UNIVERSITÄT
DES
SAARLANDES

Saarland University

Faculty of Mathematics and Computer Science

Department of Computer Science

Cryptography with Anonymity in Mind

Dissertation
zur Erlangung des Grades
des Doktors der Ingenieurwissenschaften
der Fakultät für Mathematik und Informatik
der Universität des Saarlandes

von
Ivan Pryvalov

Saarbrücken, 2021

Tag des Kolloquiums:	11. April 2022
Dekan:	Univ.-Prof. Dr. Thomas Schuster

Prüfungsausschuss:

Vorsitzender:	Prof. Dr. Sven Apel
Berichterstattende:	Prof. Dr. Dr. h.c. Michael Backes
	Prof. Dr. Amir Herzberg
	Dr. Nico Döttling
Akademischer Mitarbeiter:	Dr. Zhikun Zhang

Zusammenfassung

Fortschritte in der Informationstechnik haben leistungsstarke allgegenwärtige Rechner hervorgerufen, während uns digitale Netzwerke neue Wege für die schnelle Kommunikation ermöglicht haben. Durch die Vielzahl von Anwendungen führte dies zur Übertragung von riesigen Datenvolumen. Seit Jahrzehnten wurden bereits verschiedene kryptographische Verfahren und Technologien zum Datenschutz erforscht und analysiert. Das Ziel ist die Privatsphäre der Benutzer zu schützen und gleichzeitig nützliche Funktionalität anzubieten, was oft mit einem Kompromiss zwischen Sicherheitseigenschaften, kryptographischen Annahmen und Effizienz verbunden ist. In einer Fülle von kryptographischen Konstruktionen spielen Anonymitätseigenschaften eine besondere Rolle, da sie in vielen realistischen Szenarien sehr wichtig sind. Allerdings fehlen vielen kryptographischen Primitive Anonymitätseigenschaften oder sie stehen im Zusammenhang mit erheblichen Kosten.

In dieser Dissertation erweitern wir den Bereich von kryptographischen Primitiven mit einem Fokus auf Anonymität. Erstens definieren wir Anonymous RAM, eine Verallgemeinerung von Einzelbenutzer-Oblivious RAM für mehrere misstraute Benutzer, und stellen dazu zwei Konstruktionen mit verschiedenen Kompromissen zwischen Annahmen und Effizienz vor. Zweitens definieren wir ein Verschlüsselungsverfahren, das es erlaubt anonym eine Verbindung zwischen Geheimtexten herzustellen und deren Integrität zu überprüfen. Darüber hinaus bietet die aggregierbare Variante von diesem Verfahren an, Parallel Anonymous RAM zu bauen. Dieses verbessert Anonymous RAM, indem es mehrere Benutