathsf{Output}^{\mathsf{Rec}}_{\mathcal{A},\Pi})$, consisting of the view of the adversary after the *sharing* phase and the output of the honest parties after the *recovery* phase, is computationally indistinguishable from the pair (Sim, s) where Sim is the output of the simulator and $s \stackrel{R}{\leftarrow} \mathbb{F}$ is a uniformly chosen secret. For the case of DKG, the simulator also gets g^s as an input. Indeed, the DKG variant of this definition is essentially equivalent to the property-based definition from [19].

5.2 Canonical Schemes

While the above definition nicely captures the desired security properties of DSG and DKG, it misses some useful "efficiency" aspects such as non-interactive reconstruction or the ability to reconstruct shares via linear operations – a feature that is necessary for "reconstruction-in-the-exponent" in DLOG-based threshold systems. To capture these additional properties (which are common to most existing schemes), we introduce the notion of canonical schemes and focus throughout the paper on such schemes.[9]

Let E be a non-interactive commitment scheme. We say that a DSG is in *canonical* form if, at the end of the sharing phase, each honest party holds a share $x_i \in \mathbb{F}$ of the secret s according to some $(t_{\mathsf{p}}, t_{\mathsf{c}})$-AFS that is specified by the public values $z = (M, y, v)$ that are known to all parties (e.g., broadcast during the protocol). In addition, at the end of the sharing phase all the parties learn commitments $(\alpha_i = E(x_i; \rho_i))_{i\in[n]}$ to all the shares. In this case, the recovery phase can be implemented by the following single-round *canonical recovery* protocol (Protocol 7).

Protocol 7 (Canonical Recovery Protocol Π_{Rec}). *We assume a common reference string* crspf*, public index* $z = (M, y, v)$ *and public share commitments* $\alpha = (\alpha_i)_{i\in[n]}$*. In addition, each honest party* P_i *holds* (x_i, ρ_i) *such that* $\alpha_i = E(x_i; \rho_i)$*.*

- ***R1:*** *Each party* P_i *broadcasts* x_i *and a NIZK* π_i *(with respect to* crspf*) that* x_i *and* α_i *satisfy the equality* $\alpha_i = E(x_i; \rho_i)$ *with respect to the witness* ρ_i*.*
- ***Output*** *Let* (x'_i, α'_i) *denote the values broadcasted by the* i*th party and let* $T \subset [n]$ *be the set of indices* $i \in [n]$ *for which the proof* π'_i *passes verification with respect to* α'_i *and* x'_i*. Compute the linear recovery algorithm* $\mathsf{Rec}_{M,y,v}(T, x'[T])$ *and output the result.*

[9] Alternatively, one could try to formalize an these properties as part of the ideal functionality (e.g., by letting the functionality distribute "handles" for the secret). We feel that the current solution is simpler and more intuitive.

If we strive for the weaker variant of DSG/DKG that is mentioned in Remark 7 then we can simply open the commitment in the recovery phase and avoid the NIZK. (See the full version [3] for more details.)

Remark 8 (Canonical Recovery in the Exponent). The above protocol can be easily modified to allow the reconstruction of the secret s in the exponent of a public group element $h \in \mathbb{G}$ (which is broadcasted to all the parties by the "client"). In the first round, each party sends h^{x_i} together with NIZK that certifies that x_i is consistent with its commitment α_i. At this point any (possibly external) party can compute h^s by dropping the invalid elements (whose validity proofs fail) and by computing the *linear* reconstruction algorithm Rec_z "in the exponent". Thus canonical protocols efficiently support "reconstruction in the exponent", which is a crucial feature in the context of DLOG-based threshold cryptography. We emphasize that most threshold cryptography applications use recovery in the exponent, for example to compute BLS signatures or to compute a verifiable random function (VRF). The secret is used as the key for these functions. Furthermore, in these applications, when there are multiple invocations of the threshold function, the same secret is recovered multiple times in the exponent, using different random public bases. (One can capture this by defining the DSG/DKG functionality as a reactive multi-phase functionality in which sharing happens once during initialization and recovery-in-the-exponent can be called multiple times with different group elements h; Our protocols hold in this setting as well.)

Given the above discussion, to realize a canonical DSG it suffices to implement a secure protocol for the sharing phase. This is formalized by Functionality 1 in Fig. 1. The ideal functionality $\mathcal{F}_{\mathsf{cdsg},b}$ is parameterized with a robustness parameter b and is implicitly parameterized by a non-interactive commitment scheme E and by a b-robust $(t_\mathsf{p}, t_\mathsf{c})$-AFS. The latter is specified by a public $m \times n$ constraint matrix M, and a public extraction vector $v \in \mathbb{F}^n$ such that for every $y \in \mathbb{F}^m$ in the image of M the AFS (M, y, v) is b-robust $(t_\mathsf{p}, t_\mathsf{c})$-AFS.[10] The non-robust variant is handled by taking the robustness parameter b to be zero. Jumping ahead, we will show later (Lemma 2) that security holds even if the adversary is allowed to choose her own shares based on the residual value of the offset y and to erase up to b of entries of the resulting offset vector.

To understand the definition, let us focus on the non-robust version where the adversary does not erase entries, i.e., $B = \emptyset$. Intuitively, security holds since for any fixing of y in the image of M, and any fixing for the shares of the corrupted parties, $x[\mathsf{C}]$, if we choose the shares of the honest parties $x[\mathsf{H}]$ uniformly at random subject to $Mx = y$, then the secret $s = \sum_i v_i x_i$ is uniformly distributed.

Formally, we prove the following lemma.

Lemma 2. *Let Π be a protocol that t_p-realizes $\mathcal{F}_{\mathsf{cdsg},b}$ for some b-robust $(t_\mathsf{p}, t_\mathsf{c})$-AFS (M, y) and let Π_{Rec} denote the canonical recovery protocol. Then, $(\Pi, \Pi_{\mathsf{Rec}})$,*

[10] Asymptotically, we may assume that M and v are sampled from some b-robust $(t_\mathsf{p}, t_\mathsf{c})$-AFS sampler $Z(1^n)$ during a one-time set-up phase; Such a phase is needed any way to set the field and the underlying cyclic group $\mathbb{G}$.

Functionality 8 ($\mathcal{F}_{\mathsf{cdsg},b}$). *The functionality gets the set of corrupt parties* C.

1. *(Sampling) The functionality samples random commitment keys for the honest parties* $(\rho_i)_{i\in\mathsf{H}}$ *and samples random field elements as shares for the honest parties* $x_\mathsf{H} = (x_i)_{i\in\mathsf{H}} \stackrel{R}{\leftarrow} \mathbb{F}^{|\mathsf{H}|}$. *Then it computes the residual offset vector* $y' = M_\mathsf{H} \cdot x_\mathsf{H} \in \mathbb{F}^m$ *where* M_H *is the restriction of* M *to the columns indexed by* H. *The adversary gets the commitments of the honest parties* $(\alpha_i)_{i\in\mathsf{H}}$ *where* $\alpha_i = E(x_i;\rho_i)$ *and the offset vector* y'. *(The constraint matrix* M *and the extraction vector* v *are assumed to be public.)*
2. *(Corruption) The adversary selects her own shares* $x_\mathsf{C} = (x_i)_{i\in\mathsf{C}}$, *her own commitment keys* $\rho_\mathsf{C} = (\rho_i)_{i\in\mathsf{C}}$ *and specifies an erasure subset* $B \subset [m]$ *of size at most* b. *The tuple* $(x_\mathsf{C}, \rho_\mathsf{C}, B)$ *is sent to the functionality.*
3. *The functionality merges* $x_\mathsf{H}, x_\mathsf{C}$ *to a single vector* $x \in \mathbb{F}^n$, *computes* $y = Mx = y' + M_\mathsf{C} \cdot x_\mathsf{C}$ *where* M_C *is the restriction of* M *to the columns indexed by* C. *Finally, the functionality broadcasts the erasure set* B, *the modified, erased, offset vector* $y[\bar{B}]$, *and the commitments* $(E(x_i;\rho_i))_{i\in[n]}$. *In addition, each honest party* i *privately receives* (x_i, ρ_i).

Fig. 1. Functionality 1 – sharing phase for a canonical DSG.

viewed as a two-phase protocol, $(t_\mathsf{p}, t_\mathsf{c})$*-realizes* $\mathcal{F}_\mathsf{dsg}$ *where we assume that* Π_Rec *is applied to the index* $(M[\bar{A};\bar{B}], y[\bar{B}], v[\bar{A}])$ *where* $B, y[\bar{B}]$ *are the public output of the first phase and* A *is the sacrificed set of* B *(which is computed based on* B *and* M *by using the specification of the AFS).*

Proof (sketch). We begin with a brief intuition under the simplifying assumption that the commitment scheme is perfectly hiding. (This assumption will be waived in actual proof.) In this case, by the privacy properties of the AFS, for any fixing of the view of the adversary after the sharing phase, the distribution of the secret is uniform and independent of the view. Furthermore, assume that in the reconstruction phase, the adversary aborts all but t_c parties that submit honest (uncorrupted) shares. Then, by the correctness and robustness properties, the secret is recovered properly. In the actual proof, we argue that (1) computationally hiding commitments suffice for achieving computationally close simulation and that (2) the binding properties and the NIZK guarantee that a computationally bounded adversary cannot modify its shares during the reconstruction. The full proof is deferred to the full version [3]. □

5.3 From DSG to DKG

We reduce $\mathcal{F}_\mathsf{dkg}$ to canonical DSG, $\mathcal{F}_\mathsf{cdsg}$, by adding a single round of "reconstruction in the exponent" of the generator g. (Party i broadcasts g^{s_i} with a NIZK that proves consistency with α_i, and the valid elements can be combined into g^s as explained in Remark 8.) This allows everyone to recover the public key g^s without revealing any additional information on s. The communication

per party is constant (it depends on the security parameter but does not grow with n). Formally, we have the following lemma whose proof is deferred to the full version [3].

Lemma 3. *Let Π be a protocol that t_{p}-realizes $\mathcal{F}_{\mathsf{cdsg},b}$ for some b-robust $(t_{\mathsf{p}}, t_{\mathsf{c}})$-AFS (M, y) and let Π'_{Rec} denote the "canonical recovery in the exponent" protocol from Remark 8. Then, the functionality $\mathcal{F}_{\mathsf{dkg}}$ is $(t_{\mathsf{p}}, t_{\mathsf{c}})$-realized by the following two-phase protocol:*

- *(Sharing) Invoke Π and then apply Π'_{Rec} with the public group generator g and where the index of the AFS is taken to be $(M[\bar{A}; \bar{B}], y[\bar{B}], v[\bar{A}])$ where $B, y[\bar{B}]$ are the public output of Π and A is the sacrificed set of B (which is computed based on B and M by using the specification of the AFS).*
- *(Reconstruction in the exponent) Given public group generator h, invoke Π'_{Rec} with the generator h and where the index of the AFS is taken to be $(M[\bar{A}; \bar{B}], y[\bar{B}], v[\bar{A}])$ as defined above.*

The drawback of the above approach is that it adds a single round of communication in order to reconstruct the secret in the exponent. This can be avoided if we are willing to realize a weaker variant of the DKG functionality. The idea is to make a single call to the $\mathcal{F}_{\mathsf{cdsg}}$ functionality while setting the underlying commitment scheme in $\mathcal{F}_{\mathsf{cdsg}}$ to $E(x_i; \rho_i) := g^{x_i}$. We refer to this variant as *canonical DKG*, $\mathcal{F}_{\mathsf{cdkg}}$. Although E is not a valid commitment scheme (being deterministic it fails to satisfy semantic security), the values $(g^{x_i})_{i \in [n]}$ leak exactly the public key g^s which should be revealed anyway. Still, strictly speaking, $\mathcal{F}_{\mathsf{cdkg}}$ does not realize $\mathcal{F}_{\mathsf{dkg}}$ since the adversary can choose its inputs after seeing the "exponentiated shares" of the honest parties, and so the adversary can effectively shift the public key by an arbitrary shift $\Delta \in \mathbb{F}$. This issue is discussed by [19] who show that this variant suffices for typical applications of threshold cryptography (e.g., Schnorr's signatures). Intuitively, if the underlying hardness assumption (e.g., in-feasibility of extracting DLOG) holds over the un-shifted key, then it also holds with respect to the shifted public key since the shift Δ can be extracted from the adversary. We note that this variant can be formalized by a variant of the DKG ideal functionality in which the functionality first sends the public key g^s to the adversary who is allowed to shift it by a chosen Δ, and then forwards the shifted public key $g^{s+\Delta}$ to the honest parties.

6 Realizing Robust Canonical DSG

In Sect. 5 we showed that the task of realizing DSG/DKG reduces (with constant overhead) to the task of realizing the $\mathcal{F}_{\mathsf{cdsg},b}$ functionality. In this section we present two protocols that realize $\mathcal{F}_{\mathsf{cdsg},b}$. In both cases, each party reads/writes $O(1)$ elements from the BB such that at the end of the protocol each party holds her private output. In addition, everyone can recover the public outputs by reading the content of the BB. Our first "basic" protocol (Sect. 6.2) achieves a relatively low, yet constant, privacy threshold, and our second "extended"

protocol (Sect. 6.3) provides a near-threshold result, namely an arbitrarily small gap between the privacy and correctness thresholds, τ_{p} and τ_{c}. We begin with some preliminaries (Sect. 6.1).

6.1 Notation and Tools

Notation. Let $M = (M_{j,i})_{j\in[m],i\in[n]}$ be a sparse $m\times n$ matrix. We focus on binary matrices though the following can be easily generalized to the non-binary case. The support of column number $i \in [n]$ is denoted by $R_i = \{j \in [m] : M_{j,i} \neq 0\}$ and the support of row number $j \in [m]$ is denoted by $L_j = \{i \in [n] : M_{j,i} \neq 0\}$. We let $L_{j,-i}$ denote the set $L_j \setminus \{i\}$. The matrix M will be used as a mapping from vectors $x \in \mathbb{F}^n$ to vectors $y \in \mathbb{F}^m$ where $y = Mx$. Accordingly, each column of M corresponds to an input and each row corresponds to an output, and so $j \in R_i$ means that the output j is influenced by the input i and by the inputs in $L_{j,-i}$. For a set of inputs $I \subset [n]$, we let $R(I) = \cup_{i\in I} R_i$ denote the set of outputs that are affected by inputs in I. Throughout the section, $E_\rho(x)$ is taken to be a non-interactive commitment scheme that is specified as part of the description of $\mathcal{F}_{\mathsf{cdsg},b}$. The algorithm E takes a field element $x \in \mathbb{F}$ and a key ρ as input, and outputs some "tag".

Tools. We will need non-interactive commitments $\mathsf{Com}_{\mathsf{crscm}}(x;k)$, and for clarity we distinguish between these commitments and the "internal" commitments E that is specified by $\mathcal{F}_{\mathsf{cdsg},b}$. We will also need an ID-based simulation-sound NIZK proof system for the following relations: The E-relation, defined wrt the tagging-algorithm E, via $\mathcal{R}_E := \{(\alpha,(\rho,x)) : \alpha = E_\rho(x)\}$, and an additive commitment relation, about a value y being equal to a linear combination of committed values. For a coefficient vector $v \in \mathbb{F}^\kappa$, it is defined as

$$\mathcal{R}_v = \{(\mathsf{crscm}, y, \alpha, (c_i)_{i\in[\kappa]}), (x, \rho, (x_i, k_i)_{i\in[\kappa]} : \\ \forall i \in [\kappa], c_i = \mathsf{Com}_{\mathsf{crscm}}(x_i; k_i), \alpha = E_\rho(x), y = x + \sum_i v_i \cdot x_i\}.$$

To simplify notation, we typically omit the CRS crscm from the subscript of the commitment and from the relations.

6.2 The Basic Protocol

Protocol 2 in Fig. 2 describes a basic DSG protocol that realized $\mathcal{F}_{\mathsf{cdsg},b}$. Intuitively, the protocol $\Pi_{M,E}$ securely computes the mapping $x = (x_1,\ldots,x_n) \mapsto y = Mx$ for an arbitrary security threshold t, except that it allows the adversary to abort outputs that depend on the adversary's inputs.[11] Since the matrix

[11] We emphasize again that we used affine secret sharing in order to let each participant choose its share x_i at random, and have the system publish $y = Mx$ to enable recovering the joint secret. If, instead, we were using secret sharing where $y = Mx = 0$ then the participants would have needed to coordinate their share generation, to ensure that $Mx = 0$.

Protocol 10 (The basic protocol $\Pi_{M,E}$). *We assume a common reference string* $\mathsf{crs} = (\mathsf{crspf}, \mathsf{crscm})$. *Each party* $P_i, i \in [n]$ *locally samples random* (x_i, ρ_i) *and proceeds as follows.*

- ***R1:*** *(Sending Randomizers) For every output* $o \in R_i$ *influenced by* i *and every* $j \in L_{o,-i}$, *party* P_i *samples a random mask* $r_{o,i,j} \xleftarrow{R} \mathbb{F}$ *and a random commitment key* $k_{o,i,j}$, *broadcasts the commitment* $c_{o,i,j} = \mathsf{Com}(r_{o,i,j}; k_{o,i,j})$, *and sends the opening* $(r_{o,i,j}, k_{o,i,j})$ *of the commitment to the* j*th party* P_j *over a private channel.*
- ***R2:*** *(Resolving Private Inconsistencies) For every* $o \in R_i, j \in L_{o,-i}$, *if* P_i *does not receive from* P_j *an opening for the published commitment* $c_{o,j,i}$, *or receives an opening that is inconsistent with* $c_{o,j,i}$, *then* P_i *broadcasts a "complaint"* (j, o). *This complaint asserts that* $r_{o,j,i}$ *and* $k_{o,j,i}$ *should be set to zero and* $c_{o,j,i} = \mathsf{Com}(0; 0)$. *(We assume that* $i, j \in R_o$, *and if this is not the case, ignore the complaint.)*[a]
- ***R3:*** *(Computing shares of the vector* y*: Local Sums and Output Tags) Party* P_i *broadcasts* $\alpha_i = E_{\rho_i}(x_i)$ *together with a NIZK* π_i *for consistency. In addition, for every output* $o \in R_i$, *party* P_i *broadcasts the value*

$$y_{o,i} = x_i + \sum_{j \in L_{o,-i}} r_{o,i,j} - \sum_{j \in L_{o,-i}} r_{o,j,i},$$

  together with a NIZK $\pi_{o,i}$ *that the committed values in* $\alpha_i, (c_{o,i,j}, c_{o,j,i})_{j \in L_{o,-i}}$ *satisfy the above linear equation about* $y_{o,i}$.
- ***Private outputs:*** *The private output of the* i*th party is taken to be its private inputs* (x_i, ρ_i).
- ***Public output:*** *The broadcasted values define the public outputs of the protocol via the following decoding procedure. If, for some party* i, *the proof* π_i *fails to verify, set* $\alpha_i = E_0(0)$ *and replace* π_i *with a valid proof. In addition, initialize* $\mathsf{C}', B = \emptyset$. *For every output* $o \in [m]$*:*
  - ***If*** *there exists* $j \in L_o$ *for which the proof* $\pi_{o,j}$ *fails to verify insert* o *to* B *and* j *to* C'.
  - ***Otherwise****, set* $y_o = \sum_{j \in L_o} y_{o,j}$.

  Set $B, (y_o)_{o \notin B}$ *and* $(\alpha_i)_{i \in [n]}$, *as the public verification information of the DSG. (Treat* C' *as auxiliary output.)*

[a] Note that an adversary can issue false complaints and force $r_{o,j,i}$ and $k_{o,j,i}$ to be zero. In the proof, we show that this is not an issue (essentially since $r_{o,j,i}$ is being used to pad information known to the adversary anyway).

Fig. 2. The basic protocol $\Pi_{M,E}$.

is sparse and each column contains at most d non-zero elements, an adversary that corrupts t parties can only abort at most dt outputs. As a result, if (M, v) is a b-robust $(t_\mathsf{p}, t_\mathsf{c})$-AFS then $\Pi_{M,E}$ realizes $\mathcal{F}_{\mathsf{cdsg},b}$ with security threshold of $t'_\mathsf{p} = \min(t_\mathsf{p}, b/d)$. We also note that the protocol publicly identifies some of the corrupted parties (i.e., the "auxiliary output" C') – a feature that is not needed under the definition of $\mathcal{F}_{\mathsf{cdsg},b}$. To match the syntax of $\mathcal{F}_{\mathsf{cdsg},b}$ we can always

assume that C' is dropped from the output. The following theorem is proved in the full version [3].

Theorem 9. *Suppose that M is a (d, r)-sparse constraint matrix that together with a recovery vector v forms a b-robust $(t_\mathsf{p}, t_\mathsf{c})$-AFS. Then, $\Pi_{M,E}$ t'_p-realizes the functionality $\mathcal{F}_{\mathsf{cdsg},b}$ for $t'_\mathsf{p} = \min(t_\mathsf{p}, b/d)$.*

Remark 9 (The complexity of $\Pi_{M,E}$). The protocol has 3 rounds of interaction and each party sends (either privately or to the BB via broadcast) at most $O(d \cdot r \cdot \max(\kappa, \log|\mathbb{F}|))$ bits where d and r are the maximal number of non-zero elements in a column of M and the maximal number of non-zero elements in a row of M, respectively. Similarly, each party P_i receives at most $O(d \cdot r \cdot \max(\kappa, \log|\mathbb{F}|))$ bits via point-to-point communication and has to read a similar amount of bits from the BB (basically, only the commitments sent by parties that influence an output that is also influenced by P_i). Since $r = d = O(1)$, the communication per party is a constant that does not grow with the number of parties. The computational complexity per party is $O(rd) = O(1)$ cryptographic operations/field operations during the execution of the protocol. The final public decoding costs $O(n)$ downstream communication and $O(n)$ operations and it can be postponed to the recovery phase.

Variants: The above protocol can be tweaked in many ways to optimize different goals. See the full version [3] for a discussion.

6.3 Improving Security by Limiting Aborts

In order to achieve near-threshold results (i.e., an arbitrarily small gap between the privacy τ_p and correctness τ_c thresholds), we need to limit the number of aborts. Recall that the basic protocol $\Pi_{M,E}$ aborts each output that depends on an input from a party that was publicly identified as being corrupted. To limit the number of aborts, we invoke the basic protocol $\Pi_{M,E}$ and then try to recover the aborted outputs by using an additional sub-protocol. (Protocol 3 in Fig. 3.)

Informally, the idea is to remove the set of publicly corrupted parties, and to re-compute the corresponding outputs over the inputs of the other parties as if all the inputs of the publicly corrupted parties were taken to be zero. This approach means that all the outputs $o \in R_{\mathsf{C}'}$ that are affected by publicly corrupted parties (including valid ones) have to be re-computed. Fortunately, re-computing such values is quite simple given the information that was gathered in $\Pi_{M,E}$: All that is needed is to reveal the randomizers $r_{o,i,j}$ that correspond to a pair (i, j) consisting of a party $i \notin \mathsf{C}'$ and a publicly corrupted party $j \in \mathsf{C}'$ (or vice versa). Since each such value was already committed to, and the honest party from the pair knows it, that party can simply open the corresponding commitment.

A potential difficulty is that malicious parties that acted honestly in $\Pi_{M,E}$ and were not detected, may decide not to collaborate in this new sub-protocol. Namely, these parties will not open in the new sub-protocol the commitments that the protocol requires them to open. This behavior will be detected during

the new sub-protocol, the corresponding parties will be added to the set of publicly corrupted parties, and as a result their inputs will be set to zero and additional outputs will need to be computed. This process might cascade over multiple iterations of running the sub-protocol, if in each iteration a new party is identified as being corrupted, and as a result its inputs must be set to zero. Fortunately, it can be shown that the amount of communication is still constant per party. Moreover, if the number of aborts is linear and is equal to $b = \beta n$ for some small constant β, then the round complexity will be constant as well. The resulting protocol, $\Pi_{M,E,b}$, is described in Protocol 3 in Fig. 3. It is parameterized with the number b of outputs that the adversary is allowed to abort (which corresponds to the robustness parameter).

To simplify the presentation, the protocol description ignores communication complexity limitations. It will be also convenient to drop the distinction between online protocol operations and public-decoding operations that will be post-processed after the execution (e.g., as part of the recovery phase) based on publicly available values that appear on the BB. Still, we highlight such public operations by the label "**All:**" that indicates that the following operations can be computed based on public values. We will later explain how to obtain a communication-efficient variant of the protocol.

Analysis. It is not hard to verify that C' contains only corrupted parties, that $B \subseteq R(\mathsf{C}')$, and that if the procedure halts then $|B| \leq b$. We also prove an upper-bound on the number of iterations needed for the procedure to halt.

Claim 12. *The sub-protocol Π_2 halts after at most $1 + |\mathsf{C}| \cdot d/(b+1)$ iterations.*

Proof. At the end of each iteration, if an output o is in B, then there must exist at least one new publicly-corrupted input $i \in L_o$ that influences o. Since such a party i can influence at most d outputs, we discover at least $(b+1)/d$ new corrupt parties in each iteration (except for the last one), and the number of iterations is at most $1 + |\mathsf{C}| \cdot d/(b+1)$. □

Hence, when the robustness parameter is $b = 0$, we need a linear number of iterations, and when $b = \beta n$ for a constant β, only a constant number of $O(1/\beta)$ iterations is needed. In fact, even if $b = 0$, the proof shows that the number of rounds scales linearly with the number of (identifiable) corrupted parties. So an adversary can only slow down the process at the expense of revealing the identities of corrupted parties. (Specifically, in an optimistic execution path where all the parties behave honestly, the above extension adds no overhead.)

Intuitively, the security of the protocol relies on the following observations: (1) The information revealed during Π_2 (i.e., the randomizers adjacent to the publicly corrupted parties) does not violate privacy since it is already known to the adversary; and (2) Assuming that the adversary cannot violate the binding of the commitments, the outputs $(y_o)_{o \notin B}$ are consistent with the inputs $(x_i)_{i \in \mathsf{H}}, (x'_i)_{i \in \mathsf{C}}$ where x'_i is either the witness used to generate π_i (for parties that weren't caught cheating), or zero otherwise. Formally, in the full version [3] we prove the following theorem.

Protocol 11 (The extended protocol $\Pi_{M,E,b}$). *Execute Protocol 10, and compute but do not output the values $\mathsf{C}', B, (y_o)_{o\notin B}, (\alpha_i)_{i\in[n]}$ as in the last step of the protocol. Initialize an empty set $\mathcal{Z}$, and apply the following sub-protocol Π_2 as long as the set B is larger than b:*

1. *Every party $P_i, i \notin \mathsf{C}'$ does the following: For every output $o \in R_i$ and every publicly corrupted $j \in L_{o,-i} \cap \mathsf{C}'$, broadcast all the private randomizers $r_{o,i,j}, r_{o,j,i}$ and their commitment keys $k_{o,i,j}, k_{o,j,i}$ that were sent to/from the corrupted party j. (If these values were sent in the previous iterations there is no need to send them again.) We say that the randomizers are successfully revealed if the opening is consistent with the commitments $c_{o,i,j}, c_{o,j,i}$.*
2. ***All:** For every output o that depends on some publicly corrupted input $j \in \mathsf{C}' \setminus \mathcal{Z}$, **If** for every $i \in L_o \setminus \mathsf{C}'$ and $j \in L_o \cap \mathsf{C}'$ the randomizers $r_{o,i,j}, r_{o,j,i}$ were successfully revealed by P_i, **Then** call o* tentatively ready *and set a* tentative value

$$y'_o = \sum_{i\in L_o\setminus\mathsf{C}'} \left(y_{o,i} - \sum_{j\in L_{o,-i}\cap\mathsf{C}'} r_{o,i,j} + r_{o,j,i} \right).$$

3. ***All:** For every publicly corrupted party $j \in \mathsf{C}'$, **if** all the outputs that are influenced by j are tentatively ready **do**: (a) insert j to $\mathcal{Z}$ and redefine $\alpha_j = E_0(0)$; and (b) update all the affected outputs $o \in R_j$ to be $y_o = y'_o$.*
4. ***All:** Insert to C' every party j that did not successfully reveal one of its randomizers. (Such a party is now publicly corrupted.) Insert to B all the outputs that are influenced by these new publicly-corrupted parties.*

***All: Output** $B, (y_o)_{o\notin B}$ and $(\alpha_i)_{i\notin\mathsf{C}'}$. (Auxiliary output: C'.)*

Fig. 3. The extended protocol $\Pi_{M,E,b}$.

Theorem 13. *Suppose that M is a (d, r)-sparse constraint matrix that together with a recovery vector v forms a b-robust $(t_\mathsf{p}, t_\mathsf{c})$-AFS. Then, the protocol $\Pi_{M,E,b}$ t_p-realizes the functionality $\mathcal{F}_{\mathsf{cdsg},b}$.*

Remark 10 (Reducing the Communication). Let d and r be the maximal number of non-zero elements in a column of M and the maximal number of non-zero elements in a row of M, respectively. Observe that each party needs to communicate at most $d(r-1)$ openings during the protocol (since this is the number of randomizers that are "adjacent" to her). So the upstream communication per party is constant. To obtain constant downstream communication, we split the protocol into an online part and a post-processing public-decoding part. In the online part, we apply the first step of the protocol for $T = 1 + t_\mathsf{p} \cdot d/(b+1)$ iterations while updating the set C'. (Below we show that this can be done with constant downstream complexity.) In the decoding phase, we iteratively repeat over Steps 2–4 while in each iteration i we use the values that were computed in the ith

iteration of the online phase. We terminate the post-processing public-decoding once B is smaller than b, which, by Claim 12, takes at most T iterations.[12]

Let us get back to the online part and analyze the downstream complexity. Assuming that $b = \Omega(n)$, the number of iterations is constant. We will show that the downstream complexity of every party P_i is also constant. Call P_j a neighbor of P_i if they both influence a common output o. Recall that in each iteration P_i has to check, for each of her neighbors P_j, whether P_j publicly cheated, i.e., if $j \in \mathsf{C}'$. Let us denote by C'_k the set of parties that publicly cheated for the first time at the kth iteration where C'_0 is the set of parties that publicly cheated in $\Pi_{M,E}$. At the first iteration, the communication cost of checking if P_j is in C'_0 is constant (since it suffices to check the validity of the proofs sent by P_j during $\Pi_{M,E}$). For $k > 0$, the party P_j is in C'_k if (a) P_j is supposed to open a commitment; and (b) the opening is either invalid or was not sent. Condition (b) is easy to verify with $O(1)$ communication (by accessing the opening and verifying against the commitment). Condition (a) boils down to checking whether P_j has a neighbor that publicly cheated in the $k-1$ iteration. Denoting by c_k the downstream communication needed for checking if a party is in C'_k, we have that $c_k = O(D \cdot c_{k-1})$ where $D = d(r-1)$ is the maximal number of neighbors of a party. It follows that the communication in the kth iteration is $O(D^k)$ which is still constant since the number of rounds is constant. Furthermore, the computational complexity of each party is constant as well. (See the full version [3] for a more detailed description.)

By combining Theorem 13 with Lemma 2 (or Lemma 3 for the case of DKG) and with Corollary 5, we derive the following corollary (formal version of the main theorem).

Corollary 2 (near-threshold DSG and DKG). *Assuming the existence of NIZKs the following holds. For every constants $\tau_{\mathsf{p}} < \tau_{\mathsf{c}}$ and every field $\mathbb{F}$ of size super-polynomial $n^{\omega(1)}$, there exists a protocol that $(\tau_{\mathsf{p}}, \tau_{\mathsf{c}})$ realizes the $\mathcal{F}_{\mathsf{dsg}}$ functionality (resp., $\mathcal{F}_{\mathsf{dkg}}$ functionality) over $\mathbb{F}$ in which each party sends and receives only a constant number of field elements and commitments/NIZKs and computes a constant number of arithmetic and cryptographic operations. Moreover, the sharing phase has a constant number of rounds.*

Acknowledgements. We thank Noga Ron-Zewi for helpful discussions and the anonymous TCC reviewers for their comments.

[12] In the online phase, we do not update the size of B since this is communication expensive, and therefore just iterate T times.

References

1. Applebaum, B., Kachlon, E.: Sampling graphs without forbidden subgraphs and unbalanced expanders with negligible error. SIAM J. Comput. **52**(6), 1321–1368 (2023). https://doi.org/10.1137/22M1484134 https://doi.org/10.1137/22m1484134 https://doi.org/10.1137/22m1484134
2. Applebaum, B., Nir, O., Pinkas, B.: How to recover a secret with o(n) additions. In: Advances in Cryptology - CRYPTO 2023 - 43rd Annual International Cryptology Conference, CRYPTO 2023, Santa Barbara, CA, USA, August 20-24, 2023, Proceedings, Part I, pp. 236–262 (2023). https://doi.org/10.1007/978-3-031-38557-5_8
3. Applebaum, B., Pinkas, B.: Distributing keys and random secrets with constant complexity. Cryptology ePrint Archive, Paper 2024/876 (2024). https://eprint.iacr.org/2024/876
4. Bacho, R., Lenzen, C., Loss, J., Ochsenreither, S., Papachristoudis, D.: GRandLine: adaptively secure DKG and randomness beacon with (log-)quadratic communication complexity. Cryptology ePrint Archive, Paper 2023/1887 (2023). https://doi.org/10.1145/3658644.3690287, https://eprint.iacr.org/2023/1887
5. Beerliová-Trubíniová, Z., Hirt, M.: Perfectly-secure MPC with linear communication complexity. In: Theory of Cryptography Conference, TCC 2008. LNCS, vol. 4948, pp. 213–230. Springer (2008)
6. Bogdanov, A., Guo, S., Komargodski, I.: Threshold secret sharing requires a linear-size alphabet. Theory Comput. **16**, 1–18 (2020). https://doi.org/10.4086/TOC.2020.V016A002, https://doi.org/10.4086/toc.2020.v016a002
7. Canetti, R.: Universally composable security. J. ACM **67**(5), 28:1–28:94 (2020). https://doi.org/10.1145/3402457
8. Canetti, R., Gennaro, R., Jarecki, S., Krawczyk, H., Rabin, T.: Adaptive security for threshold cryptosystems. In: CRYPTO '99. LNCS, vol. 1666, pp. 98–115. Springer (1999)
9. Capalbo, M.R., Reingold, O., Vadhan, S.P., Wigderson, A.: Randomness conductors and constant-degree lossless expanders. In: Reif, J.H. (ed.) Proceedings on 34th Annual ACM Symposium on Theory of Computing, May 19-21, 2002, Montréal, Québec, Canada, pp. 659–668. ACM (2002)
10. Cascudo, I., David, B.: Publicly verifiable secret sharing over class groups and applications to DKG and YOSO. Cryptology ePrint Archive (2023)
11. Cramer, R., Damgård, I.B., Döttling, N., Fehr, S., Spini, G.: Linear Secret sharing schemes from error correcting codes and universal hash functions. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015. LNCS, vol. 9057, pp. 313–336. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46803-6_11
12. Das, S., Pinkas, B., Tomescu, A., Xiang, Z.: Distributed randomness using weighted VRFs. Cryptology ePrint Archive, Paper 2024/198 (2024). https://eprint.iacr.org/2024/198, https://eprint.iacr.org/2024/198
13. DKGPG developers: DKGPG: Distributed Key Generation for Pretty Good Privacy (PGP) (2017). https://www.nongnu.org/dkgpg/ Accessed 14 Feb 2024
14. drand: drand: Distributed Randomness Beacon Service (2020). https://github.com/drand/drand. Accessed 14 Feb 2024

15. Fitzi, M., Hirt, M., Maurer, U.M.: Trading correctness for privacy in unconditional multi-party computation (extended abstract). In: Krawczyk, H. (ed.) Advances in Cryptology - CRYPTO '98, 18th Annual International Cryptology Conference, Santa Barbara, California, USA, August 23-27, 1998, Proceedings. Lecture Notes in Computer Science, vol. 1462, pp. 121–136. Springer (1998). https://doi.org/10.1007/BFB0055724
16. Fouque, P.A., Stern, J.: One round threshold discrete-log key generation without private channels. In: International Workshop on Public Key Cryptography, pp. 300–316. Springer (2001)
17. Gallager, R.G.: Low-density parity-check codes. IRE Trans. Inf. Theory **8**(1), 21–28 (1962)
18. Gennaro, R., Jarecki, S., Krawczyk, H., Rabin, T.: Secure applications of pedersen's distributed key generation protocol. In: Topics in Cryptology - CT-RSA 2003, The Cryptographers' Track at the RSA Conference 2003. LNCS, vol. 2612, pp. 373–390. Springer (2003)
19. Gennaro, R., Jarecki, S., Krawczyk, H., Rabin, T.: Secure distributed key generation for discrete-log based cryptosystems. J. Cryptol. **20**(1), 51–83 (2007). https://doi.org/10.1007/S00145-006-0347-3
20. Goldreich, O.: The Foundations of Cryptography - Volume 2: Basic Applications. Cambridge University Press (2004). https://doi.org/10.1017/CBO9780511721656
21. Groth, J.: Non-interactive distributed key generation and key resharing. Cryptology ePrint Arch. (2021)
22. Groth, J., Shoup, V.: Design and analysis of a distributed ECDSA signing service. Cryptology ePrint Archive, Paper 2022/506 (2022). https://eprint.iacr.org/2022/506
23. Gurkan, K., Jovanovic, P., Maller, M., Meiklejohn, S., Stern, G., Tomescu, A.: Aggregatable distributed key generation. In: Canteaut, A., Standaert, F.-X. (eds.) EUROCRYPT 2021. LNCS, vol. 12696, pp. 147–176. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-77870-5_6
24. Håstad, J., Impagliazzo, R., Levin, L.A., Luby, M.: A pseudorandom generator from any one-way function. SIAM J. Comput. **28**(4), 1364–1396 (1999). https://doi.org/10.1137/S0097539793244708
25. Ishai, Y., Ostrovsky, R., Zikas, V.: Secure multi-party computation with identifiable abort. In: Garay, J.A., Gennaro, R. (eds.) CRYPTO 2014. LNCS, vol. 8617, pp. 369–386. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-44381-1_21
26. Kate, A., Mangipudi, E.V., Mukherjee, P., Saleem, H., Thyagarajan, S.A.K.: Non-interactive VSS using class groups and application to DKG. Cryptology ePrint Arch. (2023)
27. Katz, J.: Round optimal robust distributed key generation. IACR Cryptol. ePrint Arch. **2023**, 1094 (2023). https://eprint.iacr.org/2023/1094
28. Katz, J., Ostrovsky, R., Rabin, M.O.: Identity-based zero knowledge. In: Blundo, C., Cimato, S. (eds.) Security in Communication Networks, 4th International Conference, SCN 2004, Amalfi, Italy, September 8-10, 2004, Revised Selected Papers. Lecture Notes in Computer Science, vol. 3352, pp. 180–192. Springer (2004). https://doi.org/10.1007/978-3-540-30598-9_13
29. Komlo, C., Goldberg, I., Stebila, D.: A formal treatment of distributed key generation, and new constructions. IACR Cryptol. ePrint Arch. 292 (2023). https://eprint.iacr.org/2023/292

30. Mosheiff, J., Resch, N., Ron-Zewi, N., Silas, S., Wootters, M.: LDPC codes achieve list decoding capacity. In: Irani, S. (ed.) 61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020, pp. 458–469. IEEE (2020). https://doi.org/10.1109/FOCS46700.2020.00050
31. Naor, M.: Bit commitment using pseudo-randomness. In: CRYPTO '89. LNCS, vol. 435, pp. 128–136. Springer (1989)
32. Orbs-Network: DKG-on-EVM: A Distributed Key Generation Protocol for Ethereum Virtual Machine (2018). https://github.com/orbs-network/dkg-on-evm. Accessed 14 Feb (2024)
33. Pedersen, T.P.: Non-interactive and information-theoretic secure verifiable secret sharing. In: CRYPTO '91. LNCS, vol. 576, pp. 129–140. Springer (1991)
34. De Santis, A., Di Crescenzo, G., Ostrovsky, R., Persiano, G., Sahai, A.: Robust non-interactive zero knowledge. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 566–598. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-44647-8_33
35. Schindler, P.: ethDKG: an Ethereum-based DKG Implementation (2020). https://github.com/PhilippSchindler/ethdkg Accessed 14 Feb 2024
36. Schindler, P., Judmayer, A., Stifter, N., Weippl, E.: EthDKG: distributed key generation with ethereum smart contracts. Cryptology ePrint Archive, Paper 2019/985 (2019). https://eprint.iacr.org/2019/985
37. Shamir, A.: How to share a secret. Commun. ACM **22**(11), 612–613 (1979)
38. Shrestha, N., Bhat, A., Kate, A., Nayak, K.: Synchronous distributed key generation without broadcasts. IACR Cryptol. ePrint Arch. 1635 (2021). https://eprint.iacr.org/2021/1635
39. Stamer, H.: Gnosis DKG: A Distributed Key Generation Library (2018). https://github.com/gnosis/dkg. Accessed 14 Feb 2024
40. Vadhan, S.P.: Pseudorandomness. Found. Trends Theor. Comput. Sci. **7**(1-3), 1–336 (2012). https://doi.org/10.1561/0400000010
41. Zichron, L.: Locally Computable Arithmetic Pseudorandom Generators. Master thesis, Tel Aviv University (2017). https://www.bennyapplebaum.sites.tau.ac.il/_files/ugd/f706bf_501515c9cd7744c498935684bd1648a2.pdf

Reducing the Share Size of Weighted Threshold Secret Sharing Schemes via Chow Parameters Approximation

Oriol Farràs(✉) and Miquel Guiot

Universitat Rovira i Virgili, Tarragona, Spain
{oriol.farras,miquel.guiot}@urv.cat

Abstract. A secret sharing scheme is a cryptographic primitive that allows a dealer to share a secret among a set of parties, so that only authorized subsets of them can recover it. The access structure of the scheme is the family of authorized subsets.

In a weighted threshold access structure, each party is assigned a weight according to its importance, and the authorized subsets are those in which the sum of their weights is at least the threshold value. For these access structures, the share size of the best known secret sharing schemes is either linear on the weights or quasipolynomial on the number of parties, which leads to long shares, in general.

In certain settings, a way to circumvent this efficiency problem is to approximate the access structure by another one that admits more efficient schemes. This work is dedicated to the open problem posed by this strategy: Finding secret sharing schemes with a good tradeoff between the efficiency and the accuracy of the approximation.

We present a method to approximate weighted threshold access structures by others that admit schemes with small shares. This method is based on the techniques for the approximation of the Chow parameters developed by De et al. [Journal of the ACM, 2014]. Our method provides secret sharing schemes with share size $n^{1+o(1)}$, where n is the number of parties, and whose access structure is *close* to the original one. Namely, in this approximation the condition of being authorized or not is preserved for almost all subsets of parties.

In addition, we apply the recent results on computational secret sharing schemes by Applebaum et al. [STOC, 2023] to construct computational secret sharing schemes whose share size is polylogarithmic in the number of parties.

Keywords: Secret sharing scheme · weighted threshold access structure · threshold cryptography · Chow parameters

1 Introduction

A secret sharing scheme is a cryptographic primitive that allows a dealer to share a secret among a set of parties in such a way that only some subsets of parties,

E. Boyle and M. Mahmoody (Eds.): TCC 2024, LNCS 15367, pp. 517–547, 2025.
https://doi.org/10.1007/978-3-031-78023-3_17

called authorized, can recover the secret. The family of authorized subsets is called the access structure of the scheme.

Secret sharing schemes were introduced independently by Shamir [Sha79] and Blakley [Bla79] in 1979, when they presented methods for constructing secret sharing schemes for threshold access structures. In these schemes, the authorized subsets are those whose size is at least a given threshold. Interestingly, these constructions are ideal, in the sense that the length of the share received by each party is equal to the length of the secret. This is the best situation we can hope for [KGH83].

Secret sharing schemes are used as a building box of cryptographic protocols for secure multiparty computation and threshold encryption, for example [Bei11]. In many of these applications, schemes with threshold access structures are convenient. However, there are situations that require more general access structures. This is the case when using secret sharing schemes for protocols in the proof-of-stake model, where each validator has a stake that depends on the amount of coins it has and the importance in the system is proportional to the stake, resulting in a non-uniformly distribution [KRDO17, BCC+21, DPTX24]. Another common case can be found in the stock exchange, where the shares of a company are non-uniform distributed among the shareholders and the weight of their vote depends on the share. In such cases, there is a need of secret sharing schemes with *weighted threshold access structures* (WTASs). In these access structures, each party is assigned a weight according to its importance and the authorized subsets are those in which the sum of their weights is at least the threshold value.

The share size of best known secret sharing schemes for general weighted threshold access structures is either linear on the weights [Sha79] or quasipolynomial on the number of parties [BW06], which may lead to long shares, in general. Recently, there were proposals trying to circumvent this efficiency problem by approximating the weights by smaller ones or by relaxing the privacy and correctness requirements [BHS23, GJM+23, DPTX24, TF24], considering approximations of weighted threshold access structures. This work is dedicated to the open problem posed by this strategy: Finding secret sharing schemes with a good tradeoff between the efficiency and the accuracy of the approximation.

1.1 Our Results

In this work, we present a method that, given a weighted threshold access structure Γ, it provides a secret sharing scheme with *small* shares that realizes a weighted threshold access structure Γ' that is *close* to Γ. For that, we translate our problem into a problem of approximation of monotone Linear Threshold Functions (LTF) and, in this richer complexity theory framework, we develop techniques for the approximation of these functions. The approximation error we consider is the fraction of sets in which the access structures differ. Namely, the approximation error is $d/2^n$, where d is the number of subsets that are either in Γ but not in Γ', or viceversa. Our main result is as follows.

Theorem 1.1 (Informal). *For any weighted threshold access structure Γ on n parties, there exists a secret sharing scheme with share size $n^{1+o(1)}$ whose access structure is $o(1)$-close to Γ.*

The result is constructive, and we provide an efficient algorithm that, given a weighted threshold access structure Γ, it outputs another weighted threshold access structure Γ' whose weights are much smaller, the distance between Γ and Γ' is small, and the hierarchy among parties is preserved. Then, with the new weights, it is enough to use the construction of Shamir [Sha79] for weighted threshold access structures to obtain the secret sharing scheme. The approximation error is $o(1)$, which means that for almost all subsets of parties the condition of being authorized or not is not modified in the approximation. Moreover, our scheme is linear for finite fields $\mathbb{F}$ with size $\log |\mathbb{F}| = \Omega(\log n)$.

Previous best solutions had different trade-offs, illustrated in Fig. 1. In some previous works, the bounds are for the total share size, and not for the share size. Because of that, we have decided to present the bounds for the total share size, that is, the sum of the size of the shares of all parties. In our information-theoretic construction, it is simply $n \cdot n^{1+o(1)}$.

Our main technical contribution is the use of Chow parameters in the construction of secret sharing schemes for weighted threshold access structures. In this regard, the approximation of linear threshold functions by Chow parameters is a problem that has been thoroughly studied in the past, giving positive and negative results (see, for example, [Ser06, OS11, DDFS14]). In particular, we modify an algorithm for the approximation of Chow parameters by De et al. [DDFS14] to solve this problem in the monotone case. That is, to guarantee that all the weights of the approximation are positive. With that, we can approximate weighted threshold access structures with small weights. In this context, our main result is the following.

Theorem 1.2 (Informal). *Let f be a monotone LTF. For any $0 < \varepsilon$ there exists an ε-close monotone LTF g represented by an integer vector with norm $\sqrt{n} \cdot \text{quasipoly}\left(\frac{1}{\varepsilon}\right)$.*

Previous proposals also consider reductions of the weights, by scaling or rounding them [BHS23, DPTX24, TF24], leading to approximated access structures. These techniques can be replaced by the Chow parameters approximation technique developed in this work. Furthermore, we give a lower bound on the size of the weights obtained by any approximation technique and show that our strategy is nearly optimal for this task. This is summarized in the following statement.

Theorem 1.3 (Informal). *Let $n \in \mathbb{N}$. For any $\epsilon \in (0, \frac{1}{10})$ there exists a monotone LTF f over n variables such that any monotone LTF that is ε-close to f has integer weights of size $\Omega\left(\sqrt{n}, \text{quasipoly}\left(\frac{1}{\varepsilon}\right)\right)$.*

If, instead of constructing schemes in the information-theoretic setting, we consider computational assumptions, it is possible to build more efficient secret

	Total Share Size	Access structure	Error	Privacy
[Sha79]	$W \log W = 2^{O(n \log n)}$	WTAS	0	Perfect
[BW06]	$n^{O(\log n)}$	WTAS	0	Perfect
[GJM$^+$23]	$W = 2^{O(n \log n)}$	$(t, t + \Omega(\lambda))$-ramp WTAS	-	$2^{-\lambda}$-Stat.
[BHS23] Rounding, [TF24]	$O\left(\frac{n}{\beta-\alpha}\right)$	$(\alpha W, \beta W)-$ramp WTAS	-	Perfect
[BHS23] BS Channels	$n \cdot \max\left\{\lambda^2, \text{poly}(\frac{1}{\beta-\alpha})\right\}$	$(\alpha W, \beta W)-$ramp WTAS	-	$2^{-\lambda}$-Stat.
Theorem 1.1	$n^{2+o(1)}$	WTAS	$o(1)$	Perfect
[BW06]	$\text{poly}(n)$	WTAS	0	Comp.
[ABI$^+$23,BW06] Theorem 1.4	$n \cdot \text{polylog}(n)$	WTAS	0	Comp.

Fig. 1. Summary of secret sharing schemes for weighted threshold access structures. In the second column, we give an upper bound on the total share size of the schemes, considering secrets of 1-bit. We use the fact that the best upper bound for the size of the weights in weighted threshold access structures is $W = 2^{O(n \log n)}$ [Mur71], which is tight [Hås94]. In the fifth, sixth and eighth rows, the bound is obtained by multiplying the bound on the individual share size by n. In the *Access structure* column, we distinguish between perfect WTAS and ramp WTAS. In the *Error* column, we bound the error in the approximation of weighted threshold access structures. If the scheme realize exactly the access structure, we set the error to be 0. Ramp WTASs can also be used to approximate perfect WTAS, but we could not find any non-trivial upper bound on the error, as discussed in the full version of this work [FG24]. In the *Privacy* column, we distinguish between perfect, statistical, and computational security.

sharing schemes. This can be done by directly applying the recent results on succinct computational secret sharing schemes by Applebaum et al. [ABI+23] to the polynomial size monotone circuits for weighted threshold functions constructed by Beimel and Weinreb [BW06]. More in detail, the existence of an efficient Projective Pseudorandom Generator (pPRG) allows to obtain a computational secret sharing scheme for weighted threshold access structures with polylogarithmic share size in the number of parties. This is stated in the following theorem.

Theorem 1.4 (Informal). *Under the subexponential RSA assumption, any weighted threshold access structure over n parties admits a computational secret sharing scheme where the size of the shares is* $\text{polylog}(n)$ *and the size of the public information is* $\text{poly}(n)$.

Furthermore, we show how to improve the result of Theorem 1.4 by combining it with our approximation technique. In particular, we can give a better upper bound at the cost of considering an $o(1)$-close weighted threshold access structure.

Open Questions. Our results leave several open questions about the search of better secret sharing schemes for weighted threshold access structures. The schemes presented in this work can be improved in the two main stages, which are the approximation of the weighted threshold access structures and the construction of schemes for small weights. In the first stage, the existing approximation techniques have limitations that are similar to the ones in [DDFS14], whose bounds are close to the optimal in the worst case. However, it could be that the weight bounds could be improved in the average case, or for interesting distributions of weights.

For certain practical applications, it would be useful to find good approximations according to a ramp criteria in terms of the original weights and threshold t: Guarantee that subsets of weight at least $t_1 > t$ are authorized in the new access structure, while those of weight smaller than $t_2 < t$ are not authorized. We did not find a way to give valuable thresholds t_1 and t_2 without assuming distributions on the weights or decreasing a lot the approximation error (making the upper bound of shares much bigger). Despite that, we discuss some relations between these notions in Sect. 3.2.

In the second stage, our approach is simpler, because we directly apply existing techniques that take benefit of a short description of the access structure by means of weights or monotone circuits: for the information-theoretic scheme we use the virtualization technique from Shamir [Sha79], and for the computational scheme we apply the results from Applebaum et al. [ABI+23]. It is natural to expect improvements in this stage by using alternative descriptions of the access structure with more complex gates [LV18], with hierarchical access structures [FP12], with wiretap techniques [BHS23], or with ad-hoc linear schemes.

Improving the upper and lower bounds on the share size for weighted threshold access structures is still the most important open problem in this area. The current bounds are far: The best lower bound is $\Omega(\sqrt{n})$, while the best upper bound is $n^{O(\log n)}$. On the positive side, we have characterizations of the ideal weighted threshold access structures [BW06,FP12]. However, we do not know how to take advantage of this to find a useful characterization of access structures that admit schemes with polynomial share size.

1.2 Our Techniques

Given a weighted threshold access structure Γ, our main objective is to construct another weighted threshold access structure Γ' that is close to Γ and whose weights are small. This is because there exist weighted threshold secret sharing constructions whose share size is linear on the weights.

With this in mind, since every access structure can be described by a monotone Boolean function, the starting point of our work is to translate the problem

of approximating weighted threshold access structures to the problem of approximating monotone Boolean functions. The distance between Boolean functions is defined as the number of inputs in which they differ, and we say that two functions are ε-close if the fraction of inputs they differ is ε.

In the case of a weighted threshold access structure defined by a threshold T and a vector of positive weights $\boldsymbol{w} = (w_1, \ldots, w_n)$ assigned to the parties, the access structure is determined by a monotone Boolean function of the form

$$f(\boldsymbol{x}) = \text{sign}(\boldsymbol{w} \cdot \boldsymbol{x} - T),$$

which are known as monotone LTFs in the context of complexity theory.

We can therefore study weighted threshold access structures from the perspective of complexity theory simply by considering the monotone LTFs assigned to them. More in detail, our proposal consists in approximating Γ by reducing the weights and the threshold of its associated monotone LTF f. We do this by taking advantage of the description of f in terms of its Chow parameters. This procedure is summarized in Fig. 2 and can be done in five steps.

1. Consider the monotone LTF f associated to Γ.
2. Compute the Chow parameters $\boldsymbol{\chi}_f$ of f.
3. Find the Chow parameters $\boldsymbol{\chi}_g$ of a monotone LTF g that is ε-close to f and has small weights.
4. Construct the monotone LTF g from $\boldsymbol{\chi}_g$.
5. Consider the weighted threshold access structure Γ' associated to g.

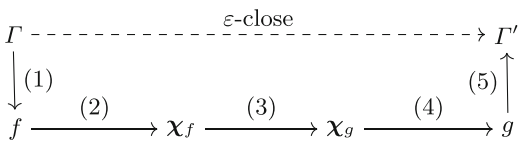


Fig. 2. Procedure for approximating any weighted threshold access structure.

Steps (1), (2), and (5) of Fig. 2 are immediate. For this reason, throughout this work we skip steps (1) and (5) and we deal directly with the monotone LTFs associated to the access structures. All the effort remains in deriving steps (3) and (4). To do so, we adapt the results of De et al. [DDFS14], in which they construct an approximate LTF with smaller weights by solving a problem related to the Chow parameters, described next.

The Chow Parameters Problem. Any Boolean function can be uniquely expressed as a real multilinear polynomial whose degree-0 and degree-1 coefficients are known as the Chow parameters [O'D14]. The notion of the Chow parameters is of greater importance in the case of LTFs, since they uniquely

determine LTFs within the space of all Boolean functions [Cho61] and, when the function is monotone, these parameters quantify the influence of each variable on the output result. In this context, the Chow parameters problem consists in efficiently reconstructing a LTF from its Chow parameters.

De et al. [DDFS14] give a solution to the approximate version of the Chow parameters problem by constructing a LTF that is close to the exact one. More specifically, they design an iterative algorithm that starts with an approximation of the Chow parameters and, step by step, it modifies the parameters until the desired LTF is obtained. Moreover, their construction has the advantage that the resulting LTF has weights that depend sublinearly in the input length and quasipolynomially in the error.

In this work, we adapt the construction of De et al. [DDFS14] to the monotone setup to guarantee that the resulting LTF is monotone. Moreover, we also analyse the running time of the procedure to ensure its efficiency.

Secret Sharing Schemes Construction. Once we have derived a low-weight approximator for the original access structure, we still need to construct the information-theoretic and the computational secret sharing schemes. In both cases, we apply already known constructions that allow us to maximize the benefits of having an approximate weighted threshold access structure with small weights.

In the case of the information-theoretic scheme, we use Shamir's virtualization technique [Sha79] . The total share size of the resulting scheme is $W \log W$, where W is the total weight, which we know that is small due to the approximation procedure.

In the computational setting, it suffices to apply the recent work of Applebaum et al. [ABI+23] that introduces a new cryptographic primitive known as Projective Pseudorandom Generator (pPRG). They use it to obtain secret sharing schemes for monotone circuits of polynomial size in which the size of the shares is polylogarithmic in the number of gates. In more detail, combining this construction with the existence of a monotone circuit of polynomial size for any weighted threshold access structure [BW06] yields a scheme with polylogarithmic shares. Furthermore, we use our approximation technique to reduce the size of the public information.

In both schemes, we also tune the parameters in order to find a good trade-off between the accuracy of the approximation and the share size. In particular, we perform a fine-grained analysis of the size of the Chow parameters to obtain secret sharing schemes with even smaller share size.

1.3 Related Work

In this section, we restrict the discussion to the previous works on secret sharing schemes for weighted threshold access structures.

In 1979, Shamir [Sha79] and Blakley [Bla79] presented the first secret sharing schemes for threshold access structures. Shamir also presented a way to realize

weighted threshold access structures with threshold schemes via virtualization: Given the weights w_i and the threshold t, the dealer treats party i as w_i different parties, sending w_i different shares of the t-threshold scheme to party i. This technique gives a scheme with total share size $O(W \log W)$ for any access structure, where W is the sum of the weights.

Weighted threshold access structures are a specific kind of hierarchical access structures. In these access structures, parties are partitioned into clusters that are hierarchically ordered, being the parties in higher levels more powerful than the ones in lower levels. Simmons [Sim88] considered some hierarchical access structures, and Brickell [Bri89] found ideal schemes for them, providing new tools for the construction of linear schemes. By using different kinds of polynomial interpolation, Tassa [Tas07], and Tassa and Dyn [TD09] proposed constructions of ideal secret sharing schemes for some kinds of hierarchies. Beimel, Tassa and Weinreb [BTW08] presented a characterization of the ideal weighted threshold access structures, generalizing some partial results in [MPSV99, PS00]. Farràs and Padró [FP12] characterized the family of ideal hierarchical access structures, and presented ideal schemes for them. That work included an alternative characterization of ideal weighted threshold access structures. Characterizations in [BTW08, FP12] use the connections between ideal access structures and matroids by Brickell and Davenport [BD91]. Recently, Mo [Mo23] and Padró [Pad24] showed that ideal hierarchical access structures are connected to lattice path matroids, and Chen, Tang and Lin presented an efficient method to construct ideal hierarchical schemes [CTL22]. Beyond the ideal case, the characterization of weighted threshold access structures that admit efficient schemes is open.

Beimel and Weinreb [BW06] proved that all weighted threshold access structures admit secret sharing schemes with share size $n^{O(\log(n))}$. This is done by first building a monotone circuit with logarithmic depth and polynomial size that describes the access structure, and then converting this circuit into a scheme. The upper bound is obtained by using the fact that every weighted threshold access structure admits an equivalent description with weights that are at most exponential [Mur71]. Taking into account that most weighted threshold access structures require weights of exponential size [SB91], this is the best general construction known to date. This result reveals that weighted threshold access structures are in a *privileged* position from the efficiency point of view, because most of the general access structures require linear schemes of normalized share size $2^{n/3+o(n)}$ [BF20]. The best lower bound on the information ratio is $\Omega(\sqrt{n})$, and it was found by Padró and Sáez in [PS00] analyzing weighted threshold access structures where there are only two possible weights. This is also the best lower bound on the share size.

In the computational side, Beimel and Weinreb [BW06] obtained an even more efficient scheme by applying Yao's technique [Yao89] for monotone circuits. Under the assumption of the existence of secure one-way functions, their computational scheme produces shares and public information of polynomial size in the number of parties. Furthermore, under the stronger assumption that subex-

ponentially secure one-way functions exist the share size in their construction can be further reduced to be polylogarithmic in the number of parties.

Recently, Benhamouda, Halevi, and Stambler [BHS23], Garg et al. [GJM+23], and Tonkikh and Freitas [TF24] explored a relaxed weighted threshold model, considering *ramp* weighted threshold access structures. In these access structures, there are privacy and correctness thresholds, and each party has a single weight. This setting is less restrictive and admits schemes that are not perfect, allowing a reduction of the share size. The schemes in [BHS23] have a privacy threshold αW and a reconstruction threshold βW for some $0 < \alpha < \beta < 1$. In this setup, their first proposal is based on a rounding technique of weights that leads to a share size of $\frac{n}{\beta-\alpha}$. For the second construction, they establish an interesting connection between wiretap channels and secret sharing schemes. Choosing specific parameters, it is possible to get a scheme with $2^{-\lambda}$-statistical security and total share size $n \cdot \max\left\{\lambda^2, \text{poly}(1/(\alpha-\beta))\right\}$.

The construction of Garg et al. [GJM+23] uses as a primitive a scheme whose security is guaranteed by the Chinese Reminder Theorem [Mig83]. The privacy is statistical, and the gap between privacy and reconstructions thresholds depends on the security parameter λ. With this security relaxation, it is possible to improve the share size bound from Shamir's scheme, obtaining $O(W)$. The resulting scheme is not linear, but there are still ways to use it as a building block [GJM+23].

Tonkikh and Freitas [TF24] present a transformation to map large real weights into smaller integer weights for a wide variety of weighted distributed protocols. They propose a method to find a ramp weighted threshold access structure with small weights where the new authorized and forbidden subsets were already authorized and forbidden, respectively. The size of the new shares can be decreased by increasing the gap.

The main difference of our approach with respect to the previous ones is that we are focused in finding efficient schemes with access structures that approximate the desired one, controlling the approximation error. In particular, we limit the number of inputs where the approximated access structure differs from the original one.

1.4 Organization

In Sect. 2 we lay out the preliminaries on the analysis of Boolean functions and secret sharing schemes. In Sect. 3 we present the technique for approximating monotone LTFs, obtaining Theorem 1.2 and proving the optimality of its bounds, i.e. Theorem 1.3. In Sect. 4 we construct the information-theoretic secret sharing scheme of Theorem 1.1, while in Sect. 5 we construct the computational secret sharing scheme of Theorem 1.4. Deferred proofs and details can be found in the full version of this work [FG24].

2 Preliminaries

Notation. We notate $\mathbb{N}$ and $\mathbb{R}_+$ for the sets of the non-negative integer and real numbers, respectively. For $n \in \mathbb{N}$, we denote the set $\{1, \ldots, n\}$ as $[n]$. For a set S, we denote its cardinal as $|S|$. We denote vectors $\boldsymbol{x}$ using bold symbols, their i-th coordinates as x_i, their Euclidean norm as $\|\boldsymbol{x}\| = \sqrt{x_1^2 + \ldots + x_n^2}$, and their support as $\mathrm{supp}(\boldsymbol{x}) = \{i \in [n] \ : \ x_i \neq 0\}$. The unary vector is denoted by $\mathbf{1}^n \in \mathbb{R}^n$ and the zero vector is denoted by $\mathbf{0}^n$. For any vectors $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$, we denote its scalar product as $\boldsymbol{x} \cdot \boldsymbol{y}$, its Hamming distance as $\mathrm{dist}_{\mathrm{Ham}}(\boldsymbol{x}, \boldsymbol{y}) = |\{i \in [n] \ : \ x_i \neq y_i\}|$, and we say that $\boldsymbol{x} \leq \boldsymbol{y}$ if and only if $x_i \leq y_i$ for all $i \in [n]$. For $x \in \{0,1\}$, $\overline{x}$ denotes its complementary, and for $\boldsymbol{x} \in \{0,1\}^n$, we set $\boldsymbol{x}^{\oplus i} = (x_1, \ldots, x_{i-1}, \overline{x_i}, x_{i+1}, \ldots, x_n)$.

Next, we define three functions over the reals. We set $\mathrm{sign} : \mathbb{R} \to \{-1, 1\}$ as the function with $\mathrm{sign}(x) = 1$ if and only if $x \geq 0$; we set $\mathrm{sign}_0 : \mathbb{R} \to \{0, 1\}$ as the function with $\mathrm{sign}_0(x) = 1$ if and only if $x \geq 0$; and we set $P_1 : \mathbb{R} \to [-1, 1]$ as the function with $P_1(x) = x$ if $x \in [-1, 1]$ and $P_1(x) = \mathrm{sign}(x)$ otherwise. For $\boldsymbol{x}, \boldsymbol{w} \in \mathbb{R}^n$ and $\sigma \in \mathbb{R}$ we define

- $\mathrm{WTF}(\boldsymbol{w}, \sigma)(\boldsymbol{x}) = \mathrm{sign}_0(\boldsymbol{w} \cdot \boldsymbol{x} - \sigma)$,
- $\mathrm{LTF}(\boldsymbol{w}, \sigma)(\boldsymbol{x}) = \mathrm{sign}(\boldsymbol{w} \cdot \boldsymbol{x} - \sigma)$, and
- $\mathrm{LBF}(\boldsymbol{w}, \sigma)(\boldsymbol{x}) = P_1(\boldsymbol{w} \cdot \boldsymbol{x} - \sigma)$.

Throughout this work, all probability distributions $\mathbf{P}$ assume picking elements uniformly at random.

2.1 Analysis of Boolean Functions

In this section, we introduce all the definitions and technical results about Boolean functions we need. First, we focus on the families of weighted and linear threshold functions and their properties. Then, we state some results about the Chow parameters. Finally, we introduce the notion of distance between Boolean functions and relate it with the Chow parameters. The statements and proofs of the switching lemmas, and the proofs of Lemma 2.13 and Lemma 2.18 can be found in [FG24].

Weighted and Linear Threshold Functions. We start by presenting the main building block of our construction: weighted threshold functions.

Definition 2.1 (Weighted Threshold Function). *Let $\boldsymbol{w} \in \mathbb{R}^n$ and $\sigma \in \mathbb{R}$. A Weighted Threshold Function (WTF) is a Boolean function $f : \{0,1\}^n \to \{0,1\}$ of the form $f(\boldsymbol{x}) = \mathrm{WTF}(\boldsymbol{w}, \sigma)(\boldsymbol{x})$. The vector $(\boldsymbol{w}, \sigma)$ is said to represent f, while the vector $\boldsymbol{x} \in \{0,1\}^n$ is said to be* authorized *(resp.* forbidden*) if $f(\boldsymbol{x}) = 1$ (resp. $f(\boldsymbol{x}) = 0$).*

Among all WTFs, we are interested in those that are monotone because they are in one-to-one correspondence with weighted threshold access structures. In this regard, the following remark gives a characterization of monotone WTFs.

Remark 2.2. A WTF given by $f(\boldsymbol{x}) = \mathrm{WTF}(\boldsymbol{w}, \sigma)(\boldsymbol{x})$ is monotone if and only if there exists a representation of f with $w_i \geq 0$ for any $i \in [n]$. Indeed, if f is monotone increasing and $w_i < 0$, then f does not depend on x_i, and we can set its weight to 0.

An important result about monotone WTFs is that they can be computed by polynomial size logarithmic depth monotone circuits of unbounded fan-in [BW06].

Theorem 2.3 ([BW06])**.** *Every weighted threshold function is in* $m\mathcal{AC}^1$.

It is also usual to define Boolean functions by taking $\{-1,1\}^n$ as domain instead of $\{0,1\}^n$. Indeed, one can pass from one domain to the other by considering the bijection $\phi(x) = (-1)^x$ for $x \in \{0,1\}$ and extending it naturally to $\{0,1\}^n$. For this reason, we now define linear threshold functions, which are an analogue of WTF in the $\{-1,1\}^n$ domain.

Definition 2.4 (Linear Threshold Function). *Let* $\boldsymbol{w} \in \mathbb{R}^n$ *and* $\sigma \in \mathbb{R}$. *A Linear Threshold Function (LTF) is a Boolean function* $f : \{-1,1\}^n \to \{-1,1\}$ *of the form* $f(\boldsymbol{x}) = \mathrm{LTF}(\boldsymbol{w}, \sigma)(\boldsymbol{x})$. *The vector* $(\boldsymbol{w}, \sigma)$ *is said to represent* f*, while the vector* $\boldsymbol{x} \in \{-1,1\}^n$ *is said to be* authorized *(resp.* forbidden*) if and only if* $f(\boldsymbol{x}) = 1$ *(resp.* $f(\boldsymbol{x}) = -1$*).*

Related to this, using ϕ we can switch from any monotone WTF to an equivalent monotone LTF and vice versa without modifying the weight of any coordinate. Namely, the switching lemmas (for more details, see [FG24]) imply that the weight vector representing a monotone LTF (resp. WTF) is not affected when converting it to a monotone WTF (resp. LTF). For this reason, along this work we will switch between both notions depending on which one is more useful for us at any given time. For technical reasons, we need to define linear bounded functions, which generalize LTFs.

Next, we define the notion of influence of a coordinate of a LTF, and the family of linear bounded functions.

Definition 2.5. *Let* $f : \{-1,1\}^n \to \{-1,1\}$ *be a Boolean function. For any* $i \in [n]$*, the influence of the* i*-th coordinate on* f *is the fraction of input values in which it affects the output, i.e.* $\mathbf{Inf}_i[f] = \mathbf{P}[f(\boldsymbol{x}) \neq f(\boldsymbol{x}^{\oplus i})]$.

Definition 2.6 (Linear Bounded Function). *Let* $\boldsymbol{w} \in \mathbb{R}^n$ *and* $\sigma \in \mathbb{R}$. *A Linear Bounded Function (LBF) is a function* $f : \{-1,1\}^n \to [1,1]$ *of the form* $f(\boldsymbol{x}) = \mathrm{LBF}(\boldsymbol{w}, \sigma)(\boldsymbol{x})$. *The vector* $(\boldsymbol{w}, \sigma)$ *is said to represent* f.

We end this section by stating two useful results about the weights of monotone LTFs. The first one shows that any monotone LTF can be expressed by a monotone formula of size polynomial in the sum of the weights. The second is a known upper bound on the size of the weights of any monotone LTF.

Theorem 2.7 ([Ser04])**.** *For any* $\boldsymbol{w} \in \mathbb{R}^n_+$ *and* $\sigma \in \mathbb{R}$*, the monotone LTF given by* $f(\boldsymbol{x}) = \mathrm{LTF}(\boldsymbol{w}, \sigma)(\boldsymbol{x})$ *has a monotone formula of size* $O(W^{5.3})$*, where* $W = \boldsymbol{w} \cdot \mathbf{1}^n$.

Theorem 2.8 ([Mur71]). *For any monotone LTF f with n variables, there exist $w_1, \ldots, w_n, \sigma \in \mathbb{N}$ smaller than $2^{\lceil n \log(n) \rceil}$ such that $f(\boldsymbol{x}) = \mathrm{LTF}(\boldsymbol{w}, \sigma)(\boldsymbol{x})$, where $\boldsymbol{w} = (w_1, \ldots, w_n)$.*

Chow Parameters. We first give the definition of the Chow parameters. We denote the expectancy of a discrete random variable by $\mathbf{E}$.

Definition 2.9 (Chow Parameters). *The Chow parameters of a function $f : \{-1,1\}^n \to \mathbb{R}$ are the $n+1$ values $\hat{f}(0) = \mathbf{E}[f(\boldsymbol{x})]$ and $\hat{f}(i) = \mathbf{E}[f(\boldsymbol{x})x_i]$ for any $i \in [n]$, taking uniform distribution on its domain. The Chow vector of f is $\boldsymbol{\chi}_f = (\hat{f}(0), \ldots, \hat{f}(n))$.*

Chow parameters are a particular case of a much more general family of parameters, the Fourier coefficients of Boolean functions. More in detail, each function $f : \{-1,1\}^n \to \mathbb{R}$ can be uniquely expressed as a multilinear polynomial whose coefficients are defined as the Fourier coefficients of f. In this setting, it can be shown that the Chow parameters simply correspond to the Fourier coefficients of degree 0 and 1. The book of O'Donnell [O'D14] offers a detailed discussion of this topic.

Following the later construction, we could also have defined the Fourier coefficients (and in particular the Chow parameters) for Boolean functions in the $\{0,1\}$ domain. However, in this case, we can no longer define the Chow parameters in terms of expectations as is done in Definition 2.9. This is because this basis is not orthonormal. Therefore, when it comes to Chow parameters, we always consider Boolean functions in the $\{-1,1\}$ domain.

In this regard, recall that for monotone LTFs we can always move from one domain to the other. Hence, by an abuse of notation, when we consider the Chow parameters of a WTF, we refer to the Chow parameters of its LTF analogue.

An important result regarding the Fourier coefficients is the following one, known as Plancherel's Theorem.

Theorem 2.10 (Plancherel's Theorem [O'D14]). *For any functions $f, g : \{-1,1\}^n \to \mathbb{R}$ we have that $\mathbf{E}[f(\boldsymbol{x})g(\boldsymbol{x})] = \sum_{S \subseteq [n]} \hat{f}(S)\hat{g}(S)$.*

Nowadays, Chow parameters (and by extension the Fourier coefficients) are among the most common tools for the study of Boolean functions. In the case of LTFs, this is mainly motivated by the next theorem, known as Chow's Theorem.

Theorem 2.11 (Chow's Theorem [Cho61]). *Any LTF is uniquely determined within the space of Boolean functions by its Chow parameters.*

Moreover, in the case of monotone Boolean functions, the Chow parameters have an additional interpretation as gauges of the influence of each coordinate. This is stated in the following proposition.

Proposition 2.12 ([O'D14]). *Let $f : \{-1,1\}^n \to \{-1,1\}$ be a monotone Boolean function. Then $\hat{f}(i) = \mathbf{Inf}_i[f]$ for any $i \in [n]$.*

We conclude this section by presenting a useful lemma relating monotonicity, projections and Chow parameters. Its proof can be found in [FG24].

Lemma 2.13. *Let $f : \{-1,1\}^n \to \mathbb{R}$ be a monotone function and $g(\boldsymbol{x}) = P_1(f(\boldsymbol{x}))$. Then $\hat{f}(i) \geq \hat{g}(i) \geq 0$ for any $i \in [n]$.*

To simplify the notation, from now on we will assume that any LTF has the weights sorted in decreasing order, which immediately implies a decreasing order in its Chow parameters except for $\hat{f}(0)$.

Distance. The concept of distance between Boolean functions gives a way to measure the similarity of two access structures. Hence, we introduce it to formally define the notion of closeness between monotone LTFs.

Definition 2.14 (Function Distance). *Let X be a finite set. The distance between two functions $f, g : X \to \mathbb{R}$ is defined as $\mathrm{dist}(f,g) = \mathbf{E}[|f(\boldsymbol{x}) - g(\boldsymbol{x})|]$. If $\mathrm{dist}(f,g) < \varepsilon$ we say that f and g are ε-close. Moreover, the Chow distance between f and g is defined as $\mathrm{dist}_{\mathrm{Chow}}(f,g) = \|\boldsymbol{\chi}_f - \boldsymbol{\chi}_g\|$.*

Remark 2.15. Notice that if f, g are WTFs then $\mathrm{dist}(f,g) = \mathbf{P}[f(\boldsymbol{x}) \neq g(\boldsymbol{x})]$, while if f, g are LTFs then $\mathrm{dist}(f,g) = 2\mathbf{P}[f(\boldsymbol{x}) \neq g(\boldsymbol{x})]$.

The notions of function distance and Chow distance are closely related. In particular, for the case of LTFs each of them can be bounded in terms of the other. This is stated in the theorems that follow.

Theorem 2.16 ([OS11])**.** *For every $f, g : \{-1,1\}^n \to \mathbb{R}$, it holds that $\mathrm{dist}_{\mathrm{Chow}}(f,g) \leq 2\sqrt{\mathrm{dist}(f,g)}$.*

Theorem 2.17 ([DDFS14])**.** *Let f be a LTF and let $g : \{-1,1\}^n \to [-1,1]$ be any function. If $\mathrm{dist}_{\mathrm{Chow}}(f,g) \leq \varepsilon$, then $\mathrm{dist}(f,g) \leq 2^{-\Omega(\sqrt[3]{\log \frac{1}{\varepsilon}})}$.*

Theorem 2.16 and Theorem 2.17 establish a relation between the upper bounds given by the distances of LTFs and their Chow distances, which provides a strategy to check closeness between LTFs simply by looking at their Chow parameters. Indeed, this is the key observation that we will exploit in the next section to approximate monotone LTFs.

Moreover, there is also a relation between the distance of monotone functions and the number of input coordinate swaps needed to guarantee the same output in both functions. This result, which is useful when viewing monotone functions as descriptions of access structures, is stated next.

Lemma 2.18. *Let $\varepsilon \in (0,1)$ and let $f, g : \{-1,1\}^n \to \{-1,1\}$ be non-constant ε-close monotone functions. For any $\boldsymbol{x} \in \{-1,1\}^n$ such that $f(\boldsymbol{x}) \neq g(\boldsymbol{x})$, there exists $\boldsymbol{y} \in \{-1,1\}^n$ such that $f(\boldsymbol{y}) = g(\boldsymbol{x})$ and $\mathrm{dist}_{\mathrm{Ham}}(\boldsymbol{x}, \boldsymbol{y}) \leq n + 1 - \log\left(\frac{1}{\varepsilon}\right)$.*

2.2 Secret Sharing Schemes

For convenience, we work with access structures described by monotone Boolean functions, which is equivalent to work with monotone increasing families of subsets.

Definition 2.19 (Access Structure). *An n-party access structure is a monotone Boolean function $f : \{0,1\}^n \to \{0,1\}$ such that $f(\mathbf{0}^n) = 0$ and $f(\mathbf{1}^n) = 1$.*

If $f(\boldsymbol{x}) = 1$, we say that the set $A = \mathrm{supp}(\boldsymbol{x})$ is authorized*, and else we say that A is* forbidden*.*

Definition 2.19 implies that non-constant monotone WTF are a particular case of access structures. In this context, they are also called *weighted threshold access structures*.

Remark 2.20. In the case of a weighted threshold access structure given by $f(\boldsymbol{x}) = WTF(\boldsymbol{w}, \sigma)(\boldsymbol{x})$, the family of authorized subsets corresponds to

$$\Gamma = \left\{ A \subseteq [n] \ : \ \sum_{i \in A} w_i \geq \sigma \right\}.$$

Moreover, note that if two functions f and f' are ε-close, the corresponding monotone families of subsets Γ and Γ' satisfy $|\Gamma \cup \Gamma'| - |\Gamma \cap \Gamma'| < \varepsilon 2^n$.

In this work, we construct information-theoretic and computational secret sharing schemes. By default, when we talk about secret sharing schemes we refer to the first ones.

Definition 2.21 (Information-Theoretic Secret Sharing Scheme). *Let $\mathcal{S}$ be a finite set and let f be an access structure over n parties. A secret sharing scheme for f is a pair consisting of a randomized algorithm* Share *and a deterministic algorithm* $\mathsf{Reconstruct}_A$ *such that*

- ***Perfect correctness.*** *For any secret $s \in \mathcal{S}$ and any authorized set A, it holds that*
$$\mathbf{P}[s = \mathsf{Reconstruct}_A(\mathsf{Share}(s)_A)] = 1,$$
where $\mathsf{Share}(s)_A$ *denotes the restriction of the output of* $\mathsf{Share}(s)$ *to the parties in A.*
- ***Perfect privacy.*** *For any secrets $s, s' \in \mathcal{S}$, any forbidden set B, and any possible set of shares $\{s_i\}_{i \in B}$, it holds that*
$$\mathbf{P}[\{s_i\}_{i \in B} = \mathsf{Share}(s)_B] = \mathbf{P}[\{s_i\}_{i \in B} = \mathsf{Share}(s')_B].$$

The best general weighted threshold secret sharing schemes in terms of the share size are given in the following result [BW06]. This result is obtained by finding a monotone circuit that computes the weighted threshold function, transforming it to a monotone formula, and then constructing a scheme for the formula [BL88]. Indeed, it is a consequence of Theorem 2.3.

Theorem 2.22 ([BW06]). *Every weighted threshold access structure over n parties admits a secret sharing scheme with share size $n^{O(\log n)}$.*

We now state the definition of computational secret sharing schemes. The security of computational secret sharing schemes is given in terms of a game between an adversary and a challenger. The details can be found in [FG24].

Definition 2.23 (Computational Secret Sharing Scheme). *Let $\mathcal{S}$ be a finite set, let $\lambda \in \mathbb{N}$ be the security parameter, and let f be an access structure over n parties. A computational secret sharing scheme for f is a pair of a randomized polynomial-time algorithm* Share *and a deterministic polynomial-time algorithm* $\mathsf{Reconstruct}_A$ *such that*

- ***Correctness.*** *For any secret $s \in \mathcal{S}$, any authorized set A, and a description D of f it holds that*

$$\mathbf{P}[s = \mathsf{Reconstruct}_A(\mathsf{Share}(s, D, \mathbf{1}^\lambda)_A, D)] = 1,$$

  where $\mathsf{Share}(s, D, \mathbf{1}^\lambda)_A$ *denotes the restriction of the output of* $\mathsf{Share}(s, D, \mathbf{1}^\lambda)$ *to the parties in A.*
- ***Privacy.*** *The scheme is $t(\lambda)$-secure if any $t(\lambda)$-time adversary wins the security game with probability at most $\frac{1}{t(\lambda)}$.*

3 Approximation of Monotone Linear Threshold Functions

The aim of this section is to approximate a monotone LTF with another monotone LTF with small integer weights. To do so, we adapt to the monotone setup the work of De et al. [DDFS14], in which an algorithm for approximating LTFs is presented. To simplify the presentation of this and the following results, we use $\tilde{O}$ notation, which ignores polylogarithmic factors. The main result of this section is the following theorem.

Theorem 3.1 (Theorem 1.2 restated). *Let $\kappa(\varepsilon) = 2^{-O(\log^3(\frac{1}{\varepsilon}))}$ and let $\mu(n) \in \mathbb{R}$ be a function that satisfies $\mu(n) \geq 2\sqrt{n+1}$. For $\delta, \varepsilon \in (0,1)$, there exists a randomized algorithm* ApproximateLTF *that given a monotone LTF over n variables f with $\kappa(\varepsilon) \leq \hat{f}(n)\mu(n)$, outputs with probability $1-\delta$ a monotone function $g(\boldsymbol{x}) = \mathrm{LTF}(\boldsymbol{v}, v_0)(\boldsymbol{x})$ with the following properties:*

1. *g is ε-close to f,*
2. *$\boldsymbol{v} \in \mathbb{N}^n$, $v_0 \in \mathbb{Z}$, and $\|\boldsymbol{v}\| = O\left(\mu(n)\left(\frac{1}{\varepsilon}\right)^{O\left(\log^2\left(\frac{1}{\varepsilon}\right)\right)}\right)$.*

Further, the algorithm runs in time $\tilde{O}\left(n\mu(n)^2 \mathrm{poly}\left(\frac{1}{\kappa(\varepsilon)}\right) \log\left(\frac{1}{\delta}\right)\right)$.

The proof of Theorem 3.1 is constructive. First, we show in Theorem 3.2 the existence of a similar algorithm ApproximateLBF whose output is a monotone LBF. The proof of Theorem 3.2 is more involved, and it deferred to Sect. 3.1. Then, we show how to use it to construct the desired ApproximateLTF algorithm.

Theorem 3.2. *Let $\mu(n) \in \mathbb{R}$ be a function that satisfies $\mu(n) \geq 2\sqrt{n+1}$. For $\delta, \varepsilon \in (0,1)$, there exists a randomized algorithm* ApproximateLBF *that given a monotone LTF over n variables f with $\varepsilon \leq \hat{f}(n)\mu(n)$, outputs with probability $1-\delta$ a monotone function $g(\boldsymbol{x}) = \mathrm{LBF}(k\boldsymbol{v}, kv_0)(\boldsymbol{x})$ with the following properties:*

1. $\mathrm{dist}_{\mathrm{Chow}}(f,g) \leq 3\varepsilon$,
2. $k \in \mathbb{R}, \boldsymbol{v} \in \mathbb{N}^n$, $v_0 \in \mathbb{Z}$, *and* $\|\boldsymbol{v}\| = O\left(\frac{\mu(n)}{\varepsilon^3}\right)$.

Further, the algorithm runs in time $\tilde{O}\left(n\frac{\mu(n)^2}{\varepsilon^4}\log\left(\frac{1}{\delta}\right)\right)$.

Proof of Theorem 3.1. A direct application of Theorem 3.2 guarantees that in a running time of $\tilde{O}\left(n\mu(n)^2\mathrm{poly}\left(\frac{1}{\kappa(\varepsilon)}\right)\log\left(\frac{1}{\delta}\right)\right)$ we obtain a monotone LBF $g(\boldsymbol{x}) = \mathrm{LBF}(\boldsymbol{v}, v_0)$ such that $\mathrm{dist}_{\mathrm{Chow}}(f,g) \leq 3\kappa(\varepsilon)$ with probability $1-\delta$. From there, adjusting properly the constants of $\kappa(\varepsilon)$ and applying Theorem 2.17 we get that $\mathrm{dist}(f,g) \leq \frac{\varepsilon}{2}$.

Now, defining the function $f'(\boldsymbol{x}) = \mathrm{LTF}(\boldsymbol{v}, v_0)(\boldsymbol{x})$, it is straightforward to check that f' is monotone and $\mathrm{dist}(f,f') \leq 2\mathrm{dist}(f,g) \leq \varepsilon$. Moreover, we have that $\|\boldsymbol{v}\| = O\left(\frac{\mu(n)}{\kappa(\varepsilon)^3}\right) = O\left(\mu(n)\left(\frac{1}{\varepsilon}\right)^{O\left(\log^2\left(\frac{1}{\varepsilon}\right)\right)}\right)$ by Theorem 3.2. □

3.1 Proof of Theorem 3.2

We first present a high-level overview of the construction of the ApproximateLBF algorithm, which is an adaptation of the algorithm from De et al. [DDFS14] to the monotone case. The construction relies in the straightforward observation that the function $\hat{f}(0) + \sum_{i=1}^n \hat{f}(i)x_i$ has exactly the same Chow parameters as the input LTF f.

Starting from there, the function $g(\boldsymbol{x}) = P_1(\hat{f}(0) + \sum_{i=1}^n \hat{f}(i)x_i)$ is a candidate for being the monotone LBF output by the algorithm because it is build with the Chow parameters of f. However, taking the projection of a function leads to a modification of the original Chow parameters, so our candidate g may not satisfy the desired condition on the Chow distance.

To solve this problem, we correct this gap in the Chow distance following a similar procedure as before. In particular, we construct a function h whose Chow parameters correspond to the difference between $\boldsymbol{\chi}_f$ and $\boldsymbol{\chi}_g$, i.e. $h(\boldsymbol{x}) = \hat{f}(0) - \hat{g}(0) + \sum_{i=1}^n (\hat{f}(i) - \hat{g}(i))x_i$, and add it to g with the aim of obtaining a better output candidate $P_1(g+h)$.

Then, we face again the problem of checking up to which point the projection operation has modified the Chow parameters of our output candidate. Hence, we can repeat the previous procedure iteratively seeking that at each step we will get closer to the desired result.

In essence, ApproximateLBF algorithm implements the idea we have just presented. Nevertheless, some minor changes are introduced to deal with technicalities regarding monotonicity, the bounds on the weights, and the running

time. The ApproximateLBF algorithm is depicted in detail in Algorithm 1, while Algorithm 2 presents an auxiliary routine.

Notice that the context of the work of De et al. [DDFS14] is slightly different. Their algorithm aims to solve the approximate Chow problem, i.e. given the approximate Chow parameters of an unknown LTF f, they find another LTF that is ε-close to f. The two main differences with our setup are that we assume we know in advance the original monotone LTF and its Chow parameters, and that in our case the constructed LTF has to be monotone. Hence, all the changes made in the statement and proof of Theorem 3.2 are due to these differences.

In this regard, from a practical point of view, it should be noted that even if the original monotone LTF is available, calculating its exact Chow parameters requires exponential time in the number of coordinates because of their definition as expected values of the monotone LTF. To circumvent this limitation, we can slightly modify ApproximateLBF (Algorithm 1) to work with an approximation of the Chow parameters of the input LTF as done in the work of De et al. [DDFS14] without affecting the properties of the output function.

Algorithm 1. ApproximateLBF

Input: Monotone $f(\boldsymbol{x}) = \mathrm{LTF}(\boldsymbol{w}, w_0)(\boldsymbol{x})$, Chow parameters $\boldsymbol{\chi}_f$, and $\delta, \varepsilon \in (0,1)$

Output: Monotone $g(\boldsymbol{x}) = \mathrm{LBF}(k\boldsymbol{v}, kv_0)(\boldsymbol{x})$ with $\mathrm{dist}_{\mathrm{Chow}}(f,g) \leq 3\varepsilon$, $\|\boldsymbol{v}\| = O\left(\frac{\mu(n)}{\varepsilon^3}\right)$

Require: $\mu(n) \geq 2\sqrt{n+1}$ and $\varepsilon \leq \hat{f}(n)\mu(n)$

$g' \leftarrow 0$, $g \leftarrow P_1(g')$
$\boldsymbol{\chi}_{\tilde{g}} \leftarrow \mathsf{VectorApprox}(\boldsymbol{\chi}_f, \boldsymbol{\chi}_g, \varepsilon)$
while $\|\boldsymbol{\chi}_f - \boldsymbol{\chi}_{\tilde{g}}\| > 2\varepsilon$ **do**
  $h \leftarrow \sum_{i=0}^{n} (\hat{f}(i) - \tilde{g}(i)) x_i$
  $g' \leftarrow g' + \frac{h}{2}$
  $g \leftarrow P_1(g')$
  Compute $\boldsymbol{\chi}_g$ with precision $\frac{\varepsilon}{2\mu(n)}$ ▷ Chow parameters of g
  $\boldsymbol{\chi}_{\tilde{g}} \leftarrow \mathsf{VectorApprox}(\boldsymbol{\chi}_f, \boldsymbol{\chi}_g, \varepsilon)$ ▷ Rounding to ensure integer difference
end while
return g

Algorithm 2. VectorApprox

Input: Chow parameters $\boldsymbol{\chi}_f, \boldsymbol{\chi}_g \in \mathbb{R}^{n+1}$ and $\varepsilon \in (0,1)$

Output: Vector $\boldsymbol{\chi}_{\tilde{g}} \in \mathbb{R}^{n+1}$

for $i = 0, \ldots, n$ **do**
  $\tilde{g}(i) \leftarrow$ the closest value to $\hat{g}(i)$ such that $\hat{f}(i) - \tilde{g}(i) = k\frac{\varepsilon}{\mu(n)}$ with $k \in \mathbb{Z}$
end for
return $\boldsymbol{\chi}_{\tilde{g}} = (\tilde{g}(0), \ldots, \tilde{g}(n))$

To prove Theorem 3.2 it suffices to check that ApproximateLBF (Algorithm 1) satisfies all the conditions of the statement. In comparison to the work of De et

al. [DDFS14], apart from slightly differences in the bounds of some norms, our proof requires a more fine-grained analysis to ensure that the resulting approximate function is also monotone. For this reason, we divide it into several lemmas: One for checking the monotonicity of the output, another to ensure the halting of the algorithm, and a last one for the bounds on the weights and the running time. The proof of all these lemmas can be found in [FG24].

We start by proving that the output of ApproximateLBF (Algorithm 1) corresponds to a monotone LBF satisfying the gap in the Chow distance. Indeed, this is the part in which our work differs the most from the one of De et al. [DDFS14], since it is where we impose the monotonicity condition on the output.

Lemma 3.3. ApproximateLBF *(Algorithm 1) outputs with probability* $1-\delta$ *a monotone LBF* g *such that* $\mathrm{dist}_{\mathrm{Chow}}(f,g) \leq 3\varepsilon$.

Lemma 3.3 ensures that the output of ApproximateLBF (Algorithm 1) is a monotone LBF satisfying the required distance in the Chow parameters. However, this is a meaningless result unless we prove that ApproximateLBF (Algorithm 1) always halts, since apparently nothing prevents the algorithm to stay indefinitely in the main loop (steps 4–8). The next lemma shows that this situation can not happen.

Lemma 3.4. *The main loop of* ApproximateLBF *(Algorithm 1, steps 4–8) requires at most* $\frac{1}{\varepsilon^2}$ *iterations.*

Finally, the remaining lemma gives the bound on the weights and the running time. It is a crucial result, since the bound on the weights is what later enables the construction of secret sharing schemes for weighted threshold access structures with small share size. Moreover, the bound on the running time shows the feasibility of this strategy.

Lemma 3.5. *The function* $g(\boldsymbol{x}) = \mathrm{LBF}(k\boldsymbol{v}, kv_0)(\boldsymbol{x})$ *output by* ApproximateLBF *(Algorithm 1) has the following properties:*

1. $k \in \mathbb{R}, \boldsymbol{v} \in \mathbb{N}^n$, $v_0 \in \mathbb{Z}$, *and* $\|\boldsymbol{v}\| = O\left(\frac{\mu(n)}{\varepsilon^3}\right)$.
2. *Its running time is* $\tilde{O}\left(n\frac{\mu(n)^2}{\varepsilon^4}\log\left(\frac{1}{\delta}\right)\right)$.

At this point, to prove Theorem 3.2 it suffices to combine the results of Lemma 3.3, Lemma 3.4 and Lemma 3.5.

3.2 Remarks on the Error and the Weight Bound of Theorem 3.1

Theorem 3.1 sets a method for approximating monotone LTFs up to any accuracy and gives a bound on the resulting weights in terms of the number of variables and the error. In this section, we show that our approximation technique preserves the hierarchy among the participants and we discuss the optimality of the technique. First, we show that the error has a negligible impact on the original hierarchy of the coordinates. Second, we prove the optimality of the

algorithm with respect to the weight bound. Later, we perform an analysis on the trade-off between both notions: The error and the size of the weights. Moreover, in the full version [FG24] we show how to relate the approximation error to a ramp criteria assuming bounds on the weights.

On the Error and the Preservation of the Weights Hierarchy. One of the biggest concerns when approximating a monotone LTF is to preserve the hierarchical properties of the original function. In this regard, we can ensure that the error produced does not cause a drastic change in the impact of each coordinate to the output of the function. Namely, we can show that the approximating technique also preserves the influence of the coordinates.

Theorem 3.6. *Let f, g be two $o(1)$-close monotone LTFs. Then, $|\mathbf{Inf}_i[f] - \mathbf{Inf}_i[g]| = o(1)$ for every $i \in [n]$.*

Proof. If $\mathrm{dist}(f,g) = o(1)$, then $\mathrm{dist}_{\mathrm{Chow}}(f,g) = o(1)$ by Theorem 2.16. By Proposition 2.12, the difference between influences is also $o(1)$. □

Another natural requirement for the approximation procedure is to maintain the hierarchy. That is, we demand that whenever one coordinate has a greater weight than another in the original function, the same happens after the approximation. If the output of our algorithm does not satisfy it, this requirement can be achieved simply by rearranging the weights of the output function. The next lemma shows that this additional modification does not increase the error of the approximation.

Lemma 3.7. *Let f be a monotone LTF, let $g(\boldsymbol{x}) = \mathrm{LTF}(\boldsymbol{w}, w_0)(\boldsymbol{x})$ be a monotone LTF ε-close to f, and let $h(\boldsymbol{x}) = \mathrm{LTF}(\boldsymbol{w}', w_0)(\boldsymbol{x})$, where $\boldsymbol{w}'$ is the weight vector $\boldsymbol{w}$ sorted in decreasing order. It holds that $\mathrm{dist}(f,h) \leq \varepsilon$.*

Proof. First, note that it suffices to prove the statement for the case where the vector $\boldsymbol{w}'$ is equal to $\boldsymbol{w}$ except for the flipping of any two coordinates $i < j$ with $w_i < w_j$, since the case where $\boldsymbol{w}'$ is decreasingly ordered can be obtained by iterating this flipping process.

Now, to show that $\mathrm{dist}(f,h) \leq \varepsilon$ we show that for any $\boldsymbol{x} \in \{-1,1\}^n$ with $x_i, x_j = -1$ it holds that

$$\left|\left\{f(\boldsymbol{y}) = g(\boldsymbol{y}) \ : \ \boldsymbol{y} \in \{\boldsymbol{x}^{\oplus i}, \boldsymbol{x}^{\oplus j}\}\right\}\right| \leq \left|\left\{f(\boldsymbol{y}) = h(\boldsymbol{y}) \ : \ \boldsymbol{y} \in \{\boldsymbol{x}^{\oplus i}, \boldsymbol{x}^{\oplus j}\}\right\}\right|.$$

Since both f and g, are monotone LTFs, each of them has only three possible outputs for any pair of inputs $\boldsymbol{x}^{\oplus i}, \boldsymbol{x}^{\oplus j}$. In more detail, either $f(\boldsymbol{x}^{\oplus i}) = f(\boldsymbol{x}^{\oplus j}) = -1$, $f(\boldsymbol{x}^{\oplus i}) = f(\boldsymbol{x}^{\oplus j}) = 1$, or $f(\boldsymbol{x}^{\oplus i}) = 1 > f(\boldsymbol{x}^{\oplus j}) = -1$ (and similarly for g). Therefore, we are left with a total of 9 different cases. Observe that the case

$$f(\boldsymbol{x}^{\oplus i}) = 1, \ f(\boldsymbol{x}^{\oplus j}) = -1, \ g(\boldsymbol{x}^{\oplus i}) = 1, \text{ and } g(\boldsymbol{x}^{\oplus j}) = -1.$$

is not possible because $w_i < w_j$ by hypothesis.

We now prove the case where

$$f(\boldsymbol{x}^{\oplus i}) = 1,\ f(\boldsymbol{x}^{\oplus j}) = -1,\ g(\boldsymbol{x}^{\oplus i}) = -1, \text{ and } g(\boldsymbol{x}^{\oplus j}) = 1.$$

By hypothesis, we have that

$$\left|\left\{f(\boldsymbol{y}) = g(\boldsymbol{y}) \ :\ \boldsymbol{y} \in \{\boldsymbol{x}^{\oplus i}, \boldsymbol{x}^{\oplus j}\}\right\}\right| = 0.$$

Moreover, by the definition of h we have that $g(\boldsymbol{x}^{\oplus i}) = -1$ implies that $h(\boldsymbol{x}^{\oplus j}) = -1$, and that $g(\boldsymbol{x}^{\oplus j}) = 1$ implies that $h(\boldsymbol{x}^{\oplus i}) = 1$. Therefore, we get that

$$\left|\left\{f(\boldsymbol{y}) = h(\boldsymbol{y}) \ :\ \boldsymbol{y} \in \{\boldsymbol{x}^{\oplus i}, \boldsymbol{x}^{\oplus j}\}\right\}\right| = 2,$$

which proves this case. The remaining cases can be proved analogously. □

Therefore, we can conclude that our technique also preserves the original hierarchy on the weights and the influence of each coordinate.

Optimality of the Weight Bound. A natural question is if the weight bound of Theorem 3.1 can be improved in terms of its dependency on n or ε. We answer this question by showing that the bound is optimal in terms of n and is close of being optimal in terms of ε. In particular, we prove the following theorem.

Theorem 3.8 (Theorem 1.3 restated). *Let* $\kappa = \min\{\sqrt{n}, \left(\frac{1}{\varepsilon}\right)^{\Omega\left(\log\left(\log\left(\frac{1}{\varepsilon}\right)\right)\right)}\}$. *For any* n *and* $\varepsilon \in (0, \frac{1}{10})$, *there exists a monotone LTF* f *over* n *variables such that any monotone LTF that is* ε*-close to* f *has integer weights of size* $\Omega(\kappa)$.

Remark 3.9. In the statement of Theorem 3.1 the weight bound, with respect to n, corresponds to any $\mu(n) \geq 2\sqrt{n+1}$. Therefore, when we look at the optimality of the bound in Theorem 3.8 we can replace $\mu(n)$ by $\sqrt{n}$.

As we did with Theorem 3.1, we split the proof of Theorem 3.8 in two steps. First, we prove that the bound is optimal in terms of its dependency on n. Later, we study the optimality of the bound with respect to ε.

To show that the bound can not be improved in terms of n it suffices to give an explicit monotone LTF f over n variables such that for $\varepsilon = \frac{1}{10}$ any LTF g ε-close to f has some weight of size $\Omega(\sqrt{n})$.[1] This is precisely what Servedio did in [Ser06], where he proved the following result.

Theorem 3.10 ([Ser06]). *Let* $\boldsymbol{w} = (1, \ldots, 1, n) \in \mathbb{Z}^n$, *let* f *be the monotone LTF given by* $f(\boldsymbol{x}) = \mathrm{LTF}(\boldsymbol{w}, n)(\boldsymbol{x})$, *and let* $g : \{-1, 1\}^n \to \{-1, 1\}$ *be an LTF that is* $\frac{1}{10}$*-close to* f. *Then any integer representation of* g *must have some weight of size* $\Omega(\sqrt{n-1})$.

[1] Since we are looking for a lower bound in terms of n, note that we can set in advance the approximation error ε to a concrete value and the approximate LTF g does not need to be monotone.

Next, we move to study the optimality of the bound in terms of $\frac{1}{\varepsilon}$. To obtain the bound stated in Theorem 3.8 we rely on a particular LTF introduced by Hästadt [Hås94] that requires integer weights of size $2^{\Omega(n\log(n))}$.

Theorem 3.11. *There exist a monotone LTF f and $\varepsilon \in (0,1)$ for which any monotone LTF g that is ε-close to f has some weight of size $\left(\frac{1}{\varepsilon}\right)^{\Omega\left(\log\left(\log\left(\frac{1}{\varepsilon}\right)\right)\right)}$.*

Proof. We argue by contradiction. Suppose that for any monotone LTF f and $\varepsilon \in (0,1)$ we can construct a monotone LTF g that is ε-close to f with weights of size $\left(\frac{1}{\varepsilon}\right)^{o\left(\log\left(\log\left(\frac{1}{\varepsilon}\right)\right)\right)}$. Now, let f be the LTF introduced by Hästadt [Hås94] that requires integer weights of size $2^{\Omega(n\log(n))}$. Without loss of generality, we can suppose that f is also monotone.[2] Then, by hypothesis, for any $\varepsilon < \frac{1}{2^n}$ we can construct a monotone LTF g with integer weights of size $2^{o(n\log(n))}$ that is ε-close to f. Since there are only $2^n < \frac{1}{\varepsilon}$ distinct input values, this implies that $f = g$. But this clearly contradicts the lower bound on the weights of f. □

If we see Theorem 3.1 as a strategy to reduce the weights of a given monotone LTF, we can interpret Theorem 3.8 not only as a limitation of this technique but also as a limitation of any approximation technique based on the reduction of the weights, as for example rounding. However, since Theorem 3.1 is a general result, it is possible that some families of monotone LTFs admit approximations with smaller weights.

Trade-Off Between the Error and the Weight Bound. Next, we study the trade-off between the size of the weights and the error of Theorem 3.1. Since the weight bound of Theorem 3.1 depends on ε, the more accurate the approximation, the higher the weight bound will be. Moreover, this factor is of the form quasipoly $\left(\frac{1}{\varepsilon}\right)$, which implies that the weight bound grows faster than the accuracy of the approximation does.

In this regard, note that, at the cost of increasing the weight bound, we can make the error ε as small as desired because there is no lower restriction apart from the trivial $\varepsilon > 0$. However, we can not increase ε freely since Theorem 3.1 requires that $2^{-O\left(\log^3\left(\frac{1}{\varepsilon}\right)\right)} < \hat{f}(n)\mu(n)$. This can be annoying in case we want to minimize the size of the weights.

To overcome this limitation, we can try to increase ε by pushing up the value of its upper bound $\hat{f}(n)\mu(n)$. In that sense, the factor $\hat{f}(n)$ is given by the original LTF and Proposition 2.12 implies that, in the worst case, it can be equal to $\frac{1}{2^{n-1}}$. Hence, our only option is to increase the value of $\mu(n)$, which is lower bounded by $2\sqrt{n+1}$.

Nevertheless, note that $\mu(n)$ also appears as a linear factor of the weight bound. For this reason, we must be careful when increasing the value of $\mu(n)$

[2] By definition, any LTF can be converted into a monotone LTF simply by flipping some of its input variables. If f is not monotone, it suffices to define f' as the monotone LTF obtained from f by this procedure. It is clear that the magnitude of the weights of f' is the same as that of the weights of f.

with the aim of decreasing the weight bound, since pushing it too high may go against our interests.

A more fine-grained analysis of this trade-off will appear later in Sect. 4 in the proof of Theorem 4.3. As we will see, a better result can be obtained by discarding the Chow parameters with the lowest values.

4 Secret Sharing Schemes for Approximate Weighted Threshold Access Structures

In this section we apply the results on low-weight approximators for monotone LTFs to construct information-theoretic secret sharing schemes for weighted threshold access structures with small share size. First, we introduce our proposal and discuss some alternatives. Later, we compare it with state-of-the-art solutions.

4.1 Scheme Construction

The main result of this section is the following theorem.

Theorem 4.1. *Let* $\kappa(\varepsilon) = 2^{-O(\log^3(\frac{1}{\varepsilon}))}$ *and let* $\mu(n) \in \mathbb{R}$ *be a function that satisfies* $\mu(n) \geq 2\sqrt{n+1}$. *For any weighted threshold access structure* f *over* n *parties and* $\varepsilon \in (0,1)$ *with* $\kappa(\varepsilon) \leq \hat{f}(n)\mu(n)$, *there exists a weighted threshold access structure over* n *parties* ε*-close to* f *admitting an information-theoretic secret sharing scheme with share size* $\tilde{O}\left(\mu(n)\left(\frac{1}{\varepsilon}\right)^{O(\log^2(\frac{1}{\varepsilon}))}\right)$.

Proof. First, recall that we can work indistinctly with f and its equivalent monotone WTF. Hence, in this proof we will not make any distinction between them.

Now, given $\mu(n) > 2\sqrt{n+1}$ and $\varepsilon \in \left(0, \hat{f}(n)\mu(n)\right]$ we apply Theorem 3.1 to obtain a monotone function $g(\boldsymbol{x}) = \mathrm{LTF}(\boldsymbol{v}, v_0)(\boldsymbol{x})$ that is ε-close to f. Moreover, it holds that $\boldsymbol{v} \in \mathbb{N}^n$, $v_0 \in \mathbb{Z}$, and $\|\boldsymbol{v}\| = O\left(\mu(n)\left(\frac{1}{\varepsilon}\right)^{O(\log^2(\frac{1}{\varepsilon}))}\right)$.

By construction, g is a weighted threshold access structure with weights $\boldsymbol{v}$, threshold v_0', and with $\mathrm{dist}(f,g) < \varepsilon$. In addition, since we know that $\|\boldsymbol{v}\| = O\left(\mu(n)\left(\frac{1}{\varepsilon}\right)^{O(\log^2(\frac{1}{\varepsilon}))}\right)$ it suffices to use Shamir's virtualization technique to obtain the desired secret sharing scheme for g. □

Remark 4.2. Since we use Shamir's virtualization technique, the secret sharing scheme constructed in Theorem 4.1 is also linear. This implies that our proposal has homomorphic properties and efficient Share and Reconstruction algorithms.

Following the arguments of Sect. 3.2, it seems inevitable that to obtain a small upper bound on the share size in Theorem 3.1, the Chow parameters associated to the weighted threshold access structure must be big enough. Otherwise, the

inequality $\kappa(\varepsilon) \leq \hat{f}(n)\mu(n)$ would imply that either ε is too small or $\mu(n)$ is too big, which in both cases would lead to an increase in the upper bound on the share size. In this regard, note that the resulting upper bound applies to the worst case, but does not provide information about the average case.

However, if some of the Chow parameters are small, we can still obtain a small upper bound on the share size simply by discarding those parties from the original weighted threshold access structure. More in detail, thanks to Proposition 2.12 we can view each Chow parameter as the influence of a specific party in the access structure. Therefore, if some of the Chow parameters are small, we can guarantee that removing them from the access structure would not modify it too much. In this way, we can always control the value of the $\hat{f}(n)$ that appears in the statement of Theorem 3.1.

As a consequence of combining this observation with Theorem 4.1 we obtain the following result.

Theorem 4.3 (Theorem 1.1 restated). *Let $k \in \mathbb{N}$. For any weighted threshold access structure f over n parties, there exists a weighted threshold access structure over n parties $\frac{1}{\log^k(n)}$-close to f admitting an information-theoretic secret sharing scheme with share size $n^{1+o(1)}$.*

Proof. Let $\boldsymbol{w}$ be the weight vector of f, let σ be its threshold value, and let $l \in [n]$ be the maximum value such that $\hat{f}(l) \geq \frac{1}{2n\log^k(n)}$.

We consider the weighted threshold access structure f' over l parties given by the weight vector $(w_1, \ldots, w_l)$ and threshold σ. Proposition 2.12 implies that $\operatorname{dist}(f, f') \leq \frac{n}{2n\log^k(n)} = \frac{1}{2\log^k(n)}$.

Next, we set $\mu(n) = n \geq 2\sqrt{n+1}$, $\varepsilon = \frac{1}{2\log(n)}$, and $\kappa(\varepsilon) \coloneqq 2^{-O\left(\log^3\left(\frac{1}{\varepsilon}\right)\right)}$. For sufficiently large n we have that

$$\kappa(\varepsilon) = \frac{1}{2^{O(\log^3(2\log^k(n)))}} \leq \frac{1}{2\log^k(n)} = \frac{n}{2n\log^k(n)} = \hat{f}(l)\mu(n).$$

Hence, we can apply Theorem 4.1 to f' to get a weighted threshold access structure over n parties g that is $\frac{1}{\log^k(n)}$-close to f' and admits an information-theoretic secret sharing scheme with share size $\tilde{O}\left(n\left(\frac{1}{\varepsilon}\right)^{O\left(\log^2\left(\frac{1}{\varepsilon}\right)\right)}\right)$. Moreover, it holds that

$$\left(\frac{1}{\varepsilon}\right)^{O\left(\log^2\left(\frac{1}{\varepsilon}\right)\right)} = n^{\frac{O\left(\log^3\left(\frac{1}{\varepsilon}\right)\right)}{\log(n)}} = n^{\frac{O(\log^3(2\log^k(n)))}{\log(n)}} = n^{o(1)},$$

which implies that the share size is $n^{1+o(1)}$. Finally, the triangle inequality implies that

$$\operatorname{dist}(f, g) \leq \operatorname{dist}(f, f') + \operatorname{dist}(f', g) \leq \frac{1}{2\log^k(n)} + \frac{1}{2\log^k(n)} = \frac{1}{\log^k(n)}.$$

□

4.2 Remarks on the Secret Sharing Techniques

At the end of the proof of Theorem 4.1, we have used Shamir's virtualization technique to construct our secret sharing scheme. Hence, one may wonder if the use of an alternative construction may lead to smaller shares. To answer this question, we move to combine our technique with the other existing proposals. First, we target the work of Benaloh and Leichter based on monotone formulae [BL88]. Later, we focus our attention on the work of Beimel and Weinreb using monotone circuits [BW06]. From there, we observe that the resulting schemes have larger share size than the one from Theorem 4.1. Finally, we present a brief discussion about lower bounds on information-theoretic secret sharing schemes for weighted threshold access structures.

Alternative Secret Sharing Schemes Constructions. Benaloh and Leichter [BL88] presented a secret sharing construction whose share size is linear in the size of any monotone formula realizing the access structure. In this regard, note that Theorem 2.7 states that any weighted threshold access structure has a monotone formula of polynomial size in the total weight. Therefore, given any weighted threshold access structure we can combine our approximation technique with these two results to obtain an information-theoretic secret sharing scheme with total share size $O\left(\mu(n)^{10.6}\left(\frac{1}{\varepsilon}\right)^{O\left(\log^2\left(\frac{1}{\varepsilon}\right)\right)}\right)$ for an ε-close weighted threshold access structure. More specifically, we first apply Theorem 3.1 to construct an ε-close weighted threshold access structure, then we use Theorem 2.7 to obtain a polynomial size monotone formula, and finally we construct the secret sharing scheme with polynomial share size.

The construction by Beimel and Weinreb [BW06] has two main steps. First, they describe logarithmic depth and polynomial size unbounded fan-in monotone circuits that compute any monotone weighted threshold function (Theorem 2.3). Later, they transform the circuit into a monotone boolean formula, that is transformed into a scheme by the technique mentioned above, obtaining the share size in Theorem 2.22. To construct these circuits they use the upper bound on the weights of $2^{\lceil n\log(n)\rceil}$ of Theorem 2.8. Hence, we can use our approximation technique to avoid using this bound. More in detail, we can apply Theorem 3.1 to construct an ε-close weighted threshold access structure and bound the weights by $\mu(n)\left(\frac{1}{\varepsilon}\right)^{O\left(\log^2\left(\frac{1}{\varepsilon}\right)\right)}$. However, this strategy only leads to secret sharing schemes with share size $n^{O\left(\log\left(\log\left(\mu(n),\frac{1}{\varepsilon}\right)\right)\right)}$, which is still quasipolynomial. This is because the quasipolynomial magnitude is due to the conversion from monotone circuits to monotone formulae, something in which our technique does not help.

On the Share Size Lower Bounds. The best lower bound on the share size is $\Omega(\sqrt{n})$ [PS00], and it is indeed a bound on the information ratio. A common technique for obtaining lower bounds on the share size for linear schemes is through counting arguments. In this regard, the next theorem gives upper and

lower bounds on the number of weighted threshold access structures over n parties and shows that they correspond to a small subset within the set of monotone Boolean functions. Counting arguments like the ones in [KW93] give trivial bounds, in this case.

Theorem 4.4. *The number of weighted threshold access structures over n parties is $2^{\Theta(n^2)}$.*

Proof. First, notice that finding the number of weighted threshold access structures over n parties is equivalent to finding the number of monotone WTFs over n variables.

Now, let $\mathcal{T}$ (resp. $\mathcal{T}_M$) denote the set of all WTFs (resp. monotone WTFs) with n variables. Muroga [Mur71] proved that

$$2^{\frac{n^2}{2}} \leq |\mathcal{T}| \leq 2^{n^2} \text{ for any } n.$$

Hence, to prove the theorem it suffices to show that $\frac{|\mathcal{T}|}{2^n} \leq |\mathcal{T}_M|$.

To do so, we define a surjective mapping $\varphi : \mathcal{T}'_M \to \mathcal{T}$, where $\mathcal{T}'_M$ is the set containing 2^n copies of each monotone LTF ordered from 0 to $2^n - 1$. In this setup, φ simply consists in taking the positive representation of any monotone LTF given by Remark 2.2 and mapping its k-th copy to the LTF obtained by negating the weights corresponding to the coordinates referred by the binary representation of k.

Hence, φ is surjective by construction because for any LTF with k weights smaller than zero we obtain a monotone LTF by taking the absolute value of these weights. In particular, its preimage consists in the k-th copy of some monotone LTF. □

4.3 Comparison with State-of-the-Art Proposals

We now compare our proposal with the state-of-the-art constructions. This is summarized in Fig. 1.

Share Size. The constructions of Tonkikh and Freitas [TF24], Benhamouda, Halevi, and Stambler [BHS23], and ours guarantee polynomial share size for approximations of weighted threshold structures. However, the share size of the approximations in [BHS23, TF24] also depend on the inverse of the gap $\beta - \alpha$, which leads to an increase of the share size when targeting ramp weighted threshold access structures with small gaps. With respect to the proposals from Shamir [Sha79] and Garg et al. [GJM+23], their share size depends on the weights, which can be exponential in terms of the number of parties as stated in Theorem 2.8. For this reason, these proposals are more suitable for the cases in which there are lots of parties with small weights. The size of the shares of the scheme by Beimel and Weinreb [BW06] does not have this direct dependence with the size of weights, and the share size is quasipolynomial in the number of parties.

Access Structure. The main general constructions for weighted threshold access structures are the ones from Shamir [Sha79] and Beimel and Weinreb [BW06]. The proposals in [BHS23, GJM+23, TF24] rely on the more flexible setting of ramp weighted threshold access structures.

In our case, we approximate a weighted threshold access structure by another one that admits small weights. The approximation error is $\frac{1}{\text{polylog}(n)} = o(1)$ Furthermore, since our proposal has the error as an input parameter, we are able to tune the accuracy of the approximation as desired. In contrast, if we approximate weighted threshold access structures by a ramp weighted threshold access structures, it is hard to establish an upper bound on the gap that guarantees efficient schemes with a certain approximation error.

Privacy and Linearity. The previous works [Sha79, BW06, BHS23, TF24] present linear schemes that have perfect privacy. These properties facilitate their use as a building block in secure multiparty computation applications. This is also the case of our construction.

5 Computational Secret Sharing Schemes for Approximate Weighted Threshold Access Structures

In this section, we construct computational secret sharing schemes for weighted threshold access structures with small share size. First, we introduce some auxiliary results necessary for our work. Later, we present the construction and show how to quantify the public information size with our approximation technique.

5.1 Succinct Computational Secret Sharing Schemes

In a recent work, Applebaum et al. [ABI+23] construct computational secret sharing schemes with small share size for a wide set of access structures. In more detail, they introduce a new cryptographic primitive known as Projective Pseudorandom Generator (pPRG), show how to construct it from several assumptions such as RSA or the existence of one-way functions, and use it to obtain succinct computational secret sharing schemes, i.e. schemes whose share size is considerably small. We defer the definition of pPRG to [FG24].

Theorem 5.1 ([ABI+23]). *Under the subexponential (resp. polynomial) RSA assumption, there exists a subexponential-robust pPRG (resp. polynomial-robust pPRG) with subexponential stretch (resp. arbitrary polynomial stretch) whose projective keys and public parameters are both strongly succinct, i.e. of length* $\log(m) \cdot \text{poly}(\lambda)$*, where* m *is the output length and* λ *is the security parameter. The running time of generating the* m*-bit output of the pPRG is* $\tilde{O}(m) \cdot \text{poly}(\lambda)$*.*

For our purposes, we require a generalization of the notion of pPRG known as block-pPRG (see [ABI+23, FG24] for details). Block-pPRG are important

because they are the building blocks for computational secret sharing schemes for monotone circuits with unbounded fan-in in which the share size is polylogarithmic in the number of gates. This is summarized in the following theorem.

Theorem 5.2 ([ABI+23]). *Let λ be the security parameter. Assume that there is a robust block-pPRG in which the length of the projective keys is $\log(m\lambda) \cdot \mathrm{poly}(\lambda)$ and the length of the public parameters is $\log(m\lambda) \cdot \mathrm{poly}(\lambda)$, where m is the output length (number of blocks) of the generator and each block is of length λ. Then, there is a computational secret sharing scheme for monotone unbounded fan-in circuits whose share size is $\log(m\lambda) \cdot \mathrm{poly}(\lambda)$ and its public information size is $\mathrm{poly}(\log(m), \lambda) + m_\wedge \lambda$, where m is the number of gates and $m_\wedge$ is the number of AND-gates.*

5.2 Scheme Construction

A direct application of Theorem 5.1 and Theorem 5.2 to the polynomial size logarithmic depth monotone circuits for monotone WTF of Theorem 2.3 leads to the construction of a computational secret sharing scheme for weighted threshold access structures with polylogarithmic share size and public information of polynomial size in the number of parties. This is stated in the next theorem.

Theorem 5.3 (Theorem 1.4 restated). *Let λ be the security parameter. Under the subexponential RSA assumption, any weighted threshold access structure over n parties admits a computational secret sharing scheme where the size of the shares is $\mathrm{poly}(\log(n), \lambda)$ and the size of the public information is $\mathrm{poly}(n, \lambda)$.*

Proof. Given a weighted threshold access structure f, we use Theorem 2.3 to compute its polynomial size logarithmic depth monotone circuit. Then, we apply Theorem 5.1 and Theorem 5.2 to this circuit to obtain the desired computational secret sharing scheme.

The main drawback of the secret sharing scheme construction of Theorem 5.3 is that its public information size corresponds to the size of the monotone circuit, which is a polynomial of unspecified degree. For this reason, it is worth trying to pin down the degree of that polynomial.

In order to do so, we combine our results on low-weight approximators for monotone LTFs with the work from Applebaum et al. [ABI+23]. In particular, we construct computational secret sharing schemes for approximate weighted threshold access structures that maintain the share size of Theorem 5.3 and whose public information size is a polynomial of concrete degree. The main statement follows.

Theorem 5.4. *Let λ be the security parameter, let $\kappa(\varepsilon) = 2^{-O(\log^3(\frac{1}{\varepsilon}))}$ and let $\mu(n) \in \mathbb{R}$ be a function that satisfies $\mu(n) \geq 2\sqrt{n+1}$. Under the subexponential RSA assumption, for any weighted threshold access structure over n parties f and $\varepsilon \in (0,1)$ with $\kappa(\varepsilon) \leq \hat{f}(n)\mu(n)$, there exists a weighted threshold access*

structure over n parties that is ε-close to f admitting a computational secret sharing scheme where the size of the shares is $\mathrm{poly}\left(\log\left(\mu(n), \frac{1}{\varepsilon}\right), \lambda\right)$ *and the size of public information is* $O\left(\mu(n)^{10.6}\left(\frac{1}{\varepsilon}\right)^{O\left(\log^2\left(\frac{1}{\varepsilon}\right)\right)}, \mathrm{poly}\left(\lambda\right)\right)$.

Proof. The first part of the proof is analogue to that of Theorem 4.1, i.e. we use the switching lemmas and Theorem 3.1 to obtain a weighted threshold access structure g with weight vector $\boldsymbol{v}$, threshold v_0, and $\mathrm{dist}(f, g) < \varepsilon$.

From there, instead of applying Shamir's virtualization technique, we use Theorem 2.7 to obtain a monotone formula for g of size $O\left(\mu(n)^{10.6}\left(\frac{1}{\varepsilon}\right)^{O\left(\log^2\left(\frac{1}{\varepsilon}\right)\right)}\right)$. Now, since a monotone formula is indeed a monotone circuit, we can combine Theorem 5.1 and Theorem 5.2 to construct a computational secret sharing scheme for g where the size of the shares is $\mathrm{poly}\left(\log\left(\mu(n), \frac{1}{\varepsilon}\right), \lambda\right)$ and the size of public information is $O\left(\mu(n)^{10.6}\left(\frac{1}{\varepsilon}\right)^{O\left(\log^2\left(\frac{1}{\varepsilon}\right)\right)}, \mathrm{poly}\left(\lambda\right)\right)$. □

Now, we can mimic the approach done in the information-theoretic setting to obtain the following result as a consequence of Theorem 5.4.

Theorem 5.5. *Let $k \in \mathbb{N}$. Under the subexponential RSA assumption, for any weighted threshold access structure f over n parties, there exists a weighted threshold access structure that is $\frac{1}{\log^k(n)}$-close to f admitting a computational secret sharing scheme where the size of the shares is* $\mathrm{poly}\left(\log(n), \lambda\right)$ *and the size of the public information is* $O\left(n^{10.6+o(1)}, \mathrm{poly}\left(\lambda\right)\right)$.

Proof. The proof follows the same steps as the proof of Theorem 4.3. The only difference is that we apply Theorem 5.4 to the intermediate weighted threshold access structure f' instead of Theorem 4.1. In this way, at the end of the procedure we obtain a computational secret sharing scheme for g with shares of size $\mathrm{poly}\left(\log\left(n\right), \lambda\right)$ and public information of size $O\left(n^{10.6+o(1)}, \mathrm{poly}\left(\lambda\right)\right)$. □

5.3 Comparison with State-of-the-Art Proposals

The construction of Beimel and Weinreb [BW06] has polynomial share size, while the one in Theorem 5.3 has polylogarithmic share size. Nevertheless, following the arguments of [ABI+23], under the stronger assumption that subexponentially secure one-way functions exist, the scheme of Beimel and Weinreb could reach the same share size as that of Theorem 5.3 because both rely on the same monotone circuit for WTFs. However, the public information in Theorem 5.3 and Theorem 5.5 is linear in the number of gates, whereas in the work of Beimel and Weinreb the size of the public information is linear in the number of wires. This results in a quadratic improvement for our proposals in the context of general circuits. The key difference lies in the use of the computational scheme by Applebaum et al. [ABI+23] for monotone circuits in Theorem 5.3, while Beimel and Weinreb construction adopts Yao's scheme [Yao89]. Another difference is

that the construction of Beimel and Weinreb [BW06] assumes the existence of secure one-way functions, while the constructions in Theorem 5.3 and Theorem 5.5 are based on the RSA assumption. This is summarized in Fig. 1.

Regarding the access structure, both the Beimel and Weinreb scheme and Theorem 5.3 can realize any weighted threshold access structure. However, in Theorem 5.5, we approximate the original weighted threshold access structure to further reduce the share size.

Acknowledgments. We thank Amos Beimel, Carles Padró, and the TCC reviewers for valuable comments and suggestions.

The authors are supported by grant 2021 SGR 00115 from the Government of Catalonia, by the project ACITHEC PID2021-124928NB-I00 funded by MCIN/AEI/10.13039/501100011033/FEDER, EU, and by the project HERMES, funded by the European Union NextGenerationEU/PRTR via INCIBE.

References

ABI+23. Applebaum, B., Beimel, A., Ishai, Y., Kushilevitz, E., Liu, T., Vaikuntanathan, V.: Succinct computational secret sharing. In: STOC 2023, pp. 1553–1566. ACM (2023)

BCC+21. Breidenbach, L., et al.: Chainlink 2.0: Next steps in the evolution of decentralized oracle networks. Technical report, Chainlink Labs (2021)

BD91. Brickell, E.F., Davenport, D.M.: On the classification of ideal secret sharing schemes. J. Cryptology **4**(73), 123–134 (1991)

Bei11. Beimel, A.: Secret-sharing schemes: a survey. In: Chee, Y.M., Guo, Z., Ling, S., Shao, F., Tang, Y., Wang, H., Xing, C. (eds.) IWCC 2011. LNCS, vol. 6639, pp. 11–46. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-20901-7_2

BF20. Beimel, A., Farràs, O.: The share size of secret-sharing schemes for almost all access structures and graphs. IACR Cryptol. ePrint Arch. **2020**, 664 (2020)

BHS23. Benhamouda, F., Halevi, S., Stambler, L.: Weighted secret sharing from wiretap channels. In: 4th Conference on Information-Theoretic Cryptography, ITC 2023, vol. 267. LIPIcs, pp. 8:1–8:19 (2023)

BL88. Benaloh, J.C., Leichter, J.: Generalized secret sharing and monotone functions. In: Goldwasser, S., (ed.) CRYPTO '88, vol. 403. LNCS, pp. 27–35. Springer (1988)

Bla79. Blakley, G.R.: Safeguarding cryptographic keys. In: Proc. of the 1979 AFIPS National Computer Conference, vol. 48. AFIPS Conference proceedings, pp. 313–317. AFIPS Press (1979)

Bri89. Brickell, E.F.: Some ideal secret sharing schemes. J. Combin. Math. Combin. Comput. **6**, 105–113 (1989)

BTW08. Beimel, A., Tassa, T., Weinreb, E.: Characterizing ideal weighted threshold secret sharing. SIAM J. Discrete Math. **22**(1), 360–397 (2008)

BW06. Beimel, A., Weinreb, E.: Monotone circuits for monotone weighted threshold functions. Inform. Process. Lett. **97**(1), 12–18 (2006). Conference version: Proc. of 20th Annu. IEEE Conf. on Computational Complexity, pp. 67–75 (2005)

Cho61. Chow, C.K.: On the characterization of threshold functions. In: 2nd Annual Symposium on Switching Circuit Theory and Logical Design (SWCT 1961), pp. 34–38 (1961)
CTL22. Chen, Q., Tang, C., Lin, Z.: Efficient explicit constructions of multipartite secret sharing schemes. IEEE Trans. Inf. Theory **68**, 601–631 (2022)
DDFS14. De, A., Diakonikolas, I., Feldman, V., Servedio, R.A.: Nearly optimal solutions for the chow parameters problem and low-weight approximation of halfspaces. J. ACM **61**(2), April 2014
DPTX24. Das, S., Pinkas, B., Tomescu, A., Xiang, Z.: Distributed randomness using weighted vrfs. Cryptology ePrint Archive, Report 2024/198 (2024). https://eprint.iacr.org/2024/198
FG24. Farràs, O., Guiot, M.: Reducing the share size of weighted threshold secret sharing schemes via chow parameters approximation. Cryptology ePrint Archive, Report 2024/772 (2024). https://eprint.iacr.org/2024/772
FP12. Farràs, O., Padró, C.: Ideal hierarchical secret sharing schemes. IEEE Trans. Inf. Theory **58**(5), 3273–3286 (2012)
GJM+23. Garg, S., Jain, A., Mukherjee, P., Sinha, R., Wang, M., Zhang, Y.: Cryptography with weights: Mpc, encryption and signatures. In: Handschuh, H., Lysyanskaya, A. (eds.) Advances in Cryptology – CRYPTO 2023, pp. 295–327. Springer, Switzerland (2023)
Hås94. Håstad, J.: On the size of weights for threshold gates. SIAM J. Discrete Math. **7**(3), 484–492 (1994)
KGH83. Karnin, E.D., Greene, J.W., Hellman, M.E.: On secret sharing systems. IEEE Trans. Inf. Theory **29**(1), 35–41 (1983)
KRDO17. Kiayias, A., Russell, A., David, B., Oliynykov, R.: Ouroboros: a provably secure proof-of-stake blockchain protocol. In: Katz, J., Shacham, H. (eds.) Advances in Cryptology – CRYPTO 2017, pp. 357–388. Springer (2017)
KW93. Karchmer, M., Wigderson, A.: On span programs. In: 8th Structure in Complexity Theory, pp. 102–111 (1993)
LV18. Liu, T., Vaikuntanathan, V.: Breaking the circuit-size barrier in secret sharing. In: 50th STOC, pp. 699–708 (2018)
Mig83. Mignotte, M.: How to share a secret. In: Beth, T., (ed.) Cryptography, pp. 371–375. Springer, Heidelberg (1983)
Mo23. Mo, S.: Ideal hierarchical secret sharing and lattice path matroids. Des. Codes Cryptogr. **91**(4), 1335–1349 (2023)
MPSV99. Morillo, P., Padró, C., Sáez, G., Villar, J.L.: Weighted threshold secret sharing schemes. Inform. Process. Lett. **70**(5), 211–216 (1999)
Mur71. Muroga, S.: Threshold Logic and Its Applications. Wiley-Interscience (1971)
O'D14. O'Donnell, R.: Analysis of Boolean Functions. Cambridge University Press, USA (2014)
OS11. O'Donnell, R., Servedio, R.A.: The chow parameters problem. SIAM J. Comput. **40**(1), 165–199 (2011)
Pad24. Padró, C.: Efficient representation of lattice path matroids. Ann. Comb. (2024)
PS00. Padró, C., Sáez, G.: Secret sharing schemes with bipartite access structure. IEEE Trans. Inf. Theory **46**(7), 2596–2604 (2000)
SB91. Siu, K.-Y., Bruck, J.: On the power of threshold circuits with small weights. SIAM J. Discret. Math. **4**(3), 423–435 (1991)
Ser04. Servedio, R.A.: Monotone Boolean formulas can approximate monotone linear threshold functions. Discrete Appl. Math. **142**(1–3), 181–187 (2004)

Ser06. Servedio, R.A.: Every linear threshold function has a low-weight approximator. In: 21st Annual IEEE Conference on Computational Complexity (CCC'06), pp. 18–32 (2006)
Sha79. Shamir, A.: How to share a secret. Commun. ACM **22**, 612–613 (1979)
Sim88. Simmons, G.J.: How to (really) share a secret. In: CRYPTO, pp. 390–448 (1988)
Tas07. Tassa, T.: Hierarchical threshold secret sharing. J. Cryptology **20**(2), 237–264 (2007)
TD09. Tassa, T., Dyn, N.: Multipartite secret sharing by bivariate interpolation. J. Cryptology **22**(2), 227–258 (2009)
TF24. Tonkikh, A., Freitas, L.: Swiper: a new paradigm for efficient weighted distributed protocols. In: Proceedings of the 43rd ACM Symposium on Principles of Distributed Computing, PODC '24, pp. 283–294. Association for Computing Machinery (2024)
Yao89. Yao, A.C.: Unpublished manuscript, 1989. Presented at Oberwolfach and DIMACS Workshops

New Upper Bounds for Evolving Secret Sharing via Infinite Branching Programs

Bar Alon[3], Amos Beimel[1], Tamar Ben David[2], Eran Omri[2(✉)], and Anat Paskin-Cherniavsky[2]

[1] Department of Computer Science, Ben Gurion University, Beer Sheva, Israel
[2] Department of Computer Science, Ariel University, Ariel Cyber Innovation Center (ACIC), Ariel, Israel
omrier@ariel.ac.il
[3] Georgetown University, Washington, USA

Abstract. Evolving secret-sharing schemes, defined by Komargodski, Naor, and Yogev [TCC 2016B], are secret-sharing schemes in which there is no a-priory bound on the number of parties. In such schemes, parties arrive one by one; when a party arrives, the dealer gives it a share and cannot update this share in later stages. The requirement is that some predefined sets (called authorized sets) should be able to reconstruct the secret, while other sets should learn no information on the secret. The collection of authorized sets that can reconstruct the secret is called an evolving access structure. The challenge of the dealer is to be able to give short shares to the current parties without knowing how many parties will arrive in the future. The requirement that the dealer cannot update shares is designed to prevent expensive updates.

Komargodski et al. constructed an evolving secret-sharing scheme for every monotone evolving access structure; the share size of the t^{th} party in this scheme is 2^{t-1}. Recently, Mazor [ITC 2023] proved that evolving secret-sharing schemes require exponentially-long shares for some evolving access structures, namely shares of size $2^{t-o(t)}$. In light of these results, our goal is to construct evolving secret-sharing schemes with non-trivial share size for wide classes of evolving access structures; e.g., schemes with share size 2^{ct} for $c < 1$ or even polynomial size. We provide several results achieving this goal: (1) We define layered infinite branching programs representing evolving access structures, show how to transform them into generalized infinite decision trees, and show how to construct evolving secret-sharing schemes for generalized infinite decision trees. Combining these steps, we get a secret-sharing scheme realizing the evolving access structure. As an application of this framework, we construct an evolving secret-sharing scheme with non-trivial share size for access structures that can be represented by layered infinite branching programs with width at layer t of at most $2^{0.15t}$. If the width is polynomial, then we get an evolving secret-sharing scheme with quasi-polynomial share size. (2) We construct efficient evolving secret-sharing schemes for dynamic-threshold access structures with high dynamic-threshold and for infinite 2-slice and 3-slice access structures. (3) We prove lower bounds on the share

E. Boyle and M. Mahmoody (Eds.): TCC 2024, LNCS 15367, pp. 548–580, 2025.
https://doi.org/10.1007/978-3-031-78023-3_18

size of evolving secret-sharing schemes for infinite k-hypergraph access structures and for infinite directed st-connectivity access structures. As a by-product of the lower bounds, we provide the first non-trivial lower bound for *finite* directed st-connectivity access structures for general secret-sharing schemes.

1 Introduction

In the common model of secret-sharing schemes [13,28,41], there are n parties and a dealer, which holds a secret. The dealer applies some randomized algorithm to the secret, resulting in n strings, called shares; it gives the i^{th} share to the i^{th} party. There are two requirements. (1) correctness: some predefined subsets of the parties can jointly reconstruct the secret from their shares, and (2) security: any other set gets no information on the secret. The collection of predefined authorized sets is called an access structure. These schemes are well-studied and have many applications. This model assumes that the number of parties is known when preparing the shares and giving the shares to the parties; furthermore, the sharing algorithm and the share size are determined by the number of parties, e.g. in the best-known secret-sharing scheme for an arbitrary n-party access structure the share size is $2^{0.585n}$ [5].

The assumption that the number of parties is known in advance is problematic in many scenarios. Of course, one can take some upper bound on the number of parties. On one hand, if this bound is big, then the share size will be large even if only few parties actually participate in the scheme. On the other hand, if this bound is small, then there is a risk that too many parties will arrive and no further shares can be produced; this will require an expensive re-sharing of the secret and updating all shares (which can be impossible if some parties are temporally off-line). Thus, we need to consider models with an unbounded number of parties.

Komargodski, Naor, and Yogev [31] defined *evolving secret-sharing schemes* with an unbounded number of parties. In this model, parties arrive one after the other and the number of parties that will arrive is not known. At the beginning of the execution, the dealer holds a secret (as in the common model). When a party arrives, the dealer computes a share and gives it to the party; this share cannot be updated in the future. Thus, when preparing the t^{th} share, the dealer cannot assume any bound on the number of parties that will eventually arrive; the size of the t^{th} share should be measured as a function of t. We require correctness and privacy with respect to an *evolving access structure*, where the parties are $\{p_i\}_{i\in\mathbb{N}}$ and the evolving access structure is a collection of finite subsets of the parties that are authorized to reconstruct the secret.[1]

We next briefly discuss the known results on evolving secret-sharing schemes. A longer discussion can be found in Sect. 1.2. Komargodski et al. [31] showed that every monotone evolving access structure can be realized by an evolving

[1] We assume that the order that the parties arrive is known in advance, or, alternatively, the t^{th} party to arrive assumes the role of the t^{th} party.

secret-sharing scheme; in this scheme the size of the t^{th} share is 2^{t-1}. Recently, Mazor [36] proved that evolving secret-sharing schemes require exponentially long shares – there is an evolving access structure such that in any evolving secret-sharing scheme realizing it the size of share of the t^{th} party is $2^{t-o(t)}$ (for infinitely many t's). On the positive side, Komargodski et al. and follow-up works [9,10,16,21–23,26,32,37–39,43,44] constructed efficient evolving secret-sharing schemes for natural access structures.

1.1 Our Results

Having the result of Mazor [36] in mind, our research has 3 goals. Our first goal is to identify a class of evolving access structures that can be realized with an evolving secret-sharing scheme that has non-trivial share size; e.g., schemes with share size 2^{ct} for $c < 1$ or sub-exponential share size. We do so by defining a computational model that represents evolving access structures, and then show how to realize it with a non-trivial evolving secret-sharing scheme. Our second goal is to construct efficient evolving secret-sharing schemes, i.e., schemes with polynomial-size shares, for several natural classes of evolving access structures. Finally, we provide lower bounds on the share size for several interesting evolving access structures. See Table 1 for a summary of our results. Below, we review our results in more detail.

Evolving Secret-Sharing Schemes for Infinite Branching Programs. We abstract and generalize the constructions of [31,32] of evolving secret-sharing schemes. We define infinite branching programs (defined informally below and formally in Definition 3.1), which represent evolving access structures, show how to transform them to a simpler computation model we call generalized infinite decision trees (abbreviated GIDT), and show how to construct evolving secret-sharing schemes for generalized infinite decision trees. We defer the discussion on GIDTs to Sect. 1.3. Thus, to construct an evolving secret-sharing scheme for an evolving access structure using our framework, one can either represent it as an infinite branching program and use our transformation to construct a generalized infinite decision tree or directly represent the evolving access structure as a generalized infinite decision tree. We note that many secret-sharing schemes for *finite* access structures use a representation of the access structure by some computational model to construct a secret-sharing scheme realizing the access structure, e.g., CNF and DNF formulas are used in [28], monotone formulas are used in [12], and monotone span programs are used in [29].

An infinite monotone non-deterministic branching program (abbreviated IBP) computes a monotone function $f : \{0,1\}^* \to \{0,1\}$; this function is the characteristic function of an evolving access structure.[2] An IBP is an infinite directed acyclic graph G, where each edge is labeled by a variable from $\{x_i\}_{i\in\mathbb{N}}$ or by the constant 1. For every input $\sigma \in \{0,1\}^t$ (interpreted as an assignment to

[2] That is, $f(\sigma_1,\ldots,\sigma_t) = 1$ if an only if $\{p_i : 1 \le i \le t, \sigma_i = 1\}$ is in the access structure.

the variables $x_1, \ldots, x_t$) it holds that $f(\sigma) = 1$ if and only if there exists a path in G from the source vertex to a leaf (a vertex without out-going edges) that is satisfied by σ, namely, each edge in the path is either labeled by 1 or labeled with a variable x_i such that $1 \leq i \leq t$ and $\sigma_i = 1$ (see Definition 3.1 for a formal definition). A layered IBP (abbreviated LIBP) is an IBP where the vertices are partitioned into finite layers, all edges are directed from some layer i to layer $i+1$, and all edges entering layer i are labeled by either 1 or by x_i. See Fig. 1 for an illustration of an LIBP. Intuitively, when using LIBPs for constructing an evolving secret-sharing scheme, passing through an edge that is labeled by x_i is interpreted as using the share of the i^{th} party in the reconstruction.

We show how to reduce the question of realizing an evolving access structure represented as an LIBP to the question of realizing certain finite access structures. One parameter that determines the share size of the resulting evolving secret-sharing scheme is the width of the LIBP, where the width of an LIBP at layer t, denoted by $w(t)$, is the number of vertices in layer t. We prove the following theorem.

Theorem 1.1 (Realizing LIBPs – Informal). *Let B be an LIBP. There exists an evolving secret-sharing scheme realizing B in which the share of party p_t is the shares of p_t in $\left(\prod_{1 \leq j \leq \log t} w(2^j)\right)$ secret-sharing schemes realizing some finite access structures.*

As an application of this framework, we construct evolving secret-sharing schemes for LIBPs with bounded width.

Theorem 1.2 (Realizing Bounded Width LIBPs – Informal). *For every function $\varepsilon(t) < 0.04$, every LIBP of width $w(t) \leq 2^{\varepsilon(t) \cdot t}$ can be realized by an evolving secret-sharing scheme in which the share size of the t^{th} party is $2^{O\left(\min\{*\}\, \varepsilon(t) \log t, \sqrt{\varepsilon(t)}\right) \cdot t}$.*

In the above theorem, $\varepsilon(t)$ can be an arbitrary function that is smaller than 1. For example, for LIBPs whose width is at most $2^{0.15t}$ (i.e., $\varepsilon(t) = 0.15$), we get share size $2^{0.97t}$. As another example, if the width is polynomial (i.e., $\varepsilon(t) = O(\log(t)/t)$), then we get an evolving secret-sharing scheme with quasi-polynomial share size. Thus, evolving access structures that can be represented by LIBPs with bounded width can be realized with non-trivial share size. This is in contrast with the lower bound of [36], proving that there exists an evolving access structure requiring shares of size at least $2^{t-o(t)}$.

As another application of our framework, we construct evolving secret-sharing schemes for evolving directed *layered* st-connectivity access structures. In this access structure, the parties are edges of an infinite layered graph, and a set is authorized if and only if it contains a path from a source vertex to a target vertex. Additionally, the order of arrival is such that first the parties from layer 0 to layer 1 arrive, then from layer 1 to layer 2, and so on. The share size in our construction is at most polynomially larger than the share size of the best-known scheme for finite directed st-connectivity access structure [12].

Theorem 1.3. *Any evolving directed layered st-connectivity access structure can be realized with an evolving secret-sharing scheme, such that for all $t \in \mathbb{N}$ the share size of the t^{th} party is $t^{O(\log t)}$.*

Efficient Evolving Secret-Sharing Schemes for Dynamic-Threshold for Large Thresholds. In an evolving $\mathrm{tr}(\cdot)$-dynamic-threshold access structure, where $\mathrm{tr} : \mathbb{N} \to \mathbb{N}$ is a function, a set of parties A is authorized if, for some $t \in \mathbb{N}$, the set A contains at least $\mathrm{tr}(t)$ parties from the first t parties. This is a natural generalization of a fixed k-threshold evolving access structure, where a set is authorized if at some point k parties have arrived [30]. Dynamic-threshold evolving access structures were first considered by Komargodski and Paskin-Cherniavsky [32], who constructed an evolving $\mathrm{tr}(\cdot)$-dynamic-threshold secret-sharing scheme in which the share size of the t^{th} party is $\tilde{O}(t^4)$. Subsequent works on dynamic-threshold appeared in [9,10,43]. Most related to our results is the work by Xing and Yuan [43], who constructed a more efficient scheme, when the dynamic-threshold function is *small*, i.e., $\mathrm{tr}(t) = t^\beta$ for some constant $0 < \beta < 1$. We, on the other hand, show how to construct a more efficient evolving secret-sharing scheme when the dynamic-threshold function is *large*, i.e., $\mathrm{tr}(t) \geq t - t^\beta$ for some constant $0 < \beta < 1$.

Theorem 1.4 ($(t - t^\beta)$-Dynamic-Threshold Secret-Sharing Schemes – Informal). *Let $\beta \in (0, 1)$ be a constant and $\mathrm{tr}(t) \geq t - t^\beta$. There exists an evolving secret-sharing scheme realizing the evolving $\mathrm{tr}(\cdot)$-dynamic-threshold access structure in which the share size of the t^{th} party is at most $\tilde{O}(t^{1+2\sqrt{\beta}+\beta})$.*

For all $\beta < 1$, our scheme is more efficient than the scheme of [32]. For example, for $\mathrm{tr}(t) = t - t^{1/4}$, the share size in our scheme is $\tilde{O}(t^{2.25})$ (compared to $\tilde{O}(t^4)$ in the scheme of [32]).

Efficient Evolving Secret-Sharing Schemes for Slice Access Structures. An infinite k-slice access structure is an access structure where all sets of size at most $k-1$ are unauthorized, all finite sets of size at least $k+1$ are authorized, and each set of size k can be either authorized or unauthorized; that is, to specify a k-slice access structure we need to specify which sets of size k (i.e., in the k^{th}-slice) are authorized. Secret sharing for finite slice access structures have been extensively studied (see, for example, the citations in [3]); they are equivalent to conditional disclosure of secrets (CDS) protocols, a cryptographic primitive introduced by Gertner et al. [27]. 2-slice access structures are also known as forbidden-graph secret-sharing schemes [42]. We construct evolving secret-sharing schemes with sub-linear size shares for evolving 2-slice and 3-slice access structures.

Theorem 1.5 (Realizing 2-Slice and 3-Slice Access Structures – Informal). *Every 2-slice and 3-slice access structure can be realized by an evolving secret-sharing scheme in which the share size of the t^{th} party is $2^{\tilde{O}\left((\log t)^{\varepsilon+1/\sqrt{2}}\right)}$, for any constant $\varepsilon > 0$.*

The share size in the best-known secret-sharing schemes for finite n-party k-slice access structures for constant k is $2^{\tilde{O}(k+\sqrt{k \log n})} = 2^{\tilde{O}(\sqrt{\log n})}$ (using the CDS protocols of [34,35] and the transformation of [2,7]). For infinite 2-slice and infinite 3-slice access structures, the share size in our evolving secret-sharing scheme is comparable.

Lower Bounds. We prove lower bounds on the share size of evolving secret-sharing schemes for two natural classes of access structures. We first consider infinite directed st-connectivity access structures; in these access structures the parties are edges of an infinite graph (with some order on the edges determining when they arrive in the evolving access structure); a set of parties (i.e., edges) is authorized if and only if it contains a path from a fixed source vertex to a fixed target vertex. We prove the following lower bound.

Theorem 1.6 (Lower Bounds for Evolving Directed st-Connectivity – Informal). *There exists an evolving directed st-connectivity access structure, such that in every evolving secret-sharing scheme realizing it the share size of the t^{th} party, for infinitely many t's, is at least $\Omega(t)$.*

As a by-product of the lower bounds, we provide the first non-trivial lower bound for finite directed st-connectivity access structures for general secret-sharing schemes.

Theorem 1.7 (Lower Bounds for Finite Directed st-connectivity – Informal). *For every $n \in \mathbb{N}$ there exists an n-party directed st-connectivity access structure, such that in every secret-sharing scheme realizing it, there exists at least one party whose share size is at least $\Omega(\sqrt{n})$.*

Previously, no non-trivial lower bound was known for finite st-connectivity access structures for general secret-sharing schemes. A lower bound of $n^{\Omega(\log n)}$ for linear secret-sharing schemes was proven by Pitassi and Robere [40].

We also prove lower bounds on the share size of evolving secret-sharing schemes for infinite k-hypergraph access structures for a constant k; in these access structures the minimal authorized sets are of size exactly k; however, there can be large unauthorized sets. Our lower bounds for infinite k-hypergraph access structures for constant k are tight as a fairly naive evolving secret-sharing scheme provides a matching upper bound on the share size.

Theorem 1.8 (Lower Bounds for Evolving k-Hypergraphs – Informal). *For every constant k there exists an evolving k-hypergraph access structure, such that in every evolving secret-sharing scheme realizing it, the share size of the t^{th} party, for infinitely many t's is at least $\Omega(t^{k-2})$.*

1.2 Previous Results

1.2.1 Evolving Secret-Sharing Schemes

We first mention two related works that preceded [31]. Cachin [15] and Csirmaz and Tardos [20] studied online secret sharing, which is similar to evolving secret-sharing schemes. As in evolving secret-sharing schemes, in online secret-sharing,

Table 1. A summary of the known lower and upper bounds on the share size in evolving secret-sharing schemes for the evolving access structures considered in this paper.

	Upper bounds	Lower bounds
General evolving access structures	2^{t-1} [31]	$2^{t-o(t)}$ [36]
LIBPs with width $\leq 2^{\varepsilon(t)\cdot t}$	$2^{O(\min\{(\sqrt{\varepsilon(t)}, \varepsilon(t)\log t\}\cdot t)}$ Theorems 3.15, 3.17	
Evolving k-hypergraphs	$O(t^{k-1})$ [1, Theorem (A.1)]	$\Omega(t^{k-2})$ [1, Theorem (6.8)]
Evolving 2, 3-slices	$2^{\tilde{O}\left((\log t)^{1/\sqrt{2}+\varepsilon}\right)}$ for any constant $\varepsilon > 0$ [1, Thms. (5.4), (5.6)]	
$(t - t^{\beta})$-dynamic threshold for a constant $\beta \in (0, 1)$.	$\tilde{O}(t^{1+2\sqrt{\beta}+\beta})$ [1, Theorem (4.1)]	
Evolving directed st-connectivity	$t^{O(\log t)}$ (layered graphs) [1, Theorem (3.19)]	$\Omega(t)$ [1, Theorem (6.7)]

parties arrive one after the other and the number of parties is unbounded. However, in [20] the number of authorized sets that a party can join is bounded and in [15] there is a large public bulletin board.

Evolving Threshold Secret-Sharing Schemes. Komargodski et al. [31] constructed an evolving k-threshold secret-sharing scheme for any constant k in which the size of the share of the t^{th} party is $O(k \log t)$. D'Arco, De Prisco, and De Santis [21] constructed an improved evolving 3-threshold secret-sharing scheme in which the size of the share of the t^{th} party is $(4/3 + \varepsilon) \log t$ for arbitrary small ε (the share size in the evolving 3-threshold scheme of [31] is at least $2 \log t$). D'Arco et al. [22] constructed probabilistic evolving k-threshold secret-sharing schemes in which the share size is $O(1)$; in these schemes, the secret is reconstructed only with a constant probability $p < 1$. Other constructions of evolving threshold secret-sharing schemes were given in [17,23,37,44].

Evolving Dynamic-Threshold Secret-Sharing Schemes. Komargodski and Paskin-Cherniavsky [32] constructed an evolving $\mathrm{tr}(\cdot)$-dynamic-threshold secret-sharing scheme in which the share size of the t^{th} party is $O(t^4 \log t)$. Xing and Yuan [43] showed an alternative construction of evolving $\mathrm{tr}(\cdot)$-dynamic secret sharing scheme. Their construction saves a factor of $\log t$ compared to the evolving scheme of a [32]. They also considered the evolving $\mathrm{tr}(t) = t^{\beta}$-dynamic-threshold secret-sharing schemes (that is, the dynamic-threshold is small) and showed that it can be realized by an evolving secrets-sharing scheme in which the share size of the t^{th} party is $O(t^{4\beta})$. We show that this can be achieved (up to a factor of $\log t$) by a variation of the scheme of [32]. In this paper we use the construction of [43] to construct evolving $\mathrm{tr}(t)$-dynamic-threshold secret-sharing schemes for large dynamic-threshold $\mathrm{tr}(\cdot) \geq t - t^{\beta}$ for some constant β.

Evolving Secret-Sharing Schemes for Other Access Structures. Chaudhury, Dutta, and Sakurai [16] constructed evolving threshold schemes that can be implemented in the complexity class AC^0. Dutta, Roy, Fukushima, Kiyomoto, and Sakurai [24] and Phalakarn, Suppakitpaisarn, Attrapadung, and Matsuura [39] constructed evolving hierarchical secret-sharing schemes (in the latter paper the schemes are homomorphic). Beimel and Othman [9,10] constructed evolving ramp secret-sharing schemes, i.e., schemes in which there is a gap between the dynamic threshold $\mathrm{tr}_2(\cdot)$ for authorized sets and the dynamic-threshold $\mathrm{tr}_1(\cdot)$ for unauthorized sets. They showed that for every constants $0 < \alpha < \beta < 1$, there is an evolving $(\mathrm{tr}_1(t) = \alpha \cdot t, \mathrm{tr}_2(t) = \beta \cdot t)$-ramp secret-sharing scheme in which the size of the shares of each party is $O(1)$.

Computational Evolving Secret-Sharing Schemes. Francati and Venturi [26] defined a computational variant of evolving secret-sharing schemes. In this setting, the dealer and the adversary are computationally bounded, and the number of parties is bounded by some polynomial in the security parameter; the polynomial is unknown to the dealer. They constructed computational evolving secret-sharing schemes for many evolving access structures, including graphs access structures, DNF and CNF formulas access structures, monotone circuits access structures, threshold access structures, and dynamic-threshold access structures.

Since their results hold in the computational setting, they are incomparable to ours. Additionally, there is a difference in the representation of evolving access structures. While we represent them using infinite branching programs (IBP), Francati and Venturi [26] represent them as infinite monotone circuits (IMC). The two models have different complexity parameters (width for IBP and number of gates for IMC). Therefore, it is unclear which model can be used to give a better construction for evolving secret-sharing schemes, even for the same type of security (namely, computational or information-theoretic).

Evolving Conditional Disclosure of Secrets. Peter [38] defined evolving conditional disclosure of secrets (CDS) protocols. In this setting, parties arrive in sequential order and there is no a-priory bound on the number of parties. Each party holds a private input, and when it arrives, it sends a random message to a referee. At any stage of the protocol, the referee should be able to reconstruct a secret string held by all the parties from the messages it gets, if and only if the inputs of the parties that arrived satisfy some condition. Peter [38] accompanied the new definition with constructions of evolving CDS protocols for general evolving predicates, evolving min-sum predicates, and evolving constrained predicates.

1.2.2 Some Related Works on Secret-Sharing Schemes for Finite Access Structures

Secret-sharing schemes for arbitrary access structures were introduced by Ito, Saito, and Nishiseki [28]; they constructed for every monotone n-party access structure a secret-sharing scheme in which the size of the share of each party is

2^n. In a breakthrough work, Liu, and Vaikuntanathan [33] constructed a secret-sharing scheme for arbitrary access structures with share size $2^{0.944n}$. This was improved in a sequence of works [3–5,35]; currently, the best known secret-sharing schemes for arbitrary access structures were constructed by Applebaum and Nir [5] and have share size $2^{0.585n}$. The best known lower bound on the share size is by Csirmaz [18,19], proving that for every $n \in \mathbb{N}$ there is an n-party access structure in which the share size of at least one party is $\Omega(n/\log n)$ and its total share size is at least $\Omega(n^2/\log n)$.

We next mention some results for finite counterparts of the evolving access structures considered in this paper. Finite *undirected* st-connectivity access structures can be realized by a secret-sharing scheme in which each share and the secret is a bit [11]. The best known secret sharing scheme for finite *directed* st-connectivity access structures is by using the formula based-secret-sharing scheme of [12] and has share size $n^{O(\log n)}$ for realizing a graph with n edges. This scheme can be used to realize an access structure represented as a finite non-deterministic branching program of size n with share size $n^{O(\log n)}$. The best constructions for k-slice access structures are by various transformations from the k-server CDS protocols of [34,35]; the best schemes known today have share size $\min\{2^{O(k)+\tilde{O}(\sqrt{k \log n})}, kn \cdot 2^{\tilde{O}(\sqrt{k \log n})}, 2^{\tilde{O}(\sqrt{n})}\}$ [2,3,8]. The naive secret-sharing scheme for k-hypergraph access structures is to share the secret independently for each minimal authorized set, this results in share size $O(\binom{n}{k-1})$ per party. This can be improved by a factor of $\log n$ using a result of Erdös and Pyber [25]. Recently, it was proved by Beimel [6] that for every n and every $3 \leq k \leq \log n$, there is a k-hypergraph with n vertices in which the share size of at least one party is $\Omega(n^{1-1/(k-1)}/k)$ and its total share size is at least $\Omega(n^{2-1/(k-1)}/k)$.

1.3 Our Techniques

Layered Infinite Branching Programs and Generalized Infinite Decisions Trees. Our task is to design an evolving secret-sharing scheme for LIBPs. We do not know how to construct such a scheme directly. We know how to realize LIBPs when the infinite graph of the infinite branching program is a tree, using the evolving secret-sharing scheme of [31] for undirected st-connectivity. However, transforming a (layered) graph of a LIBP B to a tree results in a tree whose width is huge – for every path $u_0, u_{j_1}, \ldots, u_{j_t}$ from the source vertex to a vertex in the t^{th}-layer there is a vertex $u_{j_1,\ldots,j_t}$ in the t^{th} layer of the tree.

Following [31,32], we overcome this problem by partitioning the variables of the layered branching program into consecutive sets, called generations. The generations are defined by some function $h : \mathbb{N} \to \mathbb{N}$; the i^{th} generation contains the variables $x_{h(i-1)+1}, \ldots, x_{h(i)}$. In the infinite decision tree T we construct, there is a vertex $u_{j_1,\ldots,j_i}$ for any sequence of vertices $u_{j_1}, \ldots, u_{j_i}$ in layers $h(1), \ldots, h(i)$ respectively. If the width of the LIBP B in layer t (i.e., the number of vertices in the layer) is $w(t)$, then the number of vertices in the resulting infinite branching program T is $O\left(\prod_{1 \leq j \leq i} w(h(j))\right)$ (this is the expression in Theorem 1.1, taking $h(i) = 2^i$). We add an edge $(u_{j_1,\ldots,j_{i-1}}, u_{j_1,\ldots,j_{i-1},j_i})$ to T representing all paths

in B from $u_{j_{i-1}}$ in layer $h(i-1)$ to u_{j_i} in layer $h(i)$; this edge should be satisfied by an assignment σ if and only if σ satisfies some path in B from $u_{j_{i-1}}$ to u_{j_i}.

We abstract the above construction by defining a generalized infinite decisions tree (abbreviated GIDT), which is an infinite tree together with a partition function $h : \mathbb{N} \to \mathbb{N}$; a GIDT is an infinite tree in which each edge between layer $i-1$ and layer i in the tree is labeled by a predicate that depends on the variables in the i^{th} generation, i.e., on $x_{h(i-1)+1}, \ldots, x_{h(i)}$. To construct an evolving secret-sharing scheme for a GIDT, we first execute the evolving secret-sharing scheme of [31] for the tree; in this scheme the parties are the edges of the tree. Next, for each edge in the tree we take the share sh_e of the edge and share it using a secret-sharing realizing the predicate of the edge (here, again, we represent an access structure by a predicate).

Evolving Secret-Sharing Schemes for LIBPs with Bounded Width. The main application of our construction of evolving secret-sharing schemes for LIBPs is an evolving secret-sharing scheme with non-trivial share size realizing LIBPs with bounded width. In Theorems 3.15 and 3.17, we present two constructions for LIBPs with bounded width. Both constructions use the transformation from LIBPs to GIDTs and use the evolving secret-sharing scheme realizing the GIDT, that is, we use Theorem 1.1. For example, by Theorem 1.1, if the width of the LIBP is $w(t) = 2^{0.04t}$, then the number of shares of secret-sharing schemes for finite access structures that the t^{th} party holds is

$$O\left(\prod_{1\le j\le \log t} w(2^j)\right) = O\left(\prod_{1\le j\le \log t} 2^{0.04\cdot 2^j}\right) = O\left(2^{0.04\cdot\sum_{1\le j\le \log t} 2^j}\right)$$
$$\le O\left(2^{0.04\cdot 2^{\log(t+1)}}\right) = O\left(2^{0.04(t+1)}\right).$$

Thus, the number of shares for width $w(t) = 2^{0.04t}$ is non-trivial. We still need to specify how we realize the finite access structures determined by the labels of the GIDT. In the first construction, we use the best-known secret-sharing scheme for arbitrary n-party access structures of Applebaum and Nir [5]; the share size in this scheme is $2^{0.585n}$. In the second construction, we use the formula-based secret-sharing scheme of Benaloh and Leichter [12] using a monotone formula for the graph reachability problem. When the width of the LIBP is smaller than $2^{t/\log^2 t}$, the second construction is more efficient.

Evolving Secret-Sharing Schemes for Evolving dynamic-threshold Access Structure with Large Threshold. We use the evolving secret-sharing scheme of Xing and Yuan [43] for dynamic-threshold access structures. In this scheme, the parties are partitioned into generations. In each generation, with parties $\{p_i, p_{i+1}, \ldots, p_{i+g}\}$, two schemes are executed: (1) a secret-sharing realizing the finite restriction of the evolving $\mathrm{tr}(\cdot)$-dynamic-threshold access structure to the parties of the generation, and (2) Shamir's $\mathrm{tr}(g)$-out-of-$(g + \mathrm{tr}(g))$ threshold secret-sharing scheme. Each party in the generation gets a share of each scheme

and the last $\mathrm{tr}(g)$ shares of Shamir's scheme are recursively shared using the evolving scheme of Xing and Yuan among the next generations.

We improve the share size for large dynamic-threshold access structures, i.e., when $\mathrm{tr}(t) \geq t - t^\beta$ for some constant $0 < \beta < 1$, by constructing a better secret-sharing scheme for the finite $(t - t^\beta)$-dynamic-threshold access structure. Specifically, we consider the access structure whose parties are $\{p_1, \ldots, p_g\}$ and a set A is authorized if $|A \cap \{p_1, \ldots, p_t\}| \geq \mathrm{tr}(t) = t - t^\beta$ for some $1 \leq t \leq g$. This access structure can be realized by executing g copies of Shamir's secret-sharing scheme, i.e., for each $1 \leq t \leq g$ we execute Shamir's $\mathrm{tr}(t)$-out-of-t secret-sharing scheme. We prove that for large $\mathrm{tr}(\cdot)$ it suffices to execute only t^β copies of Shamir's scheme. Assume that $\mathrm{tr}(t) \geq \mathrm{tr}(t-1) + 1$ and consider an authorized set A whose maximum party is p_t; if $|A \cap \{p_1, \ldots, p_t\}| \geq \mathrm{tr}(t)$, then $|A \cap \{p_1, \ldots, p_{t-1}\}| \geq |A \cap \{p_1, \ldots, p_t\}| - 1 \geq \mathrm{tr}(t) - 1 \geq \mathrm{tr}(t-1)$. Thus, if $\mathrm{tr}(t) \geq \mathrm{tr}(t-1) + 1$ we do not need to execute the $\mathrm{tr}(t)$-out-of-t secret-sharing scheme. We show that this leaves us with at most t^β schemes.

Evolving Secret-Sharing Scheme for Evolving Slice Access Structures. We next explain the ideas of our construction of an evolving secret-sharing scheme for a 2-slice access structure, in which the authorized sets are some sets of size two and all sets of size at least 3. First, we handle authorized sets of size at least 3 using the scheme of Komargodski et al. [31]. For authorized sets of size exactly 2 we do the following. Partition the parties into generations. Let G_i denote the i^{th} generation, and let k be a large constant. We then use the secret-sharing scheme for finite slice functions [7,34] to share s among the parties of every k consecutive generations. Finally, we need to handle pairs of parties in the access structure that are not in k consecutive generations; here for every $j \in \mathbb{N}$ we give the j^{th} party, which is in some generation G_i, a random bit r_j. Then, for every t in generation at least $i + k$, if the j^{th} and t^{th} parties are in the access structure, we give $s \oplus r_j$ to the t^{th} party. The size of the share of the t^{th} party in some generation i is dominated by the share in the secret-sharing scheme for the finite slice functions and the number of bits $s \oplus r_j$ that it gets; the latter number is at most the number of parties in the first $i - k$ generations. By choosing the correct size of the generations (namely $2^{\log^c i}$ for some constant c), we get the desired share size. By considering arbitrarily large k, we show that the share size decreases.

The evolving secret-sharing scheme for a 3-slice access structure uses similar ideas; however, it is more complicated. The complicated case in constructing an evolving secret-sharing scheme for 3-slice access structures is in the case where there are two parties in some generation and one party from a future generation. To handle this case we use a CDS protocol for the finite index function. The details on this scheme are given in the full version [1].

Lower Bounds for Evolving Secret Sharing of Some Natural Access Structures. We prove lower bounds on the share size for explicit natural evolving access structures. Toward proving these results, we first show a general lower bound. This lower bound generalizes the recent result of [36] to include more access

structures, and is inspired by the generalization of Csirmaz's lower bound [19] due to Blundo et al. [14]. The idea is to define an infinite independent sequence: we partition the parties into two sets $A = \{p_{a_i}\}_{i\in\mathbb{N}}$ and $B = \{p_{b_i}\}_{i\in\mathbb{N}}$ and consider an infinite sequence of sets $A_1, A_2, \ldots$, each of them is a finite subset of A, and consider an evolving access structure whose minimal authorized sets are $\{A_i \cup \{b_i\}\}_{i\in\mathbb{N}}$ (the definition of an infinite independent sequence is more general; see the full version [1]). Using the lower bound of [14,19] for finite access structures, we deduce that for every i the total share of $P_i \triangleq \{p_{a_1}, \ldots, p_{a_i}\}$ is at least the number of sets in the sequence contained in P_i. As in [36], we schedule the parties in B to appear sparsely in $\{p_i\}_{i\in\mathbb{N}}$ and get a lower bound on the total share size of the first t parties in the evolving access structure.

We use the above general lower bound to get lower bounds for two interesting families of access structures. We first construct an infinite independent sequence for an infinite directed st-connectivity access structure. We consider a layered graph with 3 layers. Interestingly, we also obtain a lower bound of $\Omega(\sqrt{n})$ on the share size of finite (i.e., not evolving) directed st-connectivity, by taking finite prefixes of the infinite independent sequence. We also prove lower bounds for infinite k-hypergraph access structures for constant k; this is done by generalizing the finite independent sequence for finite k-hypergraph access structures given in [6]. For example, for $k = 3$, we consider the independent sequence that contains all subsets of A of size 2 (hence the set $A_i \cup \{b_i\}$ is of size 3 as required for 3-hypergraph access structures). The number of subsets of A of size 2 contained in $\{p_{a_1}, \ldots, p_{a_i}\}$ is $\Theta(i^2)$; we deduce that the total share size of the first t parties in any evolving secret-sharing scheme realizing this access structure is $\Omega(t^2)$. By a fairly simple construction of an evolving k-hypergraph secret-sharing scheme, our lower bound is tight for k-hypergraph access structures.

2 Preliminaries

In this section, we present formal definitions of secret-sharing schemes and evolving secret-sharing schemes.

Notations. For $n \in \mathbb{N}$ we use the notation $[n]$ to denote the set $\{1, 2, \ldots, n\}$. We denote by log the logarithmic function with base 2. When we refer to a set of parties $A = \{p_{i_1}, p_{i_2}, \ldots, p_{i_t}\}$, we assume that $i_1 < i_2 < \cdots < i_t$. We let $\mathrm{poly}(t)$ denote an unspecified polynomial.

2.1 Secret-Sharing Schemes

We start by defining (perfect) secret-sharing schemes for a finite set of parties.

Definition 2.1 (Access Structures)**.** *Let $P = \{p_1, \ldots, p_n\}$ be a set of parties. A collection $\Gamma \subseteq 2^{\{p_1,\ldots,p_n\}}$ is monotone if $B \in \Gamma$ and $B \subseteq C$ imply that $C \in \Gamma$. An access structure $\Gamma \subseteq 2^{\{p_1,\ldots,p_n\}}$ is a monotone collection of non-empty sets. Sets in Γ are called authorized, and sets not in Γ are called unauthorized. We will represent an n-party access structure by a function $f : \{0,1\}^n \to \{0,1\}$,*

where an input (i.e., a string) $\sigma = \sigma_1, \sigma_2, \ldots, \sigma_n \in \{0,1\}^n$ *represents the set* $A_\sigma = \{p_i : i \in [n], \sigma_i = 1\}$*, and* $f(\sigma) = 1$ *if and only if* $A \in \Gamma$*. We will also call* f *an access structure.*

A secret-sharing scheme defines a way to distribute shares to parties. Such a scheme is said to realize an access structure Γ if the shares held by any authorized set of parties (i.e., a set in the access structure) can be used to reconstruct the secret, and the shares held by any unauthorized set of parties reveal nothing about the secret. The formal definition is given as follows.

Definition 2.2 (Secret-Sharing Schemes)**.** *A secret-sharing scheme* Π *over a set of parties* $P = \{p_1, \ldots, p_n\}$ *with domain of secrets* S *and domain of random strings* R *is a mapping from* $S \times R$ *to a set of* n*-tuples* $S_1 \times S_2 \times \cdots \times S_n$ *(the set* S_j *is called the domain of shares of* p_j*). A dealer distributes a secret* $s \in S$ *according to* Π *by first sampling a random string* $r \in R$ *with uniform distribution, computing a vector of shares* $\Pi(s;r) = (\mathsf{sh}_1, \ldots, \mathsf{sh}_n)$*, and privately communicating each share* sh_j *to party* p_j*. For a set* $A \subseteq \{p_1, \ldots, p_n\}$*, we denote* $\Pi_A(s;r)$ *as the restriction of* $\Pi(s;r)$ *to its* A*-entries (i.e., the shares of the parties in* A*).*

A secret-sharing scheme Π *with domain of secrets* S *realizes an access structure* Γ *if the following two requirements hold:*

Correctness. *The secret* s *can be reconstructed by any authorized set of parties. That is, for any authorized set* $B = \{p_{i_1}, \ldots, p_{i_{|B|}}\} \in \Gamma$*, there exists a reconstruction function* $\mathrm{Recon}_B : S_{i_1} \times \cdots \times S_{i_{|B|}} \to S$ *such that for every secret* $s \in S$ *and every random string* $r \in R$*, it holds that* $\mathrm{Recon}_B\left(\Pi_B(s;r)\right) = s$*.*

Security. *Every unauthorized set cannot learn anything about the secret from its shares. Formally, for any set* $T \notin \Gamma$*, every two secrets* $s_1, s_2 \in S$*, and every possible vector of shares* $\langle \mathsf{sh}_j \rangle_{p_j \in T}$*,* $\Pr\left[\Pi_T(s_1;r) = \langle \mathsf{sh}_j \rangle_{p_j \in T}\right] = \Pr\left[\Pi_T(s_2;r) = \langle \mathsf{sh}_j \rangle_{p_j \in T}\right]$*, where the probability is over the choice of* r *from* R *with uniform distribution.*

The size of the share of party p_j *is defined as* $\log |S_j|$ *and the size of the shares of* Π *is defined as* $\max_{1 \le j \le n} \log |S_j|$*. The total share size of* Π *is defined as* $\sum_{j=1}^{n} \log |S_j|$*.*

We will use the following result on a construction of secret-sharing schemes from monotone formulas in our constructions.

Theorem 2.3 (Secret Sharing from Monotone Formulas [12]). *Let* $f : \{0,1\}^n \to \{0,1\}$ *be a monotone function (i.e., an access structure). If there is a monotone formula with* m *leaves that computes* f*, then* f *can be realized by a secret-sharing scheme in which the total share size is* m*.*

Below we list the access structures that are of interest in this paper.

Threshold Access Structures. The most basic and well-known access structure is the threshold access structure:

Definition 2.4 (Threshold Access Structures)**.** *Let $1 \leq k \leq n$. A* k-out-of-n threshold access structure Γ *over a set of parties $P = \{p_1, \ldots, p_n\}$ is the access structure containing all subsets of size at least k, that is, $\Gamma = \{A \subseteq P : |A| \geq k\}$.*

The well-known scheme of Shamir for the k-out-of-n threshold access structure (based on polynomial interpolation) is an efficient threshold secret-sharing scheme; the properties of Shamir's scheme over $\mathbb{F}_{2^m}$ for an appropriate $m \in \mathbb{N}$ are summarized in the next theorem.

Theorem 2.5 (Shamir [41]). *For every $n \in \mathbb{N}$, and $k \in [n]$, there is a secret-sharing scheme for secrets of size ℓ (i.e., the domain of secrets is $S = \{0,1\}^\ell$) realizing the k-out-of-n threshold access structure, in which the share size is $\max\{\ell, \lceil \log(n+1) \rceil\}$. Moreover, the shares of the scheme are elements of the field $\mathbb{F}_{2^{\ell+\log n}}$.*

st-connectivity Access Structures. The second access structure we consider is the st-connectivity access structure. It is defined as follows.

Definition 2.6 (The Undirected/Directed st-connectivity Access Structures)**.** *Let $G = (V, E)$ be the complete undirected (respectively directed) graph such that $u_s, u_t \in V$. We define the* st-connectivity access structure *as follows: The parties correspond to edges of G. A set of parties is authorized if and only if it contains an undirected (respectively a directed) path from u_s to u_t.*

Benaloh and Rudich [11] constructed a secret-sharing scheme for undirected st-connectivity in which the secret and each share is a bit. The best known secret-sharing scheme for directed st-connectivity is the formula-based scheme of [12] and has share size $n^{O(\log n)}$ (we describe the monotone formula for st-connectivity in Claim 3.16).

Access Structures Defined by Hypergraphs. A *hypergraph* is a pair $H = (V, E)$ where V is a set of vertices and $E \subseteq 2^V \setminus \{\emptyset\}$ is the set of hyperedges. A *hypergraph* where all hyperedges are of size exactly k is called a k-hypergraph. We say H is finite if V is finite. A k-partite hypergraph is a k-hypergraph $H = (V, E)$, for which there is a partition of V to k disjoint sets $D_1, \ldots, D_k$ such that every hyperedge $e \in E$ satisfies $|e \cap D_i| = 1$ for every $1 \leq i \leq k$ (i.e., each hyperedge contains exactly one vertex from each part). An access structure Γ is a k-hypergraph access structure (also called k-homogeneous access structure) if all minimal authorized sets are of size exactly k (described by a hypergraph). Formally, it is defined as follows.

Definition 2.7 (k-Hypergraph Access Structures)**.** *An access structure Γ is a* k-hypergraph access structure *if there exists a finite k-hypergraph $H = (V, E)$ such that the parties of Γ are the vertices V and a set of parties is authorized if and only if it contains a hyperedge (in other words, the minimal authorized sets of Γ are the hyperedges). An access structure Γ is a k-partite access structure if its k-hypergraph is k-partite.*

Every k-hypergraph access structure with n parties has a monotone formula of size $O(n^k/\log n)$ (by using a result of [25]), thus it can be realized by the secret-sharing scheme of [12] with total share size $O(n^k/\log n)$.

k-Slice Access Structures. An access structure Γ is a k-slice access structure if all sets of size at most $k-1$ are unauthorized, all sets of size at least $k+1$ are authorized, and sets of size k can be either authorized or unauthorized; we describe the authorized sets of size k by a k-hypergraph.

Definition 2.8 (k-Slice Access Structures)**.** *An access structure Γ is a* k-slice access structure *if there exists a finite k-hypergraph $H = (V, E)$ such that the parties of Γ are the vertices V and a set of parties is authorized if and only if it contains at least $k+1$ parties or the set contains a hyperedge (in other words, the minimal authorized sets of Γ are the hyperedges and all sets of size $k+1$ that do not contain a hyperedge).*

We will use the following constructions for finite slice access structures as a building block in our evolving secret-sharing schemes for infinite slice access structures. They are implied by the CDS protocols of [34,35] and the transformation of [2,7].

Theorem 2.9. *Let Γ be a (finite) 2-slice access structure with n parties. Then there is a secret-sharing scheme realizing Γ, in which the share size of every party is at most $2^{O(\sqrt{\log n \cdot \log\log n})}$.*

Theorem 2.10. *Let $k \geq 2$ and let Γ be a (finite) k-slice access structure with n parties. Then there is a secret-sharing scheme realizing Γ, in which the share size of every party is at most $2^{O(\sqrt{\log n} \cdot \log\log n)}$.*

2.2 Evolving Secret-Sharing Schemes

In an evolving secret-sharing scheme, defined by [31], the number of parties is unbounded. Parties arrive one after the other; when a party p_t arrives the dealer gives it a share. The dealer cannot update the share later and does not know how many parties will arrive after party p_t. Thus, we measure the share size of p_t as a function of t. We start by defining an evolving access structure, which specifies the authorized sets. The number of parties in an evolving access structure is infinite, however every authorized set is finite.

Definition 2.11 (Evolving Access Structures)**.** *Let $P = \{p_i\}_{i\in\mathbb{N}}$ be an infinite set of parties. A collection of finite sets $\Gamma \subseteq 2^P$ is an* evolving access structure *if for every $t \in \mathbb{N}$ the collections $\Gamma^t \triangleq \Gamma \cap 2^{\{p_1,\ldots,p_t\}}$ is an access structure as defined in Definition 2.1. We will represent an access structure by a function $f : \{0,1\}^* \to \{0,1\}$, where an input (i.e., a string) $\sigma = \sigma_1, \sigma_2, \ldots, \sigma_n \in \{0,1\}^n$ represents the set $A_\sigma = \{p_i : i \in [n], \sigma_i = 1\}$,[3] and $f(\sigma) = 1$ if and only if $A_\sigma \in \Gamma$. We will also call f an evolving access structure.*

[3] In particular, the same set has infinitely many representations by inputs of various lengths, using sufficiently many trailing zeros.

Definition 2.12 (Evolving Secret-Sharing Schemes)**.** *Let S be a domain of secrets, where $|S| \geq 2$, and $\{R_t\}_{t\in\mathbb{N}}, \{S_t\}_{t\in\mathbb{N}}$ be two sequences of finite sets. An* evolving secret-sharing scheme *with domain of secrets S is a sequence of mappings $\Pi = \{\Pi^t\}_{t\in\mathbb{N}}$, where for every $t \in \mathbb{N}$, Π^t is a mapping $\Pi^t : S \times R_1 \times \cdots \times R_t \to S_t$ (this mapping returns the share sh_t of p_t).*

An evolving secret-sharing scheme $\Pi = \{\Pi^t\}_{t\in\mathbb{N}}$ realizes an evolving access structure Γ if for every $t \in \mathbb{N}$ the secret-sharing scheme $\Pi_t(s;(r_1,\ldots,r_t)) \triangleq \langle \Pi^1(s;r_1),\ldots,\Pi^t(s;r_1,\ldots,r_t)\rangle$ (i.e., the shares of the first t parties) is a secret-sharing scheme realizing Γ^t according to Definition 2.2.

By default, the domain of secrets of an evolving secret-sharing scheme is $\{0,1\}$. The following result shows that every evolving access structure can be realized by an evolving secret-sharing scheme (with exponentially long secrets).

We next define the evolving access structures that we will consider in this paper; they generalize the finite access structures defined in Sect. 2.1.

Definition 2.13 (Evolving Threshold Access Structures)**.** *Let $k \in \mathbb{N}$. The* evolving k-threshold access structure *is the evolving access structure Γ, where Γ^t is the k-out-of-t threshold access structure.*

Komargodski et al. [31] showed that any evolving threshold access structure can be realized by an efficient evolving secret-sharing scheme.

Theorem 2.14 ([31])**.** *For every $k \in \mathbb{N}$, there is a secret-sharing scheme realizing the evolving k-threshold access structure such that the share size of party p_t is $(k-1)\cdot\log t + \mathrm{poly}(k)\cdot o(\log t)$.*

A natural generalization of an evolving threshold access structure is to allow the threshold to depend on the index of the arriving party.

Definition 2.15 (Dynamic-Threshold Access Structures)**.** *Let $\mathrm{tr} : \mathbb{N} \to \mathbb{N}$ be a non-decreasing function. A $\mathrm{tr}(t)$-*dynamic-threshold access structure *is defined as follows: A finite set of parties A is authorized if and only if there exists t such that $|A \cap \{p_1,\ldots,p_t\}| \geq \mathrm{tr}(t)$. Stated differently, a set A is unauthorized if and only if for every t, it holds that $|A \cap \{p_1,\ldots,p_t\}| < \mathrm{tr}(t)$.*

Komargodski and Paskin-Cherniavsky [32] showed that any dynamic-threshold access can be realized with an evolving secret-sharing scheme with a polynomial share size.

Theorem 2.16 ([32])**.** *For any dynamic-threshold access structure, there exists an evolving secret-sharing scheme in which the share size of party p_t is $O(t^4 \cdot \log t)$.*

Definition 2.17 (Evolving Undirected/Directed st-connectivity Access Structures)**.** *An evolving undirected (resp. directed) st-connectivity access structure is defined as follows. The parties in the access structure are the edges of an undirected (resp. directed) graph $G = (V,E)$, where V is countably infinite, with some order on the edges that specifies the order that the parties arrive. There are two*

fixed vertices in the graph $u_s, u_t \in V$, where u_s is called the source vertex and u_t the target vertex. A finite set of parties (i.e., edges) is authorized if and only if they contain an undirected (resp. a directed) path from u_s to u_t.

Komargodski et al. [31] showed that every undirected st-connectivity access structure can be realized by an evolving secret-sharing scheme in which the share of each party is a bit.

Evolving k-Hypergraph access structures and *evolving k-slice access structures* are defined as their finite counterparts, except that the k-hypergraph H is countably infinite. In these access structures, we assume that there is some order on the vertices of the hypergraph that specifies the order that the parties (i.e., vertices) arrive.

3 Evolving Secret-Sharing Schemes for Infinite Branching Programs

Infinite decision trees were used in [31,32] to construct evolving secret-sharing schemes. In this section, we define infinite non-deterministic branching programs (see Sect. 3.1.1), show how to transform them to generalized infinite decision trees (see Sect. 3.1.3), and show how to construct secret-sharing schemes for generalized infinite decision trees (see Sect. 3.1.2). This setting will be used in our constructions of schemes for various functions, i.e., for evolving access structures that have infinite branching programs with bounded-width (see Sect. 3.3).

3.1 Constructing an Evolving Secret-Sharing Schemes for Infinite Branching Programs

3.1.1 Infinite Branching Programs and Generalized Infinite Decision Trees

An infinite monotone non-deterministic branching program (IBP) is a generalization of finite monotone non-deterministic branching programs except that the number of edges and variables is infinite. Such a branching program computes a monotone function $f : \{0,1\}^* \to \{0,1\}$ on all finite inputs; i.e., defines an evolving access structure (see Definition 2.11).

Recall that a finite monotone non-deterministic branching program is a directed acyclic graph, where each edge is labeled by a variable from $x_1, \ldots, x_t$. For every input $\sigma \in \{0,1\}^t$ (interpreted as an assignment to the variables $x_1, \ldots, x_t$) it holds that $f(\sigma) = 1$ if and only if *there exists* a satisfied path in G from the source vertex to the target vertex, that is, each edge in the path is labeled with a variable x_i that is assigned 1, i.e., $\sigma_i = 1$. Below we generalize this notion to infinite branching programs. In this definition we will allow many target vertices and we will allow the edges to be labeled by 1, meaning that every assignment satisfies them.

Definition 3.1 (Infinite Monotone Non-Deterministic Branching Programs – IBP)**.** *An infinite monotone non-deterministic branching program is a triple $B =$*

$(G = (V, E), u_0, \mu)$, where V is a countable set of vertices, G is an infinite directed acyclic graph, u_0 is a source vertex, and $\mu : E \to \{x_i : i \in \mathbb{N}\} \cup \{1\}$ is a labeling of the edges by variables or by 1 (we will sometimes use the notation μ_e instead of $\mu(e)$). We denote by U_{leaf} the set of vertices in V with out-degree 0, i.e., the leaves.

For a path P in the branching program and a finite input (an assignment for the variables on the path) $\sigma \in \{0,1\}^t$ for some $t \in \mathbb{N}$, we say that σ satisfies P and denote $\text{sat}_P(\sigma) = 1$ if σ satisfies all variables on the path, i.e., for each edge e on the path either $\mu_e = 1$ or $\mu_e = x_i$ for some $1 \leq i \leq t$ such that $\sigma_i = 1$. The branching program accepts an input σ if there exists a directed path P starting in the source vertex u_0 and leading to some leaf $u \in U_{\text{leaf}}$ such that $\text{sat}_P(\sigma) = 1$. The function $f : \{0,1\}^* \to \{0,1\}$ computed by B is the function f such that $f(\sigma) = 1$ if and only if B accepts σ.

To clarify what it means for an assignment σ to satisfy a path P, consider as an example $P = (e_1, e_2, e_3, e_4)$, with respective labels $(x_7, x_3, 1, x_{25})$. Then, the assignment $\alpha \in \{0,1\}^{30}$ with $\alpha_i = 1$ if and only if i is odd satisfies P, an assignment $\beta \in \{0,1\}^{25}$ with $\beta_3 = 0$ does not satisfy P, and the assignment $\gamma = 1^{20}$ also does not satisfy P, since it is too short.

We will construct evolving secret-sharing schemes for IBPs that are layered as defined below. Intuitively, a layered IBP is a restriction of IBP, defined over layered (infinite, directed, acyclic) graphs (i.e., with edges going only from any one layer to its consecutive layer) with the additional requirement that the label of any edge from layer $i-1$ to layer i is either x_i or 1.

Definition 3.2 (Layered IBP)**.** *An infinite monotone non-deterministic branching program (abbreviated LIBP) is* layered *if the vertices of G can be partitioned into finite sets $(L_i)_{i \in \mathbb{N} \cup \{0\}}$, such that $L_0 = \{u_0\}$, there are edges only from layer $i-1$ to layer i for some $i \in \mathbb{N}$, and all edges entering layer i are labeled either by x_i or by 1. For a vertex $u \in V$, we denote $L(u)$ as the layer of u, i.e., the index i such that $u \in L_i$. The width of the branching program at layer i, denoted $w(i)$, is the number of vertices in layer L_i. For a LIBP $B = (G, u_0, \mu)$ and vertices u, v, we define the predicate $\text{reach}_{u,v}$ as $\text{reach}_{u,v} = 0$ if there is no path in G from u to v, and otherwise*

$$\text{reach}_{u,v}(x_{L(u)+1}, \ldots, x_{L(v)}) = \bigvee_{\substack{P \text{ is a path in } G \\ \text{from } u \text{ to } v}} \text{sat}_P(x_{L(u)+1}, \ldots, x_{L(v)}),$$

i.e., an input satisfies $\text{reach}_{u,v}$ if and only if it satisfies at least one path from u to v. We stress that, unlike Definition 3.1, here we consider an assignment to the predicate sat_P that only contains the variables that can appear on the path, i.e., we consider assignments $\sigma_{L(u)+1}, \ldots, \sigma_{L(v)}$ to the variables $x_{L(u)+1}, \ldots, x_{L(v)}$.

Our goal is to construct an evolving secret-sharing scheme for the access structure described by a LIBP B; we call such a scheme an evolving secret-sharing realizing B.

Example 3.3. We next describe an LIBP $B = (G = (V, E), u_0^0, \mu)$ for the 3-threshold function access structure (as defined in Definition 2.13). The layers of the graph are $L_t = \{u_0^t, \ldots, u_{\min\{3,t\}}^t\}$ for $t \in \mathbb{N} \cup \{0\}$ and the vertices of the graph are $V = \cup_{t \in \mathbb{N} \cup \{0\}} L_t$. For every $t \in \mathbb{N}$ and $0 \leq i \leq 2$ there are two edges: an edge (u_i^{t-1}, u_i^t) labeled by 1 and an edge (u_i^{t-1}, u_{i+1}^t) labeled by x_t. Notice that the leaves are $U_{\text{leaf}} = \{u_3^t : t \in \mathbb{N}, t \geq 3\}$. An illustration of this construction appears in Fig. 1.

Informally, reaching the vertex u_i^t in the LIBP means that an input σ of length t contains at least i ones. Indeed, let $\sigma \in \{0,1\}^t$ be an input that contains $i < 3$ ones. Then, any path in G that is satisfied by σ contains at most i edges that are not 1-labeled, and hence ends in a node of the form $u_j^{t'}$ for $j < 3$, i.e., not in any leaf. Conversely, let $\sigma \in \{0,1\}^t$ be an input that contains $i \geq 3$ ones, and let $t_1 < t_2 < t_3$ be the first three coordinates in σ that are 1. It is possible to define a path from u_0^0 to the leaf $u_3^{t_3}$ that is satisfied by σ, as follows. At each step the 1-labeled edge is taken, except for steps t_1, t_2, and t_3, at which the edges $(u_0^{t_1-1}, u_1^{t_1})$, $(u_1^{t_2-1}, u_2^{t_2})$, and $(u_2^{t_3-1}, u_3^{t_3})$ are taken, respectively.

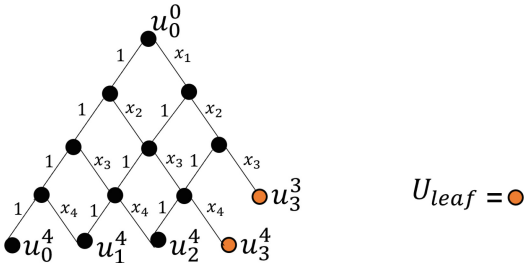


Fig. 1. The first five layers of the LIBP for the 3-threshold function. For example, u_3^4 is a leaf and the input $\sigma = 1, 1, 0, 1$ satisfies the path $u_0^0, u_1^1, u_2^2, u_2^3, u_3^4$.

In our constructions, we use *generalized* infinite decision trees. On one hand, we limit the graph to be an infinite tree. On the other hand, each edge, instead of being labeled by a variable x_i, is labeled by a predicate of some variables $x_i, \ldots, x_j$. Being a bit more specific, we divide the variables into generations $\{G_i\}_{i \in \mathbb{N}}$, and let an edge of distance i from the root be labeled with a predicate on the variables in generation G_i.

Definition 3.4 (Generalized Infinite Decision Trees – GIDT)**.** *A generalized infinite decision tree is a quadruple $T = (G = (V, E), u_0, \mu, h)$, where*

- *V is a countable set of vertices,*
- *$G = (V, E)$ is an infinite directed tree with root vertex u_0 such that the out-degree of each vertex is finite. We denote the i^{th} level L_i as $\{u \in V :$ u is at distance i from $u_0\}$, and refer to L_i as the i^{th}* layer.

- $h : \mathbb{N} \to \mathbb{N}$ *is an increasing function that partitions the variables into generations, where for* $i \in \mathbb{N}$*, generation* i *is the variables* $G_i \triangleq \{x_{h(i-1)+1}, \ldots, x_{h(i)}\}$ *(where we define* $h(0) = 0$*),*
- μ *is a labeling of the edges by* predicates*, where for every edge* e *from level* L_{i-1} *to level* L_i*, the labeling* μ_e *is some monotone predicate on the variables of generation* i*, of the form* $\varphi(x_{h(i-1)+1}, \ldots, x_{h(i)}) : \{0,1\}^{h(i)-h(i-1)} \to \{0,1\}$*.*

For a path P *in the tree ending at a vertex in layer* i*, we say that* P *is satisfied by an input* $\sigma \in \{0,1\}^t$*, denoted by* $\mathrm{sat}_P(\sigma) = 1$*, if* $h(i) \leq t$ *(that is, the variables in all predicates labeling edges in* P *are from* $x_1, \ldots, x_t$*) and for each edge* e *on the path the predicate* μ_e *is satisfied by* σ*. The GIDT* T *accepts an input* σ *if there is at least one directed path* P *starting in the source vertex* u_0 *and leading to a leaf such that* $\mathrm{sat}_P(\sigma) = 1$*. The function* $f : \{0,1\}^* \to \{0,1\}$ *computed by* T *is the function* f *such that* $f(\sigma) = 1$ *if and only if* T *accepts* σ*.*

Example 3.5. We show an example of a GIDT $T = (G = (V, E), u_0, \mu, h)$ for a dynamic 3-threshold function (i.e., defined by a function $\mathrm{tr}(t) = 3$). Let $h : \mathbb{N} \to \mathbb{N}$ be any increasing function. The layers of G are $L_i = \{u_{b_0,b_1,\ldots,b_i} : 0 = b_0 \leq b_1 \leq \cdots \leq b_{i-1} \leq b_i \leq 3, b_{i-1} \leq 2\}$. The vertices of G are $V = \cup_{i \in \mathbb{N} \cup \{0\}} L_i$. There is an edges $(u_{b_0,\ldots,b_{i-1}}, u_{b_0,\ldots,b_{i-1},b_i})$ in G for every $i \in \mathbb{N}$ and every sequence $b_0, b_1, \ldots, b_i$, where $0 = b_0 \leq b_1 \leq \cdots \leq b_{i-1} \leq b_i \leq 3, b_{i-1} \leq 2$, and this edge is labeled by $\mathrm{Thr}_{b_i - b_{i-1}}(x_{h(i-1)+1}, \ldots, x_{h(i)})$. Thr_b is the b-threshold predicate, i.e., $\mathrm{Thr}_b(y_1, \ldots, y_m) = 1$ iff $\sum_{j=1}^{m} y_j \geq b$. For example, if $b_i = b_{i-1}$, then the label of the edge is 1. The leaves of the GIDT T are $U_{\mathrm{leaf}} = \{u_{b_0,b_1,\ldots,b_i} : i \in \mathbb{N}, 0 = b_0 \leq b_1 \leq \cdots \leq b_{i-1} < b_i = 3\}$. The GIDT is described in Fig. 2.

Informally, the GIDT counts the number of ones in an input σ in each generation. More formally, there is a path from the root u_0 to a vertex $u_{b_0,b_1,\ldots,b_i}$ (for $i \in \mathbb{N}$) that is satisfied by an input $\sigma \in \{0,1\}^t$ if and only if σ contains at least $b_j - b_{j-1}$ ones from generation j for each $1 \leq j \leq i$. In particular, T accepts an input σ if and only if there is a path from u_0 to a leaf $u_{b_0,b_1,\ldots,b_i}$, where $b_i = 3$, that is satisfied by σ if and only if σ contains at least 3 ones.

3.1.2 An Evolving Secret-Sharing Scheme for GIDTs

We next present an evolving secret-sharing scheme for GIDTs. As a first step, we present an evolving secret-sharing scheme for simple infinite decision trees, defined next.

Definition 3.6 (Infinite decision trees – IDT)**.** *An infinite decision tree* $T = (G = (V, E), u_0 = 0, \mu)$ *is a special case of GIDT, where each edge* (u, v) *is either labeled by the constant 1 or by a variable* x_v*, where for simplicity we assume that* $V = \mathbb{N} \cup \{0\}$ *(i.e., a vertex is a non-negative integer). As* G *is a tree, each variable labels at most one edge. Furthermore, we assume that the vertices are ordered by the layers, i.e.,* $L_0 = \{0\}$*,* $L_1 = \{1, \ldots, w(1)\}$*, and so on (where* $w(i)$ *is the width of layer* L_i*). The variables in generation* i *are* $\{x_j : j \in L_i\}$ *(thus, we do not need to specify* h *for an IDT).*

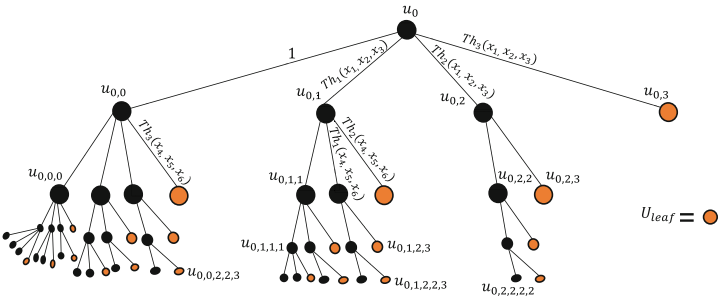


Fig. 2. The first five layers of the GIDT for 3-threshold function where $h(0) = 0$ and $h(t) = 3t$.

We note that IDT is not a special case of LIBP, since in IDT different edges of the same layer may be labeled by different variables, where in LIBP labels from layer i are labeled either by 1 or by x_{i+1}. We next recall the evolving secret-sharing scheme for infinite decision trees from [31,32]; the scheme we present also deals with edges labeled by 1. We will use this scheme to construct an evolving secret-sharing scheme for GIDTs and LIBPs. For technical reasons we assume that all edges entering leaves are labeled by a variable.[4]

Construction 3.7 (An Evolving Secret-Sharing Scheme Π_{IDT} for an IDT $T = (G, u_0, \mu)$).
Input: $s \in \{0,1\}$.
The sharing algorithm:

- For $i = 1$ to ∞:
  - For every vertex $u \in L_{i-1}$ and $v \in L_i$, when party p_v arrives choose a bit r_v as follows:
    - If v is a leaf, then let $u_0, v_1, \ldots, v_{t-1}, v$ be the path from the root u_0 to v in G and assign $r_v \leftarrow s \oplus \bigoplus_{j=1}^{t-1} r_{v_j}$.
    - If v is not a leaf and $\mu_{(u,v)} = x_v$, then r_v is a uniformly distributed random bit.
    - If v is not a leaf and $\mu_{(u,v)} = 1$, then $r_v \leftarrow 0$.
  - The share of p_v is $\mathsf{sh}_v = r_v$.

Claim 3.8. *The evolving secret-sharing scheme Π_{IDT} realizes the infinite decision tree $T = (G, u_0, \mu)$, where the share of p_t is a bit.*

Proof. First we prove the correctness of the scheme. Let $\sigma = \sigma_1, \ldots, \sigma_t$ be an input accepted by T, where $\sigma_1, \ldots, \sigma_{t-1}$ is not accepted by T (in particular, $\sigma_t = 1$), and let $A = \{p_i : \sigma_i = 1\}$ be the corresponding set of parties. There

[4] As the function computed by an IDT describes an access structure, we assume that the empty set is rejected by the IDT. For every vertex u in the tree whose in-coming edge is labeled by a variable, if there exists a path from v to a leaf such that all labels on the path are 1, we remove the subtree of v (i.e., v becomes a leaf).

is a path $u_0, v_1, \ldots, v_{t-1}, t$ in G from u_0 to the leaf t such that σ satisfies all labels on this path. By our assumption, the edge (v_{t-1}, t) is labeled by x_t. As $\sigma_t = 1$, the party p_t is in A and the parties in A have the bit $r_t = s \oplus \bigoplus_{j=1}^{t-1} r_{v_j}$. Furthermore, for every j either $\mu_{(v_{j-1},v_j)} = 1$ and $r_{v_j} = 1$ or $\mu_{(v_{j-1},v_j)} = x_{v_j}$, thus, $\sigma_j = 1$ and the parties in A hold r_{v_j}. We conclude that the parties in A can reconstruct s.

Next we prove the security of the scheme. It would be convenient to view the scheme Π_{IDT} as a recursive procedure. To share a secret s in Π_{IDT} for the subtree of T rooted at a vertex u, the dealer independently executes a secret-sharing scheme for each vertex v such that $(u, v) \in E$:

- If $\mu_{(u,v)} = 1$, the dealer shares s recursively for the subtree of G rooted at v.
- If $\mu_{(u,v)} = x_v$, the dealer chooses a random bit r_v, gives r_v to p_v, and shares $s \oplus r_v$ recursively for the subtree of G rooted at v.

(If u is a leaf then there are no recursive calls.) Let $\sigma = \sigma_{u+1}, \ldots, \sigma_t$ be an input not accepted by the subtree of T rooted at u, and let $A = \{p_i : \sigma_i = 1\}$ be the corresponding set of parties. We prove the security, that is, that the parties in A learn no information on the secret, by induction on $|\sigma|$. For the basis case when $|\sigma| = 1$, the vertex t is not a leaf and the share of p_t is either 1 or a random bit. For the induction step, as the infinite decision tree rooted at u rejects the input $\sigma = \sigma_u, \ldots, \sigma_t$, there is no path from u to a leaf that is satisfied by σ. We will show that the set A does not learn information from the secret-sharing scheme for each vertex v such that $(u, v) \in E$. Since each secret-sharing scheme is independently executed, this will imply the security. Fix such a vertex v. If $\mu_{(u,v)} = 1$, there is no path from v to a leaf that is satisfied by $\sigma_{v+1}, \ldots, \sigma_t$; by induction, the set $\{p_i : v + 1 \leq i \leq t, \sigma_i = 1\}$ learns no information on s from this execution (and the shares of $A \setminus \{p_i : v + 1 \leq i \leq t, \sigma_i = 1\}$ are independent of the shares of the recursive call to v). If $\mu_{(u,v)} = x_v$, there are two cases:

- If $\sigma_v = 1$, then there is no path from v to a leaf that is satisfied by $\sigma_{v+1}, \ldots, \sigma_t$; by induction, the set $A \setminus \{p_v\}$ learns no information on $s \oplus r_v$ from this execution, hence learns no information on s.
- If $\sigma_v = 0$, then $p_v \notin A$, and the set A learns no information on r_v, thus although it might learn $s \oplus r_v$, it learns no information on s.

□

We next show how to realize the access structure of a GIDT using the secret-sharing scheme Π_{IDT} realizing a related infinite decision tree (where edges are labeled by variables or by the constant 1). In a GIDT each edge e is labeled by a predicate a_e; in the following scheme Π_{GIDT} we consider this predicate as describing an access structure over the parties of the generation.

Construction 3.9 (An Evolving Secret-Sharing Scheme Π_{GIDT} for a GIDT $T = (G = (V, E), u_0, \mu, h)$).
Input: $s \in \{0, 1\}$.

- Construct from the GIDT $T = (G = (V, E), u_0, \mu, h)$ an IDT $T' = (G = (V, E), u_0, \mu')$ whose variables are $\{y_i : i \in \mathbb{N}\}$, where for every edge $(u, v) \in E$ if the predicate $\mu_{(u,v)}$ is the constant predicate 1, then $\mu'(u, v) = 1$; otherwise $\mu'(u, v) = y_v$.
- Execute the scheme Π_{IDT} for T' and use its shares as follows:
  ($*$ Recall that in Π_{IDT} the parties arrive according to layers, where inside a layer the order is some arbitrary fixed order $*$)
- For $i = 1$ to ∞ do:
  - When party $p_{h(i-1)+1}$ arrives do:
    - For every $(u, v) \in E$, where $u \in L_{i-1}$, $v \in L_i$, and $\mu_{(u,v)} \neq 1$, generate the share r_v of y_v in the scheme Π_{IDT} and share r_v using a secret-sharing scheme realizing the access structure defined by $\mu_{(u,v)}$ among the parties $p_{h(i-1)+1}, \ldots, p_{h(i)}$.
    - Let sh_t, for $h(i-1)+1 \leq t \leq h(i)$, be the concatenation of the shares of p_t in all these schemes.
    - Give party $p_{h(i-1)+1}$ the share $\mathsf{sh}_{h(i-1)+1}$.
  - For $t = h(i-1)+2$ to $h(i)$ do:
    - When party p_t arrives give it the share sh_t.

Claim 3.10. *The evolving secret-sharing scheme Π_{GIDT} realizes the GIDT $T = (G, u_0, \mu, h)$. For a party p_t in generation i (that is, $h(i-1)+1 \leq t \leq h(i)$), the size of the share of p_t is the sum of the sizes of the shares of p_t in the secret-sharing schemes for $\mu_{(u,v)}$ for every $(u, v) \in E$ such that $u \in L_{i-1}$, $v \in L_i$, and $\mu_{(u,v)} \neq 1$ (there are at most $w(i)$ such schemes).*

Proof. First we prove the correctness of the scheme. Let σ be an input accepted by T and $A = \{p_i : \sigma_i = 1\}$. Thus, there exists an accepting path P from u_0 to a leaf such that $\text{sat}_P(\sigma) = 1$. For every edge $e = (u, v) \in P$, the input σ satisfies μ_e. If $\mu_e \neq 1$, the parties in A can reconstruct r_v using the shares of the secret-sharing scheme realizing μ_e. If $\mu_e = 1$, according Construction 3.7, $r_v = 0$. By the correctness of Construction 3.7, the parties in A can reconstruct s by computing the exclusive-or of the bits of the vertices on P.

Next we prove the security of the scheme. Let σ be an input rejected by T and $A = \{p_i : \sigma_i = 1\}$. The parties in A can reconstruct the shares r_v in Π_{IDT} for edges (u, v) such that $\mu_{(u,v)}(\sigma) = 1$ and do not get any information on shares r_v for edge (u, v) such that $\mu_{(u,v)}(\sigma) = 0$. Since σ is rejected by T, there does not exists an accepting a path P from u_0 to a leaf such that $\text{sat}_P(\sigma) = 1$. That is, the parties in A hold shares in Π_{IDT} of an unauthorized set in T'. By the security of Π_{IDT}, the shares r_v that the parties in T can reconstruct are equally distributed when $s = 0$ and when $s = 1$.

For the share size, party p_t obtains a share in the secret-sharing scheme realizing $\mu_{(u,v)}$ for each edge (u, v) where $u \in L_{i-1}$ and $v \in L_i$. □

3.1.3 A Transformation from LIBPs to GIDTs

We next describe a transformation from an LIBP B to a GIDT T computing the same function. We start with an informal description of the transformation.

To transform an LIBP B to an IDT T (where each edge is labeled by a variable or the constant 1), we duplicate vertices and have in T a vertex $u_{0,j_1,\ldots,j_{i-1},j_i}$ for every path $u_0, u_{j_1}, \ldots, u_{j_{i-1}}, u_{j_i}$ in B starting from the root, and add an edge $(u_{0,j_1,\ldots,j_{i-1}}, u_{0,j_1,\ldots,j_{i-1},j_i})$ whose label is the label of the edge $(u_{j_{i-1}}, u_{j_i})$. The problem with this construction is that the resulting IDT is to big. To construct more efficient GIDT (which will result in more efficient evolving secret-sharing schemes), we partition the variables into generations (described by a function $h : \mathbb{N} \to \mathbb{N}$), the vertices in layer i of T are $u_{0,j_1,j_2,\ldots,j_i}$ for vertices $u_0, u_{j_1}, u_{j_2}, \ldots, u_{j_i}$ in the layers $0, h(1), h(2), \ldots, h(i)$ in B respectively. That is, the number of vertices in the resulting GIDT is much smaller. Now an edge $(u_{0,j_1,\ldots,j_{i-1}}, u_{0,j_1,\ldots,j_{i-1},j_i})$ represents *all paths* in B from $u_{j_{i-1}}$ to u_{j_i}, i.e., the predicate of this edge is satisfied by an input σ if and only if σ satisfies some path in B from $u_{j_{i-1}}$ to u_{j_i}. The formal construction is described below.

Construction 3.11 (A Transformation from a LIBP to a GIDT).
Input: A LIBP $B = (G = (V, E), u_0, \mu)$ and an increasing function $h : \mathbb{N} \to \mathbb{N}$. ($*$ We use the following notation for the vertices of the LIBP B – the vertices in the i-layer of B are $L_i = \{u^i_1, \ldots, u^i_{w(i)}\}$ for $i \in \mathbb{N} \cup \{0\}$. $*$)
Output: A GIDT $T = (G' = (V', E'), u'_0, \mu', h)$.
The transformation:

- The vertices in layer i of the tree G' are $L'_0 = \{u_0\}$ and for $i \in \mathbb{N}$ define

$$L'_i = \{u_{0,j_1,\ldots,j_i} : 1 \le j_1 \le w(h(1)), j_1 \notin U_{\text{leaf}}, \ldots, 1 \le j_i \le w(h(i)), j_i \notin U_{\text{leaf}}\} \bigcup \{v_{0,j_1,\ldots,j_{i-1}} : 1 \le j_1 \le w(h(1)), j_1 \notin U_{\text{leaf}}, \ldots, 1 \le j_{i-1} \le w(h(i-1)), j_{i-1} \notin U_{\text{leaf}}\}.$$

  The vertices of G' are $V' = \cup_{i \in \mathbb{N} \cup \{0\}} L'_i$. The leaves are $U'_{\text{leaf}} = \cup_{i \in \mathbb{N}} \{v_{0,j_1,\ldots,j_{i-1}} : 1 \le j_1 \le w(h(1)), j_1 \notin U_{\text{leaf}}, \ldots, 1 \le j_{i-1} \le w(h(i-1)), j_{i-1} \notin U_{\text{leaf}}\}$.
- The edges are

$$E' = \left\{(u_{0,j_1,\ldots,j_{i-1}}, u_{0,j_1,\ldots,j_{i-1},j_i}) : i \in \mathbb{N}, u_{0,j_1,\ldots,j_{i-1},j_i} \in V'\right\} \bigcup \left\{(u_{0,j_1,\ldots,j_{i-1}}, v_{0,j_1,\ldots,j_{i-1}}) : i \in \mathbb{N}, u_{0,j_1,\ldots,j_{i-1}} \in V'\right\}.$$

- For every $e = \left(u_{0,j_1,\ldots,j_{i-1}}, u_{0,j_1,\ldots,j_{i-1},j_i}\right) \in E'$, let $u = u^{h(i-1)}_{j_{i-1}}$, $v = u^{h(i)}_{j_i}$ and define

$$\mu'_e(x_{h(i-1)+1}, \ldots, x_{h(i)}) = \text{reach}_{u,v}(x_{h(i-1)+1}, \ldots, x_{h(i)}).$$

- For every $e = \left(u_{0,j_1,\ldots,j_{i-1}}, v_{0,j_1,\ldots,j_{i-1}}\right) \in E'$, let $u = u^{h(i-1)}_{j_{i-1}}$ and define

$$\mu'_e(x_{h(i-1)+1}, \ldots, x_{h(i)}) = \bigvee_{\substack{v \text{ is a leaf in layers} \\ h(i-1)+1,\ldots,h(i) \text{ in } B}} \text{reach}_{u,v}(x_{h(i-1)+1}, \ldots, x_{L(v)}).$$

Claim 3.12. *Construction 3.11 outputs a GIDT T which computes the same function as B. Furthermore, the number of vertices in layer i of T is $|L'_i| = \left(\prod_{1\le j<i}(w(h(j))\right)\cdot(w(h(i))+1)$.*

Proof. We first prove the equivalence of B and T, that is, we prove that B accepts an input $\sigma=\sigma_1,\ldots,\sigma_t$ if and only if T accepts σ. Let ℓ be the generation of t, that is $h(\ell-1)+1\le t\le h(\ell)$.

First assume that B accepts σ. W.l.o.g., assume that no proper prefix of σ is accepted by B (otherwise apply the following arguments to such minimal prefix). Then, there exists a path $P=(u_0^0,u_{j_1}^1,\ldots,u_{j_t}^t)$ in G where $u_{j_t}^t\in U_{\text{leaf}}$ and $\text{sat}_P(\sigma)=1$. Consider the path

$$P'=(u_0,u_{0,j_{h(1)}},\ldots,u_{0,j_{h(1)},j_{h(2)},\ldots,j_{h(\ell-1)}},v_{0,j_{h(1)},j_{h(2)},\ldots,j_{h(\ell-1)}})$$

in G'. We partition the path P in G to sub-paths – for every $1\le i\le\ell-1$, let $P^i=(u_{j_{h(i-1)}}^{h(i-1)},\ldots,u_{j_{h(i)}}^{h(i)})$ and let $P^\ell=(u_{j_{h(\ell-1)}}^{h(\ell-1)},\ldots,u_{j_t}^t)$. Since $\text{sat}_P(\sigma)=1$, we deduce that $\text{sat}_{P^i}(\sigma_{h(i-1)+1},\ldots,\sigma_{h(i)})=1$ for $1\le i\le\ell-1$ and $\text{sat}_{P^\ell}(\sigma_{h(i-1)+1},\ldots,\sigma_t)=1$. By the definition of $\text{reach}_{u,v}$, this implies that

$$\text{reach}_{u_{j_{h(i-1)}}^{h(i-1)},u_{j_{h(i)}}^{h(i)}}(\sigma_{h(i-1)+1},\ldots,\sigma_{h(i)})=1$$

for every $1\le i\le\ell-1$ and $\text{reach}_{u_{j_{h(\ell-1)}}^{h(\ell-1)},u_{j_t}^t}(\sigma_{h(i-1)+1},\ldots,\sigma_t)=1$, where $u_{j_t}^t$ is a leaf in B. Thus, in T we have

$$\text{sat}_{P'}(\sigma)=\left(\bigvee_{\substack{v\text{ is a leaf in layers}\\ h(\ell-1)+1,\ldots,h(\ell)\text{ in }B}}\text{reach}_{u_{j_{h(\ell-1)}}^{h(\ell-1)},v}(\sigma_{h(\ell-1)+1},\ldots,\sigma_{L(v)})\right)$$
$$\wedge\left(\bigwedge_{1\le i\le\ell-1}\text{reach}_{u_{j_{h(i-1)}}^{h(i-1)},u_{j_{h(i)}}^{h(i)}}(\sigma_{h(i-1)+1},\ldots,\sigma_{h(i)})\right)=1.$$

In the other direction, assume that T accepts σ. W.l.o.g., assume that no proper prefix of σ is accepted by T. Then in T there exists a path $P'=(u_0,u_{0,j_1},\ldots,u_{0,j_1,\ldots,j_{\ell-1}},v_{0,j_1,\ldots,j_{\ell-1}})$ where $v_{0,j_1,\ldots,j_{\ell-2}}$ is a leaf and $\text{sat}_{P'}(\sigma)=1$. This implies that for every $1\le i\le\ell-1$

$$\mu'_{(u_{0,j_1,\ldots,j_{i-1}},u_{0,j_1,\ldots,j_{i-1},j_i})}(\sigma_{h(i-1)+1},\ldots,\sigma_{h(i)})$$
$$=\text{reach}_{u_{j_{i-1}}^{i-1},u_{j_i}^i}(\sigma_{h(i-1)+1},\ldots,\sigma_{h(i)})=1.$$

Thus, for each $1\le i\le\ell-1$, there exists some path P^i from $u_{j_i}^{i-1}$ to $u_{j_i}^i$ in G such that $\text{sat}_{P^i}(\sigma)=1$. Furthermore, since $\mu'_{(u_{0,j_1,\ldots,j_{\ell-1}},v_{0,j_1,\ldots,j_{\ell-1}})}(\sigma_{h(\ell-1)+1},\ldots,\sigma_t)=1$ there exists a leaf v in B such that $\text{reach}_{u_{j_{\ell-1}}^{\ell-1},v}(\sigma_{h(\ell-1)+1},\ldots,\sigma_t)=1$ and, therefore, in G there exists a path

P^ℓ from $u^{\ell-1}_{j_{\ell-1}}$ to a leaf v such $\mathrm{sat}_{P^\ell}(\sigma) = 1$. By concatenating the paths $P^1, \ldots, P^\ell$, we obtain a path P in G from u_0 to a leaf for which $\mathrm{sat}_P(\sigma) = 1$; thus, B accepts σ.

To bound $|L'_i|$, recall that a vertex in layer i of T is either $u_{0,j_1,\ldots,j_i}$ or a leaf $v_{0,j_1,\ldots,j_{i-1}}$, thus $|L'_i| \leq \left(\prod_{1\leq j\leq i-1} w(h(i))\right)\left(w(h(i))+1\right)$. □

3.1.4 Putting Everything Together

Next, we combine our results from Sects. 3.1.1 to 3.1.3 and construct an evolving secret-sharing scheme for LIBPs.

Theorem 3.13. *Let $B = (G = (V, E), u_0, \mu)$ be an LIBP and $h : \mathbb{N} \to \mathbb{N}$ be an increasing function, where $h(0) = 0$. There exists an evolving secret-sharing scheme realizing B in which the share of a party p_t in generation i, i.e., $h(i-1)+1 \leq t \leq h(i)$, is the shares of p_t in the $\left(\prod_{1\leq j\leq i-1} w(h(j))\right)(w(h(i))+1)$ secret-sharing schemes realizing the predicates of the edges between layer $i-1$ and layer i in the GIDT constructed in Construction 3.11.*

Proof. We apply Construction 3.11 to transform B into an equivalent GIDT T with the specified h. Then apply Construction 3.9 to T to obtain an evolving secret-sharing scheme realizing T. By Claim 3.12 and Claim 3.10, we obtain an evolving secret-sharing scheme realizing B with the shares as stated in the theorem. □

Remark 3.14. Theorem 3.13 does not state how to choose the function h for a given LIBP B. The choice of h that will minimize the share size depends on the particular LIBP B. On one hand, when h grows slowly, the number of vertices in each level of the GIDT T obtained from B becomes larger. On the other hand, if h grows fast, there are more parties in each generation and the predicates labeling the edges of T become more complicated. The optimal choice of h should balance these two conflicting complexities.

3.2 Evolving Secret-Sharing Schemes for Dynamic-Threshold via LIBPs

Komargodski and Paskin-Cherniavsky [32] have constructed an evolving secret-sharing scheme for dynamic-threshold access structures; their construction uses GIDTs. As an example of our construction of an evolving secret-sharing schemes for LIBPs, we describe their construction using our framework.

Fix a non-decreasing function $\mathrm{tr} : \mathbb{N} \to \mathbb{N}$. We first describe an LIBP $B_{\mathrm{DynTr}} = (G_{\mathrm{DynTr}} = (V, E), u^0_0, \mu)$ for the $\mathrm{tr}(t)$-dynamic-threshold function access structure (as defined in Definition 2.15); this construction generalizes the ideas of Example 3.3. The construction we describe can be optimized; we choose the specific B_{DynTr} such that the predicates obtained after transforming it to a GIDT are simple. For $t \in \mathbb{N} \cup \{0\}$, the t^{th} layer of the graph G_{DynTr} is $L_t = \{u^t_0, \ldots, u^t_t, v^t_{\mathrm{tr}(t)}\}$. The vertices of G_{DynTr} are $V = \cup_{t\in\mathbb{N}\cup\{0\}} L_t$. The source

of B_{DynTr} is u_0^0 and the leaves are $U_{\mathrm{leaf}} = \{v^t_{\mathrm{tr}(t)} : t \in \mathbb{N}\}$. For every $t \in \mathbb{N}$ and $0 \leq i \leq t-1$ there are two edges: an edge (u_i^{t-1}, u_i^t) labeled by 1 and an edge (u_i^{t-1}, u_{i+1}^t) labeled by x_t. There are additional edges entering into the leaves: for every $t \in \mathbb{N}$ there are two edges: an edge $(u^{t-1}_{\mathrm{tr}(t)}, v^t_{\mathrm{tr}(t)})$ labeled by 1 and an edge $(u^{t-1}_{\mathrm{tr}(t)-1}, v^t_{\mathrm{tr}(t)})$ labeled by x_t.

Informally, the LIBP counts the number of ones in an input σ. Formally, let $\sigma \in \{0,1\}^t$ be an input. Then, there is a path from the source vertex u_0^0 to a vertex u_i^t (for some $0 \leq i \leq t$) that is satisfied by the input σ if and only if σ contains at least i ones; furthermore there is a path from the source u_0^0 to a leaf $v^t_{\mathrm{tr}(t)}$ that is satisfied by σ if and only if σ contains at least $\mathrm{tr}(t)$ ones (this is proved by a simple induction). That is, the LIBP B_{DynTr} computes the $\mathrm{tr}(t)$-dynamic-threshold function. The width B_{DynTr} is $w(t) = t+1$.

Let $h : \mathbb{N} \to \mathbb{N}$ be any increasing function. We next describe a GIDT T_{DynThr} – the output of the transformation described in Construction 3.11 given B_{DynThr} and h. The i^{th} layer of T_{DynThr} is

$$\begin{aligned}&\{u_{0,j_1,\ldots,j_i} : 0 \leq j_1 \leq h(1), \ldots, 0 \leq j_i \leq h(i)\}\\ &\quad \cup \{v_{0,j_1,\ldots,j_{i-1}} : 0 \leq j_1 \leq h(1), \ldots, 0 \leq j_{i-1} \leq h(i-1)\}.\end{aligned}$$

In B_{DynThr}, an assignment $\sigma = \sigma_{h(i-1)+1}, \ldots, \sigma_{h(i)}$ satisfies a path from $u^{h(i-1)}_{j_{i-1}}$ to $u^{h(i)}_{j_i}$ if and only if σ contains at least $j_i - j_{i-1}$ ones. Thus, in T_{DynThr} there is an edge $(u_{0,j_1,\ldots j_{i-1}}, u_{0,j_1,\ldots,j_{i-1},j_i})$ labeled by the threshold function $\mathrm{Thr}_{j_i - j_{i-1}}(x_{h(i-1)+1}, \ldots, x_{h(i)})$ (where Thr_b is the b-threshold function, that is, $\mathrm{Thr}_b(y_1, \ldots, y_m) = 1$ if and only if $\sum_{j=1}^m y_j \geq b$). Furthermore, an assignment $\sigma = \sigma_{h(i-1)+1}, \ldots, \sigma_t$ satisfies a path from a vertex $u^{h(i-1)}_{j_{i-1}}$ to a leaf $v^t_{g(t)}$ (where $h(i-1)+1 \leq t \leq h(i)$ and $\mathrm{tr}(t) - t + h(i-1) \leq j_{i-1} \leq \mathrm{tr}(t)$) if and only if σ contains at least $\mathrm{tr}(t) - j_{i-1}$ ones. Thus, in T_{DynThr} there is an edge $(u_{0,j_1,\ldots j_{i-1}}, v_{0,j_1,\ldots j_{i-1}})$ labeled by: $\bigvee_{h(i-1)+1 \leq t \leq h(t)} \mathrm{Thr}_{\mathrm{tr}(t)-j_{i-1}}(x_{h(i-1)+1}, \ldots, x_t)$. Informally, this label is satisfied in T_{DynThr} if in B_{DynThr} at least one path from $u_{0,j_1,\ldots j_{i-1}}$ to a leaf in the i-the generation is satisfied. By Claim 3.12, the GIDT T_{DynThr} computes the $\mathrm{tr}(t)$-dynamic-threshold function.

We implement T_{DynThr} using the secret-sharing scheme of Π_{GIDT}, where each threshold function is implemented using Shamir's t-out-of-n secret-sharing scheme with share size $\log(n+1)$. We next analyze the share size in this scheme for a party p_t in generation i, i.e., $h(i-1)+1 \leq t \leq h(i)$. First, p_t participates in one secret-sharing scheme for each edge $(u_{0,j_1,\ldots,j_{i-1}}, u_{0,j_1,\ldots,j_i})$. There are $\prod_{1 \leq j \leq i}(h(j)+1)$ such edges and the share size of Shamir's scheme realizing the label of each edge is $\log(h(i) - h(i-1)) \leq \log(h(i))$. Second, p_t participates in one secret-sharing scheme for each edge $(u_{0,j_1,\ldots,j_{i-1}}, v_{0,j_1,\ldots,j_{i-1}})$. There are $\prod_{1 \leq j \leq i-1}(h(j)+1)$ such edges. Realizing the label of each edge requires $h(i) - h(i-1)$ applications of Shamir's secret-sharing scheme, resulting in share size $\sum_{t=h(i-1)+1}^{h(i)} \log(t - h(i-1)) < h(i)\log(h(i))$ per edge. To conclude, the share size p_t is $\log(h(i)) \prod_{1 \leq j \leq i}(h(j)+1)$.

To complete the analysis of the share size, we need to choose the function h and compute the share size as a function of t. This was done in [32], taking $h(i) = 2^{2^i} - 1$. Thus, the share size is smaller than $2^i \cdot \prod_{1 \le j \le i} 2^{2^j} = 2^i \cdot 2^{\sum_{j=1}^{i} 2^j} < 2^i \cdot 2^{2^{i+1}}$. As t is in generation i, it must be that $t \ge h(i-1)+1 = 2^{2^{i-1}}$ and the share size is $O(t^4 \log t)$ (as $2^{2^{i+1}} = 2^{4 \cdot 2^{i-1}} = \left(2^{2^{i-1}}\right)^4 \le t^4$).

Improved Evolving Secret-Sharing Schemes for Small $\mathrm{tr}(t)$. Xing and Yuan's result [43] showed that $\mathrm{tr}(t)$-dynamic-threshold access structures when $\mathrm{tr}(t) = t^\beta$ some for $0 < \beta < 1$ can be realized by an evolving secret-sharing scheme with share size $O(t^{4\beta})$. We show that we can get a similar result using our framework; we get a secret-sharing scheme with share size $O(t^{4\beta} \log t)$ for t^β-dynamic-threshold access structures. The main observation is that in the LIBP B_{DynThr} for $\mathrm{tr}(t) = t^\beta$ we can reduce the number of vertices in layers $h(1), h(2), \ldots$ – we take $L_{h(i)} = \{u_0^{h(i)}, \ldots, u_{h(i)^\beta - 1}^{h(i)}\} \cup \{v_{h(i)^\beta}^{h(i)}\}$. That is, if the number of ones in a prefix of length $h(i)$ of an input σ is at least $\mathrm{tr}(h(i)) = h(i)^\beta$, then the LIBP can accept σ without looking at bits beyond this input. We do not change other layers, thus the labels on the edges in the GIDT resulting from applying the transformation of Construction 3.11 to the optimized LIBP remains the same threshold functions. Again we take $h(i) = 2^{2^i} - 1$ and get share size

$$\log(h(i)) \prod_{j=1}^{h(i)} h(i)^\beta = 2^i \cdot \prod_{1 \le j \le i} 2^{\beta \cdot 2^j} = 2^i \cdot 2^{\beta \cdot \sum_{j=1}^{i} 2^j} < 2^i \cdot 2^{\beta \cdot 2^{i+1}}.$$

As t is in generation i, it must be that $t \ge h(i-1)+1 = 2^{2^{i-1}}$ and the share size is $O(t^{4\beta} \log t)$ (as $2^{\beta \cdot 2^{i+1}} = 2^{4\beta \cdot 2^{i-1}} = \left(2^{2^{i-1}}\right)^{4\beta}$).

3.3 Evolving Secret-Sharing Schemes for LIBPs with Bounded Width

In this section we show the main application of our construction of evolving secret-sharing schemes for LIBPs, showing that LIBPs with small width can be realized with small share size. We show two results for LIBPs with small width:

- A construction presented in Theorem 3.15 showing that if the width is at most $2^{0.15t}$, then the LIBP can be realized with non-trivial share size of $O(2^{0.97t})$ and if the width is $2^{\varepsilon t}$ for $\varepsilon < 0.04$ the share size is $2^{2\sqrt{\varepsilon} \cdot t}$. This is in contrast with the lower bound of [36], proving that there exists an evolving access structure such that in every evolving secret-sharing realizing it the share size of infinitely many parties p_t is at most $2^{t - o(t)}$.
- A construction presented in Theorem 3.17 that LIBPs with width at most $w(t)$ can be realized by an evolving secret-sharing scheme with share size $(w(2t))^{O(\log t)}$; for example, if $w(t) = t^c$, i.e., the width is polynomial, then the share size of p_t is quasi-polynomial. The second construction achieves smaller share size than the first construction when $w(t) \ll 2^{t/\log^2 t}$. As an

application of Theorem 3.17, we show an evolving secret-sharing scheme for the evolving directed layered st-connectivity access structure, where for every $t \in \mathbb{N}$ the share size of p_t is at most $t^{O(\log t)}$. Due to space limitation, the construction is deferred to the full version [1].

Theorem 3.15. *Every LIBP B of width $w(t) \leq 2^{0.15t}$ can be realized by an evolving secret-sharing scheme with share size $2^{0.97t}$. Furthermore, for $\varepsilon(t) < 0.04$, every LIBP B of width $w(t) \leq 2^{\varepsilon t}$ can be realized by an evolving secret-sharing scheme with share size $2^{2\sqrt{\varepsilon(t)}\cdot t}$.*[5]

Proof. Let $\varepsilon = \varepsilon(t)$, and let $c = c(t) > 1$ to be determined later. We apply Theorem 3.13 to B of width at most $2^{\varepsilon t}$ and $h(i) = c^i$. To realize the predicates of the edges, we use the best known secret-sharing scheme for arbitrary n-party access structures of [5]. Fix a party p_t and let i be the generation of p_t, that is, $c^{i-1} + 1 \leq t \leq c^i$, in particular

$$c^i \leq tc. \tag{1}$$

By Theorem 3.13, the number of shares of secret-sharing schemes of predicates that p_t holds is

$$\begin{aligned}\left(\prod_{1\leq j\leq i-1} w(h(j))\right)(w(h(i))+1) &= O\left(\prod_{1\leq j\leq i} 2^{\varepsilon c^j}\right) = O\left(2^{\varepsilon\sum_{1\leq j\leq i} c^j}\right)\\ &\leq O\left(2^{\varepsilon c^{i+1}/(c-1)}\right) \leq O\left(2^{\varepsilon c^2 t/(c-1)}\right), \end{aligned} \tag{2}$$

where the last inequality is from (1). The share size of the secret-sharing scheme of [5] for an n-party access structure is $2^{0.585n}$; we apply it with the parties of a generation i – there are $h(i) - h(i-1) = c^i - c^{i-1} \leq tc - t = (c-1)t$ and the share size for each invocation of the secret-sharing of [5] is $2^{0.585(c-1)t}$. Thus, the size of the share of p_t is

$$\begin{aligned} O\left(2^{\varepsilon c^2 t/(c-1)} \cdot 2^{0.585(c-1)t}\right) &= O\left(2^{\left(\frac{\varepsilon c^2}{c-1}+0.585(c-1)\right)t}\right)\\ &= O\left(2^{\left(\varepsilon(c+1)+\frac{\varepsilon}{c-1}+0.585(c-1)\right)t}\right). \end{aligned} \tag{3}$$

Taking $\varepsilon = 0.15$ and $c = 1.46$, we obtain from (3) that for every LIBP of width $w(t) \leq 2^{0.15t}$ the share size of p_t is less than $2^{0.97t}$. For the second item of the theorem, take $c = 1 + \sqrt{\frac{\varepsilon}{0.585}}$; for $\varepsilon < 0.04$ we get that $c < 1.27$ and the share size we obtain from (3) is $O\left(2^{\left(2.27\varepsilon+2\sqrt{0.585\varepsilon}\right)t}\right) = 2^{2\sqrt{\varepsilon}\cdot t}$. □

For LIBPs of width $2^{\varepsilon t}$ for $\varepsilon < 0.04$, the share size decreases as the width decreases. When the width is $w(t) \leq 2^{o(t/\log^2 t)}$, we can get better share size by using special-purpose secret-sharing schemes to realize the labels of the edges. Recall in Construction 3.11, the label on each edge is $\text{reach}_{u,v}$ (or a conjunction

[5] The constants in the theorem are not optimized.

of such predicates). When the width of the LIBP is somewhat small, we can use the formula-based secret-sharing scheme of [12] to realize $\text{reach}_{u,v}$; in the secret-sharing scheme of [12] the total share size is the number of leaves of the monotone formula. For completeness, we next describe a monotone Boolean formula for $\text{reach}_{u,v}$.

Claim 3.16. *For every layered finite non-deterministic branching program $B_{\text{finite}} = (G, u_0, \mu)$ of width at most w and ℓ layers, there exists a monotone Boolean formula with at most $(2w)^{1+\log \ell}$ leaves.*

Proof. The construction of the monotone Boolean formula is as follows, where w.l.o.g., we assume that $\ell + 1$ is a power of 2 (this can at most double ℓ); we number the layers of G by layer 0 to layer $\ell - 1$. We construct the formula $F_{u,v}$ recursively. If $\ell = 2$ (i.e., u and v are in consecutive layers), then if $(u, v) \notin E$ then $F_{u,v} = 0$. Otherwise $F_{u,v}$ is the label of the edge (u, v), i.e., either $F_{u,v} = 1$ or $F_{u,v} = x_i$ for some variable x_i. If $\ell > 2$, let $u_1, \ldots, u_w$ be the vertices in G in layer $(\ell - 1)/2$; any path from u to v in G passes via exactly one vertex u_i in layer $(\ell - 1)/2$, hence $F_{u,v} = \bigvee_{i=1}^{w}(F_{u,u_i} \wedge F_{u_i,v})$, where F_{u,u_i} and $F_{u_i,v}$ were defined by the recursion.

We next compute the number of leaves in $F_{u,v}$. If $\ell = 2$ the number of leaves is at most 1. If $\ell > 2$, the formula contains $2w$ formulas for graphs with $(\ell-1)/2+1$ layers; by induction, the number of leaves in $F_{u,v}$ is at most $(2w)^{\log \ell}$. □

Theorem 3.17. *Let $w : \mathbb{N} \to \mathbb{N}$ be a non-decreasing function. Every LIBP B of width $w(t)$ can be realized by evolving secret-sharing scheme with share size $(w(2t))^{O(\log(t))}$.*

Proof. We apply Theorem 3.13 to B and $h(i) = 2^i$. To realize the predicates labeling the edges we use the formula-based secret-sharing scheme of [12] with the formula for $\text{reach}_{u,v}$ described in Claim 3.16. Observe that the predicate $\bigvee_{v \text{ is a leaf in layers } h(i-1)+1,\ldots,h(i) \text{ in } B} \text{reach}_{u,v}$ can be transformed to one instance of $\text{reach}_{u,v'}$ by adding a new vertex v' and connection each leaf v to v' by adding one path to the graph labeled by 1.

Fix a party p_t and let i be the generation of p_t, that is, $2^{i-1} + 1 \leq t \leq 2^i$, in particular $2^i \leq 2t$. The number of shares of secret-sharing schemes for $\text{reach}_{u,v}$ that p_t holds is at most

$$\left(\prod_{1 \leq j \leq i-1} w(h(j))\right)(w(h(i)) + 1) \leq O\left(\prod_{1 \leq j \leq i} w(2t)\right) \leq (w(2t))^{O(\log t)}, \quad (4)$$

where the inequalities are obtained from (1) and the monotinicity of h. The share size of the secret-sharing scheme of [12] for the monotone formula for $\text{reach}_{u,v}$ for the graph with $h(i) - h(i-1) \leq h(i)$ layers and width at most $w(h(i)) \leq w(2t)$ is $(w(2t))^{O(\log h(i))} = (w(2t))^{O(\log(2^i))} = (w(2t))^{O(\log t)}$. Thus, the size of the share of p_t is $(w(2t))^{O(\log t)} \cdot (w(2t))^{O(\log t)} = (w(2t))^{O(\log t)}$. □

Acknowledgments. The first and second authors are supported by the ISF grant 391/21 and by the Frankel center for computer science.

References

1. Alon, B., Beimel, A., Ben David, T., Omri, E., Paskin-Cherniavsky, A.: New upper bounds for evolving secret sharing via infinite branching programs. Cryptology ePrint Archive, Paper 2024/419 (2024). https://eprint.iacr.org/2024/419
2. Applebaum, B., Arkis, B.: On the power of amortization in secret sharing: d-uniform secret sharing and CDS with constant information rate. In: TCC 2018, vol. 11239. LNCS, pp. 317–344 (2018)
3. Applebaum, B., Beimel, A., Farràs, O., Nir, O., Peter, N.: Secret-sharing schemes for general and uniform access structures. In: Ishai, Y., Rijmen, V. (eds.) EUROCRYPT 2019. LNCS, vol. 11478, pp. 441–471. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-17659-4_15
4. Applebaum, B., Beimel, A., Nir, O., Peter, N.: Better secret sharing via robust conditional disclosure of secrets. In: STOC 2020, pp. 280–293 (2020)
5. Applebaum, B., Nir, O.: Upslices, downslices, and secret-sharing with complexity of 1.5^n. In: CRYPTO 2021, vol. 12827, pp. 627–655. Springer (2021)
6. Beimel, A.: Lower bounds for secret-sharing schemes for k-hypergraphs. In Kai-Min Chung, editor, *ITC 2023*, vol. 267. LIPIcs, pp. 16:1–16:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2023)
7. Beimel, A., Ishai, Y., Kumaresan, R., Kushilevitz, E.: On the cryptographic complexity of the worst functions. In: Lindell, Y. (ed.) TCC 2014. LNCS, vol. 8349, pp. 317–342. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-642-54242-8_14
8. Beimel, A., Kushilevitz, E., Nissim, P.: The complexity of multiparty PSM protocols and related models. In: Nielsen, J.B., Rijmen, V. (eds.) EUROCRYPT 2018. LNCS, vol. 10821, pp. 287–318. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-78375-8_10
9. Beimel, A., Othman, H.: Evolving ramp secret-sharing schemes. In: Catalano, D., De Prisco, R. (eds.) SCN 2018. LNCS, vol. 11035, pp. 313–332. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-98113-0_17
10. Beimel, A., Othman, H.: Evolving ramp secret sharing with a small gap. In: Canteaut, A., Ishai, Y. (eds.) EUROCRYPT 2020. LNCS, vol. 12105, pp. 529–555. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-45721-1_19
11. Benaloh, J., Rudich, S.: Private communication (1989)
12. Benaloh, J.C., Leichter, J.: Generalized secret sharing and monotone functions. In: CRYPTO '88, vol. 403. LNCS, pp. 27–35 (1988)
13. Blakley, G.R.: Safeguarding cryptographic keys. In: 1979 International Workshop on Managing Requirements Knowledge (MARK), pp. 313–318 (1979)
14. Blundo, C., De Santis, A., De Simone, R., Vaccaro, U.: Tight bounds on the information rate of secret sharing schemes. Des. Codes Cryptography **11**(2), 107–122 (1997)
15. Cachin, C.: On-line secret sharing. In: Proc. of the 5th IMA International Conference on Cryptography and Coding, vol. 1025. LNCS, pp. 190–198 (1995)
16. Chaudhury, S.S., Dutta, S., Sakurai, K.: Ac^0 constructions of secret sharing schemes - accommodating new parties. In: NSS 2020, vol. 12570. LNCS, pp. 292–308 (2020)
17. Cheng, Q., Cao, H., Lin, S.-J., Yu, N.: A construction of evolving k-threshold secret sharing scheme over a polynomial ring. arXiv preprint arXiv:2402.01144 (2024)
18. Csirmaz, L.: The dealer's random bits in perfect secret sharing schemes. Studia Sci. Math. Hungar. **32**(3–4), 429–437 (1996)

19. Csirmaz, L.: The size of a share must be large. J. Cryptol. **10**(4), 223–231 (1997)
20. Csirmaz, L., Tardos, G.: On-line secret sharing. Des. Codes Cryptography **63**(1), 127–147 (2012)
21. D'Arco, P., De Prisco, R., De Santis, A.: Secret sharing schemes for infinite sets of participants: a new design technique. Theor. Comput. Sci. **859**, 149–161 (2021)
22. D'Arco, P., De Prisco, R., De Santis, A., Pérez del Pozo, A., Vaccaro, U.: Probabilistic Secret Sharing. In: MFCS 2018, vol. 117. LIPIcs, pp. 64:1–64:16 (2018)
23. Desmedt, Y., Dutta, S., Morozov, K.: Evolving perfect hash families: a combinatorial viewpoint of evolving secret sharing. In: Cryptology and Network Security, pp. 291–307 (2019)
24. Dutta, S., Roy, P.S., Fukushima, K., Kiyomoto, S., Sakurai, K.: Secret sharing on evolving multi-level access structure. In: Information Security Applications, pp. 180–191 (2020)
25. Erdös, P., Pyber, L.: Covering a graph by complete bipartite graphs. Discret. Math. **170**(1–3), 249–251 (1997)
26. Francati, D., Venturi, D.: Evolving secret sharing made short. Cryptology ePrint Archive, Paper 2023/1534 (2023). https://eprint.iacr.org/2023/1534
27. Gertner, Y., Ishai, Y., Kushilevitz, E., Malkin, T.: Protecting data privacy in private information retrieval schemes. In: Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing, STOC '98, pp. 151–160. Association for Computing Machinery, New York (1998)
28. Ito, M., Saito, A., Nishizeki, T.: Secret sharing scheme realizing general access structure. Electronics and Communications in Japan (Part III: Fundamental Electronic Science) **72**(9), 56–64 (1989)
29. Karchmer, M., Wigderson, A.: On span programs. In: 8th Structure in Complexity Theory, pp. 102–111 (1993)
30. Komargodski, I., Naor, M., Yogev, E.: How to Share a Secret, Infinitely. In: Hirt, M., Smith, A. (eds.) TCC 2016. LNCS, vol. 9986, pp. 485–514. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53644-5_19
31. Komargodski, I., Naor, M., Yogev, E.: How to share a secret, infinitely. IEEE Trans. Inf. Theory **64**(6), 4179–4190 (2018)
32. Komargodski, I., Paskin-Cherniavsky, A.: Evolving secret sharing: dynamic thresholds and robustness. In: Kalai, Y., Reyzin, L. (eds.) TCC 2017. LNCS, vol. 10678, pp. 379–393. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-70503-3_12
33. Liu, T., Vaikuntanathan, V.: Breaking the circuit-size barrier in secret sharing. In: Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, pp. 699–708, New York, NY, USA. Association for Computing Machinery (2018)
34. Liu, T., Vaikuntanathan, V., Wee, H.: Conditional Disclosure of Secrets via Non-linear Reconstruction. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017. LNCS, vol. 10401, pp. 758–790. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63688-7_25
35. Liu, T., Vaikuntanathan, V., Wee, H.: Towards breaking the exponential barrier for general secret sharing. In: Nielsen, J.B., Rijmen, V. (eds.) EUROCRYPT 2018. LNCS, vol. 10820, pp. 567–596. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-78381-9_21
36. Mazor, N.: A lower bound on the share size in evolving secret sharing. In: Chung, K.-M. (ed.) 4th Conference on Information-Theoretic Cryptography, ITC 2023, June 6-8, 2023, Aarhus University, Aarhus, Denmark, vol. 267. LIPIcs, pp. 2:1–2:9. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2023)

37. Okamura, R., Koga, H.: New constructions of an evolving 2-threshold scheme based on binary or d-ary prefix codes. In: 2020 International Symposium on Information Theory and Its Applications (ISITA), pp. 432–436 (2020)
38. Peter, N.: Evolving conditional disclosure of secrets. In: Information Security: 26th International Conference, ISC 2023, Groningen, The Netherlands, November 15–17, 2023, Proceedings, pp. 327–347. Springer, Heidelberg (2023)
39. Phalakarn, K., Suppakitpaisarn, V., Attrapadung, N., Matsuura, K.: Evolving homomorphic secret sharing for hierarchical access structures. In: Nakanishi, T., Nojima, R. (eds.) IWSEC 2021. LNCS, vol. 12835, pp. 77–96. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-85987-9_5
40. Pitassi, T., Robere, R.: Lifting nullstellensatz to monotone span programs over any field. In: 50th STOC, pp. 1207–1219 (2018)
41. Shamir, A.: How to share a secret. Commun. ACM **22**, 612–613 (1979)
42. Sun, H.-M., Shieh, S.-P.: Secret sharing in graph-based prohibited structures. In: INFOCOM '97, pp. 718–724. IEEE (1997)
43. Xing, C., Yuan, C.: Evolving secret sharing schemes based on polynomial evaluations and algebraic geometry codes. IEEE Trans. Inf. Theory **70**(5), 3718–3728 (2024)
44. Yan, W., Lin, S.-J., Han, Y.S.: A new metric and the construction for evolving 2-threshold secret sharing schemes based on prefix coding of integers. IEEE Trans. Commun. **71**(5), 2906–2915 (2023)

Secret-Sharing Schemes for High Slices

Amos Beimel[1(✉)], Oriol Farràs[2], Or Lasri[1], and Oded Nir[3]

[1] Ben-Gurion University of the Negev, Be'er-Sheva, Israel
beimel@bgu.ac.il, orshlomo@post.bgu.ac.il
[2] Universitat Rovira i Virgili, Tarragona, Spain
oriol.farras@urv.cat
[3] Tel Aviv University, Tel Aviv, Israel
odednir@mail.tau.ac.il

Abstract. In a secret-sharing scheme, a secret is shared among n parties such that the secret can be recovered by authorized coalitions, while it should be kept hidden from unauthorized coalitions. In this work we study secret-sharing for k-slice access structures, in which coalitions of size k are either authorized or not, larger coalitions are authorized and smaller are unauthorized. Known schemes for these access structures had smaller shares for small k's than for large ones; hence our focus is on "high" $(n-k)$-slices where k is small.

Our work is inspired by several motivations: 1) Obtaining efficient schemes (with perfect or computational security) for natural families of access structures; 2) Making progress in the search for better schemes for general access structures, which are often based on schemes for slice access structures; 3) Proving or disproving the conjecture by Csirmaz (J. Math. Cryptol., 2020) that an access structures and its dual can be realized by secret-sharing schemes with the same share size.

The main results of this work are:

Perfect schemes for high slices. We present a scheme for $(n-k)$-slices with information-theoretic security and share size $kn \cdot 2^{\tilde{O}(\sqrt{k \log n})}$. Using a different scheme with slightly larger shares, we prove that the ratio between the optimal share size of k-slices and that of their dual $(n-k)$-slices is bounded by n.

Computational schemes for high slices. We present a scheme for $(n-k)$-slices with computational security and share size $O(k^2 \lambda \log n)$ based on the existence of one-way functions. Our scheme makes use of a non-standard view point on Shamir secret-sharing schemes that allows to share many secrets with different thresholds with low cost.

Multislice access structures. $(a:b)$-*multislices* are access structures that behave similarly to slices, but are unconstrained on coalitions in a wider range of cardinalities between a and b. We use our new schemes for high slices to realize multislices with the same share sizes that their duals have today. This solves an open question raised by Applebaum and Nir (Crypto, 2021), and allows to realize hypergraph access structures that are chosen uniformly at random under a natural set of distributions with share size $2^{0.491n+o(n)}$ compared to the previous result of $2^{0.5n+o(n)}$.

E. Boyle and M. Mahmoody (Eds.): TCC 2024, LNCS 15367, pp. 581–613, 2025.
https://doi.org/10.1007/978-3-031-78023-3_19

1 Introduction

Secret-sharing schemes, introduced by Shamir [Sha79] and Blakley [Bla79], is a pivotal cryptographic primitive that has many applications in cryptography and in neighboring fields. In a secret-sharing scheme, a dealer that holds a secret shares it among n parties, by sending each party a single message (called a share). It is required that predefined authorized coalitions will be able to recover the secret from their shares and that the secret will remain hidden from all unauthorized coalitions. A scheme is called *perfect* when the secret is kept information-theoretically hidden from unauthorized sets (i.e. they cannot learn anything about the secret from their shares even if they are computationally unbounded); it is called *computational* if secrecy is held against parties that are computationally bounded. The collection of authorized coalitions is called an *access structure*, and it can be captured by a monotone function $f : \{0,1\}^n \to \{0,1\}$ that outputs 1 on an input $x \in \{0,1\}^n$ iff x is the characteristic vector of an authorized set.

The most important efficiency measure for secret-sharing schemes is the size of the shares dealt to the participating parties. Hence, the goal of many research works has been to realize all (general) n-party access structures with small shares. Towards this end, modern schemes (following the seminal work of Liu and Vaikuntanathan [LV18]) typically first realize restricted families of access structures with non-trivially small shares and then compose them, in some "sophisticated" way, to get better schemes for general access structures. Improving the share size of such restricted families of access structure became a relatively central problem in the field. Among the families used in the above-mentioned paradigm, we can list k-slices (also known as k-uniform access structures) and $(a : b)$-*multislices*. A k-slice function can output arbitrary values for inputs of Hamming weight k, and must output 0 on lighter inputs and 1 on heavier ones. $(a : b)$-*multislices* are monotone functions that are unconstrained on inputs of Hamming weight between a and b, but must take the value 0 on lighter inputs, and the value 1 on heavier inputs. Note that a $(k : k)$-multislice is a k-*slice*.

Despite the growing importance of these access-structure families, works that have studied them so far have been, in some sense, incomprehensive, as they mainly focused on the regime where $k \ll n$. For example, there are at least a dozen papers dealing with secret-sharing schemes for 2-slices (also known as *forbidden graph access structures* (see, e.g., [SS97, BIKK14, BFMP22]). To the best of our knowledge, no previous papers study $(n-2)$-slices. Moreover, the best perfect schemes for slices have much smaller shares when k is small ("low slices"), compared to when k is large ("high slices"). For computational schemes the situation is even worse, as we do not know of any work that studied computational secret-sharing schemes for slices based on most basic assumption of the existence of one-way functions (OWF).[1] We therefore bring forward the following questions:

[1] Computational schemes for low slices follow from combining results of [ABF+19, ABI+23b].

- Can high slices be realized by a perfect scheme with the same share sizes as low slices?
- Can better schemes for high slices help improve schemes for general access structures?
- Can natural families of functions like slices be realized with smaller shares assuming OWFs exist?

Before we move on to survey the literature regarding the topics discussed so far, we note that these questions are also closely related to the concept of secret-sharing *duality*. In the notation of functions, the dual of an access structure $f : \{0,1\}^n \to \{0,1\}$ is the function f^* that satisfies $f^*(x) = 1 - f(\overline{x})$, where $\overline{x}$ is the string for which $\overline{x}_i = 1 - x_i$ in every $i \in [n]$. We observe that if f is monotone then so is f^*, that for every function f it holds that $(f^*)^* = f$, that the duals of k-slices are $(n-k)$-slices, and that the duals of $(a:b)$-multislices are $(n-b:n-a)$-multislices. For linear and multi-linear secret-sharing schemes (see Definition 2.2), the optimal share size of every function and its dual are identical [Gál95, Feh98, FHKP17], but it is not known whether this property holds for general schemes. In his work from 2020, Csirmaz [Csi20] focused on this question and formalized the following conjecture:

Conjecture 1.1 (Csirmaz's conjecture). The optimal share size per bit of the secret (also known as the *information ratio*) of primal and dual access structures is equal.

Csirmaz showed that in a relaxed model of secret sharing (where errors in recovery or security may occur with negligible probability), the conjecture is false. It is entirely unclear whether this says anything about duality in the standard (error-free) model when sharing one-bit secrets. In fact, we can neither exclude the possibility that the conjecture is true nor the possibility of an exponential gap between the share size of access structures and their duals. On the positive side, it is known that for simple functions such as thresholds the dual and primal share sizes are equal for large enough secrets, and Bogdanov [Bog23] recently proved that the optimal share size of 2-thresholds and $(n-1)$-thresholds is exactly the same for every n. Hence, studying duality of more complex families of functions seems like a natural next step in better understanding Csirmaz's conjecture. We ask the following question about the share size of slices:

- Can we bound the gap between the share sizes of slices and their duals? I.e., given a scheme for k-slices with share size L, can we realize their dual $(n-k)$-slices with shares of size $L \cdot t(n,k)$ where $t(n,k)$ is small?

1.1 Related Work

Perfect Schemes for General Access Structures. The first perfect schemes for general access structures that were introduced by Ito, Saito and Nishizeki. [ISN87] had shares of size $O(2^n)$. A generalization of these schemes was presented by Benaloh and Leiscter [BL88], showing that if a function can be computed by a

monotone formula of size L then it can be realized by a secret-sharing scheme with total share size L. In [KW93], it was shown that the share size of any function can also be tied to its monotone span-program complexity, but this is still $O(2^n)$ in the worst case. A breakthrough result of Liu and Vaikuntanathan [LV18] followed decades later, describing a scheme with share size $O(2^{0.994n})$. Since then, the state of the art has further improved several times [ABF+19, ABNP20, AN21]. In the latest among these developments, Applebaum and Nir [AN21] showed how to realize general access structures with shares of size $1.5^{n+o(n)}$. On the lower bound front, Csirmaz [Csi97] described an access structure that requires shares of size $\Omega(n/\log n)$, and it was proved in [ABN+22] that the modern techniques following the breakthrough of [LV18] cannot realize all access structures with shares smaller than $2^{o(n/\log^2 n)}$.

Perfect Schemes for Slices and Multislices. Secret-sharing schemes for slice access structures, also called *uniform access structures*, were previously studied in several works as [AA18, BKN18, BP18, LV18, ABF+19, AN21, ABN+22]. There exists a simple scheme that realizes every k-slice f by taking a monotone DNF or CNF formulas for f and applying the formula-to-scheme transformation of [BL88]. For one-bit secrets,[2] this scheme has shares of size $\binom{n-1}{k-1}$ for k-slices and $\binom{n-1}{n-k}$ for $(n-k)$-slices for $k \leq n/2$. The best-known upper bounds on the share size of slices in literature outperform the naive scheme above in some regimes, as detailed in the following g Fig. 1. Prior to this work (see the bounds stated in Fig. 1), there exists a gap between the share sizes of slices and their duals. When k is constant, k-slices have share sizes of $n^{o(1)}$ while their dual $(n-k)$-slices have share sizes of $O\left(\binom{n-1}{n-k}\right) = O(n^{k-1})$. When $k = \log n$ (a regime is relevant for realizing multislices and general functions) the gap will be between $n^{O(\log\log n)}$ for low slices and $O(n^{\log n})$ for high ones. We also note that the multi-linear scheme of [AA18] for k-slices has information ratio $2^{O(k)}$ for secrets with size that is double-exponential in n. By the duality closure properties for multi-linear schemes, this implies that there exists a scheme for $(n-k)$-slices with the same information ratio.

For multislice access structures, the situation is similar. Applebaum and Nir [AN21] designed a scheme for $(a : b)$-multislices as a stepping stone for schemes for general functions with shares of size $\min\{\binom{b}{\geq a} \cdot 2^{o(n)}, 2^{0.585n+o(n)}\}$, where $\binom{b}{\geq a} := \sum_{a \leq i \leq b} \binom{b}{i}$. It is not hard to see that this scheme is not "balanced" with respect to duality. For example, the share size for $(0 : 0.1n)$-multislices is $2^{0.1n+o(n)}$ while that of their dual $(0.9n : n)$-multislices is $2^{\mathrm{H}_2(0.9)n+o(n)} > 2^{0.45n}$, where H_2 is the binary entropy function.

[2] For long secrets it is sometimes known how to realize schemes with smaller share sizes per secret bit (better *information ratio*) with amortization techniques. The share size of mentioned scheme based on formulas can be improved by a factor of $\log n$ for moderately long secrets and $k \leq n/2$ [EP97, Bei23], and a k-slice scheme of [ABF+19] has information ratio k^2 for secrets of size that is doubly-exponential in n.

The share size of perfect schemes for k-slices and $(n-k)$-slices				
Slice height	Below $\log n$	Between $\log n$ and $n/\log n$	Between $n/\log n$ and $n - n/\log n$	$(n-k)$-slices for $k \leq n/\log n$
Upper bounds	$2^{O(k)+\tilde{O}(\sqrt{k \log n})}$ [AA18]	$kn \cdot 2^{\tilde{O}(\sqrt{k \log n})}$ [ABF+19]	$2^{\tilde{O}(\sqrt{n})}$ [LV18]	$kn \cdot 2^{\tilde{O}(\sqrt{k \log n})}$ Theorem 3.5, compared to $O(n^{k-1})$ in the formula-based scheme
Lower bounds	$\Omega(\log n)$ [KN90,CCX13,BGK16]			

Fig. 1. The best-known bounds on the share size of perfect secret-sharing schemes for k-slices for $1 < k < n-2$. For $k = 1$ there exist simple schemes with share size $\log n$, and for $k = n-2$ shares of size $O(\sqrt{n})$ can be obtained by taking the dual scheme of the 2-slice scheme of [GKW15]. The $\Omega(\log n)$ lower bound by [KN90, CCX13] was proved for the 2-threshold function (which is also a 2-slice function). The same bound was later proved in [BGK16] for all k-slices. The borders between the ranges of parameters are written without asymptotical notation for better readability (e.g., should be $\Theta(\sqrt{n})$ instead of $\sqrt{n}$).

Computational Secret-Sharing Schemes Based on OWF. Computational secret-sharing schemes (CSSS) can be based on a variety of cryptographic hardness assumptions. In this work, we will focus on the most basic one: the existence of one-way functions (OWFs). In the computational setting, the efficiency of schemes will also be measured with respect to a security parameter λ.[3] Yao [Yao89] (see also [VNS+03]) was the first to consider secret-sharing schemes in the computational setting. He showed that assuming the existence of one-way functions, any function that can be computed by a monotone circuit with C wires can be realized by a CSSS with share size $O(\lambda C)$.[4]

Krawczyk [Kra94] showed how to share large secrets of size S according to a k-threshold function with shares of size $|S|/k + \lambda$, thus bypassing an information-theoretic lower bound [KGH83] that states that shares cannot be smaller than the secret size. In this example, as opposed to the perfect schemes mentioned so far, the share sizes decrease when the cardinality k of the authorized sets increases.

[3] In the computational setting, the share size may also be reduced by using public information.

[4] Alternatively, with a CSSS with shares of size $O(\lambda)$ and public information of size $O(\lambda C)$. As mentioned before, in the information-theoretic setting, a similar result is only known for monotone formulas [BL88].

In the latest exciting study of computational schemes by Applebaum et al. [ABI+23b], they introduced new efficient schemes based on one-way functions for several families of access structures, including DNF formulas with long terms and CDS protocols (which are essentially a special class of slice functions, see discussion below). Their k-server CDS protocols have messages of size $\lambda + O(1)$ and $\text{poly}(t(\lambda))$-security (for a binary domain of inputs). By the connections between CDS protocols and secret-sharing schemes for slices [ABF+19, AA18], it can be shown (similarly to the proof in Sect. 4) that this implies that k-slices can be realized with shares of size $O\left(\lambda \log n \cdot \min\left\{kn, 2^{O(k)}\right\}\right)$ for $k \leq \sqrt{n}$ or $k \geq n - \sqrt{n}$ if OWF exist. Unlike the scheme of Krawczyk for thresholds, here the share size grows with k, the cardinality of the minimal authorized sets, and high slices are more expensive than low ones. Constructing computational secret-sharing schemes based on one-way functions for additional families of access structures, or even for all access structures, is an interesting open problem.

Besides the results discussed so far that are based on one-way functions, some schemes in the literature were based on stronger assumptions. In [ABI+23b], they designed several such schemes. Under the RSA assumption, they describe a CSSS that, given an arbitrary access structure f, represented by a truth table of size $N = 2^n$, produces shares of size $\text{poly}(n)$ in time $\tilde{O}(N)$. Weaker results are obtained under the decisional Diffie-Hellman and the decisional bilinear Diffie-Hellman assumptions. Under the RSA assumption, they also realize monotone CNF formulas with share size $\text{polylog}(m)$, where m is the number of clauses in the CNF formula. When considering $(n-k)$-slices that can be computed by a CNF with $O(n^k)$ clauses, the RSA based scheme with shares of size $\text{poly}(k \log n)$. In [KNY17], they give a construction of a computational secret-sharing scheme for any monotone function in NP assuming witness encryption for NP and the existence of one-way functions.

1.2 Our Results

We present several secret-sharing schemes for high slice functions, aiming to narrow or close as many gaps as possible between the share size of low slices and that of high slices. Our computational scheme for high slices will perform even better than its counterpart for low slices.

We prove the following theorem for perfect schemes:

Theorem 1.2 (Perfect Schemes for High Slices). *Let $k \leq n/2$ be positive integers. For every $(n-k)$-slice function f, there exists a secret-sharing scheme realizing f with share size $kn \cdot 2^{\tilde{O}(\sqrt{k \log n})}$.*

Our scheme closes the current gap in share sizes between slices and their duals when k is logarithmic in n (share size of $n^{O(\log \log n)}$ in both cases), and narrows it down substantially when k is constant ($n^{1+o(1)}$ compared to $n^{o(1)}$ for low constant-k slices). We note that given any constant integer k, our scheme for $(n-k)$-slices even outperform the scheme by Applebaum et al. [ABF+19] that only works for long secrets of size at least $2^{n^{n-k}}$, and has shares of size $O(n^2)$ per secret bit.

We also present a scheme for $(n-k)$-slices with a simpler structure that proves the following theorem:

Theorem 1.3 (Duality and Slices). *For every two integers $k < n$, if there exists an n-party secret-sharing scheme for k-slices with share size L, then there exists an n-party secret-sharing scheme for $(n-k)$-slices with share size $L \cdot n$.*

Our scheme works for every k, and so it allows to realize high slice functions with low ones or the other way around. Thus, by Theorem 1.3 the ratio between the share size of slice functions and their duals is bounded by n in both directions. We remark that for a given $(n-k)$-slice, our construction uses a k-slice that is *not* its dual.

Next, we present a computational scheme for high slices, which implies the following theorem:

Theorem 1.4 (Computational Scheme for High Slices, Informal). *Let f be an $(n-k)$-slice with $k \leq \sqrt{n}$. Then if OWF exist, f can be realized by a computationally-secure secret-sharing scheme with share size $O(k^2 \lambda \log n)$ (where λ is the security parameter). The running time of the sharing and reconstruction algorithms in the CSSS is* $\mathrm{poly}(\binom{n}{k}, \lambda)$.[5]

Recall that by the previously-best scheme for k-slices based on OWFs has shares of size $\min\{kn, 2^{O(k)}\} \cdot \lambda \log n$ [AA18, ABF+19, ABI+23b]. Similarly to the computational scheme of Krawczyk and unlike perfect schemes, by Theorem 1.4 high slices now have smaller shares than low slices in CSSS (Fig. 2).

The share size of computational schemes for k-slices based on OWF		
Slice height	k-slices	$(n-k)$-slices
Upper bounds	$\min\{kn, 2^{O(k)}\} \cdot \lambda \log n$ [AA18,ABF$^+$19,ABI$^+$23b]	$O(k^2 \lambda \log n)$ Theorem 4.1
Lower bounds	$\Omega(\log n)$ [ABI$^+$23a]	2 [ABI$^+$23a]

Fig. 2. The best-known upper and lower bounds on the share size for computational secret-sharing schemes for k-slice and $(n-k)$-slices based on OWF, for $k \leq n/2$. The lower bound [ABI+23a] does not require the OWF assumption; if we allow public information only a weaker bound of $\Omega(\log \log n)$ is proved in [ABI+23a].

Applications for Multislices. As applications of our perfect schemes for high slices, we present two schemes for $(a:b)$-multislices. The first one, optimized for

[5] As implied by [LS20, ABI+23b], this running time is necessary for every CSSS for $(n-k)$-slices.

the case where a and b are linear in n, solves an open question raised in [AN21], and has implications on the share size of general access structures. The second scheme is optimized for the regime where $a = n - k$ and $b = n$. The first scheme allows us to prove the following theorem:

Theorem 1.5 (Share Size of General Multislices). *For every $1 \leq a \leq b \leq n$, every $(a : b)$-multislice can be realized by a secret-sharing scheme with share size $\binom{n-a}{\geq n-b} \cdot 2^{o(n)}$.*

This scheme closes the duality gap for multislices in the relevant regime. I.e., if we combine our scheme with that of [AN21] the share sizes of $(a : b)$-multislices and of their dual $(n - b : n - a, n)$-multislices are equal up to sub-exponential factors in n. We also prove the following theorem based on our second scheme for multislices, which is taylor-made for $(n - k : n)$-multislices for small k's.

Theorem 1.6 (Share Size of $(n - k : n)$-Multislices). *For every $k < \log n(\log\log n)^2$, every $(n - k : n)$-multislice can be realized by a secret-sharing scheme with share size $k^{5k} n 2^{\tilde{O}(\sqrt{k \log n})}$. For every $\log n(\log\log n)^2 \leq k \leq n/\log^2 n$, every $(n - k : n)$-multislice can be realized by a secret-sharing scheme with share size $2^{O(k)}$.*

For example, the share size for constant k's in this scheme is $n^{1+o(1)}$, and for $k = \log n$ it is $n^{O(\log\log n)}$, similarly to the $(n - k)$-slice schemes. When $\log n \leq k \leq \log n(\log\log n)^2$ the share size is $k^{5k+o(k)}$ and for larger values of k, the share size is $2^{O(k)}$. We note that our second construction also works for $(n - k)$-hypergraph functions, which are a specific subclass of $(n - k : n)$-multislices where the minterms are all of size $(n - k)$. I.e., a k-hypergraph acts the same as slices for inputs with weight $\leq k$, but outputs 1 on a heavier input y only if there exists a 1-input x of weight k such that $y \geq x$.[6] Hypergraph access structures were studied, e.g., in [AN21, Bei23]. We also present computational, linear, and multi-linear secret-sharing schemes for $(n - k : n)$-multislices (see Theorems 5.9 and 5.10).

Applications for Random Hypergraph Access Structures. Applebaum and Nir [AN21] studied the share size of "random hypergraphs". They showed that if a k-hypergraph f is chosen by drawing m_k minterms uniformly at random then with high probability the share size of f would be smaller than that of general k-hypergraphs. A result in the same spirit was proved in [BF20a] for small k's. More formally (yet still omitting some technical details), using multislices they proved that for every k and m_k the share size of hypergraphs generated according to the above-mentioned procedure is $\sqrt{\binom{n}{k}} \cdot 2^{o(n)}$ with probability $1 - 2^{-\Omega(n)}$. The hardest random k-hypergraph in this case is when $k = n/2$ with shares of size $2^{n/2+o(n)}$. Applebaum and Nir also showed that balancing existing schemes for multislices with respect to duality (i.e., proving Theorem

[6] We say that $y \geq x$ if in every coordinate $y_i \geq x_i$.

1.5), would further improve this result. Hence, we prove the following corollary. We only state the improvement for the hardest random hypergraph, and refer the reader to [AN21, Theorem 6.2] for the general expression for every k which is somewhat involved.

Corollary 1.7 (Schemes for Random Hypergraphs). *For every $k \in [n]$, $m_k \leq \binom{n}{k}$, if a k-hypergraph is chosen by drawing m_k minterms of size k uniformly at random, then it can be realized with share size $2^{0.491n+o(n)}$ with probability $1 - 2^{-\Omega(n)}$.*

We note that general access structures can be easily realized given schemes for k-hypergraphs for $1 \leq k \leq n$; so this result may give hope for obtaining better schemes for general access structures with share size below $2^{0.5n}$.

Due to the space restrictions of this publication, we omit some results, proofs, and comments that are available in full version of this work [BFLN24].

1.3 Our Techniques

Perfect Schemes for High Slices and Duality. Our first scheme for $(n-k)$-slices relies on existing schemes for k-slices. The description below provides correctness and security for sets of size exactly $n-k$; correctness and security for sets of sizes below or above $n-k$ can be easily achieved with additional threshold schemes. To realize an $(n-k)$-slice access structure f, we start by generating shares to a k-slice function $\overline{f}$ determined by f, defined as $\overline{f}(x) = f(\overline{x})$ for every x of weight k. Then, each share sh_i of $\overline{f}$ is distributed with an $(n-k)$-out-of-$(n-1)$ threshold scheme among all of the parties except for the i-th one. Following this, let A be set of size $(n-k)$ whose characteristic vector x satisfies $f(x) = 1$, i.e., $A = \{P_i : x_i = 1\}$. For each i such that $\overline{x}_i = 1$, i.e., $x_i = 0$, $P_i \notin A$ and the $n-k$ parties of A can recover sh_i. For P_i such that $P_i \in A$, the parties only hold $n-k-1$ shares of an $(n-k)$-threshold scheme, and thus will learn nothing about sh_i. In total, the parties of A can recover k shares of $\overline{f}$ that correspond to the coalition $\overline{A} = \{P_i : x_i = 0\}$, and since $\overline{f}(\overline{x}) = f(x) = 1$ this suffices to recover the secret. Similarly, unauthorized sets will recover $\overline{f}$-shares of unauthorized sets under f, and thus will learn nothing about the secret. This construction uses a "trick" introduced by Berkowitz [Ber82] of replacing a negated variable x_i with a threshold gate over all variables but the i'th one. Berkowitz used this idea to construct monotone formulas for k-slices from non-monotone formulas for k-slices, and Beimel, Kushilevitz and Nissim [BKN18] used it to construct secret-sharing schemes for slices from CDS protocols

We now return to the open problem discussed earlier: Can we bound the gap between the share sizes of slices and their duals? It is evident that our scheme solves this problem. Given better schemes for k-slices, we would be able to plug them into our construction and immediately get a better scheme for $(n-k)$-slices. This proves Theorem 1.3 stated above. Following this theorem, we make explicit some properties of our construction that we think may find future use by defining *duality compilers*:

Definition 1.8 (Duality Compilers). *Let $\mathcal{F}$ be a family of n-variable functions and let $\mathcal{F}^* \stackrel{def}{=} \{f^* : f \in \mathcal{F}\}$ (where f^* is the dual of f). A duality compiler for $\mathcal{F}$ is a transformation that takes as input secret-sharing schemes with share size $c_{\mathcal{F}}(n)$ for every function in a family $\mathcal{F}$ and a function $f^* \in \mathcal{F}^*$ and outputs a secret-sharing scheme realizing f^* with share size $c_{\mathcal{F}^*}(n)$. The goal in designing such compilers is to have a small blow-up ratio $c_{\mathcal{F}^*}(n)/c_{\mathcal{F}}(n)$.*

This definition expands the standard viewpoint on secret-sharing duality. Instead of examining specific functions and their duals, it shifts the focus to families of functions and their duals, and enables to draw new conclusions. We stress that in order to realize the function f, such duality compilers do not have to use a scheme for f^*, and they may, for example, use a scheme for a different function $f' \in \mathcal{F}$ or for a set of functions $S \subseteq \mathcal{F}$. Our construction provides a duality compiler for $(n-k)$-slices to their duals, i.e., k-slices, with blow-up n (where for every $(n-k)$-slice f, the construction uses a secret-sharing scheme for the k-slice $\overline{f}$).

Perfect Schemes for High Slices via CDS. Our second scheme for $(n-k)$-slices is based on *conditional disclosure of secrets* (CDS) protocols [GIKM00]. In a CDS protocol, there are k servers $\mathcal{S}_1, \ldots, \mathcal{S}_k$, each holding a private input x_i, the secret s, and a common random string r, and a referee is holding the inputs $x_1, \ldots, x_k$. Each server computes a message as a function of its input x_i, the secret s, and the common random string r (the message of each server is independent of the other inputs and is computed without seeing the other messages). Each server sends its message to the referee. We say that the CDS protocol realizes a function g if the referee can reconstruct s (from the k messages and the k inputs) if and only if $g(x_1, \ldots, x_k) = 1$.

Given an $(n-k)$-slice, we will use a CDS protocol for a function $g_f : [n]^k \to \{0,1\}$ that encodes the way f behaves on inputs with weight $n-k$: On an input $(i_1, \ldots, i_k)$ the function g_f outputs the same as f when given an input with 0's in the indices $(i_1, \ldots, i_k)$ and 1's in all other indices. For example, on $i_1 = 2$, $i_2 = 4$ and $n = 5$ we define $g_f(2,4) = f(10101)$. Then, our goal is to distribute CDS messages generated according to g_f in a way that for every input x of weight $n-k$ the set of parties $A = \{P_i : x_i = 1\}$ will be able to recover k CDS messages, one from each server, that correspond to the input $\overline{x}$. I.e., in the previous example, $\{P_1, P_3, P_5\}$ should be able to reconstruct the message of the first server on input $i_1 = 2$ and the message of the second server on input $i_2 = 4$. Keeping the scheme based on slices in mind, a natural approach to do so would be to share every CDS message of the j-th server with the input i with an $(n-k)$-out-of-$(n-1)$ threshold scheme to all parties but the i-th one. However, this time a set of $n-k$ parties will be able to recover k messages of every CDS server, and in this case, the protocol does not guarantee any privacy. We will solve this issue by sharing the CDS messages in a more sophisticated way, inspired by the scheme for slices of [ABF+19]. See examples and more technical details in Sect. 3.

Computational Schemes for High Slices. The starting point of our computational schemes for $(n-k)$-slices is to take the previously described perfect scheme based on CDS protocols and plug into it the computational CDS protocol of [ABI+23b]. The share size in this implementation would be $O(nk\lambda \log n)$. While this is better than existing perfect schemes for high slices, we still need to save a multiplicative factor of n/k to prove Theorem 1.4. To do so, we notice that most of the shares dealt in the CDS-based scheme are of Shamir's threshold secret-sharing schemes with high thresholds. In an $(n'-t)$-out-of-n' Shamir scheme, $(n'-t-1)$ shares are independent random strings. Instead of dealing these random strings directly to the parties, we give each party only a (shorter) seed of a PRG, and the party generates its share from its seed. Our observation is that the same seed can be used for all schemes, which allows for further savings in the share size. In Shamir's scheme, $t+1$ of the shares are correlated with previous shares and we need to give them explicitly to the parties; with careful load balancing we can still get small shares as desired. For the full technical details, see Sect. 4.

Schemes for Multislices. Existing schemes for $(a:b)$-multislices are better when a and b are small. The scheme in [AN21] is aimed for the case where $a=\alpha n$, $b=\beta n$ for constants α, β, and it has smaller shares when these constants are small. A scheme in [BF20b] implicitly realizes $(0:k)$-multislices, and has huge shares for their dual $(n-k:n)$-multislices. We build schemes that complement the mentioned schemes and equalize the best-known share size for primal and dual multislices.

The constructions in [BF20b, AN21] both rely on formulas for multislices over CDS gates. By simple duality properties of formulas (Lemma 5.1), given such a formula F that computes a function f, if we replace in F every gate that computes a function g with a gate that computes the dual g^* of g, we will get a formula of the same size that computes f^*. Hence, in order to transform the known schemes for low multislices to schemes for high multislices it essentially suffices to realize the duals of CDS gates with small shares. The duals of CDS gates are functions that are somewhat contrived and hard to work with, and the key observation in our scheme is that the duals of k-server CDS gates can be replaced by $(n-k)$-slices. Hence, if we use our schemes for high slices we can realize high multislices with the same share size as low ones, employing the standard formula-to-scheme transformation (see Lemma 5.2 for a formal version).

1.4 Open Questions

Better Perfect Schemes for General Access Structures. In this work, we construct schemes for random hypergraphs. The obvious question to ask is whether these ideas (and shares of size $2^{\alpha n}$ for $\alpha < 1/2$) can be extended to schemes for worst-case hypergraphs, and from there to general access structures. The other side of this coin would be that random hypergraphs are easier for secret sharing than worst case ones.

Better Computational Schemes for Multislices from OWFs. When we construct $(n-k:n)$-multislices from $(n-k)$-slices, we follow a black-box transformation that is analogous to the construction of Robust CDS protocols (a generalization of CDS protocols defined in [ABNP20]) that adds a multiplicative factor of $k^{O(k)}$ to the share size. An improvement of this technique would lead to a reduction of the share size for multislices, and such an improvement may be easier to obtain taking advantage of one-way functions.

A Candidate for Duality-Separation. The best-known share size for k-slices with a constant $k \geq 2$ is $n^{o(1)}$, while that of their dual $(n-k)$-slices is now $n^{1+o(1)}$. It will be interesting to see whether this gap can be closed, or rather to prove that it is inherent. A possible path towards closing this gap may be to realize the dual of k-server CDS gates directly and more cheaply than our implementations of general $(n-k)$-slices.

Duality-Compilers. Duality-compilers seem like a useful abstraction that may help obtain new bounds for secret sharing for families of functions. A natural next step would be to describe duality-compilers with a small blow-up for other families of functions. For example, the well-studied family of *graph access structures* where every minimal authorized coalition is of size 2, or its more general version of k-hypergraphs where every minimal authorized set is of size k.

2 Preliminaries

Perfect Secret-Sharing Schemes. We define perfect secret-sharing scheme as given in [CK93,BC94]; in these schemes the security is information theoretic. Secret-sharing schemes with computational security will be defined in Sect. 2. For more information about this definition and secret-sharing in general, see [Bei11]. We start by defining an access structure, which is the collection of sets of parties that are authorized to reconstruct the secret. We describe an access structure by a monotone Boolean function.

Notation on Monotone Boolean Functions. The *weight* of an input $x \in \{0,1\}^n$, denoted $\mathrm{wt}(x)$, is the number of bits in x that are one, i.e., $\mathrm{wt}(x) = |\{i : x_i = 1\}|$. For two strings $x = (x_1, \ldots, x_n), y = (y_1, \ldots, y_n) \in \{0,1\}^n$, we say that $x \leq y$ if $x_i \leq y_i$ for every $1 \leq i \leq n$. A function $f : \{0,1\}^n \to \{0,1\}$ is *monotone* if $x \leq y$ implies $f(x) \leq f(y)$.

We will also consider partially defined functions, where $f(x) = *$ denotes that f is undefined on x. A partially defined function $f : \{0,1\}^n \to \{0,1,*\}$ is monotone if there does not exist $x, y \in \{0,1\}^n$ such that $x \leq y$, $f(x) = 1$, and $f(y) = 0$. A *minterm* of a monotone function f is a minimal input $x \in \{0,1\}^n$ such that $f(x) = 1$, i.e., for every $y \neq x$ if $y \leq x$ then $f(y) \in \{0,*\}$. A *maxterm* of a monotone function f is a maximal input $x \in \{0,1\}^n$ such that $f(x) = 0$, i.e., for every $y \neq x$ if $y \geq x$ then $f(y) \in \{1,*\}$.

Definition 2.1 (Access Structures). *An n-party access structure is a monotone function $f : \{0,1\}^n \to \{0,1,*\}$ such that $f(0^n) \neq 1$. Let $P = \{P_1, \ldots, P_n\}$ be a set of parties; for an input $x = (x_1, \ldots, x_n) \in \{0,1\}$, we define the set of parties that it represents as $I_x = \{P_i : x_i = 1\}$. For every $x \in \{0,1\}^n$, if $f(x) = 1$, then we say that I_x is authorized; if $f(x) = 0$, then we say that I_x is forbidden.*

A secret-sharing scheme is a randomized mapping $\Pi(s;r)$ whose input is a secret and a random string. A dealer distributes a secret $s \in S$ according to Π by first sampling a random string $r \in R$ with uniform distribution, computing a vector of shares $\Pi(s;r) = (\mathsf{sh}_1, \ldots, \mathsf{sh}_n)$, and privately communicating each share sh_j to party P_j. We require that any authorized set of parties can reconstruct the secret from its shares and any forbidden set cannot learn any information on the secret.

Definition 2.2 (Secret-Sharing Schemes). *A* secret-sharing scheme *Π with domain of secrets S, such that $|S| \geq 2$, is a mapping from $S \times R$, where R is some finite set called the set of random strings, to a tuple of n-sets $S_1 \times S_2 \times \cdots \times S_n$, where S_j is called the* domain of shares *of P_j. For an input $x \in \{0,1\}^n$, we denote $\Pi_x(s;r)$ as the restriction of $\Pi(s;r)$ to its I_x-entries, i.e., $(\mathsf{sh}_j)_{j:x_j=1}$.*

A secret-sharing scheme Π with domain of secrets S realizes an access structure $f : \{0,1\}^n \to \{0,1,*\}$ if the following two requirements hold:

Correctness. *For any input $x \in \{0,1\}^n$ such that $f(x) = 1$ there exists a* reconstruction function $\mathrm{Recon}_x : \prod_{\{i:x_i=1\}} S_i \to S$ *such that* $\mathrm{Recon}_x(\Pi_x(s;r)) = s$ *for every secret $s \in S$ and every random string $r \in R$.*

Security. *For any input $x \in \{0,1\}^n$ s.t. $f(x) = 0$ and every pair of secrets $s, s' \in S$, the distributions $\Pi_x(s;r)$ and $\Pi_x(s';r)$ are identical, where the distributions are over the choice of r from R with uniform distribution.*

Given a secret-sharing scheme Π, define the size *of the secret as* $\log|S|$*, the* share size *of party P_j as* $\log|S_j|$*, the* share size *as* $\max_{1\leq j\leq n}\{\log|S_j|\}$*, the* total share size *as* $\sum_{j=1}^{n}\log|S_j|$*, and the* information ratio *as* $\frac{\max_{1\leq j\leq n}\{\log|S_j|\}}{\log|S|}$.

By default, when we talk about the share size of secret-sharing schemes for an access structure, we consider schemes for one-bit secrets. Note that in Definition 2.2, there are no requirements for inputs x for which $f(x)$ is undefined, e.g., the parties in I_x can have partial information on the secret without being able to reconstruct it.

We next define multi-linear and linear secret-sharing schemes, which are schemes in which the mapping that the dealer uses to generate the shares is linear. Many of the known constructions of secret-sharing schemes are linear and multi-linear.

Definition 2.3 (Multi-Linear and Linear Secret-Sharing Schemes). *Let Π be a secret-sharing scheme with domain of secrets S. We say that Π is a* multi-linear *secret-sharing scheme over a finite field $\mathbb{F}$ if there are integers*

$\ell_d, \ell_r, \ell_1, \ldots, \ell_n$ such that $S = \mathbb{F}^{\ell_d}$, $R = \mathbb{F}^{\ell_r}$, $S_1 = \mathbb{F}^{\ell_1}, \ldots, S_n = \mathbb{F}^{\ell_n}$, and the mapping Π is a linear *mapping over $\mathbb{F}$ from $\mathbb{F}^{\ell_d+\ell_r}$ to $\mathbb{F}^{\ell_1+\cdots+\ell_n}$. We say that a scheme is linear over $\mathbb{F}$ if $S = \mathbb{F}$ (i.e., when $\ell_d = 1$).*

Slice, Mutislice, and Hypergraph Access Structures. In this work we construct secret-sharing schemes for slices and for multislice access structures. A k-slice (also called uniform access structure) is an access structure where all sets of size smaller than k are forbidden, all sets of size larger than k are authorized, and sets of size k can be either forbidden or authorized. An $(a : b)$-multislice is an access structure where all sets of size smaller than a are forbidden, all sets of size larger than b are authorized, and sets of size between a and b can be either forbidden or authorized. A k-hypergraph access structure is an access structure whose minimal authorized sets are of size k, and it can have forbidden sets of size much larger than k.

Definition 2.4 (Slices, Multislices, and Hypergraphs). *Let k, n be integers such that $k \leq n$. A (k, n)-*slice *is a function $f : \{0,1\}^n \to \{0,1\}$ such that if $\mathrm{wt}(x) < k$, then $f(x) = 0$ and if $\mathrm{wt}(x) > k$, then $f(x) = 1$. A* partially defined (k, n)-slice *is a function that is defined on all inputs of weight k and is undefined on all other inputs. When n is clear from the context, we write k slice instead of (k, n)-slice. Let a, b, n be integers such that $1 \leq a \leq b \leq n$. An (a, b)-*multislice *is a monotone function $f : \{0,1\}^n \to \{0,1\}$ such that if $\mathrm{wt}(x) < a$, then $f(x) = 0$ and if $\mathrm{wt}(x) > b$, then $f(x) = 1$. A k-*hypergraph access structure *is a function $f : \{0,1\}^n \to \{0,1\}$ such that all its minterms have weight exactly k.*

Note that a k-slice is a (k, k)-multislice and a k-hypergraph is a (k, n)-multislice.

Remark 2.5. To construct a secret-sharing scheme for a (fully-defined) k-slice, it suffices to construct a secret-sharing scheme for the partially defined function $f' : \{0,1\}^n \to \{0, 1, *\}$, where $f'(x) = f(x)$ if $\mathrm{wt}(x) = k$ and $f'(x) = *$ otherwise. Given a secret-sharing scheme Π' realizing f', we construct a secret-sharing scheme Π with secret $s \in \{0,1\}$ realizing f as follows:

1. Share the secret s using a $(k+1)$-out-of-n secret-sharing scheme and give each party one share of this scheme.
2. Choose a random bit r_1 with uniform distribution and compute $r_2 = r_1 \oplus s$.
3. Share r_1 using a k-out-of-n secret-sharing scheme and give each party one share of this scheme.
4. Share r_2 using the secret-sharing scheme Π' and give each party its share of this scheme.

It can be verified that Π realizes the fully-defined k-slice f. The share size in Π is equal to the share size in Π' up to an additive term of $O(\log n)$. Thus, in this paper, we will realize partially defined slices.

Protocols for Conditional Disclosure of Secrets. We next define conditional disclosure of secrets (CDS) protocols, a useful cryptographic primitive introduced in [GIKM00]. In particular, this primitive is used to construct secret-sharing schemes for general access structures, starting in the work of [LV18]. An informal presentation of CDS protocols appears in the introduction.

Definition 2.6 (Conditional Disclosure of Secrets (CDS) Protocols). *A* k-server CDS protocol $\mathcal{P}$*, with domain of secrets* S*, domain of common random strings* R*, and finite message domains* $M_1, \ldots, M_k$*, consists of* k *encoding functions* $\text{ENC}_1, \ldots, \text{ENC}_k$*, where* $\text{ENC}_i : X_i \times S \times R \to M_i$ *for every* $i \in [k]$*. For an input* $x = (x_1, \ldots, x_k) \in X_1 \times \cdots \times X_k$*, secret* $s \in S$*, and randomness* $r \in R$*, we let* $\text{ENC}(x, s; r) = (\text{ENC}_1(x_1, s; r), \ldots, \text{ENC}_k(x_k, s; r))$.

Let $g : X_1 \times \cdots \times X_k \to \{0,1\}$ *be a* k*-input function. We say that* $\mathcal{P}$ *is a CDS protocol for* g *if it satisfies the following properties:*

Correctness. *There is a deterministic reconstruction function* $\text{DEC} : X_1 \times \cdots \times X_k \times M_1 \times \cdots \times M_k \to S$ *such that for every input* $x = (x_1, \ldots, x_k) \in X_1 \times \cdots \times X_k$ *for which* $g(x_1, \ldots, x_k) = 1$*, every secret* $s \in S$*, and every common random string* $r \in R$*, it holds that* $\text{DEC}(x, \text{ENC}(x, s; r)) = s$.

Security. *For every input* $x = (x_1, \ldots, x_k) \in X_1 \times \cdots \times X_k$ *satisfying* $g(x_1, \ldots, x_k) = 0$ *and every pair of secrets* $s, s' \in S$*, the distributions* $\text{ENC}(x, s; r)$ *and* $\text{ENC}(x, s'; r)$ *are equally distributed, where the probability distributions are over the choice of* r *from* R *with uniform distribution.*

The message size *of a CDS protocol* $\mathcal{P}$ *is defined as the size of the largest message sent by the servers, i.e.,* $\max_{1 \le i \le k} \log |M_i|$.

Computational Secret-Sharing Schemes and CDS Protocols. We quote the definition of *computational secret-sharing schemes (CSSS)* from [ABI+23b]. In a $t(\lambda)$-secure CSSS the sharing and reconstruction are efficient, and no adversary running in time $t(\lambda)$ can learn non-negligible information about the secret from the shares of any unauthorized set of parties (where λ is the security parameter). When defining "efficiency" it is important to consider the way the access structure is represented. In this paper, we will mainly represent an access structure as a k-slice function, explicitly describing $f(x)$ for every input x of weight k. Nevertheless, in the definition of CSSS we use the abstract definition of a representation model.

Definition 2.7 (Representation Model [ABI+23b]). *A* representation model *is a polynomial time computable function* $U : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}$*, where* $U(\text{Prog}, x)$ *is referred to as the value returned by a "program"* Prog *on an input* $x \in \{0,1\}^n$*. We assume that each* Prog *specifies the input size* n *and* $|\,\text{Prog}\,| \ge n$*. We say that* Prog *represents the function* $f : \{0,1\}^n \to \{0,1\}$ *in the representation model* U *if* $U(\text{Prog}, x) = f(x)$.

Definition 2.8 (Comp. Secret-Sharing Schemes. (CSSS) [ABI+23b]). *A CSSS for a representation model* U *consists of a pair of algorithms* $\text{CSSS} = (\text{CSSS.SHARE}, \text{CSSS.RECON})$ *with the following syntax.*

Sharing. $\text{CSSS.}\textsc{Share}(1^\lambda, \text{Prog}, s) \to (\mathsf{sh}_1, \ldots, \mathsf{sh}_n)$ *(where n denotes the input length of* Prog*) is a randomized poly-time algorithm that takes as input a security parameter λ, a program* Prog*, and a secret $s \in \{0,1\}$; it outputs n shares $\mathsf{sh}_1, \ldots, \mathsf{sh}_n$, where sh_i, for $1 \le i \le n$, is the share of party P_i.* [7]

Reconstruction. $\text{CSSS.}\textsc{Recon}(\text{Prog}, x, (\mathsf{sh}_i)_{i:x_i=1}) \to s$ *is a deterministic poly-time algorithm that takes as input a program* Prog*, an input $x \in \{0,1\}^n$ (where n denotes the input size of* Prog*), and shares of the parties in $I_x = \{P_i : x_i = 1\}$. The algorithm outputs a secret $s \in \{0,1\}$.*

We say that CSSS *is correct (with respect to U) if for every λ, s, program* Prog*, and input $x \in \{0,1\}^n$ such that $U(\text{Prog}, x) = 1$ (where n denotes the input length of* Prog*), the process of invoking $\text{CSSS.}\textsc{Share}(1^\lambda, \text{Prog}, s) \to (\mathsf{sh}_1, \ldots, \mathsf{sh}_n)$ and then invoking $\text{CSSS.}\textsc{Recon}(\text{Prog}, x, (\mathsf{sh}_i)_{i:x_i=1})$ always returns s.*

To define the security of CSSS *we consider the following game between a non-uniform $t(\lambda)$-time adversary $\mathcal{A}$ and a challenger:*

1. *The adversary $\mathcal{A}$ on input 1^λ chooses* Prog *and an input $x \in \{0,1\}^n$ such that $U(\text{Prog}, x) = 0$ (where n is the input size of* Prog*) and sends them to the challenger.*
2. *The challenger chooses a secret $s \leftarrow_U \{0,1\}$ uniformly at random. It computes $(\mathsf{sh}_1, \ldots, \mathsf{sh}_n) \leftarrow \text{CSSS.}\textsc{Share}(1^\lambda, \text{Prog}, s)$ and sends $(\mathsf{sh}_i)_{x_i=1}$ to the adversary.*
3. *The adversary outputs a bit s'.*

The adversary wins the game if $s' = s$. We say that CSSS *is $t(\lambda)$-secure if for every non-uniform $t(\lambda)$-time adversary $\mathcal{A}$ and sufficiently large λ, the probability that $\mathcal{A}$ wins is at most $1/2 + 1/t(\lambda)$. By default, we require $t(\lambda)$-security for every polynomial $t(\cdot)$. In any case, we always assume that $1/t(\lambda)$ is negligible.*[8]

A computational CDS (CCDS) protocol is defined similarly to CSSS. We refer the reader to the full version of this work [BFLN24] for details. We use the following result.

Theorem 2.9 ([ABI+23b]). *Assuming $t(\lambda)$-secure one-way functions exist, for all k-input functions $g : (\{0,1\}^\ell)^k \to \{0,1\}$, represented by truth tables of size $N = 2^{\ell k}$, there exists a $\text{poly}(t(\lambda))$-secure CCDS protocol with message size $O(\lambda\ell)$. The running time of the encoding and decoding algorithms is $O(2^{\ell k}\lambda)$.*

[7] In [ABI+23b], the scheme also returns public information sh_0 given to all parties (or published in the cloud); in this work we do not use this public information.

[8] A function $\varepsilon(\lambda)$ is negligible if for every positive polynomial (λ) there exits λ_0 such that $\varepsilon(\lambda) \le 1/p(\lambda)$ for every $\lambda > \lambda_0$. Our results remain valid also when $t(\lambda) \ge \lambda$, as in [ABI+23b]; for simplicity of our notations we prefer to only consider negligible functions.

3 Perfect Secret-Sharing Schemes for $(n-k)$-Slices

We provide two new constructions of perfect secret-sharing schemes for $(n-k)$-slice functions. The first one is based on schemes for k-slices, and the second one on k-server CDS protocols. By the current state of the art of CDS protocols and secret-sharing schemes for slices, the second scheme is more efficient by a factor of n. For small k's it has shares of size $kn^{1+o(1)}$, compared to shares of size $kn^{2+o(1)}$ for the first simpler scheme. However, if more efficient schemes for slices will be constructed, the first scheme may become the leading one.

3.1 Construction from Schemes for k-Slices

In this section, we prove Theorem 3.1, which is a reformulation of Theorem 1.3. We describe a simple scheme for $(n-k)$-slices based on a scheme for k-slices. Specifically, we will show how to realize an $(n-k)$-slice f given a scheme for the partially-defined k-slice $\overline{f}$, where $\overline{f}(x) = f(\overline{x})$ for every input x of weight k and undefined for other inputs.

The secret: An element $s \in \{0,1\}$.
The scheme:

1. Share the secret s with a secret-sharing scheme $\overline{\Pi}$ realizing $\overline{f}$; denote by $\mathsf{sh}_1, \ldots, \mathsf{sh}_n$ the resulting shares.
2. For every $t \in [n]$, share sh_t with Shamir's $(n-k)$-out-of-$(n-1)$ secret-sharing scheme, and distribute one share to each party in $\{P_1, \ldots, P_n\} \setminus \{P_t\}$.

The share of each P_j: A share of each sh_t for every $t \neq j$.

Fig. 3. A secret-sharing scheme realizing a partially defined $(n-k)$-slice f using a scheme $\overline{\Pi}$ for the partially defined k-slice $\overline{f}$.

Theorem 3.1. *Let $f : \{0,1\}^n \to$ bit be an $(n-k)$-slice. If there is a secret-sharing scheme for the partially defined k-slice $\overline{f}$ with share size $c_{\text{slice}}(k,n)$, then there is a secret-sharing scheme realizing the slice f with share size $O(n \cdot \max\{\log n, c_{\text{slice}}(k,n)\})$.*

Proof. By Remark 2.5, it suffices to realize partially-defined slice functions, only defined on inputs of weight $n-k$. The scheme for such a function f is described in Fig. 3. We next prove the correctness and security of the scheme, only considering inputs of weight $n-k$.

For correctness, if $f(x) = 1$ then by definition $\overline{f}(\overline{x}) = f(x) = 1$, and hence the shares $\{\mathsf{sh}_i : \overline{x}_i = 1\} = \{\mathsf{sh}_i : x_i = 0\}$ of $\overline{\Pi}$ reveal the secret. For every i such that $x_i = 0$, the parties of I_x can compute every sh_i generated in Step 1 by combining their corresponding $n-k$ shares in the threshold sharing of sh_i dealt in Step 2 (since $P_i \notin I_x$), and thus can recover the secret s.

For security, if $f(x) = 0$ then by definition $\overline{f}(\overline{x}) = f(x) = 0$. For every i such that $x_i = 1$, the parties in I_x only hold the shares of $I_x \setminus \{P_i\}$, i.e., they hold $n-k-1$ shares in a secret-sharing scheme with threshold $n-k$ and these shares are uniformly distributed. Thus, the parties in I_x can only obtain the shares $\{\mathsf{sh}_i : x_i = 0\} = \{\mathsf{sh}_i : \overline{x}_i = 1\}$ of $\overline{\Pi}$ from the shares generated in Step 1 for the access structure $\overline{f}$. These are shares of the set $I_{\overline{x}}$ which is an unauthorized set of $\overline{f}$; therefore they reveal no information on s.

The share of each party consists of $n-1$ shares of shares of $\overline{\Pi}$; each share in a Shamir threshold scheme has size $O(\max\{\log n, c_{\text{slice}}(n,k)\}$. □

Using the k-slice secret-sharing scheme of [ABF+19], which has share size $kn \cdot 2^{\tilde{O}(\sqrt{k \log n})}$, in Theorem 3.1, results in a scheme for $(n-k)$-slices with share size $n^2 k \cdot 2^{\tilde{O}(\sqrt{k \log n})}$. In the following section we prove Theorem 3.5 by presenting a better scheme.

3.2 Construction from k-Server CDS Protocols

We now preset the second construction for $(n-k)$-slices. The structure of this construction is similar to the construction from [ABF+19] of secret-sharing schemes for k-slices from k-server CDS protocols. For that, we need to define the following functions; in these functions we encodes an input of weight $(n-k)$ by the k indices in which the input is 0.

Definition 3.2 (The Function g_f). *Let $f : \{0,1\}^n \to \{0,1\}$ be an $(n-k)$-slice. For a sequence $j_1, \ldots, j_k$ of k distinct numbers in $[n]$ we define an input $X^{j_1,\ldots,j_k} = (x_1, \ldots, x_n)$ as $x_{j_1} = x_{j_2} = \cdots = x_{j_k} = 0$ and all other bits of x are 1; the weight of $X^{j_1,\ldots,j_k}$ is exactly $n-k$. We define the k-input function $g_f : [n]^k \to \{0,1\}$, where $g_f(j_1, \ldots, j_k) = 1$ if and only if $1 \le j_1 < \cdots < j_k \le n$ and $f(X^{j_1,\ldots,j_k}) = 1$.*

Example 3.3. For the sequence $(2,3)$, the input $X^{2,3}$ is 1001^{n-3}. Let f be the $(n-2)$-slice function, where $f(x) = 1$ for an input $x = (x_1, \ldots, x_n)$ of weight $n-2$ if and only if there is an index $1 \le j \le n-1$ such that $x_j = x_{j+1} = 0$. In this case $g_f(j_1, j_2) = 1$ if and only if $j_2 = j_1 + 1$. E.g., for $n = 5$,

$$g_f(2,3) = f(X^{2,3}) = f(10011) = 1 \text{ and } g_f(2,4) = f(X^{2,4}) = f(10101) = 0.$$

In Fig. 4, we describe the secret-sharing scheme realizing a partially defined $(n-k)$-slice f using a k-server CDS protocol for g_f. We next describe the ideas of the scheme, considering an $(n-2)$-slice f. We execute a 2-server CDS protocol for g_f; let $m_{i,j}$ be the message of server $\mathcal{S}_i$ in the 2-server CDS protocol with input $j \in [n]$. Consider an input x of weight $n-2$ such that $f(x) = 1$ and let $j_1 < j_2$ be the indices such that $x_{j_1} = x_{j_2} = 0$, i.e., $x = X^{j_1,j_2}$. Thus, $g_f(j_1, j_2) = 1$ and the secret can be reconstructed from m_{1,j_1}, m_{2,j_2}. We can try and apply the same strategy as in the scheme described in Fig. 3, that is, sharing each message $m_{i,j}$ in an $(n-2)$-out-of-$(n-1)$ secret-sharing scheme and give the shares to all parties except for P_j. In this case, the parties in I_x can reconstruct m_{1,j_1}, m_{2,j_2}

and reconstruct the secret. However, when $f(x = X^{j_1,j_2}) = 0$ the parties in I_x can also reconstruct m_{1,j_2}, m_{2,j_1} (as, for example, the shares of m_{1,j_2} are given to all parties except for P_{j_2}). In this case, the parties in I_x can reconstruct two messages of the first server and there are no security guarantees from the CDS protocol.[9]

We need to ensure that the parties in I_x can only reconstruct the message $m_{1,j}$, where j is the smallest index such that $x_j = 0$. In this case $x_1 = \cdots = x_{j-1} = 1$ and all the $n - j$ bits $x_{j+1}, \ldots, x_n$ are 1 except for exactly one bit. Thus, for every j we share $m_{1,j}$ in a 2-out-of-2 secret-sharing scheme. We share the first share in a $(j-1)$-out-of-$(j-1)$ secret-sharing scheme and give the shares to the first $j-1$ parties. Similarly, we share the second share in a $(n-j-1)$-out-of-$(n-j)$ secret-sharing scheme and give the shares to the last $n-j$ parties. We treat $m_{2,j}$ symmetrically. Some technical details arise in the first and last indices. For example, $j_2 \geq 1$, so we do not need to share $m_{2,1}$. As another example, if $j_1 = 1$, there are no parties with index smaller than 1 and we share $m_{1,1}$ in a $(n-k)$-out-of-$(n-1)$ scheme (without sharing it in a 2-out-of-2 scheme).

The scheme for $(n-k)$-slice functions generalizes this idea, where each message $m_{i,j}$ is shared using a 2-out-of-2 secret-sharing scheme, the first and second shares are shared among the first $j-1$ parties and last $n-j$ parties respectively with appropriate thresholds.

The secret: An element $s \in \{0,1\}$.
Auxiliary protocol: Let $\mathcal{P}$ be a k-server CDS protocol with message size $c = c_{\text{cds}}(k,n)$ for the function $g_f : [n]^k \to \{0,1\}$ (defined in Definition 3.2).

The scheme:

1. Choose a common random string r as chosen in the k-server CDS protocol $\mathcal{P}$ and execute $\mathcal{P}$ with the secret s and the common random string r. For every $i \in [k], j \in [n]$, let $m_{i,j} \in \{0,1\}^c$ be the message of server $\mathcal{S}_i$ in $\mathcal{P}$ with input $j \in [n]$, i.e., $m_{i,j} = \text{Enc}_i(j, s; r)$.
2. For $i+1 \leq j \leq n-k+i-1$, share $m_{i,j}$ in a 2-out-of-2 secret-sharing scheme, that is, choose $m^1_{i,j}$ with uniform distribution from $\{0,1\}^c$ and compute $m^2_{i,j} = m_{i,j} \oplus m^1_{i,j}$.
3. Let $m^2_{i,i} = m_{i,i}$ and $m^1_{i,n-k+i} = m_{i,n-k+i}$ for every $i \in [k]$.
4. For every $i \in [k]$ and $i+1 \leq j \leq n-k+i$, share the string $m^1_{i,j}$ using the $(j-i)$-out-of-$(j-1)$ secret-sharing scheme among the first $j-1$ parties.
5. For every $i \in [k]$ and every $i \leq j \leq n-k+i-1$, share the string $m^2_{i,j}$ using an $(n-k-j+i)$-out-of-$(n-j)$ secret-sharing scheme among the last $n-j$ parties.

Fig. 4. A secret-sharing scheme realizing a partially defined $(n-k)$-slice f using a k-server CDS protocol $\mathcal{P}$ for g_f.

[9] This problem can be solved by using robust CDS protocols (as defined in [ABNP20]); however, the known robust CDS protocols have very large message size. We use an idea of [ABF+19] to solve this problem.

Lemma 3.4. *Let $f : \{0,1\}^n \to \{0,1\}$ be an $(n-k)$-slice. If there is a k-server CDS protocol for $g_f : [n]^k \to \{0,1\}$ with message size $c_{\mathrm{cds}}(k,n)$, then there is a secret-sharing scheme realizing f with share size $O(k \cdot n \cdot \max\{\log n, c_{\mathrm{cds}}(k,n)\})$.*

Proof. By Remark 2.5, it suffices to realize partially-defined slice functions, only defined on inputs of weight $n-k$. The scheme for such a function f is described in Fig. 4. We next prove the correctness security of the scheme, only considering inputs of weight $n-k$. That is, we consider an input x such that $x_{j_i} = \cdots = x_{j_k} = 0$ for some indices $1 \le j_i \le \cdots \le j_k \le n$ and all other bits in x are 1, i.e., $x = X^{j_i,\cdots,j_k}$.

Assume that $f(x) = 1$. We next explain how the parties in I_x can recover m_{i,j_i} for every $i \in [k]$. First observe that $i \le j_i \le n-k+i$ (since there are $i-1$ bits in x that are zero before x_{j_i} and there are $k-i$ bits that are zero after x_{j_i}). If $i = j_i$, then $x_1 = x_2 = \cdots = x_i = 0$ and I_x has $(n-i)-(k-i) = n-k$ parties with index greater than $j_i = i$, i.e., the parties in I_x can recover $m^2_{i,j_i} = m_{i,j_i}$, which is shared via a secret-sharing scheme with threshold $n-k-j_i+i = n-k$. Analogously, if $j_i = n-k+i$, then $x_{n-k+i} = x_{n-k+i+1} = \cdots = x_n = 0$ and I_x has $(j_i-1)-(i-1) = n-k+i-1-i+1 = n-k$ parties with index smaller than $j_i = n-k+i$, i.e., the parties in I_x can recover $m^1_{i,j_i} = m_{i,j_i}$. Now consider the case that $i+1 \le j_i \le n-k+i-1$. The subset I_x has $j_i - i$ parties with index smaller than j_i and $n-j_i-(k-i)$ parties with index greater than j_i. So the subset I_x can recover both m^1_{i,j_i} and m^2_{i,j_i} and so can compute m_{i,j_i}. As $g_f(j_1,\ldots,j_k) = 1$ and the parties in I_x can recover the messages $m_{1,j_1},\ldots,m_{k,j_k}$, they can recover the secret.

Assume that $f(x) = 0$. In this case $g_f(j_1,\ldots,j_k) = 0$, hence the parties in I_x cannot obtain any information on s from the messages $m_{i,j_1},\ldots,m_{k,j_k}$ they can recover. We next claim that the parties in I_x have no information on any message $m_{i,j}$, where $j \neq j_i$. Since $j \neq j_i$, the number of bits that are zero in x in the first j bits of x is not i, i.e., either there are at least i bits that are zero among the first $j-1$ bits of x or there are at least $k-i$ bits that are zero among the last $n-j$ bits of x (since exactly k bits of x are zero). In the former case, the parties in I_x hold at most $j-1-i$ shares in a secret-sharing scheme of $m^1_{i,j}$ with threshold $j-i$, hence they have no information on $m^1_{i,j}$, thus, they have no information on $m_{i,j}$. In the latter case, the parties in I_x hold at most $(n-j)-(k-i) = n-k-j$ shares in a secret-sharing scheme of $m^2_{i,j}$ with threshold $n-k-j+1$, hence they have no information on $m^2_{i,j}$, thus, they have no information on $m_{i,j}$. As each $m_{i,j}$ is shared independently, the parties in I_x gain no information from the sharing of all the messages $(m_{i,j})_{i\in[k],j\neq i_j}$. To conclude, the set I_x only obtains the messages $m_{i,j_1},\ldots,m_{k,j_k}$, which by the security of the CDS protocol give no information on the secret.

The share of each party is composed of $O(nk)$ shares in Shamir's threshold secret-sharing schemes with $O(n)$ parties. Thus, the share size is $O(nk \cdot \max\{\log n, c_{\mathrm{cds}}(k,n)\})$. □

Theorem 3.5 (Perfect Schemes for High Slices, Theorem 1.2 Restated). *Let $k \leq n/2$ be positive integers. For every $(n-k)$-slice function f, there exists a secret-sharing scheme realizing f with share size*

$$kn \cdot 2^{O(\sqrt{k \log n}) \log(k \log n)} = kn \cdot 2^{\tilde{O}(\sqrt{k \log n})}.$$

Proof. By [LVW18], there is a k-server CDS protocol for functions $g : [n]^k \to \{0,1\}$ with message size $2^{O(\sqrt{k \log n} \log(k \log n))}$. Using this protocol in Lemma 3.4, we get for every $(n-k)$-slice a secret-sharing scheme with the share size stated in the theorem. □

4 Computationally-Secure Schemes for $(n-k)$-Slices

In this section, we construct CSSSs for $(n-k)$-slices. The first observation is that for $k \leq \sqrt{n}$ we can use the CCDS protocol of [ABI+23b] (see Theorem 2.9) in the scheme of Lemma 3.4 and obtain a secret-sharing scheme for $(n-k)$-slices with share size $O(k\lambda n \log n)$; this is a slight improvement compared to the perfect scheme we constructed. Similarly, for $k \leq \sqrt{n}$ we can obtain a CSSS realizing k-slices with share size $O(\min\{2^k, nk\} \cdot k\lambda \log n)$ by plugging the CCDS protocol of [ABI+23b] in the schemes of [AA18, ABF+19].[10]

Our goal is to save a factor of n/k in the share size of CSSS realizing $(n-k)$-slices compared to the above-mentioned CSSS for $(n-k)$-slices. Recall that in our scheme described in Lemma 3.4, the share of each party contains $O(kn)$ shares in threshold secret-sharing schemes with secrets of size $c_{\text{ccds}}(\lambda, k, n) = O(\lambda \log n)$. We show how to realize these secret-sharing schemes such that the share size of each party in the $O(kn)$ threshold schemes is $O(k^2 \lambda \log n)$. The idea is to give each party a seed of a pseudorandom generator (PRG) that is expanded to a pseudorandom string containing the $O(kn)$ shares. The obstacle is that the n shares are correlated. We use the fact that in the t-out-of-n secret-sharing of Shamir [Sha79] the first $t-1$ shares are uniformly distributed and independent. While the schemes described in the previous paragraph are relatively simple – they take a formula for f and realize some of its gates with computational schemes instead of perfect ones, our scheme for $(n-k)$-slices treats the perfect scheme as "white box", replacing the shares that are random by pseudorandom strings. This methodology is not new; however, the way we utilize it is new.

In Fig. 6, we present the sharing in Shamir's scheme making this fact explicit; in our presentation, the dealer gives a random element to a set A of $t-1$ parties and then picks a polynomial Q that interpolates the $t-1$ shares of A and the secret to generate the shares for the rest of the parties B. Note that in this scheme the polynomial Q is a uniformly distributed polynomial of degree at most $t-1$ such that $Q(0) = s$, that is, the sharing is exactly as in the more common description of Shamir's secret-sharing scheme. This is similar to the systematic

[10] The proofs of the constructions of [AA18, ABF+19] and Lemma 3.4 are when the CDS protocol is perfect; however, they can be updated to the computational setting, similarly to the proof of Claim 4.2.

encoding of Reed-Solomon codes. We use Procedure Interpolate, described in Fig. 5, to compute the polynomial. We will use this procedure in our scheme for $(n-k)$-slices; the above sets A and B will be carefully chosen to minimize the share size of each party.

Procedure Interpolate$(t, A, B, s, (\mathsf{sh}_i)_{P_i \in A})$**:**
Parameters: A threshold t, a set of $t-1$ parties A, a set of parties B disjoint from A, a secret s, and the shares $(\mathsf{sh}_i)_{P_i \in A}$ of the parties in A.

1. The dealer computes the unique polynomial Q of degree at most $t-1$ in $\mathbb{F}_p[x]$ that satisfies $Q(0) = s$ and $Q(i) = \mathsf{sh}_i$ for every $P_i \in A$.
2. The dealer computes $\mathsf{sh}_i = Q(i)$ for every $P_i \in B$ and returns $(\mathsf{sh}_i)_{P_i \in B}$.

Fig. 5. A description of Procedure Interpolate that, given shares of $t-1$ parties and a secret, uses interpolation to find the polynomial that passes via these points and computes the other shares using this polynomial.

The secret: An element $s \in \mathbb{F}_p$, where $p > n$ is a prime.
Public parameters: A set $A \subset P = \{P_1, \ldots, P_n\}$ such that $|A| = t-1$.
The scheme:

1. For every $P_i \in A$, choose sh_i independently with uniform distribution from $\mathbb{F}_p$.
2. Interpolate$(t, A, P \setminus A, s, (\mathsf{sh}_i)_{P_i \in A})$.

The shares: $\mathsf{sh}_1, \ldots, \mathsf{sh}_n$.

Fig. 6. Shamir's t-out-of-n secret-sharing scheme with a systematic choice of the polynomial.

We can change Shamir's scheme as described in Fig. 6, giving each party in A an independent seed w_i of a pseudorandom generator (PRG), and for $P_i \in A$, the party P_i and the dealer compute $\mathsf{sh}_i = \mathrm{PRG}(w_i)$. The dealer also computes shares $(\mathsf{sh}_i)_{P_i \in B}$ from $(\mathsf{sh}_i)_{P_i \in A}$ and the secret (using Procedure Interpolate) and gives these shares to the parties in B. The total share size in this scheme is $O(t\lambda + (n-t)\log p)$ (where Shamir's scheme is executed over $\mathbb{F}_p$ for a prime $p > n$).[11] In a single execution of this scheme, using PRGs reduces the total share size when $n - t$ is small and the length of the secret is bigger than the security parameter (i.e., it avoids the lower bound of the length of the secret

[11] The same results also hold for $\mathbb{F}_q$, where $q > n$ is a prime power. For the sake of simplicity, we restrict the presentation to the case that q is a prime number.

that holds for perfect secret-sharing schemes [KGH83]). Note that Krawczyk's construction [Kra94] gives a smaller share size for this case.[12]

When using the scheme many times with different secrets, the saving is more dramatic – the dealer can give each of the first $t-1$ parties only one seed w_i, which will be expanded to the shares of the party in all the schemes, that is, the share size of these parties is $O(\lambda)$. This can be done even when the thresholds in the various secret-sharing schemes are not the same, and for different sets A. Specifically, in our secret-sharing scheme for $(n-k)$-slices we execute $O(nk)$ threshold t-out-of-n' secret-sharing schemes with various t, n' such that $n'-t \leq k$ and $\log p = O(\lambda \log n)$ (this is the length of the messages in the CCDS protocol, which we need to share with Shamir's scheme). The total share size in these executions is $O(n\lambda + k^2 \lambda n \log n)$, i.e., one seed per party and k "extra" shares for each of the $O(nk)$ executions of the threshold scheme, each "extra" share is of length $O(\lambda \log n)$. We can balance the share size in these $O(nk)$ executions by giving each of the n parties "extra" shares only in $O(k^2)$ schemes; this results in share size $O(k^2 \lambda \log n)$ per party.

The following Theorem 4.1 is a formal statement of Theorem 1.4 from the introduction; in the formal statement of the theorem we deal with the family of slice functions rather than a specific function (as required by Definition 2.8 – the definition of CSSS).

Theorem 4.1. *Let $k : \mathbb{N} \to \mathbb{N}$ be a function such that $2 \leq k(n) \leq \sqrt{n}$ for every $n \in \mathbb{N}$. If $t(\lambda)$-secure one-way functions exist for some negligible function $1/t(\lambda)$, then there exists a $\mathrm{poly}(t(\lambda))$-secure CSSS for $(n-k(n))$-slice functions represented as a truth table of size $\binom{n}{k(n)}$ with share size $O(k^2 \lambda \log n)$. The running time of the sharing and reconstruction algorithms of the CSSS is $\tilde{O}(n^k) \cdot \mathrm{poly}(\lambda) = \mathrm{poly}(\binom{n}{k(n)}, \lambda)$.*

Proof. By Remark 2.5, it suffices to realize partially-defined slice functions, only defined on inputs of weight $n-k$. The CSSS for such a function f is described in Fig. 7.

We first elaborate on the assumptions used in the scheme and prove that the running time of the sharing algorithm is polynomial in the size of the representation of the slice function and the security parameter (as required in Definition 2.8). In the scheme, we assume that the PRG is $\mathrm{poly}(t(\lambda))$-secure. By [HILL99], such PRG can be built from a $t(\lambda)$-secure one-way function and its running time for a pseudorandom string of length knc is $knc \cdot \mathrm{poly}(\lambda)$. Furthermore, if we use the CCDS protocol of [ABI+23b], the message size is $c = O(\lambda \log n)$ and the running time of the encoding algorithm (for computing the nk messages) is $n^k \cdot \mathrm{poly}(\lambda)$ (see Theorem 2.9). Finally, interpolation can be implemented using $\tilde{O}(n)$ arithmetic operations over a field $\mathbb{F}_p$ (using FFT), where $\log p \approx c = O(\lambda \log n)$; each arithmetic operation can be performed in time $\tilde{O}(\log p) = \tilde{O}(\lambda \log n)$ (using Schönhage-Strassen multiplication algorithm). As

[12] We cannot use Krawczyk's construction as we have a few schemes with different thresholds and different sets of parties.

Input: A secret $s \in \{0,1\}$ and an $(n-k)$-slice access structure f described by a truth table of size $\binom{n}{k}$.

Auxiliary tools:

- A k-server CCDS protocol CCDS $= (\text{CCDS.Enc}, \text{CCDS.Dec})$ with message size $c = c_{\text{ccds}}(\lambda, k, n)$ for $g_f : [n]^k \to \{0,1\}$ (defined in Definition 3.2).
- A pseudorandom generator PRG that gets as an input a seed w of length λ and outputs a string of length knc; denote $\text{PRG}(w) = (\text{PRG}_{i,j}(w))_{i\in[k],j\in[n]}$, where $\text{PRG}_{i,j}(w) \in \{0,1\}^c$.
- Procedure Interpolate from Fig. 5 over a field $\mathbb{F}_p$, where $\log p \approx c$ is the message size of the CCDS protocol.

The scheme:

1. For every $\ell \in [n]$, sample with uniform distribution a seed $w_\ell \in \{0,1\}^\lambda$ for the party P_ℓ and let $y_\ell^{i,j} \leftarrow \text{PRG}_{i,j}(w_\ell)$ for every $i \in [k], j \in [n]$.
2. Choose a common random string r as chosen in the k-server CDS protocol CCDS and execute CCDS with the secret s and the common random string r. For every $i \in [k], j \in [n]$, let $m_{i,j} \in \{0,1\}^c$ be the message $m_{i,j} \leftarrow \text{CCDS.Enc}(1^\lambda, g_f, i, x_i = j, s; r)$.
3. For every $i \in [k]$ and $i+1 \le j \le n-k+i-1$, share $m_{i,j}$ in a 2-out-of-2 secret-sharing scheme, that is, choose $m_{i,j}^1$ with uniform distribution from $\{0,1\}^c$ and compute $m_{i,j}^2 = m_{i,j} \oplus m_{i,j}^1$.
4. For every $i \in [k]$, let $m_{i,i}^2 = m_{i,i}$ and $m_{i,n-k+i}^1 = m_{i,n-k+i}$.
5. For every $i \in [k]$ and $i+1 \le j \le n-k+i$, share the string $m_{i,j}^1$ with a $(j-i)$-out-of-$(j-1)$ secret-sharing scheme among the first $j-1$ parties [a]: Let $A = \{P_1, \ldots, P_{j-i-1}\}, B = \{P_{j-i}, \ldots, P_{j-1}\}$, $\mathsf{sh}_\ell^{i,j,1} \leftarrow y_\ell^{i,j}$ for $1 \le \ell \le j-i-1$, $t = j-i$, and
$$\left(\mathsf{sh}_\ell^{i,j,1}\right)_{\ell=j-i}^{j-1} \leftarrow \text{Interpolate}\left(t, A, B, s = m_{i,j}^1, (\mathsf{sh}_\ell^{i,j,1})_{\ell=1}^{j-i-1}\right).$$
6. For every $i \in [k]$ and every $i \le j \le n-k+i-1$, share the string $m_{i,j}^2$ with an $(n-k-j+i)$-out-of-$(n-j)$ secret-sharing scheme among the last $n-j$ parties: Let $A = \{P_{j+k-i+2}, \ldots, P_n\}, B = \{P_{j+1}, \ldots, P_{j+k-i+1}\}$, $\mathsf{sh}_\ell^{i,j,2} \leftarrow y_\ell^{i,j}$ for $j+k-i+2 \le \ell \le n$, $t = n-k-j+1$, and
$$\left(\mathsf{sh}_\ell^{i,j,2}\right)_{\ell=j+1}^{j+k-i+1} \leftarrow \text{Interpolate}\left(t, A, B, s = m_{i,j}^2, (\mathsf{sh}_\ell^{i,j,2})_{\ell=j+k-i+2}^{n}\right).$$
7. The share sh_ℓ of P_ℓ is formed by w_ℓ, $\mathsf{sh}_\ell^{i,j,1}$ for $i \in [k]$ and $\max\{i+1, \ell+1\} \le j \le \min\{n-k+i, \ell+i\}$, and $\mathsf{sh}_\ell^{i,j,2}$ for $i \in [k]$ and $\max\{i+1, \ell-k+i-1\} \le j \le \min\{n-k+i, \ell-1\}$.

[a] If $j = i+1$ then $A = \emptyset$ and Procedure Interpolate gives the secret (i.e., $m_{i,j}^1$) to the first $j-1$ parties; this is a 1-out-of-$(j-1)$ scheme as required.

Fig. 7. A CSSS realizing a partially defined $(n-k)$-slice f.

the PRG is executed n times and there are $O(nk)$ interpolations, the total running time of the sharing is $n^k \cdot \mathrm{poly}(\lambda) + \tilde{O}\ (n^2 k) \cdot \mathrm{poly}(\lambda)$. As the size of the representation of the $(n-k)$-slice function f is $\binom{n}{k(n)} \geq (n/k)^k \geq \sqrt{n}^k = (n^k)^{0.5}$ (as $k(n) \leq \sqrt{n}$), the running time is polynomial in the representation and the security parameter.

We next prove the correctness and security of the scheme, only considering inputs of weight $n-k$. This scheme is an optimization of the perfect secret-sharing scheme described in Fig. 4, where we use a CCDS protocol instead of a perfect CDS protocol and use procedure Interpolate to share the shares, using correlated pseudorandom strings as the "random shares"; however these are valid shares in Shamir's secret-sharing scheme. Thus, the correctness follows from the correctness of the scheme from Fig. 4.

For the security of the scheme described in Fig. 7, we assume that there is an adversary $\mathcal{A}_{\mathrm{CSSS}}$ trying to break the CSSS and prove that if $\mathcal{A}_{\mathrm{CSSS}}$ succeeds then there is either an adversary breaking the CCDS protocol or an adversary breaking the PRG, contradicting their security. Recall that $\mathcal{A}_{\mathrm{CSSS}}$, on input 1^λ, chooses an $(n-k)$-slice function f and an input $x \in \{0,1\}^n$ such that $\mathrm{wt}(x) = n-k$ and $f(x) = 0$ and gets from the challenger shares $(\mathsf{sh}_i)_{x_i=1}$ generated by the scheme for a random secret $s \in \{0,1\}$.

We define $n+1$ hybrids, where in the d-th hybrid a secret s is chosen with uniform distribution, and the shares given to the adversary are generated similar to the scheme in Fig. 7, where we replace some pseudorandom strings in step 1 with truly random strings as follows: For every $1 \leq \ell \leq d$, if $x_\ell = 0$ we use for every $i \in [k], j \in [n]$ a truly random string for $y_\ell^{i,j}$ (instead of $y_\ell^{i,j} \leftarrow \mathrm{PRG}_{i,j}(w_\ell)$). All other strings $y_\ell^{i,j}$ are generated as pseudorandom strings. Let pr_d be the probability that the adversary $\mathcal{A}_{\mathrm{CSSS}}$ guesses the secret given the shares generated in the d-th hybrid for a uniformly distributed secret $s \in \{0,1\}$.

Notice that in the 0-th hybrid the shares given to the adversary are generated as in the scheme described in Fig. 7. On the other hand, in the n-th hybrid, for all parties not in I_x, the shares are generated using truly random strings. In this case, we will show that by the security of the CCDS protocol, the probability that the adversary guesses s, i.e., pr_n is at most $1/2 + 1/\mathrm{poly}(t(\lambda))$. The proofs of the following claims are in the full version [BFLN24].

Claim 4.2. *Assume that the CCDS protocol used in the protocol described in Fig. 4 is $t^\alpha(\lambda)$-secure for some constant $\alpha < 1$ and that $\mathcal{A}_{\mathrm{CSSS}}$ runs in time $t^{0.4\alpha}(\lambda)$. Then, $\mathrm{pr}_n \leq 1/2 + 1/t^\alpha(\lambda)$.*

We next show that, by the security of the PRG, the probability that an adversary guesses the secret in hybrid d is at most $1/t^\alpha(\lambda)$ greater than the probability that it guesses the secret in hybrid $d+1$.

Claim 4.3. *Assume that the PRG used in the protocol described in Fig. 4 is $t^\alpha(\lambda)$-secure for some constant $\alpha < 1$ and that $\mathcal{A}_{\mathrm{CSSS}}$ runs in time $t^{0.4\alpha}(\lambda)$. Then, $\mathrm{pr}_{d-1} - \mathrm{pr}_d \leq 1/t^\alpha(\lambda)$ for every $1 \leq d \leq n$.*

We have assumed that the PRG and the CCDS protocol used in the scheme of Fig. 7 are $1/t^{\alpha}(\lambda)$ secure for some constant $\alpha < 1$. Consider an adversary $\mathcal{A}_{\text{CSSS}}$ that runs in time $t^{0.4\alpha}(\lambda)$. By Claim 4.2 and Claim 4.3,

$$\text{pr}_0 = (\text{pr}_0 - \text{pr}_1) + \cdots + (\text{pr}_{n-1} - \text{pr}_n) + \text{pr}_n \leq \frac{n}{t^{\alpha}(\lambda)} + \frac{1}{2} + \frac{1}{t^{\alpha}(\lambda)}.$$

Thus, pr_0 – the probability that an adversary guesses the secret in the 0-th hybrid i.e., in the scheme described in Fig. 7 – is at most $1/2 + \frac{n+1}{t^{\alpha}(\lambda)}$. As $n \leq \lambda$ and $1/t(\lambda)$ is a negligible function, this probability is less than $1/2 + 1/t^{0.4\alpha}(\lambda)$. To conclude, we have proved that the probability that any adversary running in time at most $t^{0.4\alpha}(\lambda)$ guesses the secret with probability at most $1/2+1/t^{0.4\alpha}(\lambda)$, i.e., the CSSS is $1/t^{0.4\alpha}(\lambda)$-secure.

We complete the proof by analyzing the share size. There are $O(kn)$ executions of threshold t'-out-of-n' secret-sharing schemes in the scheme described in Fig. 7; in each one of them $n' - t' \leq k - 1$. In each such scheme there are at most k "extra" shares. The scheme distributes these "extra" shares such that each party gets $O(k^2)$ "extra" shares, i.e., in the scheme for $m^1_{i,j}$, which is shared among the first $j-1$ parties only the last $i-1$ parties $P_{j-i}, \ldots, P_{j-1}$ get the "extra" shares and in the scheme for $m^2_{i,j}$, which is shared among the last $n-j-$ parties only the first $k-i$ parties $P_{j+1}, \ldots, P_{j+k-i+1}$ get the "extra" shares. To conclude, the share of each party contains one seed of size λ and $O(k^2)$ shares in Shamir's scheme with secrets of length $O(\lambda \log n)$ – the message size of the CCDS protocol of [ABI+23b] (see Theorem 2.9). All together, the share size of each party is $O(k^2 \lambda \log n)$. □

5 Applications to Multislices

In this section, we present implications for multislices of the improved schemes for $(n-k)$-slices presented in Sects. 3 and 4. We split the results into different subsections. In Sect. 5.1, we present some results we use about the construction of secret-sharing schemes from monotone Boolean formulas. In Sect. 5.2, we present general results for $(a : b)$-multislices, proving Theorem 1.5. For $(n : n-k)$-multislices with $k = o(n)$, we are able to find better schemes, and the rest of the section is dedicated to this case. The proofs of the results about this particular case, intermediate results, and additional remarks can be found in [BFLN24].

5.1 The Framework

First, we discuss the framework in which we use the schemes for $(n-k)$-slices. Many general and multislice schemes from recent years are based on the following paradigm:

1. Given a function f over n bits, build a constant-depth formula F for f that uses AND and OR gates, together with gates that compute (k, N)-slice functions for some N, and $k = O(\log N)$. In these formulas, all (k, N)-slice gates are in the same level, that is, in every path from the root to a leaf there is at most one (k, N)-slice gate.

2. Apply the closure properties of secret-sharing schemes over formulas to realize a scheme for f according to F (Lemma 5.2). Whenever necessary, plug in a black-box way an efficient scheme for k-slices (or k-server CDS protocols) that has shares of size $2^{\tilde{O}(\sqrt{k \log N})}$ [LVW18].

Given a scheme for a multislice function f that follows the above paradigm, we will be able to use the following simple duality lemmas to realize the dual of f. The first lemma is a folklore result that was stated and proved in [ABN+22]. The second one is a natural generalization of the [BL88] whose proof also appears in [ABN+22].

Lemma 5.1 (Formulas and Duality). *Let C be a formula that computes a function $f : \{0,1\}^n \to \{0,1\}$ and let $G_1, \ldots, G_k$ be its gates. For any gate G that computes a function g, denote by G^* a gate that computes g^*. Then, a formula C' with the same structure as C and with every gate G_i replaced with G_i^* computes the dual function f^*.*

Lemma 5.2 (Formulas and Secret Sharing). *Suppose that a monotone function $f : \{0,1\}^n \to \{0,1\}$ can be implemented by a formula F over some collection of monotone gates G, and assume that every gate $g \in G$ can be realized by a secret-sharing scheme whose share-size is w_g. Then, f can be realized by a secret-sharing scheme whose share size is $\max_{i \le i \le n} \{w_{F,i}\}$, where the weight function $w_{F,i}$ is defined as follows.*

- *The weight $w_F(v)$ of a leaf v in F is the product $\prod_j w_{g_j}$ where g_j is the jth gate in the (unique) path from the root to v.*
- *The weight $w_{F,i}$ of the ith variable in the formula F is the sum of $w_F(v)$ of all leaves v labeled by x_i.*

Similarly, if every gate g can be realized by a secret-sharing scheme whose information ratio is w'_g, then f can be realized by a secret-sharing scheme with information ratio of $\max_{i \le i \le n} \{w'_{F,i}\}$.

It is worth noticing that if we consider a formula F whose gates can be realized by a secret-sharing schemes with good information ratio, then the resulting share size of party P_i will be between $w_{F,i}$ and $w'_{F,i}$. For instance, if F has k layers of t-out-of-n threshold gates, the share size increases by $\log n$ in each path, but not by $\log^k n$.

5.2 Schemes for $(a : b)$-Multislices

We restate and prove Theorem 1.5, which implies that the current gap between the share sizes of the family of $(a : b)$-multislices and its dual can be narrowed down to $2^{o(n)}$. Recall that this has implications on the share size of random hypergraph access structures (Corollary 1.7).

Theorem 5.3 (Share Size of Multislices, Theorem 1.5 Restated). *For every $a < b \in [n]$, every $(a : b)$-multislice can be realized by a secret-sharing scheme with share size $\binom{n-a}{\ge n-b} \cdot 2^{o(n)}$.*

Proof. As analyzed by [ABN+22], the scheme of Applebaum and Nir [AN21] for $(a:b)$-multislices that has share sizes $\binom{b}{\geq a} \cdot 2^{o(n)}$ works according to the paradigm specified in the introduction of this section. For each multislice, they construct a formula F over AND and OR gates, combined with gates that compute (k, N)-slice gates with $k = \sqrt{n}$, $N = \sqrt{n}2^{\sqrt{n}}$.

By Lemma 5.1, if we replace every gate G in F by its dual gate G^* we get a formula F^* that computes f^*, the dual of f. This formula will consist of AND and OR gates, together with $(N-k, N)$-slice gates with $k = \sqrt{n}$, $N = \sqrt{n}2^{\sqrt{n}}$, which are the duals of the slice gates that appeared in F. By Lemma 5.2, the overhead of realizing a secret-sharing scheme for f^* based on F^* compared to realizing f based on F boils down solely to the difference in the cost of the different types of slice gates used in each of the formulas. In [ABNP20] the (k, N)-slices of F with the above parameters were realized with share size $2^{\tilde{O}(\sqrt{n})}$; by Theorem 3.5 we get the same asymptotical share size for the $(N-k, N)$-slices of F^*, i.e., we get share size $kN2^{\tilde{O}(\sqrt{k \log N})} = 2^{\tilde{O}(\sqrt{n})}$. Hence, since the scheme of [ABNP20] realizes $(n-b : n-a)$-multislices with shares of size $\binom{n-a}{\geq n-b} \cdot 2^{o(n)}$, their dual $(a:b)$-multislices can also be realized with the same share size, as desired. □

5.3 Schemes for $(n-k:n)$-Multislices

In the rest of this section, we construct schemes for $(n-k:n)$-multislices that are more efficient than the ones described in Theorem 1.5 when $k = o(n)$. Our main result for perfect schemes is in Theorem 5.10. The current best upper and lower bounds for $(n-k:n)$-multislices are summarized in Fig. 8.

The share size of perfect schemes for $(n-k:n)$-multislices				
	$k \geq t_1$	$t_1 \geq k \geq t_2$	$t_2 \geq k \geq t_3$	$t_3 \geq k$
Upper bounds	$2^{0.59n}$ [AN21]	$O(\binom{n-1}{k-1})$	$2^{O(k)}$ Theorem 5.6	$k^{5k} n 2^{\tilde{O}(\sqrt{k \log n})}$ Theorem 5.6
Lower bounds	$\Omega(k \log k)$ [Csi97]		$\Omega(\log n)$ [BGK16]	

Fig. 8. The best-known bounds on the share size of perfect secret-sharing schemes for $(n-k:n)$-multislices. See [BFLN24] for more details. The thresholds for the bounds in the table are are $t_1 = 0.14n$, $t_2 = n/\log^2 n$, and $t_3 = \log n(\log\log n)^2$.

Definition 5.4 ((h_i, k)-Hypergraphs). *Let $i \leq k \leq n$ and let $\mathrm{TR}_{k+1,n}$ be the threshold $k+1$ function on n variables. Given an i-hypergraph h_i (i.e., a function whose minterms are of size exactly i), the (h_i, k)-hypergraph is defined as $h_i \vee \mathrm{TR}_{k+1,n}$.*

We next show how to realize $(n-k:n)$-multislices from secret-sharing schemes for the dual of (h_i, k)-hypergraphs

Lemma 5.5 (Multislices from duals of (h_i, k)-hypergraphs). *Let f be an $(n-k:n)$-multislice. Assume that for every $i \leq k < n$ and every i-hypergraph h_i, the dual of the (h_i, k)-hypergraph can be realized by a secret-sharing scheme with share size $c_{\text{dual}}(i,k,n)$. Then f can be realized by a secret-sharing schemes with share size $\sum_{i=1}^{k} c_{\text{dual}}(i,k,n)$. If the secret-sharing schemes for the duals of (h_i, k)-hypergraphs are linear, the resulting scheme is linear.*

In our schemes, we use a construction of [BF20b] for (h_i, k) hypergraphs (which follows from results of [ABNP20]) and an adaptation of a construction of [ABNP20]. These constructions implicitly follow the framework of Sect. 5.1.

Theorem 5.6 (Share Size of $(n-k:n)$-Multislices, Theorem 1.6 Restated). *For $k \leq \log n(\log\log n)^2$, every $(n-k:n)$-multislice can be realized by a secret-sharing scheme with share size $k^{5k} \cdot n \cdot 2^{\tilde{O}(\sqrt{k \log n})}$. For $\log n(\log\log n)^2 < k < n/\log^2 n$, every $(n-k:n)$-multislice can be realized by a secret-sharing scheme with share size $2^{O(k)}$.*

Remark 5.7. In Theorem 5.6, we implicitly construct a constant depth formula for $(n-k:n)$-multislices of size $O(k\ell_k)$ with AND, OR, threshold, and dual of partite slice gates, where the dual of partite slice gates are in the same level. That is, we construct a formula for duals of (h_i, k)-hypergraphs and construct from it a formula for $(n-k:n)$-multislices. In most of our constructions, we replace the dual of $(\kappa_i N_i - \kappa_i, \kappa_i N_i)$-partite slice gates by $(\kappa_i, \kappa_i N_i)$-slices.

5.4 CSSS for $(n-k:n)$-Multislices

We next use the construction of Sect. 5.3 to construct CSSS for $(n-k:n)$-multislices. For the CSSS we use a computational analogue of the formula-based perfect secret-sharing scheme of Benaloh and Leichter [BL88]; in our scheme we use CSSS to realize the gates. Proving the security of the resulting CSSS requires analyzing the representation size of the gates and the share size as well as running time of the CSSS implementing the gates. We prove the security for the specific formula of Lemma 5.2. We remark that in the computational setting, Yao [Yao89] showed that a CSSS can use a monotone *circuit*; however, we do not need this generalization.

Remark 5.8. The CSSS for $(n-k:n)$-multislices as described in Theorem 5.9 is interesting when $k \leq \log n$. For $\log n < k < \log n(\log\log(n))^2$, it has share size that is worse than the perfect secret-sharing scheme described in Theorem 5.6 by a factor of λ. Furthermore, for $k > \log n(\log\log(n))$, we can obtain a CSSS with share size $O(2^{O(k)}\lambda \operatorname{polylog}(n))$. As this secret-sharing scheme will only outperform the perfect secret-sharing scheme described in Theorem 5.6 in a small range of parameters, we omit these details.

Theorem 5.9 (CSSSs for $(n-k:n)$-Multislices). *Let $k \leq \sqrt{n}$ and $t(\lambda) \geq k^{ck} \cdot \mathrm{polylog}(n)$ for a sufficiently large constant c. Assuming the existence of $t(\lambda)$-secure OWFs, there is a $\mathrm{poly}(t(\lambda))$-secure CSSS for $(n-k:n)$-multislices, represented as a truth table of all inputs of weight at least $n-k$. The share size in the CSSS is $O(k^{5k}\lambda\,\mathrm{polylog}(n))$, where λ is the security parameter, and the running time of the sharing and reconstruction algorithms of the CSSS is $\tilde{O}(n^{4k}) \cdot \mathrm{poly}(\lambda) = \mathrm{poly}(\binom{n}{k}, \lambda)$.*

5.5 Linear and Multi-linear Schemes for $(n-k:n)$-Multislices

Next, we provide upper and lower bounds on the share size and information ratio for linear and multi-linear schemes. Notice that the gap between the bounds is asymptotically tight when k is constant.

Theorem 5.10 (Linear Schemes for $(n-k:n)$-Multislices). *Let $1 < k < n/\log^2 n$. Then $(n-k:n)$-multislices can be realized by a linear secret-sharing scheme with share size $\tilde{O}(k^{5k}n^{(k-1)/2})$.*

Theorem 5.11 (Multi-linear Secret-Sharing Schemes for $(n-k:n)$-Multislices). *For $1 < k < \log n \log\log(n)$, every $(n-k:n)$-multislice f can be realized by a multi-linear secret-sharing scheme with secrets of size $2^{n^{k-1}}$ with information ratio $O(k^{5k}\log^2 n)$. For $\log n \log\log(n) < k < n/\log^2 n$, every $(n-k:n)$-multislice f can be realized by a multi-linear secret-sharing scheme with secrets of size $2^{n^{O(k)}}$ with information ratio $2^{O(k)}$.*

Theorem 5.12. *For almost all $(n-k:n)$-multislices, the total share size in every linear secret-sharing scheme with a one-bit secret realizing these access structures is $\Omega(n^{(k-1)/2}/k^{(k+1)/2})$.*

Acknowledgments. We thank Benny Applebaum and Eliran Kachlon for valuable comments on earlier drafts of this paper. The first author is supported by the ISF grant 391/21 and by the ERC grant 742754 (project NTSC). The second author is supported by the grant 2021SGR 00115 from the Government of Catalonia, the project ACITHEC PID2021-124928NB-I00 from the Government of Spain, and the project HERMES funded by the European Union NextGenerationEU/PRTR via INCIBE. The third author is supported by the ISF grant 391/21 and by the Frankel center for computer science. The forth author is supported by ISF grant no. 2805/21 and by the European Union (ERC, NFITSC, 101097959). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Research Council. Neither the European Union nor the granting authority can be held responsible for them.

References

AA18. Applebaum, B., Arkis, B.: On the power of amortization in secret sharing: d-Uniform Secret Sharing and CDS with Constant Information Rate. In: Beimel, A., Dziembowski, S. (eds.) TCC 2018. LNCS, vol. 11239, pp. 317–344. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-03807-6_12

ABF+19. Applebaum, B., Beimel, A., Farràs, O., Nir, O., Peter, N.: Secret-sharing schemes for general and uniform access structures. In: EUROCRYPT 2019, vol. 11478. LNCS, pp. 441–471 (2019)

ABI+23a. Abram, D., Beimel, A., Ishai, Y., Kushilevitz, E., Narayanan, V.: Cryptography from planted graphs: Security with logarithmic-size messages. In: TCC 2023, vol. 14369. LNCS, pp. 286–315 (2023)

ABI+23b. Applebaum, B., Beimel, A., Ishai, Y., Kushilevitz, E., Liu, T., Vaikuntanathan, V.: Succinct computational secret sharing. In: STOC **2023**, 1553–1566 (2023)

ABN+22. Applebaum, B., Beimel, A., Nir, O., Peter, N., Pitassi, T.: Secret sharing, slice formulas, and monotone real circuits. In: ITCS 2022, vol. 215. LIPIcs, pp. 8:1–8:23 (2022)

ABNP20. Applebaum, B., Beimel, A., Nir, O., Peter, N.: Better secret sharing via robust conditional disclosure of secrets. In: STOC 2020, pp. 280–293 (2020)

AN21. Applebaum, B., Nir, O.: Upslices, downslices, and secret-sharing with complexity of 1.5^n. In: CRYPTO 2021, vol. 12827. LNCS, pp. 627–655 (2021)

BC94. Beimel, A., Chor, B.: Universally ideal secret sharing schemes. IEEE Trans. Inf. Theory **40**(3), 786–794 (1994)

Bei11. Beimel, A.: Secret-sharing schemes: a survey. In: Coding and Cryptology – Third International Workshop, IWCC 2011, vol. 6639. LNCS, pp. 11–46 (2011)

Bei23. Beimel, A.: Lower bounds for secret-sharing schemes for k-hypergraphs. In: ITC 2023, vol. 267. LIPIcs, pp. 16:1–16:13 (2023)

Ber82. Berkowitz, S.: On some relationships between monotone and nonmonotone circuit complexity. Technical report, Department of Computer Science, University of Toronto (1982)

BF20a. Beimel, A., Farràs, O.: The share size of secret-sharing schemes for almost all access structures and graphs. IACR Cryptol. ePrint Arch. **2020**, 664 (2020)

BF20b. Beimel, A., Farràs, O.: The share size of secret-sharing schemes for almost all access structures and graphs. In: TCC 2020, vol. 12552. LNCS, pp. 499–529 (2020)

BFLN24. Beimel, A., Farràs, O., Lasri, O., Nir, O.: Secret-sharing schemes for high slices. Technical Report 2024/602, IACR Cryptology ePrint Archive (2024)

BFMP22. Beimel, A., Farràs, O., Mintz, Y., Peter, N.: Linear secret-sharing schemes for forbidden graph access structures. IEEE-TIT **68**(3), 2083–2100 (2022)

BGK16. Bogdanov, A., Guo, S., Komargodski, I.: Threshold secret sharing requires a linear size alphabet. In: TCC 2016, vol. 9986. LNCS, pp. 471–484 (2016)

BIKK14. Beimel, A., Ishai, Y., Kumaresan, R., Kushilevitz, E.: On the cryptographic complexity of the worst functions. In: TCC 2014, vol. 8349. LNCS, pp. 317–342 (2014)

BKN18. Beimel, A., Kushilevitz, E., Nissim, P.: The complexity of multiparty PSM protocols and related models. In: EUROCRYPT 2018, vol. 10821. LNCS, pp. 287–318 (2018)

BL88. Benaloh, J.C., Leichter, J.: Generalized secret sharing and monotone functions. In: CRYPTO '88, vol. 403. LNCS, pp. 27–35 (1988)
Bla79. Blakley, G.R.: Safeguarding cryptographic keys. In: Proc. of the 1979 AFIPS National Computer Conference, vol. 48. AFIPS Conference proceedings, pp. 313–317 (1979)
Bog23. Bogdanov, A.: Csirmaz's duality conjecture and threshold secret sharing. In: ITC, vol. 267. LIPIcs, pp. 3:1–3:6 (2023)
BP18. Beimel, A., Peter, N.: Optimal linear multiparty conditional disclosure of secrets protocols. In: ASIACRYPT 2018, vol. 11274. LNCS, pp. 332–362 (2018)
CCX13. Cascudo, I., Cramer, R., Xing, C.: Bounds on the threshold gap in secret sharing and its applications. IEEE Trans. Inf. Theory **59**(9), 5600–5612 (2013)
CK93. Chor, B., Kushilevitz, E.: Secret sharing over infinite domains. J. Cryptology **6**(2), 87–96 (1993)
Csi97. Csirmaz, L.: The size of a share must be large. J. Cryptology **10**(4), 223–231 (1997)
Csi20. Csirmaz, L.: Secret sharing and duality. J. Math. Cryptol. **15**(1), 157–173 (2020)
EP97. Erdös, P., Pyber, L.: Covering a graph by complete bipartite graphs. Discret. Math. **170**(1–3), 249–251 (1997)
Feh98. Fehr, S.: Span programs over rings and how to share a secret from a module. Master's thesis, ETH Zurich (1998)
FHKP17. Farràs, O., Hansen, T.B., Kaced, T., Padró, C.: On the information ratio of non-perfect secret sharing schemes. Algorithmica **79**(4), 987–1013 (2017)
Gál95. Gál. A.: Combinatorial Methods in Boolean Function Complexity. Ph.D. thesis, U. of Chicago (1995)
GIKM00. Gertner, Y., Ishai, Y., Kushilevitz, E., Malkin, T.: Protecting data privacy in private information retrieval schemes. J. Comput. Syst. Sci. **60**(3), 592–629 (2000)
GKW15. Gay, R., Kerenidis, I., Wee, H.: Communication complexity of conditional disclosure of secrets and attribute-based encryption. In: CRYPTO 2015, vol. 9216. LNCS, pp. 485–502 (2015)
HILL99. Håstad, J., Impagliazzo, R., Levin, L.A., Luby, M.: Construction of a pseudo-random generator from any one-way function. SIAM J. Comput. **28**(4), 1364–1396 (1999)
ISN87. Ito, M., Saito, A., Nishizeki, T.: Secret sharing schemes realizing general access structure. In Globecom **87**, 99–102 (1987)
KGH83. Karnin, E.D., Greene, J.W., Hellman, M.E.: On secret sharing systems. IEEE Trans. Inf. Theory **29**(1), 35–41 (1983)
KN90. Kilian, J., Nisan, N.: Unpublished result (1990)
KNY17. Komargodski, I., Naor, M., Yogev, E.: Secret-sharing for NP. J. Cryptol. **30**(2), 444–469 (2017)
Kra94. Krawczyk, H.: Secret sharing made short. In: CRYPTO '93, vol. 773. LNCS, pp. 136–146 (1994)
KW93. Karchmer, M., Wigderson, A.: On span programs. In: 8th Structure in Complexity Theory, pp. 102–111 (1993)
LS20. Larsen, K.G., Simkin, M.: Secret sharing lower bound: either reconstruction is hard or shares are long. In: SCN 2020, vol. 12238. LNCS, 566–578 (2020)
LV18. Liu, T., Vaikuntanathan, V.: Breaking the circuit-size barrier in secret sharing. In: 50th STOC, pp. 699–708 (2018)

LVW18. Liu, T., Vaikuntanathan, V., Wee, H.: Towards breaking the exponential barrier for general secret sharing. In: EUROCRYPT 2018, vol. 10820. LNCS, pp. 567–596 (2018)

Sha79. Shamir, A.: How to share a secret. Commun. ACM **22**, 612–613 (1979)

SS97. Sun, H.-M., Shieh, S.-P.: Secret sharing in graph-based prohibited structures. In: INFOCOM '97, pp. 718–724 (1997)

VNS+03. Vinod, V., Narayanan, A., Srinathan, K., Pandu Rangan, C., Kim, K.: On the power of computational secret sharing. In: Indocrypt 2003, vol. 2904. LNCS, pp. 162–176 (2003)

Yao89. Yao, A.C.: Unpublished manuscript. Presented at Oberwolfach and DIMACS Workshops (1989)

Homomorphic Secret Sharing with Verifiable Evaluation

Arka Rai Choudhuri[1], Aarushi Goel[2], Aditya Hegde[3(✉)], and Abhishek Jain[3,4]

[1] Nexus, San Francisco, USA
[2] Purdue University, West Lafayette, USA
aarushi@purdue.edu
[3] Johns Hopkins University, Baltimore, USA
{ahegde,abhishek}@cs.jhu.edu
[4] NTT Research, Sunnyvale, USA

Abstract. A homomorphic secret sharing (HSS) scheme allows a client to delegate a computation to a group of untrusted servers while achieving input privacy as long as at least one server is honest. In recent years, many HSS schemes have been constructed that have, in turn, found numerous applications to cryptography.

Prior work on HSS focuses on the setting where the servers are semi-honest. In this work we study HSS in the setting of malicious evaluators. We propose the notion of HSS *with verifiable evaluation* (ve-HSS) that guarantees correctness of output *even when all the servers are corrupted*. ve-HSS retains all the attractive features of HSS and adds the new feature of succinct public verification of output.

We present *black-box* constructions of ve-HSS by devising generic transformations for semi-honest HSS schemes (with negligible error). This provides a new non-interactive method for verifiable and private outsourcing of computation.

1 Introduction

In a t-private homomorphic secret sharing (HSS) scheme [16], a client uses a sharing algorithm to split a secret x into multiple shares such that any subset of at most t shares do not reveal anything about x. Such a scheme additionally supports homomorphic computation on the shares via the following algorithms:

- A local evaluation algorithm Eval that given a function f, maps a share x_i to an output share y_i.
- A reconstruction algorithm Rec that computes the output $y = f(x)$ from the output shares $\{y_i\}_i$.

Similar to additive secret sharing schemes, the reconstruction algorithm Rec is simply an additive function over the output shares. This in turn implies that

E. Boyle and M. Mahmoody (Eds.): TCC 2024, LNCS 15367, pp. 614–650, 2025.
https://doi.org/10.1007/978-3-031-78023-3_20

the size of each share output by Eval is *succinct* and only depends on the output length of f[1], and is otherwise independent of the description of f.

A t-private HSS allows a client to delegate a (possibly resource-intensive) computation to a group of untrusted servers while achieving input privacy against any t number of servers.[2] This feature, in turn, has many applications in cryptography, including sublinear-communication secure multiparty computation (MPC) [16,39,40,44], multi-server private information retrieval [16,54], correlated-randomness generators [12,13], reusable MPC [3], constrained pseudorandom functions [41], and more. By now, several constructions of HSS are known from standard group-based [1,14,16,18,73,76], lattice-based [23,45] and code-based assumptions [13,40,44]. Some of these schemes differ in the number of servers they can support (two or more), the class of functions that can be evaluated, as well as whether or not they have (non-negligible) correctness error.

Prior work in this area focuses on semi-honest servers who perform the local evaluations honestly (see Sect. 1.3 for some exceptions). This means, in particular, that the output is only guaranteed to be correct when the servers are honest. In this work, we study HSS in the setting where the servers may be *malicious* and can deviate from the honest evaluation strategy. In particular, the new property we seek is public verifiability of the output by any external auditor, even when all the servers are malicious. Viewed from the lens of delegation of computation, we seek *verifiable* and private delegation of computation.

HSS with Verifiable Evaluation. We propose a new notion of HSS *with verifiable evaluation* (ve-HSS) that retains the attractive features of HSS and adds public verifiability to the output of the computation. Similar to (semi-honest) HSS, a ve-HSS scheme is equipped with sharing, Eval and Rec algorithms that provide the same functionality and efficiency properties. Since the evaluators may be malicious, the reconstruction algorithm Rec in ve-HSS produces a "candidate" output. We require an additional algorithm Verify that either accepts or rejects this output. Importantly, verification is *succinct*, namely, Verify runs in time independent of the complexity of the evaluated function and the number of servers.

We first augment the HSS privacy notion to require that the privacy of the client's input should hold even if the malicious servers learn whether the candidate output produced was accepted or rejected by the client. This augmentation is necessary since malicious servers may produce maliciously computed shares, and attempt to glean information based on the client behavior. As in the case of HSS, we parameterize the privacy requirements by the number of corrupt servers t.

Motivated by the application to verifiable delegation of computation, we consider the following notions of soundness:

[1] There is an additional dependence on the security paramter, which we omit here.

[2] This is analogous to fully homomorphic encryption (FHE) which allows private delegation of computation to a single server.

- *Local Soundness*: This notion guarantees that for any shared input $x = (x_{\mathsf{pub}}, x_{\mathsf{priv}})$—where x_{pub} is the public input and x_{priv} is the private input known only to the client—the Verify algorithm, given x, accepts a candidate output y if and only if y is the result of evaluating the function f on x.
- *Public Soundness*: This notion guarantees that for any shared input $x = (x_{\mathsf{pub}}, x_{\mathsf{priv}})$, the Verify algorithm, given x_{pub}, accepts a candidate output y if and only if $y = f(x_{\mathsf{pub}}, \cdot)$ for "some" private input.

Crucially, we require these soundness properties to hold *even when all the servers are corrupted*. This means that an external auditor can publicly verify the correctness of the output without knowing anything about (or placing any trust assumptions in) the entities who computed the output. This property makes our notion *stronger* than standard secure multiparty computation (MPC), which can only guarantee output correctness when at least one party is honest.

It is easy to see that ve-HSS with succinct public verification implies succinct non-interactive arguments (SNARGs) [71]. Indeed, the local soundness property implies SNARGs for deterministic computations while public soundness implies SNARGs for non-deterministic computations. The class of languages supported by the SNARG is determined by the family of functions supported by the ve-HSS scheme.

Multi-Client HSS with Verifiable Evaluation. We also study verifiability for *multi-client* HSS, where a set of mutually distrusting clients can share their inputs to a set of servers who can then perform joint evaluations on the inputs of the clients. The notions of privacy and soundness in this setting are strengthened to allow for collusions between the clients and servers. In particular, local soundness now allows each client to verify whether its private input x_i was used in the computation, i.e., $y = f(\cdot, x_i, \cdot)$ even if all the remaining clients and all the servers are corrupted. Public soundness, as before, guarantees that the output was obtained by evaluating f on the fixed public input and "some" set of private inputs, i.e., $y = f(x_{\mathsf{pub}}, \cdot, \cdot)$, even if all parties are corrupt.

1.1 Our Results

In this work, we study HSS with verifiable evaluation with a focus on building solutions that make only *black-box* use of cryptography. We obtain new results both for the single-client and multi-client settings.

Single Client HSS with Verifiable Evaluation. We present two sets of results for single-client homomorphic secret sharing (HSS) with verifiable evaluation: one in the generic group model and the other in the standard model. Below, we present the results for each model separately, starting with our result in the generic group model.

Theorem 1 (Generic Group Model, Single Client, Informal). *In the generic group model, we construct a single client m-server HSS with verifiable evaluation that only makes black-box use of cryptography:*

- *For* NC^1*: from (i) class groups assumptions; (ii) Decisional Composite Residuosity (DCR) assumption; or (iii) Learning with Errors (LWE) assumption.*
- *For* $\mathsf{P/poly}$*: from (i) the security of indistinguishability obfuscation (iO) + one-way functions (OWF); or (ii) security of fully homomorhpic encryption (FHE).*

In the standard model, we consider boolean circuits that are SIMD (single instruction multiple data). We obtain analogous results as in the generic group model, but in the SIMD setting.

Theorem 2 (Standard Model, Single Client, Informal). *In the standard model, we construct a single client m-server HSS with verifiable evaluation that only makes black-box use of cryptography:*

- *For* SIMD NC^1*: from the subgroup decision assumption in addition to either (i) class groups assumptions; (ii) Decisional Composite Residuosity (DCR) assumption; or (iii) Learning with Errors (LWE) assumption.*
- *For* SIMD $\mathsf{P/poly}$*: from the subgroup decision assumption in addition to either (i) the security of indistinguishability obfuscation (iO) + one-way functions (OWF); or (ii) security of fully homomorhpic encryption (FHE).*

Main Compiler. The results stated above for the single-client setting are based on our current understanding of the HSS landscape. However, we derive these results using a general compiler that takes as its starting point any standard semi-honest HSS scheme, allowing for further instantiations as our understanding of HSS schemes evolves.

At a high level, our compiler adds verifiability to a semi-honest HSS scheme by requiring the servers to additionally compute a zero-knowledge succinct non-interactive arguments (zkSNARGs) of correct computation *under the hood* using the semi-honest HSS, i.e. the zkSNARG is computed in a distributed manner by all the servers. In order to ensure that the compiler only makes black-box use of cryptography, we enforce certain structural (and security) constraints on the zkSNARGs. We formalize these properties via a new notion of of zkSNARGs that we call *splittable* zkSNARGs.

We will describe the security properties shortly, but the primary 'structural' properties we require from such a splittable zkSNARG is that the computation of the zkSNARG proof can be divided into an initial *non-linear* computation phase over all of its input, followed by a computation phase that exclusively performs *linear* operations on the output of the non-linear phase. Importantly, we further require that the non-linear phase is non-cryptographic. We make this distinction since in our compiler the non-linear phase will be computed via the (semi-honest) HSS scheme, and the linear phase will be computed on the additive shares obtained obtained from the aforementioned HSS evaluation.

With our new tool splittable zkSNARGs, we present below our main theorem statement: a compiler for transforming any semi-honest single-client HSS scheme with at most negligible correctness error into a single-client HSS with verifiable evaluation (ve-HSS).

Theorem 3 (Main Compiler, Informal). *Let* HSS *be a single-client m-server HSS scheme for a class of circuits* $\mathbb{C}$ *with negligible correctness error. Then given a splittable zkSNARG for* $\mathbb{C}'$ *where the non-linear computation phase can be implemented using a circuit in* $\mathbb{C}$*, there exists a single client m-server HSS with verifiable evaluation* ve-HSS *for circuit class* $\min\{\mathbb{C}, \mathbb{C}'\}$*, that only makes black-box use of cryptography.*

Here $\min\{\mathbb{C}, \mathbb{C}'\}$ simply refers to the smaller class among the two circuit classes.

We describe next our results for the aforementioned tool of splittable zkSNARGs, needed to instantiate our compiler.

Splittable zkSNARGs. We have already described the structural requirements for splittable SNARGs, where the computation of the proof can be split into non-linear and linear phases of computation. For security, in addition to the regular soundness properties of zkSNARGs, we require a splittable zkSNARG to have *robust verification*, i.e. the verifier will reject all proofs that contain additive errors. We defer a formal definition of this notion to later. We present two instantiations of splittable zkSNARGs that can be used to instantiate our compilers.

Spittable zkSNARGs for NP. Our first instantiation supports general NP languages and is based on the Groth16 zkSNARK[3] [58] that achieve perfect zero-knowledge and computational soundness (proven in the generic group model). We show that the Groth16 zkSNARKs satisfy all the properties of a splittable zkSNARG.

Moreover, we show that the non-linear computation phase in the Groth16 zkSNARK requires evaluating a circuit of depth $\mathcal{O}(D + \log \lambda)$ if the relation circuit has depth D.

Spittable zkSNARGs for batch-NP. Our second instantiation supports batch-NP languages and can be based on *standard assumptions*. Recall that while SNARGs for all NP languages are not known based on standard assumption, SNARGs for batch-NP (also referred to as BARGs) [25,37,63] are known from standard assumptions [34,37,38,60,80]. BARGs allow a prover to convince a verifier of the validity of k NP statements using a proof that is significantly smaller than the combined witness of the k statements. Our splittable zkSNARG for batch-NP is based on the scheme of [80] that uses composite-order pairing groups and achieves computational soundness under the subgroup-decision assumption.

We first show how to lift the BARGs of [80] to achieve the *perfect zero-knowledge* property. Our construction only makes *black-box* use of cryptography.

Theorem 4 (Black-box Perfect Zero-knowledge BARGs, Informal). *Assuming the hardness of the subgroup decision problem in a composite-order pairing group, there exists a BARG for boolean circuit satisfiability that achieves perfect zero-knowledge and makes only black-box use of the group. The proof*

[3] We use zkSNARK instead of zkSNARG when the underlying scheme is also an *argument of knowledge*.

size is $\mathsf{poly}(\lambda, |C|)$*, the CRS size is* $m^2 \cdot \mathsf{poly}(\lambda)$ *and the verification complexity is* $\mathsf{poly}(\lambda, |C|) + \mathsf{poly}(\lambda, m, n)$*, where* λ *is a security parameter,* $C : \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$ *is the Boolean circuit,* n *is the statement size, and* m *is the number of instances.*

We note that, in general, zkBARGs can be obtained by compiling BARGs with non-interactive zero knowledge proofs. Furthermore, recent works [8,24,27] have shown how to construct NIZKs from BARGs, which in turn, yields zkBARGs from BARGs. While these transformations are generic, they make *non-black-box* use of cryptography in several steps. Theorem 4, in contrast, yields a new and direct construction of zkBARGs based on pairing groups that makes black-box use of the group.

We additionally demonstrate that our zkBARG construction obtained in the above theorem is indeed a splittable zkSNARG for batch-NP. We shall use this to construct ve-HSS schemes that can evaluate SIMD boolean circuits. In particular, we show that when computing the zkBARG for any SIMD computation, the non-linear computation phase can be computed by a circuit of depth $O(D + \log \lambda)$, where D is the depth of the SIMD computation.

Instantiating our compiler in Theorem 3 using (i) semi-honest HSS schemes for NC^1 (assuming DCR [73,76] or LWE [23] or class groups [1]) and $\mathsf{P/poly}$ (assuming iO + OWF [15] or FHE [19,45]) ; and (ii) splittable zkSNARGs for NC^1 and $\mathsf{P/poly}$ (in the generic group model) and SIMD NC^1 and SIMD $\mathsf{P/poly}$ (assuming hardness of the decision subgroup problem) give us our main results in Theorems 1 and 2.

Multi-Client HSS with Verifiable Evaluation. Finally, we extend our results to the *multi-client* setting. Starting from semi-honest public-key HSS schemes[4] with negligible error, we present a compiler for public-key multi-client ve-HSS using Splittable zkSNARGs and obtain the following results:

1. *Semi-Honest Clients:* When clients are semi-honest, we present a compiler for transforming any public-key HSS scheme with at most negligible correctness error into a multi-client ve-HSS using a suitable Splittable zkSNARG, assuming a secure "public-key setup".
   When instantiated with HSS schemes based on DCR [73,76] and class group [1], our compiler yields a fully black-box solution for multi-client ve-HSS, where the public key setup can either be generated by a trusted third party or jointly by the servers using a black-box one-round maliciously secure distributed protocol from [1]. When using other HSS schemes to instantiate our compiler, the setup must be done by a trusted entity. This is because black-box protocols for public-key setup are not known for these other HSS schemes.

[4] There are secret-key and public-key variants of HSS schemes. In the public-key variant, anyone with access to the public-key can generate shares of their secret whereas in the secret-key variant one needs to know the key to generate secret shares.

2. *Malicious Clients:* When the clients are malicious, we instantiate our compiler using HSS schemes based on DCR [73,76] and class groups [1].
   If some clients are malicious, it is crucial to ensure that they send well-formed HSS shares of their respective inputs. In addition to the distributed key setup protocol from [1], in this setting, we also use the black-box, one-round, maliciously secure, input sharing protocol for these HSS schemes from [1] to instantiate our compiler.

1.2 Applications

We now discuss some applications of our ve-HSS constructions.

Private Delegation of Computation. Our notion of ve-HSS provides a black-box solution to the problem of private and verifiable delegation of computation (by a single or multiple clients) to a group of untrusted servers. Our HSS-based approach offers a significant advantage over the FHE-based approaches explored in [2,51]. In the FHE-based approach, to make the final proof publicly verifiable, the client must perform some post-processing on the output sent by the server. In contrast, our approach yields a truly non-interactive solution, where the output computed by the servers is already publicly verifiable and requires no additional intervention from the client. To the best of our knowledge, this yields the first such non-interactive and black-box solution for delegating the computation of *non-cryptographic functions*.

Next, we demonstrate that our approach can also be used to delegate the computation of some *cryptographic functions*, while also being black-box in the cryptographic operations involved in the computation of these functions. Specifically, we highlight the application of our constructions to the problem of privacy-preserving delegation of zkSNARG computations in a black-box manner—an area that has recently garnered significant interest [32,50,51,69,74].

Private Delegation of zkSNARK Computation. Let $\mathcal{R} = \{x, w \mid C(x, w) = 1\}$ be an NP relation defined by a circuit C. Consider a client who wishes to compute a proof for the statement x using a particular zkSNARG proof system Π. We show how if Π is a splittable zkSNARG, then the client can use our single-client ve-HSS scheme—built using Π itself—to outsource this proof computation.

A few remarks are in order: first, we note that our group-based ve-HSS schemes can only support NC^1 circuits (a limitation of known semi-honest HSS schemes). Second, the above solution offers several advantages over recent works [32,50,51,69] that employ general-purpose MPC techniques (or FHE) for private-outsourcing of zkSNARGs: (i) MPC-based approaches [32,50,69] several rounds of the interaction between the servers, with communication growing with the size of the circuit C. In contrast, our approach necessitates *zero* interaction amongst the servers. (ii) As discussed above, unlike the FHE-based approach in [51], the proof computed by the servers in our solution is already publicly verifiable and requires no additional intervention from the client. Moreover, unlike [51], the final zkSNARK generated using our delegation framework does not

reveal to the verifier whether it was generated by the client himself or using a delegation scheme.

Private Delegation of Collaborative zkSNARK Computation. A recent work by Ozdemir and Boneh [74] introduced the notion of *collaborative zkSNARKs* as a general tool for publicly auditable privacy-preserving computations. Collaborative zkSNARKs is a distributed protocol enabling a set of parties, each holding a share of an NP witness, to collectively compute a zero-knowledge succinct non-interactive argument (zkSNARG) for a given statement. Using similar ideas as in the single-client setting, our construction of multi-client ve-HSS can be used by a group of clients to delegate the generation of a *splittable zkSNARG* to a group of untrusted servers.

Ozdemir and Boneh's approach requires communication proportional to the size of the NP relation circuit. In contrast, a key advantage of our approach is that it requires *minimal interaction* between the servers, which are only needed for reconstructing the proof. The clients, meanwhile, only need to interact when computing and transmitting HSS shares of their combined witness to the servers. After this, the servers can independently compute the collaborative zkSNARG.

We refer the reader to Sect. 2.5, for a detailed overview of these applications.

1.3 Related Work

We provide a brief overview of related work.

Verifiable Function Secret Sharing. In a verfiable function secret sharing scheme [10,17,26], the *client* may be malicious, and the servers want the guarantee that the *function shares that they received from the client are well-formed*, thus preventing any attempt by the client to learn unauthorized information about the servers' database. This is quite different from our notion of verifiability that aims to establish the *correctness of evaluation* in the face of malicious servers.

Maliciously Secure Sublinear MPC Based on HSS. A recent work by Abram, Damgård, Orlandi and Scholl [1] presents a black-box construction of maliciously-secure sublinear MPC using HSS. Specifically, they construct two-round sublinear MPC with a public-key setup from DCR-based HSS schemes [73,76] as well as new HSS schemes based on class groups. The public-key setup can be instantiated with a one round distributed protocol (using a trusted setup for RSA parameter generation in the DCR case).

By virtue of being a maliciously-secure MPC protocol, their protocol guarantees correctness of output as long as at least one of the servers is honest. However, it does *not* imply our notion of HSS with verifiable evaluation which requires local and public soundness properties even when all parties are corrupted.

Homomorphic Secret Sharing with Verifiability. Some prior works [28–30,59,72,77–79,81,82] have previously considered different notions of verifiability in homomorphic secret sharing (under the umbrella term "verifiable HSS") for limited class of functions. The constructions proposed in [77–79] are only for

computing a sum or product of all the clients' inputs and the clients are always assumed to be honest. While these results claim to achieve soundness (or verifiability) even when all servers are corrupt, they were later shown to be broken by [59,72]. None of the other results [28–30,81,82] consider soundness when all servers are corrupt. [30,81] present HSS with verifiability of degree-d multiplications, where d depends on the number of servers. Verifiable HSS schemes in [28,29,82] are based on LWE and are designed for different degree polynomials (where the degree of the polynomial is dependent on the number of servers or the security parameter).

2 Technical Overview

As outlined in Sect. 1, our goal in this work is to design a ve-HSS scheme that uses cryptographic operations in a *black-box* manner. In this section, we discuss the main ideas underlying our result.

2.1 Single Client HSS with Verifiable Evaluation

We now describe the ideas underlying the construction of our *HSS with verifiable evaluation*. Throughout this discussion, we will operate in a simplified setting, assuming that there are only two servers[5]. These ideas can be easily generalized to settings with more than two server.

Strawman Approach. A ve-HSS scheme proposes to add "verifiability" to existing HSS schemes. Within Cryptography, zero-knowledge proofs [56] are the most widely used tool for adding "verifiability" to any computation. The celebrated result by Goldreich, Micali, Wigderson (GMW) [55] showcased how any semi-honest multiparty computation protocol can be upgraded to withstand malicious corruptions, by requiring each participant to include a zero-knowledge proof (to attest to their honest behavior) with every message they send. Similarly, the folklore way to add verifiability to delegation of computation via FHE involves asking the server to additionally compute a zero-knowledge proof attesting that the FHE evaluation was done honestly.

This motivates the following natural (GMW-inspired) approach to adding verifiability to existing HSS constructions: the servers each compute a zero-knowledge proof alongside their outputs from the evaluation algorithm, certifying that they executed the evaluation algorithm honestly. Since the evaluation phase must be non-interactive and its output must be sublinear in the size of the function being evaluated, the servers can employ a zero-knowledge proof that is also a succinct non-interactive argument (SNARG) [65,71] that admits a succinct proof and sublinear verification time. This approach, however, inherently

[5] Existing constructions of HSS [1,23,40,73,76] that meet our criteria of correctness (and support more than constant-depth functions, without relying on heavy tools such as iO and FHE) only support two servers. Other known construction of Multiparty HSS [44] only satisfy weak correctness (which is not sufficient for us).

requires non-black-box use of cryptography. Specifically, since the evaluation algorithms of known HSS constructions rely heavily on cryptographic operations, it is unclear if the servers can compute a zero-knowledge SNARG (zkSNARG) to prove the honest execution of these operations without employing non-black-box techniques.

We remark that, other malicious MPC techniques [31,43,52] involving information-theoretic MAC based checks[6] or the use of distributed zero-knowledge proofs [9,20–22] to verify correctness of computation are also not helpful in our setting. The soundness argument in these approaches hold only if a subset of the servers are allowed to be corrupt. In contrast, as discussed in Sect. 1, we demand stronger notions of soundness, that hold even if *all* servers are corrupt.

Our Starting Idea. In order to avoid such non-black-box use of the cryptography, our initial insight is to have the *servers jointly compute a single SNARG proof to prove that the computation is correct*. In particular, instead of proving that each server performed the evaluation honestly, this is a proof attesting that the reconstructed output y is indeed a correct result of evaluating function f on input x.

Now, we need to determine how the servers can compute such a proof. Our second insight is to *utilize HSS itself to compute this proof*. In other words, rather than generating the proofs "externally" to validate the correctness of HSS Eval, we want to use HSS Eval itself to generate the joint proof. We would like to rely on the self-authenticating property of SNARGs to prevent incorrect outputs from being accepted. Implementing this idea, however, is not straightforward:

- A naïve implementation of this approach would entail non-black-box use of the cryptographic operations involved in computing the underlying SNARG proof, contrary to our requirements.
- To prove the security of the resulting ve-HSS scheme, the simulator/ideal adversary in the ideal world must be able to abort the output if a malicious server misbehaves. The standard method to simulate aborts in the GMW paradigm is to check individual proofs sent by the corrupt parties. However, in this approach, since we no longer require servers to attach individual proofs, it is not immediately clear how one would detect and simulate aborts in the ideal world.

Note that even if we are willing to forgo the black-box requirement, implementing the aforementioned idea using non-black-box cryptographic operations of the underlying SNARGs presents a significant obstacle. This is because soundness of all known constructions of SNARGs for NP [4–6,42,49,53,57,58,68,71] are proven in idealized models such as the random oracle model (ROM), generic group model (GGM), or the algebraic group model (AGM). Employing the above non-black-box approach for SNARGs based on ROM or GGM would inevitably

[6] An information-theoretic MAC based approach is also used in the design of maliciously secure sublinear MPC based on HSS [1].

require non-black-box descriptions of these oracles. For SNARGs based on AGM, it remains unclear if the soundness of the SNARG computed using HSS (which also relies on cryptographic assumptions) can be still be established in the AGM model, or if it requires making additional hardness assumptions.

Consequently, a black-box approach appears to be the only viable option for instantiating the above idea. However, it is not immediately clear whether existing HSS schemes can support computation of a SNARG inside them, while still being black-box in the underlying cryptographic operations. Nevertheless, this idea of computing SNARGs using HSS Eval forms the starting point of our approach, and we develop new ideas to overcome these challenges.

Splittable SNARGs. Since all SNARGs rely on cryptographic hardness assumptions, generating a SNARG proof for arithmetic relations involves two types of operations: cryptographic and non-cryptographic (i.e., simple field arithmetic). To compute the SNARG proof within HSS (while being black-box), our next insight is to make use of HSS Eval for the field arithmetic, and devise a separate method for the servers to compute the necessary cryptographic operations *outside* of HSS. However, since these two types of operations within SNARG generation may be highly intertwined, it may not be always be feasible to compute the cryptographic operations in a distributed and non-interactive manner. As such, it is unclear if it is even possible to implement this idea.

Our main contribution lies in showing that this idea is indeed feasible to implement. All we require is for the SNARG at hand to satisfy the following "split-prover" property. Namely, it must be possible to clearly divide the generation of splittable SNARG proofs into two steps:

1. *Low Depth Non-Cryptographic Operations:* The first step must only entail non-cryptographic operations over a ring. The computational depth of this step should be such that this overall computation is within the class of functions that are currently supported by existing HSS constructions.[7]
2. *Linearly Distributable Cryptographic Operations:* The second step of the computation may involve cryptographic operations. However, these operations must be feasible to execute in a distributed manner without requiring interaction. Specifically, considering that the first step is computed using HSS Eval, the servers will possess additive shares of the output from this phase. Using these additive shares, each server should be capable of independently performing certain operations and generating a partial SNARG proof. We will sometimes refer to these partial proofs as shares of the proof. Importantly, there must exist an efficient and simple algorithm to reconstruct the final proof using the partial proofs computed by all servers.

We refer to such SNARGs that satisfy these two properties as *splittable SNARGs*. Looking ahead, we will also require splittable SNARGs to satisfy a third property - a weaker form of the well-studied notion called *unambiguity* [75].

[7] Most existing HSS schemes support evaluating a limited class of circuits e.g., branching programs [1,14,16,18,23,44,47,73,76] and low-degree multi-variate polynomials [13,33,40,44,48,61,66].

We discuss this in more detail towards the end of this subsection. In subsequent subsections, we demonstrate how some existing SNARGs do satisfy these properties.

Candidate Approach. Given a splittable SNARG, as described above, and any HSS scheme with negligible correctness error, the servers run the evaluation algorithm in our ve-HSS scheme as follows:

- Given HSS shares of the input (say $x = (x_{\mathsf{priv}}, x_{\mathsf{pub}})$), the servers execute the evaluation algorithm of the underlying HSS scheme (say $\mathsf{HSS.Eval}$) to compute additive shares of $y = f(x)$.
- Let $\mathcal{R}_{\mathsf{local}} : \{(f, x, y) \mid \mathsf{st},\ f(x) = y\}$ be a deterministic relation, i.e., $\mathcal{R}_{\mathsf{local}} \in \mathsf{P}$. Using HSS shares of x, the servers use HSS Eval to compute additive shares of the output of the first part of the computation (i.e., involving low depth non-cryptographic operations) required for generating a SNARG for the relation $\mathcal{R}_{\mathsf{local}}$[8].
- Using additive shares of x, y and additive shares of the outcome of the previous computation, the servers then perform the second step of the computation (i.e., linearly distributable cryptographic operations) in SNARG generation to obtain shares of the final SNARG proof.
- The servers send additive shares of y as well as shares of the SNARG proof to the client.

Finally, the client can reconstruct these shares, verify the proof (say π_{local}) w.r.t., x, y and accept y as the correct output only if this proof is successfully verified. It is easy to see that if the servers do not have access to the shares of the output or the proof and do not learn whether or not this proof verified, this approach trivially achieves privacy against malicious servers. This is because, in this non-interactive protocol, the view of the malicious server only consists of the HSS share of x that was provided to it in the input sharing phase. From the privacy guarantee of the underlying HSS scheme, it follows that this share computationally hides the input x. Moreover, local soundness of this construction follows from the soundness of the underlying SNARG.

Public Soundness. While the above approach suffices for local soundness, as previously discussed, we require the vHSS scheme to also satisfy a notion of public soundness. Moreoever, we require a stronger notion of privacy, where the servers can learn the output as well as the outcome of the honest client's verification. Recall that public soundness necessitates that upon observing the output of the evaluation phase, anyone (not solely the client) can be convinced that there exists some x_{local} such that y is a valid output of f given public input x_{pub}. To accomplish this, we can modify the aforementioned protocol as follows: in addition to providing a "local proof" Π_{priv} for the client, as described previously, the servers will also compute a "publicly verifiable SNARG" (say

[8] Note that given x, since the computation of $f(x)$ is deterministic, we do not need to start with HSS shares of y for this computation. It can be assumed to be implicitly computed as part of this computation.

Π_{pub}) for the relation $\mathcal{R}_{\mathsf{pub}} = \{(f, x_{\mathsf{pub}}, y) \mid \exists x_{\mathsf{priv}}, \text{ s.t., } f(x_{\mathsf{pub}}, x_{\mathsf{priv}}) = y\}$ in a similar manner. Shares of Π_{local} will only be sent to the clients, while the shares of the output y and this publicly verifiable proof Π_{pub} will be published publicly. The client will accept the output if and only if both Π_{local} and Π_{pub} verify. Note that unlike $\mathcal{R}_{\mathsf{local}}$, $\mathcal{R}_{\mathsf{pub}}$ is not in P. Therefore, while it suffices to use a SNARG for P for the computation of Π_{local}, we crucially require a SNARG for NP for the computation of Π_{pub}.

Privacy. Recall that our privacy notion requires that the view of the malicious server, consisting of its input share and the decision of the Verify algorithm, can be simulated solely based on the function output without the client's input. This mirrors the requirement in any MPC protocol achieving security with abort. This strong privacy guarantee is crucial to prevent an adversary from causing selective aborts based on x_{priv} (see [67], for a detailed discussion on this issue in MPC protocols). An observant reader might notice that achieving this notion of privacy is not as straightforward as before. This presents the following challenges:

Challenge 1: Since we allow for malicious corruptions of one of the servers, the shares of the candidate y and Π_{pub} computed in the evaluation phase are not guaranteed to have been honestly computed.[9] This *incorrect* output and proof could potentially leak information about the private input.

Addressing this Challenge: We observe that unlike in an interactive protocol, where the computation performed by the honest parties can be influenced by the strategy adopted by the adversary, this is not feasible in a non-interactive protocol. Since our evaluation phase in a non-interactive, the shares of y and Π_{pub} output by the honest party, cannot be influenced by the adversary. Therefore, an incorrect output y-which can only result from the adversary sending an incorrect share of y-does not compromise privacy.

Furthermore, since Π_{pub} is publicly revealed, ensuring privacy necessitates the use of a zero-knowledge SNARG (zkSNARG) to generate Π_{pub}. We assume that the randomness used to generate the zkSNARG Π_{pub} is sampled and HSS-shared by the client with the servers in the input sharing phase. Given HSS shares of this randomness, the computation carried out by the servers in the evaluation phase is deterministic.

Challenge 2: A rushing adversary might opt to wait for the honest server to publish their additive share of y and Π_{pub}, and then adjust their output shares of y, Π_{local} and Π_{pub}, accordingly. To prove security, we require the simulator/ideal adversary in the ideal world to be able to efficiently simulate the joint view of the adversarial server and the output of the honest client. However, if the adversary chooses to transmit malformed shares of the proofs Π_{local}, Π_{pub} (while still providing an honestly computed share of y), it becomes unclear how the simulator in the ideal world can accurately predict whether the resulting reconstructed proofs will verify with respect to x_{priv}, x_{pub} and an honestly computed y (i.e., it cannot simulate the output of the honest client).

[9] Since our evaluation phase is non-interactive, the servers have no way to verify that this output and proof were honestly generated before reconstructing them.

Note that since Π_{pub} is publicly reconstructed, the simulator in the ideal world can reconstruct this proof on its own and check whether it verifies w.r.t. the reconstructed y. However, since it does not have x_{local} and Π_{local} (i.e., the output of the honest client), the same strategy cannot be used for deciding the outcome of Π_{local}, which is verified w.r.t. x_{local}.

Addressing this Challenge: As discussed earlier, this issue does not arise in a GMW-style approach [55], where each server provides an individual proof confirming honest behavior. This is because each of these proofs can be independently verified with respect to a *public* state. In our setting however, we have a jointly computed proof that needs to be verified w.r.t. a *private* state. Nevertheless, in order to simulate the outcome of Verify, we would like our simulator to perform a similar local verification based on the proof share of the corrupt sever.

To enable this, we require the splittable zkSNARG used for generating Π_{local} to satisfy an additional property, which we term *distributed-prover robust verification (DP-robust verification)*. Informally speaking, this property dictates that given a fixed (and honestly generated) share of Π_{local} computed by the honest server (using HSS shares of x_{priv}), a computationally bounded malicious server can (w.h.p.) only find a unique share of Π_{local}, such that the reconstructed Π_{local} verifies with respect to x_{priv}, y and x_{pub}. Using this property, the simulator in the security argument can determine the output of the honest client based on whether (1) Π_{pub} verifies; and (2) the share of Π_{local} provided by the adversarial server matches the one it should have computed given its share of x_{priv}.

Summary. We now summarize how we prove security of our proposed construction. As discussed earlier, local and public soundness follow from the soundness of the underlying zkSNARGs. To prove privacy, our simulator in the ideal world will proceed as follows:

1. The simulator queries the ideal functionality to learn y. In this non-interactive protocol, the adversary can always learn the correct output.
2. It computes HSS shares of $x_{\mathsf{priv}} = 0$ and some randomness to be used for generating π_{pub}. Indistinguishability of these shares and the honestly generated shares sent by an honest client in the real world follows from privacy of the underlying HSS scheme.
3. It uses the simulator of the underlying zkSNARG to simulate an accepting proof Π_{pub}.
4. The simulator computes the "expected" shares of y (say $y^{\mathcal{A}}$), Π_{local} (say $\Pi^{\mathcal{A}}_{\mathsf{local}}$) and Π_{pub} (say $\Pi^{\mathcal{A}}_{\mathsf{pub}}$) using HSS Eval and the HSS shares sent to the adversary.
5. It simulates the shares of y, Π_{pub} sent by the honest server as follows: $y^{\mathcal{H}} = y - y^{\mathcal{A}}$ and $\Pi^{\mathcal{H}}_{\mathsf{pub}} = \Pi_{\mathsf{pub}} - \Pi^{\mathcal{A}}_{\mathsf{pub}}$. It is easy to see that $y^{\mathcal{H}}$ is identical to the share sent by an honest server in the real world. Indistinguishability of $\Pi^{\mathcal{H}}_{\mathsf{pub}}$ from that sent by the honest server in the real world follows from zero-knowledge of the underlying zkSNARG.
6. Finally, upon receiving the shares $\bar{y}^{\mathcal{A}}, \bar{\Pi}^{\mathcal{A}}_{\mathsf{local}}, \bar{\Pi}^{\mathcal{A}}_{\mathsf{pub}}$ sent by the adversarial server, the simulator checks:

- If $y^{\mathcal{A}} \neq \bar{y}^{\mathcal{A}}$, then from soundness of the underlying zkSNARGs, it follows that the adversary will not be able to produce shares such that the local proof verifies. And therefore the output of the honest client in this case must be $\perp$.
- If $y^{\mathcal{A}} = \bar{y}^{\mathcal{A}}$, but $\bar{\Pi}^{\mathcal{A}}_{\mathsf{local}} \neq \Pi^{\mathcal{A}}_{\mathsf{local}}$, then from DP-robust verification of the underlying zkSNARG, it follows that the output of the honest client in this case must be $\perp$.
- If $y^{\mathcal{A}} = \bar{y}^{\mathcal{A}}$ and $\bar{\Pi}^{\mathcal{A}}_{\mathsf{local}} = \Pi^{\mathcal{A}}_{\mathsf{local}}$, then the simulator checks if $\bar{\Pi}^{\mathcal{A}}_{\mathsf{pub}} + \Pi^{\mathcal{H}}_{\mathsf{pub}}$ is an accepting proof. If so, the output of the honest clients must be y, else it must be $\perp$.

This concludes a proof sketch for our proposed construction.

DP-Robust Verification. A few remarks on DP-robust verification are in order:

- *Robust verification implies DP-robust verification:* Looking ahead in Sect. 3, we first introduce a notion called *robust verification* (a single-prover analogue of DP-robust verification) and then demonstrate that this notion implies DP-robust verification. At a high level, we say that a zkSNARG satisfies robust verification if, given a valid statement x and witness w of choice, an adversary cannot, with high probability, find a non-zero additive error ϵ such that, if Π is an honestly generated accepting proof for x, then $\Pi + \epsilon$ is also an accepting proof for x.
- *Robust Verification vs Unambiguity:* It is worth noting that robust verification represents a weaker variant of a well-studied notion [75] called unambiguity of a proof system, which demands that a computationally bounded adversary cannot produce an accepting proof different from the "true/prescribed proof" even for true statements. This notion has recently found application in connection to PPAD hardness [7,35,36,46,62,64,70]. However, achieving this property is challenging and most proof systems (including the ones we will instantiate our approach with) do not satisfy this property. Fortunately, we only require a weaker variant (as described above) of this property. It is easy to see that a proof system that satisfies the standard notion of unambiguity will also satisfy robust verification. However, the inverse implication does not hold true.
- *Native Robust verification in randomized zkSNARGs:* We are only able to prove robust verification natively for randomized variants of the splittable zkSNARGs that we use to instantiate our approach. Therefore, even though Π_{local} could potentially have been generated using a deterministic-prover zkSNARG, we will compute it using a randomized-prover zkSNARG.
- *Instantiations of Splittable zkSNARGs:* We observe that all linear PCP-based zkSNARGs [6] satisfy the structural properties of splittable zkSNARGs. Specifically, the prover algorithm in these zkSNARGs can be divided into two steps: low-depth non-cryptographic operations and linearly distributable cryptographic operations. Additionally, we show that one such zkSNARG [58] also satisfies robust verification. Determining whether other instantiations

of linear PCP-based zkSNARGs satisfy robust verification remains an open question. Furthermore, we present a zero-knowledge variant of the Waters-Wu [80] BARG scheme, and demonstrate that it satisfies all the structural properties of splittable zkSNARGs and has robust verification.

– *DP-robust verification using information-theoretic MACs:* It is natural to ask whether one can leverage information-theoretic message authentication codes (MACs) to achieve DP-robust verification. As a plausible candidate approach, consider a scenario where the servers possess HSS shares of a key for an information-theoretic MAC scheme. Alongside computing shares of the output, local, and public proofs, the servers also compute shares of a MAC on the local proof by exploiting the structural properties of a Splittable SNARG. Specifically, the servers compute a MAC on the output of the low-depth non-cryptographic computation and then exploit the linearity of the distributable cryptographic operations to compute share of a MAC on the local proof. Unforgeability of the MAC scheme (with sufficient repetitions) now ensures that the corrupt party can w.h.p. find only a unique share of the local proof and a unique share of the MAC on the local proof such that the following property is satisfied: when combined with the MAC share and proof share computed by the honest server, the reconstructed MAC verifies with respect to the secret shared MAC key. While this serves as a generic approach to ensure DP-robust verification, we show that some SNARGs inherently satisfy the robust verification property; without any modifications. In particular, this guarantees robust verification without requiring any private state (like the MAC key) which might be a property of independent interest.

2.2 Groth16 ZkSNARKs [58] Are Splittable

[58] (henceforth referred to as the Groth16 zkSNARK) is a constant-sized zkSNARK, wherein the final proof consists of only three group elements. This scheme has perfect zero-knowledge, while its soundness holds in the generic group model (GGM). We observe that this scheme satisfies robust verification and all the other properties that we require from an splittable zkSNARK. In this section, we briefly recall this scheme and then discuss why it can be used to instantiate our approach.

Overview of Groth16 zkSNARKs. Let $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2, g_T)$ be bilinear groups such that $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ are groups of prime order p with generators g_1, g_2, g_T respectively, that satisfy the relation $e(g_1, g_2) = g_T$.

– *Correlated Random Setup (CRS):* The CRS is highly structured and is relation-specific. It comprises of correlated group elements in $\mathbb{G}_1$ and $\mathbb{G}_2$.
– *Proof Generation:* The relation being proven is encoded as an instance of Quadratic Arithmetic Program (QAP) [53]. The proof generation algorithm comprises of the following steps:
  1. *Extended Witness Generation:* The initial step involves deriving an "extended witness" by using the statement and witness for the given relation. This extended witness can essentially be perceived as the complete

computation trace or the list of all intermediate wire values computed during the evaluation of the relation circuit using the statement and witness as input.

2. *Additional Constant-Depth Field Operations:* The proof generation algorithm requires performing a constant-depth computation over this extended witness (and some additional randomness sampled by the prover to ensure zero-knowledge).
3. *Combining with CRS Terms:* The output of the previous step is then used to linearly combine (using group exponentiations) the group elements from the CRS to compute the final proof, which comprises of three group elements: $A, C \in \mathbb{G}_1$ and $B \in \mathbb{G}_2$.

– *Verification:* Given the CRS, statement and the proof elements A, B, C, the verifier performs a constant number of pairing operations to verify if the proof is accepting or not.

Splittable zkSNARG Properties. Recall that, in our application, the relation that will be proved using this zkSNARK is of the form: $\{f, x_{\mathsf{pub}}, x_{\mathsf{priv}}, y \mid \mathsf{st.},\ f(x_{\mathsf{pub}}, x_{\mathsf{priv}}) = y\}$, where f, y are always part of the statement and depending on the type of proof (i.e., local/public), x_{priv} is either part of the statement or the witness.

The depth of the computation necessary for generating each element of the extended witness for such relations is less than or equal to the depth of the circuit representation of f. Since the additional computation over the extended witness requires constant depth field operations, overall, the depth of the non-cryptographic operations involved in the generation of Groth16 zkSNARKs is asymptotically equivalent to that of f. However, most existing HSS schemes only support evaluating a restricted class of circuits, called Restricted Multiplication Straight-line (RMS) programs. Informally, an RMS program is an arithmetic circuit over bounded integers, where one of the two inputs of a multiplication gate must always be a circuit input wire. We show in the full version that for any function f that can be computed by an NC^1 circuit, an HSS scheme for RMS programs[10] can compute the non-cryptographic operations involved in Groth16. We refer the reader to the full version for more details.

The only cryptographic operations involved in the generation of this SNARK are the group exponentiations required when combing the CRS terms with the output of the non-cryptographic operations. We now explain (using a simplified example) how the servers can generate shares of A, B, C: Let $a_1, \ldots, a_\ell \in \mathbb{Z}_p$ be some terms output by the non-cryptographic operations in the previous step, and $Q_1, \ldots, Q_\ell \in \mathbb{G}_1$ be some terms in the CRS. In Groth16 zkSNARKs, the proof term A is computes as $A = \prod_1^\ell Q_i^{a_i}$. Given additive shares of $a_1, \ldots, a_\ell$ over $\mathbb{Z}_p$, the j^{th} server can compute $A^j = \prod_1^\ell Q_i^{a_i^j}$, where a_i^j is the share of a_i held by the j^{th} server (for each $j \in \{1, 2\}$). The final A can be reconstructed as

[10] We require the HSS scheme to support evaluating RMS programs over super-polynomially bounded integers. This is true for all HSS schemes with negliglble correctness error.

$A = A^1 \cdot A^2$ using the shares computed by each of the servers[11]. Shares of the other two terms B and C can also be computed in a similar manner. This idea of computing group exponentiations in a distributed manner using additive shares of the exponent has been explored in several prior works (see [74] and references contained therein for more details).

Robust Verification. Given the above discussion, it is clear that Groth16 zkSNARK meets all the structural requirements we demand from splittable zkSNARG. To establish that this proof system is applicable within our framework, all that remains is to demonstrate that it also achieves robust verification.

Given the proof terms A, B, C, and the statement $\mathsf{st} :- (x = (x_{\mathsf{priv}}, x_{\mathsf{pub}}), y)$, the verifier in Groth16 checks the following equation:

$$e(A, B) \stackrel{?}{=} e(Q_1, Q_2) \cdot e(Q_3^{x_{\mathsf{priv}}} \cdot Q_4^{x_{\mathsf{pub}}} \cdot Q_5^{y}, Q_6) \cdot e(C, Q_7)$$

where, $Q_1, \ldots, Q_7$ correspond to some group elements in the CRS that are used for this verification check.

To establish robust verification, we want to show that given some errors $g_1^{\epsilon_A}, g_2^{\epsilon_B}, g_1^{\epsilon_C}$ by the adversary, if $\epsilon_A \neq 0$ or $\epsilon_B \neq 0$ or $\epsilon_C \neq 0$, and (A, B, C) is an accepting proof, then the following is not an accepting proof (w.h.p):

$$\bar{A} = A \cdot g_1^{\epsilon_A} \quad \bar{B} = B \cdot g_2^{\epsilon_B} \quad \bar{C} = C \cdot g_1^{\epsilon_C}$$

The verification check given these malformed proof terms can now be written as

$$\begin{aligned}
& e(A \cdot g_1^{\epsilon_A}, B \cdot g_2^{\epsilon_B}) \stackrel{?}{=} e(Q_1, Q_2) \cdot e(Q_3^{x_{\mathsf{priv}}} \cdot Q_4^{x_{\mathsf{pub}}} \cdot Q_5^{y}, Q_6) \cdot e(C \cdot g_1^{\epsilon_C}, Q_7) \\
\Rightarrow\ & e(A, B) \cdot e(g_1^{\epsilon_A}, B) \cdot e(A, g_2^{\epsilon_B}) \cdot e(g_1^{\epsilon_A}, g_2^{\epsilon_B}) \\
& \stackrel{?}{=} e(Q_1, Q_2) \cdot e(Q_3^{x_{\mathsf{priv}}} \cdot Q_4^{x_{\mathsf{pub}}} \cdot Q_5^{y}, Q_6) \cdot e(C, Q_7) \cdot e(g_1^{\epsilon_C}, Q_7) \\
\Rightarrow\ & e(g_1^{\epsilon_A}, B) \cdot e(A, g_2^{\epsilon_B}) \cdot e(g_1^{\epsilon_A}, g_2^{\epsilon_B}) \stackrel{?}{=} e(g_1^{\epsilon_C}, Q_7) \\
\Rightarrow\ & e(g_1^{\epsilon_A}, B) \cdot e(A, g_2^{\epsilon_B}) \stackrel{?}{=} e(g_1^{\epsilon_C}, Q_7) \cdot e(g_1^{\epsilon_A}, g_2^{\epsilon_B})^{-1}
\end{aligned}$$

We observe that because of the randomness used in the generation of A, B is uniformly sampled from $\mathbb{Z}_p$, these proof terms A and B are uniformly distributed over $\mathbb{G}_1$ and $\mathbb{G}_2$. Moreover, we know that Q_7 is guaranteed to not be g_2^0. Therefore, the probability that an adversarial server who does not have the randomness used in the generation of A and B can find non-zero field elements $\epsilon_A, \epsilon_B, \epsilon_C$ such that the above check verifies, is exponentially small (in the size of the field). As a result, this randomized variant of Groth16 satisfies robust verification. We refer the reader to the full version for a formal proof for why Groth16 zkSNARKs satisfy robust verification.

Recall that in the single-client scenario, for local proof we only need to prove a relation in P. Moreover, since this proof is only sent to the client, one might

[11] Note that this multiplicative reconstruction can also be viewed as additive reconstruction over $\mathbb{Z}_p$ in the exponent of g_1.

intuitively consider using a de-randomized variant of Groth16 zkSNARKs. However, given that our proof of robust verification only holds for the randomized version of Groth16, we must use this variant for local proofs as well. Overall, instantiating our approach from Sect. 2.1 using the Groth16 zkSNARKs gives us a single-client ve-HSS scheme. Since soundness of this proof system holds in the generic group model, our resulting ve-HSS scheme is also secure in the generic group model.

2.3 Splittable Zero-Knowledge BARGs

BARGs (Succinct Non-Interactive Batch Arguments) are SNARGs designed for a specific set of relations where a batch of statements, all requiring verification against the same NP relation, are involved. Thanks to recent advancements [34, 37,38,60,80], BARGs are now known from various standard assumptions, (i.e., without relying on any idealized models). As discussed in Sect. 1.1, [80] (referred to hereafter as Waters-Wu BARGs) are the only known BARGs that avoid non-black-box usage of cryptography, but they do not achieve zero-knowledge. In this section, we discuss how we can adapt the Waters-Wu BARGs to obtain a black-box construction of zero-knowledge BARG (zkBARGs). We then proceed to show that this zkBARG is a splittable zkSNARG and can be used to instantiate our approach.

Recap of Waters-Wu BARGs. We start by recalling the design of Waters-Wu BARGs. At a high level, this BARG scheme is a *commit-and-prove* SNARG that achieves non-adaptive soundness. Let $\mathbb{G}$ and $\mathbb{G}_T$ be symmetric composite order pairing groups of order $N = pq$ for primes p and q. Thus $\mathbb{G} \cong \mathbb{G}_p \times \mathbb{G}_q$ where $\mathbb{G}_p$ is a subgroup of order p generated by $g_p = g^q$ where g is the generator for $\mathbb{G}$, and $\mathbb{G}_p$ is a subgroup of order q generated by $g_q = g^p$. Further, there exists an efficiently computable non-degenerate bilinear map, $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$, such that for all $a, b \in \mathbb{Z}_N$ $e(g^a, g^b) = e(g, g)^{ab}$. The subgroups $\mathbb{G}_p$ and $\mathbb{G}_q$ themselves are orthogonal with respect to the pairing operation, i.e. $e(g_p, g_q) = g_T^0$ where g_T is the generator for $\mathbb{G}_T$. The security of the construction relies on the *decision subgroup assumption* [11] which states that a random element from the subgroup $\mathbb{G}_p$ is indistinguishable from a random element in the full group $\mathbb{G}$.

Let $C : \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$ be the Boolean circuit (composed of only NAND gates) used in the BARG, where the BARG proof is constructed over a batch of T instances $(x_1, \ldots, x_T)$ such that for each $i \in [T]$, there exists witness w_i such that $C(x_i, w_i) = 1$. The CRS is defined to be

$$\mathsf{crs} = \left(\{A_i = g_p^{\alpha_i}\}_{i\in[T]}, A = \prod_{i\in[T]} A_i = g_p^{\sum_{i\in[T]} \alpha_i}, \{B_{i,j} = g_p^{\alpha_i \alpha_j}\}_{i,j\neq i}\right)$$

where the α_is are independently and randomly sampled from $\mathbb{Z}_N$.

– *Algebraic Vector Commitment:* For each wire k in the circuit C, a commitment U_k is computed as $U_k = \prod_{i\in[T]} A_i^{w_{i,k}} \in \mathbb{G}_p$ where $w_{i,k}$ is the k-th wire value in the i-th instance. The number of such commitments produced is equal to the number of wires in the circuit.

- *Wire Validity Check:* To prove that each commitment commits to a binary vector, the prover computes for each wire k, a proof term $V_k = \prod_{i \neq j} B_{i,j}^{(1-w_{i,k})w_{j,k}}$.
- *Gate Validity Check:* To prove that the (NAND) gate computation was done correctly, for every gate ℓ indexed by the two incoming wires k_1 and k_2 and outgoing wire k_3, the prover computes $W_\ell = \prod_{i \neq j} B_{i,j}^{w_{j,k_3} - w_{i,k_1} w_{j,k_2} - 1}$.
- *Verification:* Given crs, the commitments $\{U_k\}_k$, the wire validity proofs $\{V_k\}_k$ and gate validity proofs $\{W_\ell\}_\ell$. The verifier checks for each k, if $e(A, U_k) = e(U_k, U_k) \cdot e(g_p, V_k)$; and for each gate tuple $\ell = (k_1, k_2, k_3)$, if $e(A, U_{k_3}) \cdot e(U_{k_1}, U_{k_2}) = e(A, A) \cdot e(g_p, W)$. Additionally, the verifier recomputes the commitments for the input wires corresponding to the statements to check if they were honestly generated.

Adding Zero-Knowledge. Since the algebraic vector commitments to internal wire values in this scheme are deterministically computed and hence are not *hiding*, it is easy to see that as such these BARGs do not achieve zero-knowledge. Towards adding zero-knowledge to this construction, our first idea is to use a hiding commitment. Such commitments can be obtained by simply adding randomness to the above algebraic vector commitments. In particular, for each wire k in the circuit C, the prover first samples a random $r_k \in \mathbb{Z}_p$ and then computes $U_k = g_p^{r_k} \cdot \prod_{i \in [T]} A_i^{w_{i,k}} \in \mathbb{G}_p$.[12] This randomized U_k now information-theoretically hides the $w_{i,k}$ values.

For *completeness*, we now also need to modify V_k and W_ℓ, in such a way that they can somehow be verified w.r.t. the randomized commitments U_k. Importantly, these modification should not break soundness of this proof system. We propose to modify V_k as follows:

$$V_k = \frac{\prod_{i \neq j} B_{i,j}^{(1-w_{i,k})w_{j,k}} \cdot \prod_{i \in [1,m]} A_i^{r_k(1-2w_{i,k})}}{g_p^{r_k^2}}. \tag{1}$$

Interestingly, in the full version, we show that given the randomized commitments U_k and these V_k terms, we can continue to use the same verification check as in Waters-Wu BARGs for wire validity checks. In the full version, we show that W_ℓ can also be modified in a similar manner, such that, together with the randomized U_k, gate validity checks can be performed in a similar manner as in Waters-Wu BARGs. While these modifications suffice for completeness, we still need to show the resulting BARG achieves soundness and zero-knowledge.

Zero-Knowledge: Next, we show that our modified BARG achieves perfect zero-knowledge. Our first observation here is that, each U_k is identically distributed to an element that is obtained by first unformly sampling a random value (say z_k) from $\mathbb{Z}_N$ and then computing $g_p^{z_k}$. Moreover, given these values of U_k and the crs,

[12] Since the statement in public and need not be hidden, the prover can simply set $r_k = 0$ when computing commitments for the input wires corresponding to the statements.

the remaining proof terms V_k and W_ℓ are uniquely determined by the verification equations for wire-validity and gate-validity checks. Our proof for perfect zero-knowledge, now follows from these observations in a straightforward manner. Formally speaking, a simulator with knowledge of all the α_i terms used for generation of the crs can simulate an accepting proof for any batch of statements as follows: (1) for each wire k, it first samples a random $z_k \in \mathbb{Z}_N$ and then computes $U_k = g_p^{z_k}$. (2) Using knowledge of α_is and the z_k values, it can now compute all the V_k and W_ℓ terms such that all the verification checks go through. It is easy to see that this proof is perfectly indistinguishable from an honestly generation proof for a true batch of statements.

Soundness: Observe that, our idea for adding zero-knowledge to the Waters-Wu BARGs only entails making modifications to the proof generation algorithm. Importantly, we do not require modifying the crs and the verification algorithm. In other words, both the crs and the verification algorithm in this modified construction are identical to that in Waters-Wu BARGs. As a result, soundness of our scheme simply follows from the soundness of Waters-Wu BARGs.

Splittable zkSNARG Properties. Similar to the Groth16 zkSNARKs, the first step in generating our zkBARG, involves computing all wire values corresponding to the statements, followed by the computation of all the exponent terms needed to compute V_k and W_ℓ. For instance, for each V_k, these exponent terms are $(1-w_{i,k})w_{j,k}, r_k(1-2w_{i,k})$ and r_k^2. Since these only involve operations over $\mathbb{Z}_N$, they can be computed in a distributed manner using HSS Eval.

Given the crs terms, all wire values, and these exponent terms, the subsequent step is to compute U_k, V_k, W_ℓ using group exponentiations. As discussed in Sect. 2.2, these operations can be performed in a non-interactive distributed manner, provided additive shares of the wire values and the exponent terms are available (which we obtain from the previous step). This establishes that our zkBARG satisfies all the structural properties of a splittable zkSNARG.

Robust Verification. Finally, we show via a sequence of the following claims that this scheme also has robust verification:

- Firstly, we observe that an adversary cannot add a non-zero error to the *vector commitments corresponding to the statement*. Recall that, since the statements are public, we do not randomize these commitments in our zkBARG. Similar to Water-Wu BARGs, the verifier verifies the correctness of these commitments by recomputing them. Therefore, any deviation from an honestly computed commitment to the statements will result in the proof being rejected.
- Next, we establish that the probability of an adversary successfully choosing a non-zero error on any of the remaining commitments U_k or the V_k terms, while still surpassing the wire-validity checks, is exponentially small. The proof of this claim employs a similar argument to our proof of robust verification for Groth16 zkSNARKs and crucially relies on the fact that the randomness r_k used in the computation of U_k remains unknown to the adversary.

– Lastly, if no errors are introduced on the U_k terms, then there exist unique values of W_ℓ that will allow the gate-validity checks to pass. This ensures that if the adversary sends errors for any of the W_ℓ terms, the verification will fail.

We defer a formal proof to the full version. Instantiating our approach from Sect. 2.1 using this new construction of zkBARGs gives us a single-client ve-HSS scheme for SIMD computations. The security of this scheme holds in the standard model.

2.4 Multi-client HSS with Verifiable Evaluation

So far, our discussion has been limited to the single-client setting. Now, we shift our focus to the scenario where multiple (say n) mutually distrustful clients seek to share their inputs via HSS with the servers to compute a joint function on these inputs. In this section, we now consider the setting where multiple mutually distrusting clients (we consider both consider both semi-honest and malicious client settings) have inputs that they wish to HSS share with the servers to have them compute a joint function on these inputs.

Candidate Approach. In the single-client setting, the local proof assures the client that the output was honestly computed using their input. In the multi-client setting, we require the same assurance for every client who shares their inputs via HSS with the servers. A straightforward way to achieve this would be to have the servers compute multiple local proofs, one for each server. Specifically, the local proof sent to the i^{th} server will pertain to the relation $\mathcal{R}_i = \{x_i, y, f, x_{\mathsf{pub}} : \exists \{x_j\}_{j=1, i\neq j}^{n}, \mathsf{st}., f(x_{\mathsf{pub}}, x_1, \ldots, x_n) = y\}$. Observe that, unlike in the single-client setting, this is no longer a deterministic relation. To ensure privacy (of the other clients' inputs), we must use a zero-knowledge SNARG to generate this proof. The publicly verifiable proofs can be computed using any splittable zkSNARG as in the single-client setting, albeit for the relation $\mathcal{R}_{\mathsf{pub}} = \{y, f, x_{\mathsf{pub}} : \exists \{x_i\}_{i=1}^{n}, \mathsf{st}., f(x_{\mathsf{pub}}, x_1, \ldots, x_n) = y\}$. Together these two types of proofs ensure local and public soundness in the multi-client setting. However, privacy of this candidate approach does not follow as easily as before.

Challenges in the Multi-Client Setting. Unlike in the single-client setting, corrupt clients can also have inputs in the multi-client setting. This presents some unique challenges in this setting.

– *Privacy:* In the single-client setting, the output y is uniquely determined by the input of the honest client. As a result, if the local proof verifies, then the correctness of the output y follows from the soundness of the underlying splittable zkSNARG. However, in the multi-client setting, the output y might not be determined by the input of any single client. Since the statements proved using the local and public proofs consist of only a subset of the inputs, we can no longer rely solely on these proofs to verify the correctness of the reconstructed output.

We explain this using a simple example. Let us assume that we have two clients $\mathsf{C}_0, \mathsf{C}_1$ with one bit input each (let C_i have input bit b_i), who wish to delegate the computation of the product of their inputs using ve-HSS. Let us further assume that the malicious server employs the following strategy: it first waits for the honest server to send its share of the output and then chooses its share of the output adaptively, such that the reconstructed output is always 0. Now consider the scenario, where both b_0 and b_1 are 1 and therefore the correct output is $y = 1$. However, using the aforementioned attack strategy, if the adversarial server succeeds in forcing the reconstructed output to be $y = 0$, it can still potentially generate convincing local proofs. This is because the local proof for C_i only guarantees that there exists b_{1-i} such that the AND of b_i and b_{1-i} is 0.

The issue with our previous approach based on DP-robust verification is that our simulator can only check validity of the proof share sent by an adversarial server for the "correct" y. Here, since the resulting proof maybe w.r.t. a different $\bar{y}$, DP-robust verification does not suffice. We need a stronger version of DP-robust verification in this setting, that also allows the adversary to maul the statement w.r.t. which the proof is generated.

Unfortunately, we do not know how to prove this *strong* DP-robust verification for the splittable zkSNARGs considered in Sect. 2.2 and Sect. 2.3. To circumvent this issue, in the full version, we show how to combine our ideas from the single-client setting with information-theoretic MAC-based ideas from [1] (see Sect. 1.3) to obtain a secure multi-client vHSS.

– *Input Sharing:* Unlike in the single-client setting, where the client can use a *secret-key* HSS scheme to compute shares of its input, in the multi-client setting, it is easier for the clients to compute HSS shares of their respective inputs if they start with a public-key HSS scheme. However, this requires the public key and the requisite evaluation keys to be generated in a secure manner. Moreover, given the outcome of this setup procedure, in case some of the clients may also be malicious, we also need to ensure that they send well-formed HSS shares of their inputs. We address this as follows:

  1. *Semi-Honest Clients:* When clients are semi-honest, our compiler can be used to transform any public-key HSS scheme with at most negligible correctness error into a multi-client ve-HSS using a suitable Splittable zkSNARG, assuming a secure "public-key setup".

     Abram et al. [1] demonstrate that for DCR-based [73,76] and class group-based [1] HSS schemes, there exists a maliciously secure one-round distributed protocol for public-key setup that only requires making a black-box use of cryptography. When instantiated with these HSS schemes, our compiler yields a fully black-box solution for multi-client ve-HSS, where the public key setup can be done jointly by the servers using the distributed protocol from [1].

     Alternatively, we can instantiate our compiler using other HSS schemes. However, since black-box protocols for public-key setup are not known

for these other HSS schemes, the setup in these cases must be done by a trusted entity.[13]

2. *Malicious Clients:* Abram et al. [1] present a black-box, one-round, maliciously secure input sharing protocol (in the random oracle model) for HSS schemes based on DCR [73,76] and class groups [1]. We leverage this (and their distributed protocol for key setup) to demonstrate that our compiler can be instantiated with these HSS schemes to obtain a black-box solution for multi-client ve-HSS.[14]

2.5 Applications

In this section, we discuss applications of our ve-HSS schemes to private delegation of different types of functions.

Delegation of Non-Cryptographic Functions. Our notion of ve-HSS naturally provides a framework for private outsourcing of computation, whether by a single client or a group of clients, to two or more servers. It is easy to see that for non-cryptographic functions, our constructions offer a black-box solution to this problem. As discussed in Sect. 1.2, this offers several advantages over an FHE based delegation.

Delegation of zkSNARK Computation. We now address the problem of private outsourcing of zkSNARK computation to a group of untrusted servers by a single client. Specifically, given a public statement x, consider a client who wishes to obtain a proof (using a specific zkSNARK scheme Π) attesting that it knows some witness w such that $C(x, w) = 1$. Instead of computing the proof itself, the client wants to outsource this computation to a group of untrusted servers. This problem has been explored in several recent works [32,50,51,69].

Since the computation of a zkSNARK is inherently cryptographic, naïvely using any delegation framework for outsourcing this task may lead to non-black-box use of the cryptographic operations involved in generating the zkSNARK, even if the original delegation scheme is black-box in the underlying cryptographic primitives used in its construction. However, we demonstrate that our specific construction of single-client ve-HSS yields a solution for delegating zkSNARK computation that remains black-box in the cryptographic operations used in its generation.

In particular, we observe that if the zkSNARK Π the client wishes to employ is a splittable zkSNARK, then our construction of a single-client ve-HSS provides a fully black-box solution as follows:

1. The client first uses HSS Share to compute shares of $(x_{\mathsf{priv}} = w)$.

[13] This setup could also be implement jointly by the servers using a generic interactive maliciously secure MPC protocol. However, this would result in a non-black-box use of cryptography during the key setup phase.

[14] We note that initializing our compiler with other HSS schemes, for which such black-box protocols for public-key setup and input sharing are not currently known, does not result in a fully black-box solution.

2. The client then sends $(x_{\mathsf{pub}} = x)$ along with these HSS shares of x_{priv} to the servers and instructs them to compute C using our single-client ve-HSS (instantiated with Π).
3. Recall that, in our ve-HSS construction, the output of Rec algorithm contains the output $C(x, w) = 1$ a local proof Π_{local} and a public proof Π_{pub}. Here Π_{pub} is a zkSNARK proof for the relation $\mathcal{R}_{\mathsf{pub}} = \{(C, x = x_{\mathsf{pub}}, 1) \mid \exists w = x_{\mathsf{priv}}, \text{s.t.,} C(x, w) = 1\}$. This corresponds exactly to the zkSNARK that the client wanted to delegate in the first place, so the servers in this construction simply output their respective shares of Π_{pub} and ignore the shares of the local proofs.

This yields a fully non-interactive solution for the problem of private delegation of zkSNARK computation. As discussed in Sect. 1.2, in contrast, prior MPC-based approaches [32,50,69] require a large amount of communication amongst the servers. An advantage of our solution over the FHE based approach proposed in [51] is that the servers can independently compute shares of the delegated zkSNARK proof, without requiring additional intervention from the client.

Delegation of Collaborative zkSNARK Computation. Next, we consider the setting where a group of mutually distrustful clients wish to *jointly* compute a zkSNARK for some statement x using their combined witnesses $(w_i)_{i\in[n]}$ for the relation: $\mathcal{R} : \{(C, x) \mid \exists (w_i)_{i\in[n]}, \mathsf{st.}, C(x, (w_i)_{i\in[n]}) = 1\}$. Such zkSNARKs were recently introduced in [74], and are referred to as Collaborative zkSNARKs.

We demonstrate that our construction for multi-client ve-HSS, yields a black-box approach for private delegation of splittable collaborative zkSNARKs. This can essentially be viewed as a multi-client analogue of the previous application. While the main ideas used for enabling this application are similar to those used in the previous application, there are some distinctions:

1. Unlike in the single-prover setting, where the prover knows whether it has a valid witness corresponding to the given statement and relation, in collaborative zkSNARKs, no individual party owns the entire witness. Consequently, the clients do not know if the statement is true or not. As a result, the definition of zero-knowledge is slightly different in collaborative zkSNARKs. Specifically, in this setting, privacy of the honest clients' witnesses must be guaranteed irrespective of whether statement is true or false.
   When using our multi-client ve-HSS scheme for delegating the computation of collaborative zkSNARKs, we propose the following approach to always ensure privacy of the honest clients' witnesses: similar to our previous application, upon receiving x and HSS shares of the witnesses, the servers compute (using the a simplified version of our multi-client ve-HSS) shares of $C(x, (w_i)_{i\in[n]})$ and the public proof (while ignoring the local proofs). In this application, before revealing their shares of the public proof, we instruct the servers to use HSS Eval to multiply the output of $C(x, (w_i)_{i\in[n]})$ with the public proof. In case $C(x, (w_i)_{i\in[n]}) = 1$, the resulting value corresponds the the desired collaborative zkSNARK. Privacy of the honest clients' witnesses follows

from the zero-knowledge property of the underlying zkSNARK. And in case $C(x, (w_i)_{i\in[n]}) = 0$, above modification ensures that no information (apart from the outcome of $C(x, (w_i)_{i\in[n]})$) about the honest clients' witnesses is revealed.

2. As discussed in Sect. 2.4, we require clients to perform information-theoretic MAC checks in the ve-HSS construction, based on ideas from [1], to ensure that adversarial servers cannot tamper with the inputs of clients and deceive them into accepting an incorrect output. Thus, our collaborative zkSNARK construction requires the servers to also participate in the input-sharing phase to send HSS input shares of their individual MAC keys. The servers then locally perform the MAC check to verify that the relation circuit was evaluated correctly. Since at least one server is assumed to be honest in the collaborative zkSNARK model, the soundness of the MAC check ensures that the statement is indeed verified with the witness shared by clients. Moreover, since the MAC check is run by the servers, the clients are no longer required after the input-sharing phase.

2.6 Full Version

In the remainder of this paper we only include the formal definition of a Splittable zkSNARG, its properties in the distributed prover setting, the definition of a ve-HSS scheme, and our ve-HSS construction. We refer the reader to the full version for the remaining constructions and proofs.

3 Splittable zkSNARG

In this section, we first define Splittable zkSNARGs and then state its properties.

Definition 1 (Splittable zkSNARG). *Let $\mathbb{C}_{\mathsf{NL}}$ and $\mathbb{C}_{\mathsf{ZK}}$ be two classes of circuits. A zkSNARG $\mathsf{SNARG} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ is said to be a $\mathbb{C}_{\mathsf{NL}}$-simple Splittable zkSNARG for $\mathbb{C}_{\mathsf{ZK}}$-circuit satisfiability if it additionally satisfies the following properties.*

- ***Split Prover:*** *The prover algorithm can be split into a non-linear phase $\mathsf{Prove}_{\mathsf{NL}}$ and a linear phase $\mathsf{Prove}_{\mathsf{L}}$ such that for any circuit $\mathsf{C} \in \mathbb{C}_{\mathsf{ZK}}$, we have the following.*
  - $\mathsf{Prove}_{\mathsf{NL}}(\mathsf{C}, x, y, w) \to M_{\mathsf{NL}}$ *is a* PPT *algorithm that takes the circuit* C*, statement (x, y) and witness w as input and outputs a tuple M_{NL} over a ring R.*
  - $\mathsf{Prove}_{\mathsf{L}}(\mathsf{crs}, M_{\mathsf{NL}}) =: \pi$ *computes an R-linear map on M_{NL} defined by* crs*.*

  We require that for all $\mathsf{C} \in \mathbb{C}_{\mathsf{ZK}}$

$$\left\{ \pi \;\middle|\; \begin{array}{r} \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, \mathsf{C}) \\ \pi \leftarrow \mathsf{Prove}(\mathsf{crs}, x, y, w) \end{array} \right\} \overset{c}{\approx} \left\{ \pi \;\middle|\; \begin{array}{r} \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, \mathsf{C}) \\ M_{\mathsf{NL}} \leftarrow \mathsf{Prove}_{\mathsf{NL}}(\mathsf{C}, x, y, w) \\ \pi := \mathsf{Prove}_{\mathsf{L}}(\mathsf{crs}, M_{\mathsf{NL}}) \end{array} \right\}$$

  where the ensembles are indexed by (λ, x, y, w) for $\lambda \in \mathbb{N}$ and $\mathsf{C}(x, w) = y$.

- ***Efficiency:*** *A $\mathbb{C}_{\mathsf{NL}}$-simple Splittable zkSNARG should satisfy the following efficiency properties.*
  - $\mathbb{C}_{\mathsf{NL}}$***-simple:*** *We require that for every circuit $\mathsf{C} \in \mathbb{C}_{\mathsf{ZK}}$ there exists a circuit $\mathsf{C}_{\mathsf{NL}} \in \mathbb{C}_{\mathsf{NL}}$ such that for all circuit inputs $\mathbf{x}$ and $\mathbf{w}$, and randomness r we have $\mathsf{C}_{\mathsf{NL}}(x, w, r) = \mathsf{Prove}_{\mathsf{NL}}(\mathsf{C}, x, y, w; r)$ where $y = \mathsf{C}(x, w)$.*
  - ***Succinct Randomness:*** *The length of the random tape required by $\mathsf{Prove}_{\mathsf{NL}}$ is $\mathsf{poly}(\lambda \ell_{\mathsf{inp}}) \cdot o(|\mathsf{C}|)$.*
- ***Robust Verification:*** *For all polynomial sized adversaries $\mathcal{A}$ there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$ and circuits $\mathsf{C} \in \mathbb{C}_{\mathsf{ZK}}$ we have*

$$\Pr\left[\begin{array}{c} \mathsf{C}(x,w) = y \wedge \\ \epsilon \neq 0 \wedge \\ \mathsf{Verify}(\mathsf{crs}, x, y, \overline{\pi}) = 1 \end{array} \middle| \begin{array}{r} \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, \mathsf{C}) \\ (x, y, w, \epsilon) \leftarrow \mathcal{A}(1^\lambda, \mathsf{crs}) \\ \pi \leftarrow \mathsf{Prove}(\mathsf{crs}, x, y, w) \\ \overline{\pi} := \pi + \epsilon \end{array}\right] \leq \mathsf{negl}(\lambda).$$

We next state the completeness and security properties of Splittable zkSNARGs, when the proofs are computed in a distributed manner.

Lemma 1 (DP-Completeness). *Let* SNARG *be a Splittable zkSNARG for $\mathbb{C}_{\mathsf{ZK}}$-circuit satisfiability. Then for all polynomial sized adversaries $\mathcal{A}$ there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$ and circuits $\mathsf{C} \in \mathbb{C}_{\mathsf{ZK}}$ we have*

$$\Pr\left[\begin{array}{c} \mathsf{C}(x,w) = y \wedge \\ \mathsf{Verify}(\mathsf{crs}, x, y, \pi) = 0 \end{array} \middle| \begin{array}{c} (x, y, w, m, \mathcal{I}, [M_{\mathsf{NL}}]_{\mathcal{I}}) \leftarrow \mathcal{A}(1^\lambda, \mathsf{crs}) \\ \forall i \in [1, m],\ [\pi]_i := M_{\mathsf{crs}} \cdot [M_{\mathsf{NL}}]_i \end{array}\right] \leq \mathsf{negl}(\lambda)$$

where $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, \mathsf{C})$, $M_{\mathsf{NL}} \leftarrow \mathsf{Prove}_{\mathsf{NL}}(\mathsf{C}, x, y, w)$, $[M_{\mathsf{NL}}]_{i^*} := M_{\mathsf{NL}} - \sum_{i \in \mathcal{I}} [M_{\mathsf{NL}}]_i$, $\pi := \sum_{i=1}^{m} [\pi]_i$, $m \in \mathbb{Z}^+$ *is the number of distributed provers, $\mathcal{I}$ is a set of $m-1$ corrupt provers, and i^* is the index of the honest prover.*

Lemma 2 (DP-Robust Verification). *Let* SNARG *be a Splittable zkSNARG for $\mathbb{C}_{\mathsf{ZK}}$-circuit satisfiability and let $m \in \mathbb{N}$. Then for all polynomial sized adversaries $\mathcal{A}$ there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$ and circuits $\mathsf{C} \in \mathbb{C}_{\mathsf{ZK}}$ we have*

$$\Pr\left[\begin{array}{c} \mathsf{C}(x,w) = y \wedge \\ \epsilon \neq 0 \wedge \\ \mathsf{Verify}(\mathsf{crs}, x, y, \overline{\pi}) = 1 \end{array} \middle| \begin{array}{r} (x, y, w, m, \mathcal{I}, [M_{\mathsf{NL}}]_{\mathcal{I}}, [\overline{\pi}]_{\mathcal{I}}) \leftarrow \mathcal{A}(1^\lambda, \mathsf{crs}) \\ \forall i \in [1, m],\ [\pi]_i := M_{\mathsf{crs}} \cdot [M_{\mathsf{NL}}]_i \\ \epsilon := \sum_{i=1}^{m} [\overline{\pi}]_i - [\pi]_i \end{array}\right] \leq \mathsf{negl}(\lambda)$$

where $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, \mathsf{C})$, $M_{\mathsf{NL}} \leftarrow \mathsf{Prove}_{\mathsf{NL}}(\mathsf{C}, x, y, w)$, $[M_{\mathsf{NL}}]_{i^*} := M_{\mathsf{NL}} - \sum_{i \in \mathcal{I}} [M_{\mathsf{NL}}]_i$, $[\overline{\pi}]_{i^*} := [\pi]_{i^*}$, $\overline{\pi} := \sum_{i=1}^{m} [\overline{\pi}]_i$, $m \in \mathbb{Z}^+$ *is the number of distributed provers, $\mathcal{I}$ is a set of $m-1$ corrupt provers, and i^* is the index of the honest prover.*

In the full version, we prove that the [58] satisfies the Splittable zkSNARG properties and that the non-linear phase of the prover, for NC^1 circuit satisfiability, can be computed by an RMS program.

We also show that a randomized variant of the Waters-Wu BARG is both zero-knowledge, and additionally meets the requirements for a Splittable zkSNARG. The non-linear phase of the zero-knowledge BARG, for SIMD-NC^1 circuit satisfiability, can be computed by an RMS program.

4 HSS with Verifiable Evaluation

In this section we first define HSS with verifiable evaluation and then present our construction in Fig. 2.

$\mathsf{Privacy}^0_{\mathcal{A},\mathsf{Sim},\mathsf{C}}(1^\lambda)$:

1. Compute $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, \mathsf{C})$ and send crs to $\mathcal{A}$.
2. Receive $(\mathcal{I}, \mathbf{x}_{\mathsf{pub}}, \mathbf{x})$ from $\mathcal{A}$. Output 0 if $\mathcal{I} \not\subset [1, m]$ or if $|\mathcal{I}| = m$.
3. Compute
$$\left(\langle \mathbf{x} \rangle_1, \ldots, \langle \mathbf{x} \rangle_m\right) \leftarrow \mathsf{Share}(\mathsf{crs}, \mathbf{x})$$
and send $\langle \mathbf{x} \rangle_{\mathcal{I}}$ to $\mathcal{A}$.
4. Receive $\overline{[\mathsf{out}]}_{\mathcal{I}}$ from $\mathcal{A}$. Compute
$$\begin{aligned} &\forall i \in [1, m] \setminus \mathcal{I}, \\ &[\mathsf{out}]_i := \mathsf{Eval}(\mathsf{crs}, i, \mathbf{x}_{\mathsf{pub}}, \langle \mathbf{x} \rangle_i), \\ &(\overline{\mathbf{y}}, \overline{\pi_{\mathsf{loc}}}, \overline{\pi_{\mathsf{pub}}}) := \mathsf{Recon}(\\ &\qquad \mathsf{crs}, \overline{[\mathsf{out}]}_{\mathcal{I}}, [\mathsf{out}]_{[1,m]\setminus\mathcal{I}}), \\ &b \leftarrow \mathsf{LVerify}(\mathsf{crs}, \mathbf{x}_{\mathsf{pub}}, \mathbf{x}, \overline{\mathbf{y}}, \overline{\pi_{\mathsf{loc}}}). \end{aligned}$$
5. Send $(b, \overline{\pi_{\mathsf{pub}}})$ to $\mathcal{A}$ and output b' returned by $\mathcal{A}$.

$\mathsf{Privacy}^1_{\mathcal{A},\mathsf{Sim},\mathsf{C}}(1^\lambda)$:

1. Compute $(\mathsf{crs}, \mathsf{st}) \leftarrow \mathsf{Sim}(1^\lambda, \mathsf{C})$ and send crs to $\mathcal{A}$.
2. Receive $(\mathcal{I}, \mathbf{x}_{\mathsf{pub}}, \mathbf{x})$ from $\mathcal{A}$. Output 0 if $\mathcal{I} \not\subset [1, m]$ or if $|\mathcal{I}| = m$.
3. Compute
$$(\langle \mathbf{x} \rangle_{\mathcal{I}}, \mathsf{st}_s) \leftarrow \mathsf{Sim}(\mathsf{st}, \mathcal{I}, |\mathbf{x}|)$$
and send $\langle \mathbf{x} \rangle_{\mathcal{I}}$ to $\mathcal{A}$.
4. Receive $\overline{[\mathsf{out}]}_{\mathcal{I}}$ from $\mathcal{A}$. Compute
$$(b, \overline{\pi_{\mathsf{pub}}}) \leftarrow \mathsf{Sim}\left(\mathsf{st}, \mathbf{x}_{\mathsf{pub}}, \mathsf{C}(\mathbf{x}_{\mathsf{pub}}, \mathbf{x}), \overline{[\mathsf{out}]}_{\mathcal{I}}\right).$$
5. Send $(b, \overline{\pi_{\mathsf{pub}}})$ to $\mathcal{A}$ and output b' returned by $\mathcal{A}$.

Fig. 1. ve-HSS privacy experiment.

Definition 2 (HSS with verifiable evaluation). *Let m be a positive integer and $\mathbb{C}$ be a class of boolean circuits. An m-server HSS with verifiable evaluation (ve-HSS) scheme for $\mathbb{C}$ is a tuple of algorithms* $\mathsf{ve\text{-}HSS} = (\mathsf{Setup}, \mathsf{Share}, \mathsf{Eval}, \mathsf{Recon}, \mathsf{LVerify}, \mathsf{PVerify})$ *with the following syntax.*

- $\mathsf{Setup}(1^\lambda, \mathsf{C}) \to \mathsf{crs}$ *is a* PPT *algorithm that takes the description of a circuit* $\mathsf{C} \in \mathbb{C}$ *as input and outputs the common reference string* crs.
- $\mathsf{Share}(\mathsf{crs}, \mathbf{x}) \to (\langle \mathbf{x} \rangle_1, \ldots, \langle \mathbf{x} \rangle_m)$ *is a* PPT *algorithm that takes as input* crs *and private input* $\mathbf{x} \in \mathcal{M}^*$ *and outputs a sharing of the inputs* $\langle \mathbf{x} \rangle$.
- $\mathsf{Eval}(\mathsf{crs}, i, \mathbf{x}_{\mathsf{pub}}, \langle \mathbf{x} \rangle_i) =: [\mathsf{out}]_i$ *is a polynomial time algorithm that takes* crs, *a server index* i, *the public input* $\mathbf{x}_{\mathsf{pub}} \in \mathcal{M}^*$ *and the share of the* i*-th server and computes a share of the output* $[\mathsf{out}]_i$.
- $\mathsf{Recon}(\mathsf{crs}, [\mathsf{out}]_1, \ldots, [\mathsf{out}]_m) =: (\mathbf{y}, \pi_{\mathsf{loc}}, \pi_{\mathsf{pub}})$ *is a polynomial time algorithm that takes* crs *and the shares of the output and reconstructs it to compute the circuit output* $\mathbf{y} \in \{0,1\}^*$ *and proofs* π_{loc} *and* π_{pub}.

Public Parameters. An m-server HSS scheme HSS and Splittable zkSNARG SNARG. For a circuit C, we use $\mathsf{C}_{\mathsf{loc}}$ and $\mathsf{C}_{\mathsf{pub}}$ to denote the circuit satisfiability instances for the local and public proof. We use $\mathsf{C}^{\mathsf{loc}}_{\mathsf{NL}}$ and $\mathsf{C}^{\mathsf{pub}}_{\mathsf{NL}}$ to denote the circuits that compute the non-linear phase $\mathsf{Prove}_{\mathsf{L}}$ of the Splittable zkSNARG for $\mathsf{C}_{\mathsf{loc}}$ and $\mathsf{C}_{\mathsf{pub}}$ respectively.

$\mathsf{Setup}(1^\lambda, \mathsf{C})$

$\mathsf{crs}_{\mathsf{loc}} \leftarrow \mathsf{SNARG.Setup}(1^\lambda, \mathsf{C}_{\mathsf{loc}})$
$\mathsf{crs}_{\mathsf{pub}} \leftarrow \mathsf{SNARG.Setup}(1^\lambda, \mathsf{C}_{\mathsf{pub}})$
$\mathsf{crs} := (\mathsf{crs}_{\mathsf{loc}}, \mathsf{crs}_{\mathsf{pub}}, \mathsf{C})$
return crs

$\mathsf{Share}(\mathbf{x})$

$r_{\mathsf{loc}}, r_{\mathsf{pub}} \leftarrow \mathcal{M}^{\ell_{\mathsf{rand}}}$
$(\langle \mathbf{x} \rangle, \langle r_{\mathsf{loc}} \rangle, \langle r_{\mathsf{pub}} \rangle) \leftarrow$
$\quad \mathsf{HSS.Share}(1^\lambda, \mathbf{x}, r_{\mathsf{loc}}, r_{\mathsf{pub}})$
return $(\langle \mathbf{x} \rangle, \langle r_{\mathsf{loc}} \rangle, \langle r_{\mathsf{pub}} \rangle)$

$\mathsf{Recon}([\mathsf{out}])$

parse $[\mathsf{out}] = ([\mathbf{y}], [\pi_{\mathsf{loc}}], [\pi_{\mathsf{pub}}])$
$\mathbf{y} := \sum_{i=1}^{m} [\mathbf{y}]_i$
$\pi_{\mathsf{loc}} := \sum_{i=1}^{m} [\pi_{\mathsf{loc}}]_i$
$\pi_{\mathsf{pub}} := \sum_{i=1}^{m} [\pi_{\mathsf{pub}}]_i$
$\mathsf{out} := (\mathbf{y}, \pi_{\mathsf{loc}}, \pi_{\mathsf{pub}})$
return out

$\mathsf{Eval}(\mathsf{crs}, i, \mathbf{x}_{\mathsf{pub}}, \langle \mathbf{x} \rangle_i, \langle r_{\mathsf{loc}} \rangle_i, \langle r_{\mathsf{pub}} \rangle_i)$

parse $\mathsf{crs} = (\mathsf{crs}_{\mathsf{loc}}, \mathsf{crs}_{\mathsf{pub}}, \mathsf{C})$
$[\mathbf{y}]_i := \mathsf{HSS.Eval}(i, \mathsf{C}, \mathbf{x}_{\mathsf{pub}}, \langle \mathbf{x} \rangle_i)$
$[M_{\mathsf{loc}}]_i := \mathsf{HSS.Eval}($
$\quad i, \mathsf{C}^{\mathsf{loc}}_{\mathsf{NL}}, \mathbf{x}_{\mathsf{pub}}, \langle \mathbf{x} \rangle_i, \langle r_{\mathsf{loc}} \rangle_i)$
$[\pi_{\mathsf{loc}}]_i := \mathsf{SNARG.Prove}_{\mathsf{L}}(\mathsf{crs}_{\mathsf{loc}}, [M_{\mathsf{loc}}]_i)$
$[M_{\mathsf{pub}}]_i := \mathsf{HSS.Eval}($
$\quad i, \mathsf{C}^{\mathsf{pub}}_{\mathsf{NL}}, \mathbf{x}_{\mathsf{pub}}, \langle \mathbf{x} \rangle_i, \langle r_{\mathsf{pub}} \rangle_i)$
$[\pi_{\mathsf{pub}}]_i := \mathsf{SNARG.Prove}_{\mathsf{L}}($
$\quad \mathsf{crs}_{\mathsf{pub}}, [M_{\mathsf{pub}}]_i)$
$[\mathsf{out}]_i = ([\mathbf{y}]_i, [\pi_{\mathsf{loc}}]_i, [\pi_{\mathsf{pub}}]_i)$
return $[\mathsf{out}]_i$

$\mathsf{LVerify}(\mathsf{crs}, \mathbf{x}_{\mathsf{pub}}, \mathbf{x}, \mathbf{y}, \pi_{\mathsf{loc}})$

parse $\mathsf{crs} = (\mathsf{crs}_{\mathsf{loc}}, \mathsf{crs}_{\mathsf{pub}}, \mathsf{C})$
$b := \mathsf{SNARG.Verify}($
$\quad \mathsf{crs}_{\mathsf{loc}}, \mathbf{x}_{\mathsf{pub}}, \mathbf{x}, \mathbf{y}, \pi_{\mathsf{loc}})$
return b

$\mathsf{PVerify}(\mathsf{crs}, \mathbf{x}_{\mathsf{pub}}, \mathbf{y}, \pi_{\mathsf{pub}})$

parse $\mathsf{crs} = (\mathsf{crs}_{\mathsf{loc}}, \mathsf{crs}_{\mathsf{pub}}, \mathsf{C})$
$b := \mathsf{SNARG.Verify}(\mathsf{crs}_{\mathsf{pub}}, \mathbf{x}_{\mathsf{pub}}, \mathbf{y}, \pi_{\mathsf{pub}})$
return b

Fig. 2. ve-HSS construction from HSS and Splittable zkSNARG.

- $\mathsf{LVerify}(\mathsf{crs}, \mathbf{x}_{\mathsf{pub}}, \mathbf{x}, \mathbf{y}, \pi_{\mathsf{loc}}) \rightarrow b$ *is a* PPT *algorithm that takes* crs*, the circuit inputs* $\mathbf{x}_{\mathsf{pub}}$ *and* $\mathbf{x}$*, the circuit output* $\mathbf{y}$ *and the local proof* π_{loc}*, and outputs a bit* b *indicating if the proof is accepted or rejected.*
- $\mathsf{PVerify}(\mathsf{crs}, \mathbf{x}_{\mathsf{pub}}, \mathbf{y}, \pi_{\mathsf{pub}}) \rightarrow b$ *is a* PPT *algorithm that takes* crs*, the public inputs* $\mathbf{x}_{\mathsf{pub}}$*, the circuit output* $\mathbf{y}$ *and the public proof* π_{pub}*, and outputs a bit* b *indicating if the proof is accepted or rejected.*

A ve-HSS scheme should satisfy the following properties.

- ***Efficiency:*** *The ve-HSS scheme is said to be efficient if it has linear reconstruction, succinct shares and succinct verification as defined below.*
  - ***Linear Reconstruction:*** Recon *is linear in* $[\mathsf{out}]_1, \ldots, [\mathsf{out}]_m$.
  - ***Succinct Shares:*** *The runtime of* Share *is* $\mathsf{poly}(m\lambda|\mathbf{x}|) \cdot o(|\mathsf{C}|)$ *and the output length of* Eval *is* $\mathsf{poly}(\lambda|\mathbf{y}|) \cdot o(|\mathsf{C}|)$.
  - ***Succinct Verification:*** *The runtime of* $\mathsf{LVerify}$ *and* $\mathsf{PVerify}$ *are* $\mathsf{poly}(\lambda \cdot (|\mathbf{x}_{\mathsf{pub}}| + |\mathbf{x}| + |\mathbf{y}|)) \cdot o(|\mathsf{C}|)$ *and* $\mathsf{poly}(\lambda \cdot (|\mathbf{x}_{\mathsf{pub}}| + |\mathbf{y}|)) \cdot o(|\mathsf{C}|)$, *respectively.*
- ***Correctness:*** *For all polynomial sized adversaries* $\mathcal{A}$ *there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$ *and circuits* $\mathsf{C} \in \mathbb{C}$ *we have*

$$\Pr\left[\begin{array}{c}\mathbf{y} \neq \mathsf{C}(\mathbf{x}_{\mathsf{pub}}, \mathbf{x}) \ \vee \ \mathsf{LVerify}(\mathsf{crs}, \mathbf{x}_{\mathsf{pub}}, \mathbf{x}, \mathbf{y}, \pi_{\mathsf{loc}}) = 0 \vee \\ \mathsf{PVerify}(\mathsf{crs}, \mathbf{x}_{\mathsf{pub}}, \mathbf{y}, \pi_{\mathsf{pub}}) = 0\end{array}\right] \leq \mathsf{negl}(\lambda)$$

  where the probability is over $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, \mathsf{C})$, $(\mathbf{x}_{\mathsf{pub}}, \mathbf{x}) \leftarrow \mathcal{A}(\mathsf{crs})$, $(\langle\mathbf{x}\rangle_1, \ldots, \langle\mathbf{x}\rangle_m) \leftarrow \mathsf{Share}(\mathsf{crs}, \mathbf{x})$, $[\mathsf{out}]_i := \mathsf{Eval}(\mathsf{crs}, i, \mathbf{x}_{\mathsf{pub}}, \langle\mathbf{x}\rangle_i)$ *for each* $i \in [1, m]$, *and* $(\mathbf{y}, \pi_{\mathsf{loc}}, \pi_{\mathsf{pub}}) := \mathsf{Recon}(\mathsf{crs}, [\mathsf{out}]_1, \ldots, [\mathsf{out}]_m)$.
- ***Privacy:*** *There exists a* PPT *simulator* Sim *such that for all polynomial sized adversaries* $\mathcal{A}$ *there exists a negligible function* $\mathsf{negl}(\cdot)$ *and for all* $\lambda \in \mathbb{N}$ *and circuits* $\mathsf{C} \in \mathbb{C}$,

$$\left|\Pr\left[\mathsf{Privacy}^0_{\mathcal{A},\mathsf{Sim},\mathsf{C}}(1^\lambda) = 1\right] - \Pr\left[\mathsf{Privacy}^1_{\mathcal{A},\mathsf{Sim},\mathsf{C}}(1^\lambda) = 1\right]\right| \leq \mathsf{negl}(\lambda)$$

  where $\mathsf{Privacy}^b_{\mathcal{A},\mathsf{Sim},\mathsf{C}}(1^\lambda)$ *is defined in Fig. 1.*
- ***Local Soundness:*** *For all polynomial sized adversaries* $\mathcal{A}$ *there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$ *and circuits* $\mathsf{C} \in \mathbb{C}$ *we have*

$$\Pr\left[\mathbf{y} \neq \mathsf{C}(\mathbf{x}_{\mathsf{pub}}, \mathbf{x}) \ \wedge \ \mathsf{LVerify}(\mathsf{crs}, \mathbf{x}_{\mathsf{pub}}, \mathbf{x}, \mathbf{y}, \pi_{\mathsf{loc}}) = 1\right] \leq \mathsf{negl}(\lambda)$$

  where the probability is over $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, \mathsf{C})$, $(\mathbf{x}_{\mathsf{pub}}, \mathbf{x}, \mathsf{st}) \leftarrow \mathcal{A}(\mathsf{crs})$, $(\langle\mathbf{x}\rangle_1, \ldots, \langle\mathbf{x}\rangle_m) \leftarrow \mathsf{Share}(\mathsf{crs}, \mathbf{x})$, $([\mathsf{out}]_1, \ldots, [\mathsf{out}]_m) \leftarrow \mathcal{A}(\mathsf{st}, \langle\mathbf{x}\rangle_1, \ldots, \langle\mathbf{x}\rangle_m)$, *and* $(\mathbf{y}, \pi_{\mathsf{loc}}, \pi_{\mathsf{pub}}) := \mathsf{Recon}(\mathsf{crs}, [\mathsf{out}]_1, \ldots, [\mathsf{out}]_m)$.
- ***Public Soundness:*** *For all polynomial sized adversaries* $\mathcal{A}$ *there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$ *and circuits* $\mathsf{C} \in \mathbb{C}$ *we have*

$$\Pr\left[\begin{array}{c}(\mathsf{C}, \mathbf{x}_{\mathsf{pub}}, \mathbf{y}) \notin \mathsf{SAT} \wedge \\ \mathsf{PVerify}(\mathsf{crs}, \mathbf{x}_{\mathsf{pub}}, \mathbf{y}, \pi_{\mathsf{pub}}) = 1\end{array} \middle| \begin{array}{c}\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, \mathsf{C}) \\ (\mathbf{x}_{\mathsf{pub}}, \mathbf{y}, \pi_{\mathsf{pub}}) \leftarrow \mathcal{A}(\mathsf{crs})\end{array}\right] \leq \mathsf{negl}(\lambda).$$

Theorem 5. *Let* HSS *be an* m*-server HSS scheme for a class of circuits* $\mathcal{P}_{\mathsf{hss}}$ *over* R. *If there exists a* $\mathcal{P}_{\mathsf{hss}}$*-complex Splittable zkSNARG for* $\mathbb{C}$*-circuit satisfiability such that* $\mathbb{C} \subseteq \mathcal{P}_{\mathsf{hss}}$ *then Fig. 2 is an* m*-server ve-HSS scheme for* $\mathbb{C}$ *that is black-box in* HSS *and the linear phase of* SNARG.

Acknowledgements. A majority of this work was done while the first and second authors were at NTT Research. The third and fourth authors were supported in part by NSF CNS-1814919, NSF CAREER 1942789 and Johns Hopkins University Catalyst award. The fourth author was additionally supported in part by JP Morgan Faculty Award, and research gifts from Ethereum, Stellar and Cisco.

References

1. Abram, D., Damgård, I., Orlandi, C., Scholl, P.: An algebraic framework for silent preprocessing with trustless setup and active security. In: Dodis, Y., Shrimpton, T. (eds.) Advances in Cryptology – CRYPTO 2022, Part IV. LNCS, vol. 13510, pp. 421–452. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-15985-5_15
2. Aranha, D.F., Costache, A., Guimarães, A., Soria-Vazquez, E.: HELIOPOLIS: verifiable computation over homomorphically encrypted data from interactive oracle proofs is practical. Cryptology ePrint Archive, Report 2023/1949 (2023). https://eprint.iacr.org/2023/1949
3. Bartusek, J., Garg, S., Masny, D., Mukherjee, P.: Reusable two-round MPC from DDH. In: Pass, R., Pietrzak, K. (eds.) TCC 2020: 18th Theory of Cryptography Conference, Part II. LNCS, vol. 12551, pp. 320–348. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-64378-2_12
4. Bitansky, N., et al.: The hunting of the SNARK. J. Cryptology **30**(4), 989–1066 (2017). https://doi.org/10.1007/s00145-016-9241-9
5. Bitansky, N., Canetti, R., Chiesa, A., Tromer, E.: Recursive composition and bootstrapping for SNARKS and proof-carrying data. In: Boneh, D., Roughgarden, T., Feigenbaum, J. (eds.) 45th Annual ACM Symposium on Theory of Computing, pp. 111–120. ACM Press, Palo Alto, CA, USA (2013). https://doi.org/10.1145/2488608.2488623
6. Bitansky, N., Chiesa, A., Ishai, Y., Ostrovsky, R., Paneth, O.: Succinct non-interactive arguments via linear interactive proofs. In: Sahai, A. (ed.) TCC 2013: 10th Theory of Cryptography Conference. LNCS, vol. 7785, pp. 315–333. Springer, Berlin (2013). https://doi.org/10.1007/978-3-642-36594-2_18
7. Bitansky, N., et al.: PPAD is as hard as LWE and iterated squaring. In: Kiltz, E., Vaikuntanathan, V. (eds.) TCC 2022: 20th Theory of Cryptography Conference, Part II. LNCS, vol. 13748, pp. 593–622. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-22365-5_21
8. Bitansky, N., Kamath, C., Paneth, O., Rothblum, R., Vasudevan, P.N.: Batch proofs are statistically hiding. Cryptology ePrint Archive, Report 2023/754 (2023). https://eprint.iacr.org/2023/754
9. Boneh, D., Boyle, E., Corrigan-Gibbs, H., Gilboa, N., Ishai, Y.: Zero-knowledge proofs on secret-shared data via fully linear PCPs. In: Boldyreva, A., Micciancio, D. (eds.) Advances in Cryptology – CRYPTO 2019, Part III. LNCS, vol. 11694, pp. 67–97. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-26954-8_3
10. Boneh, D., Boyle, E., Corrigan-Gibbs, H., Gilboa, N., Ishai, Y.: Lightweight techniques for private heavy hitters. In: 2021 IEEE Symposium on Security and Privacy, pp. 762–776. IEEE Computer Society Press, San Francisco, CA, USA (2021). https://doi.org/10.1109/SP40001.2021.00048
11. Boneh, D., Goh, E.J., Nissim, K.: Evaluating 2-DNF formulas on ciphertexts. In: Kilian, J. (ed.) TCC 2005: 2nd Theory of Cryptography Conference. LNCS, vol. 3378, pp. 325–341. Springer, Berlin (2005). https://doi.org/10.1007/978-3-540-30576-7_18
12. Boyle, E., Couteau, G., Gilboa, N., Ishai, Y.: Compressing vector OLE. In: Lie, D., Mannan, M., Backes, M., Wang, X. (eds.) ACM CCS 2018: 25th Conference on Computer and Communications Security, pp. 896–912. ACM Press, Toronto, ON, Canada (2018). https://doi.org/10.1145/3243734.3243868

13. Boyle, E., Couteau, G., Gilboa, N., Ishai, Y., Kohl, L., Scholl, P.: Efficient pseudorandom correlation generators: silent OT extension and more. In: Boldyreva, A., Micciancio, D. (eds.) Advances in Cryptology – CRYPTO 2019, Part III. LNCS, vol. 11694, pp. 489–518. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-26954-8_16
14. Boyle, E., Couteau, G., Gilboa, N., Ishai, Y., Orrù, M.: Homomorphic secret sharing: optimizations and applications. In: Thuraisingham, B.M., Evans, D., Malkin, T., Xu, D. (eds.) ACM CCS 2017: 24th Conference on Computer and Communications Security, pp. 2105–2122. ACM Press, Dallas, TX, USA (2017). https://doi.org/10.1145/3133956.3134107
15. Boyle, E., Gilboa, N., Ishai, Y.: Function secret sharing. In: Oswald, E., Fischlin, M. (eds.) Advances in Cryptology – EUROCRYPT 2015, Part II. LNCS, vol. 9057, pp. 337–367. Springer, Berlin (2015). https://doi.org/10.1007/978-3-662-46803-6_12
16. Boyle, E., Gilboa, N., Ishai, Y.: Breaking the circuit size barrier for secure computation under DDH. In: Robshaw, M., Katz, J. (eds.) Advances in Cryptology – CRYPTO 2016, Part I. LNCS, vol. 9814, pp. 509–539. Springer, Berlin (2016). https://doi.org/10.1007/978-3-662-53018-4_19
17. Boyle, E., Gilboa, N., Ishai, Y.: Function secret sharing: Improvements and extensions. In: Weippl, E.R., Katzenbeisser, S., Kruegel, C., Myers, A.C., Halevi, S. (eds.) ACM CCS 2016: 23rd Conference on Computer and Communications Security, pp. 1292–1303. ACM Press, Vienna, Austria (2017). https://doi.org/10.1145/2976749.2978429
18. Boyle, E., Gilboa, N., Ishai, Y.: Group-based secure computation: optimizing rounds, communication, and computation. In: Coron, J.S., Nielsen, J.B. (eds.) Advances in Cryptology – EUROCRYPT 2017, Part II. LNCS, vol. 10211, pp. 163–193. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-56614-6_6
19. Boyle, E., Gilboa, N., Ishai, Y., Lin, H., Tessaro, S.: Foundations of homomorphic secret sharing. In: Karlin, A.R. (ed.) ITCS 2018: 9th Innovations in Theoretical Computer Science Conference. vol. 94, pp. 21:1–21:21. LIPIcs, Cambridge, MA, USA (2018). https://doi.org/10.4230/LIPIcs.ITCS.2018.21
20. Boyle, E., Gilboa, N., Ishai, Y., Nof, A.: Practical fully secure three-party computation via sublinear distributed zero-knowledge proofs. In: Cavallaro, L., Kinder, J., Wang, X., Katz, J. (eds.) ACM CCS 2019: 26th Conference on Computer and Communications Security, pp. 869–886. ACM Press, London, UK (2019). https://doi.org/10.1145/3319535.3363227
21. Boyle, E., Gilboa, N., Ishai, Y., Nof, A.: Efficient fully secure computation via distributed zero-knowledge proofs. In: Moriai, S., Wang, H. (eds.) Advances in Cryptology – ASIACRYPT 2020, Part III. LNCS, vol. 12493, pp. 244–276. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-64840-4_9
22. Boyle, E., Gilboa, N., Ishai, Y., Nof, A.: Sublinear GMW-style compiler for MPC with preprocessing. In: Malkin, T., Peikert, C. (eds.) Advances in Cryptology – CRYPTO 2021, Part II. LNCS, vol. 12826, pp. 457–485. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-84245-1_16
23. Boyle, E., Kohl, L., Scholl, P.: Homomorphic secret sharing from lattices without FHE. In: Ishai, Y., Rijmen, V. (eds.) Advances in Cryptology – EUROCRYPT 2019, Part II. LNCS, vol. 11477, pp. 3–33. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-17656-3_1
24. Bradley, E., Waters, B., Wu, D.J.: Batch arguments to NIZKs from one-way functions. Cryptology ePrint Archive, Report 2023/1938 (2023). https://eprint.iacr.org/2023/1938

25. Brakerski, Z., Holmgren, J., Kalai, Y.T.: Non-interactive delegation and batch NP verification from standard computational assumptions. In: Hatami, H., McKenzie, P., King, V. (eds.) 49th Annual ACM Symposium on Theory of Computing, pp. 474–482. ACM Press, Montreal, QC, Canada (2017). https://doi.org/10.1145/3055399.3055497
26. de Castro, L., Polychroniadou, A.: Lightweight, maliciously secure verifiable function secret sharing. In: Dunkelman, O., Dziembowski, S. (eds.) Advances in Cryptology – EUROCRYPT 2022, Part I. LNCS, vol. 13275, pp. 150–179. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-06944-4_6
27. Champion, J., Wu, D.J.: Non-interactive zero-knowledge from non-interactive batch arguments. In: Handschuh, H., Lysyanskaya, A. (eds.) Advances in Cryptology – CRYPTO 2023, Part II. LNCS, vol. 14082, pp. 38–71. Springer, Cham (2023). https://doi.org/10.1007/978-3-031-38545-2_2
28. Chen, X.: Verifiable homomorphic secret sharing for machine learning classifiers. IEEE Access **11**, 43639–43647 (2023). https://doi.org/10.1109/ACCESS.2023.3271319
29. Chen, X., Zhang, L.F.: Two-server verifiable homomorphic secret sharing for high-degree polynomials. In: Susilo, W., Deng, R.H., Guo, F., Li, Y., Intan, R. (eds.) ISC 2020: 23rd International Conference on Information Security. LNCS, vol. 12472, pp. 75–91. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-62974-8_5
30. Chen, X., Zhang, L.F.: Two-server delegation of computation on label-encrypted data. IEEE Trans. Cloud Comput. **9**(4), 1645–1656 (2021). https://doi.org/10.1109/TCC.2019.2913375
31. Chida, K., et al.: Fast large-scale honest-majority MPC for malicious adversaries. In: Shacham, H., Boldyreva, A. (eds.) Advances in Cryptology – CRYPTO 2018, Part III. LNCS, vol. 10993, pp. 34–64. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-96878-0_2
32. Chiesa, A., Lehmkuhl, R., Mishra, P., Zhang, Y.: EOS: efficient private delegation of zkSNARK provers. In: Calandrino, J.A., Troncoso, C. (eds.) USENIX Security 2023: 32nd USENIX Security Symposium, pp. 6453–6469. USENIX Association, Anaheim, CA, USA (2023)
33. Chillotti, I., Orsini, E., Scholl, P., Smart, N.P., Van Leeuwen, B.: Scooby: improved multi-party homomorphic secret sharing based on FHE. In: Galdi, C., Jarecki, S. (eds.) SCN 22: 13th International Conference on Security in Communication Networks. LNCS, vol. 13409, pp. 540–563. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-14791-3_24
34. Choudhuri, A.R., Garg, S., Jain, A., Jin, Z., Zhang, J.: Correlation intractability and SNARGs from sub-exponential DDH. In: Handschuh, H., Lysyanskaya, A. (eds.) Advances in Cryptology – CRYPTO 2023, Part IV. LNCS, vol. 14084, pp. 635–668. Springer, Cham (2023). https://doi.org/10.1007/978-3-031-38551-3_20
35. Choudhuri, A.R., Hubácek, P., Kamath, C., Pietrzak, K., Rosen, A., Rothblum, G.N.: Finding a NASH equilibrium is no easier than breaking Fiat-Shamir. In: Charikar, M., Cohen, E. (eds.) 51st Annual ACM Symposium on Theory of Computing, pp. 1103–1114. ACM Press, Phoenix, AZ, USA (2019). https://doi.org/10.1145/3313276.3316400
36. Choudhuri, A.R., Hubacek, P., Kamath, C., Pietrzak, K., Rosen, A., Rothblum, G.N.: PPAD-hardness via iterated squaring modulo a composite. Cryptology ePrint Archive, Report 2019/667 (2019). https://eprint.iacr.org/2019/667

37. Choudhuri, A.R., Jain, A., Jin, Z.: Non-interactive batch arguments for NP from standard assumptions. In: Malkin, T., Peikert, C. (eds.) Advances in Cryptology – CRYPTO 2021, Part IV. LNCS, vol. 12828, pp. 394–423. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-84259-8_14
38. Choudhuri, A.R., Jain, A., Jin, Z.: SNARGs for $\mathcal{P}$ from LWE. In: 62nd Annual Symposium on Foundations of Computer Science, pp. 68–79. IEEE Computer Society Press, Denver, CO, USA (2022). https://doi.org/10.1109/FOCS52979.2021.00016
39. Couteau, G.: A note on the communication complexity of multiparty computation in the correlated randomness model. In: Ishai, Y., Rijmen, V. (eds.) Advances in Cryptology – EUROCRYPT 2019, Part II. LNCS, vol. 11477, pp. 473–503. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-17656-3_17
40. Couteau, G., Meyer, P.: Breaking the circuit size barrier for secure computation under quasi-polynomial LPN. In: Canteaut, A., Standaert, F.X. (eds.) Advances in Cryptology – EUROCRYPT 2021, Part II. LNCS, vol. 12697, pp. 842–870. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-77886-6_29
41. Couteau, G., Meyer, P., Passelègue, A., Riahinia, M.: Constrained pseudorandom functions from homomorphic secret sharing. In: Hazay, C., Stam, M. (eds.) Advances in Cryptology – EUROCRYPT 2023, Part III. LNCS, vol. 14006, pp. 194–224. Springer, Cham (2023). https://doi.org/10.1007/978-3-031-30620-4_7
42. Damgård, I., Faust, S., Hazay, C.: Secure two-party computation with low communication. In: Cramer, R. (ed.) TCC 2012: 9th Theory of Cryptography Conference. LNCS, vol. 7194, pp. 54–74. Springer, Berlin (2012). https://doi.org/10.1007/978-3-642-28914-9_4
43. Damgård, I., Pastro, V., Smart, N.P., Zakarias, S.: Multiparty computation from somewhat homomorphic encryption. In: Safavi-Naini, R., Canetti, R. (eds.) Advances in Cryptology – CRYPTO 2012. LNCS, vol. 7417, pp. 643–662. Springer, Berlin (2012). https://doi.org/10.1007/978-3-642-32009-5_38
44. Dao, Q., Ishai, Y., Jain, A., Lin, H.: Multi-party homomorphic secret sharing and sublinear MPC from sparse LPN. In: Handschuh, H., Lysyanskaya, A. (eds.) Advances in Cryptology – CRYPTO 2023, Part II. LNCS, vol. 14082, pp. 315–348. Springer, Cham (2023). https://doi.org/10.1007/978-3-031-38545-2_11
45. Dodis, Y., Halevi, S., Rothblum, R.D., Wichs, D.: Spooky encryption and its applications. In: Robshaw, M., Katz, J. (eds.) Advances in Cryptology – CRYPTO 2016, Part III. LNCS, vol. 9816, pp. 93–122. Springer, Berlin (2016). https://doi.org/10.1007/978-3-662-53015-3_4
46. Ephraim, N., Freitag, C., Komargodski, I., Pass, R.: Continuous verifiable delay functions. In: Canteaut, A., Ishai, Y. (eds.) Advances in Cryptology – EUROCRYPT 2020, Part III. LNCS, vol. 12107, pp. 125–154. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-45727-3_5
47. Fazio, N., Gennaro, R., Jafarikhah, T., Skeith III, W.E.: Homomorphic secret sharing from paillier encryption. In: Okamoto, T., Yu, Y., Au, M.H., Li, Y. (eds.) ProvSec 2017: 11th International Conference on Provable Security. LNCS, vol. 10592, pp. 381–399. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-68637-0_23
48. Fosli, I., Ishai, Y., Kolobov, V.I., Wootters, M.: On the download rate of homomorphic secret sharing. In: Braverman, M. (ed.) ITCS 2022: 13th Innovations in Theoretical Computer Science Conference. vol. 215, pp. 71:1–71:22. LIPIcs, Berkeley, CA, USA (2022). https://doi.org/10.4230/LIPIcs.ITCS.2022.71

49. Gabizon, A., Williamson, Z.J., Ciobotaru, O.: PLONK: permutations over Lagrange-bases for oecumenical noninteractive arguments of knowledge. Cryptology ePrint Archive, Report 2019/953 (2019). https://eprint.iacr.org/2019/953
50. Garg, S., Goel, A., Jain, A., Policharla, G.V., Sekar, S.: zkSaaS: zero-knowledge SNARKs as a service. In: Calandrino, J.A., Troncoso, C. (eds.) USENIX Security 2023: 32nd USENIX Security Symposium, pp. 4427–4444. USENIX Association, Anaheim, CA, USA (2023)
51. Garg, S., Goel, A., Wang, M.: How to prove statements obliviously? In: Reyzin, L., Stebila, D. (eds.) Advances in Cryptology – CRYPTO 2024, Part X. LNCS, vol. 14929, pp. 449–487. Springer, Cham, (2024). https://doi.org/10.1007/978-3-031-68403-6_14
52. Genkin, D., Ishai, Y., Prabhakaran, M., Sahai, A., Tromer, E.: Circuits resilient to additive attacks with applications to secure computation. In: Shmoys, D.B. (ed.) 46th Annual ACM Symposium on Theory of Computing, pp. 495–504. ACM Press, New York, NY, USA (2014). https://doi.org/10.1145/2591796.2591861
53. Gennaro, R., Gentry, C., Parno, B., Raykova, M.: Quadratic span programs and succinct NIZKs without PCPs. In: Johansson, T., Nguyen, P.Q. (eds.) Advances in Cryptology – EUROCRYPT 2013. LNCS, vol. 7881, pp. 626–645. Springer, Berlin (2013). https://doi.org/10.1007/978-3-642-38348-9_37
54. Gilboa, N., Ishai, Y.: Distributed point functions and their applications. In: Nguyen, P.Q., Oswald, E. (eds.) Advances in Cryptology – EUROCRYPT 2014. LNCS, vol. 8441, pp. 640–658. Springer, Berlin (2014). https://doi.org/10.1007/978-3-642-55220-5_35
55. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game or a completeness theorem for protocols with honest majority. In: Aho, A. (ed.) 19th Annual ACM Symposium on Theory of Computing, pp. 218–229. ACM Press, New York City, NY, USA (1987). https://doi.org/10.1145/28395.28420
56. Goldwasser, S., Micali, S., Rackoff, C.: The knowledge complexity of interactive proof systems. SIAM J. Comput. **18**(1), 186–208 (1989)
57. Groth, J.: Short pairing-based non-interactive zero-knowledge arguments. In: Abe, M. (ed.) Advances in Cryptology – ASIACRYPT 2010. LNCS, vol. 6477, pp. 321–340. Springer, Berlin (2010). https://doi.org/10.1007/978-3-642-17373-8_19
58. Groth, J.: On the size of pairing-based non-interactive arguments. In: Fischlin, M., Coron, J.S. (eds.) Advances in Cryptology – EUROCRYPT 2016, Part II. LNCS, vol. 9666, pp. 305–326. Springer, Berlin (2016). https://doi.org/10.1007/978-3-662-49896-5_11
59. He, Y., Zhang, L.F.: Cheater-identifiable homomorphic secret sharing for outsourcing computations. J. Ambient. Intell. Humaniz. Comput. **11**(11), 5103–5113 (2020). https://doi.org/10.1007/S12652-020-01814-5
60. Hulett, J., Jawale, R., Khurana, D., Srinivasan, A.: SNARGs for P from sub-exponential DDH and QR. In: Dunkelman, O., Dziembowski, S. (eds.) Advances in Cryptology – EUROCRYPT 2022, Part II. LNCS, vol. 13276, pp. 520–549. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-07085-3_18
61. Ishai, Y., Lai, R.W.F., Malavolta, G.: A geometric approach to homomorphic secret sharing. In: Garay, J. (ed.) PKC 2021: 24th International Conference on Theory and Practice of Public Key Cryptography, Part II. LNCS, vol. 12711, pp. 92–119. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-75248-4_4
62. Jawale, R., Kalai, Y.T., Khurana, D., Zhang, R.Y.: SNARGs for bounded depth computations and PPAD hardness from sub-exponential LWE. In: Khuller, S., Williams, V.V. (eds.) 53rd Annual ACM Symposium on Theory of Computing, pp. 708–721. ACM Press, Italy (2021). https://doi.org/10.1145/3406325.3451055

63. Kalai, Y.T., Paneth, O., Yang, L.: How to delegate computations publicly. In: Charikar, M., Cohen, E. (eds.) 51st Annual ACM Symposium on Theory of Computing, pp. 1115–1124. ACM Press, Phoenix, AZ, USA (2019). https://doi.org/10.1145/3313276.3316411
64. Kalai, Y.T., Paneth, O., Yang, L.: Delegation with updatable unambiguous proofs and PPAD-hardness. In: Micciancio, D., Ristenpart, T. (eds.) Advances in Cryptology – CRYPTO 2020, Part III. LNCS, vol. 12172, pp. 652–673. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-56877-1_23
65. Kilian, J.: A note on efficient zero-knowledge proofs and arguments (extended abstract). In: 24th Annual ACM Symposium on Theory of Computing, pp. 723–732. ACM Press, Victoria, BC, Canada (1992). https://doi.org/10.1145/129712.129782
66. Lai, R.W.F., Malavolta, G., Schröder, D.: Homomorphic secret sharing for low degree polynomials. In: Peyrin, T., Galbraith, S. (eds.) Advances in Cryptology – ASIACRYPT 2018, Part III. LNCS, vol. 11274, pp. 279–309. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-03332-3_11
67. Lindell, Y., Pinkas, B.: Secure two-party computation via cut-and-choose oblivious transfer. In: Ishai, Y. (ed.) TCC 2011: 8th Theory of Cryptography Conference. LNCS, vol. 6597, pp. 329–346. Springer, Berlin (2011). https://doi.org/10.1007/978-3-642-19571-6_20
68. Lipmaa, H.: Progression-free sets and sublinear pairing-based non-interactive zero-knowledge arguments. In: Cramer, R. (ed.) TCC 2012: 9th Theory of Cryptography Conference. LNCS, vol. 7194, pp. 169–189. Springer, Berlin (2012). https://doi.org/10.1007/978-3-642-28914-9_10
69. Liu, X., Zhou, Z., Wang, Y., Zhang, B., Yang, X.: Scalable collaborative zk-SNARK: fully distributed proof generation and malicious security. Cryptology ePrint Archive, Report 2024/143 (2024). https://eprint.iacr.org/2024/143
70. Lombardi, A., Vaikuntanathan, V.: Fiat-Shamir for repeated squaring with applications to PPAD-hardness and VDFs. In: Micciancio, D., Ristenpart, T. (eds.) Advances in Cryptology – CRYPTO 2020, Part III. LNCS, vol. 12172, pp. 632–651. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-56877-1_22
71. Micali, S.: CS proofs (extended abstracts). In: 35th Annual Symposium on Foundations of Computer Science, pp. 436–453. IEEE Computer Society Press, Santa Fe, NM, USA (1994). https://doi.org/10.1109/SFCS.1994.365746
72. Mondal, A., Tiwari, P.R., Gupta, D.: Poster: fully homomorphic secret sharing with output verifiability. NDSS Poster (2022). https://www.ndss-symposium.org/wp-content/uploads/NDSS2022Poster_paper_36.pdf
73. Orlandi, C., Scholl, P., Yakoubov, S.: The rise of paillier: homomorphic secret sharing and public-key silent OT. In: Canteaut, A., Standaert, F.X. (eds.) Advances in Cryptology – EUROCRYPT 2021, Part I. LNCS, vol. 12696, pp. 678–708. Springer, Cham, (2021). https://doi.org/10.1007/978-3-030-77870-5_24
74. Ozdemir, A., Boneh, D.: Experimenting with collaborative zk-SNARKs: zero-knowledge proofs for distributed secrets. In: Butler, K.R.B., Thomas, K. (eds.) USENIX Security 2022: 31st USENIX Security Symposium, pp. 4291–4308. USENIX Association, Boston, MA, USA (2022)
75. Reingold, O., Rothblum, G.N., Rothblum, R.D.: Constant-round interactive proofs for delegating computation. SIAM J. Comput. **50**(3) (2021)
76. Roy, L., Singh, J.: Large message homomorphic secret sharing from DCR and applications. In: Malkin, T., Peikert, C. (eds.) Advances in Cryptology – CRYPTO 2021, Part III. LNCS, vol. 12827, pp. 687–717. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-84252-9_23

77. Tsaloli, G., Banegas, G., Mitrokotsa, A.: Practical and provably secure distributed aggregation: verifiable additive homomorphic secret sharing. Cryptogr. **4**(3), 25 (2020). https://doi.org/10.3390/CRYPTOGRAPHY4030025
78. Tsaloli, G., Liang, B., Mitrokotsa, A.: Verifiable homomorphic secret sharing. In: Baek, J., Susilo, W., Kim, J. (eds.) ProvSec 2018: 12th International Conference on Provable Security. LNCS, vol. 11192, pp. 40–55. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-01446-9_3
79. Tsaloli, G., Mitrokotsa, A.: Sum it up: verifiable additive homomorphic secret sharing. In: Seo, J.H. (ed.) ICISC 19: 22nd International Conference on Information Security and Cryptology. LNCS, vol. 11975, pp. 115–132. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-40921-0_7
80. Waters, B., Wu, D.J.: Batch arguments for NP and more from standard bilinear group assumptions. In: Dodis, Y., Shrimpton, T. (eds.) Advances in Cryptology – CRYPTO 2022, Part II. LNCS, vol. 13508, pp. 433–463. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-15979-4_15
81. Yoshida, M., Obana, S.: Verifiably multiplicative secret sharing. IEEE Trans. Inf. Theory **65**(5), 3233–3245 (2019). https://doi.org/10.1109/TIT.2018.2886262
82. Zhang, L.F., Wang, H.: Multi-server verifiable computation of low-degree polynomials. In: 2022 IEEE Symposium on Security and Privacy, pp. 596–613. IEEE Computer Society Press, San Francisco, CA, USA (2022). https://doi.org/10.1109/SP46214.2022.9833792

Author Index

E. Boyle and M. Mahmoody (Eds.): TCC 2024, LNCS 15367, pp. 651–652, 2025.
https://doi.org/10.1007/978-3-031-78023-3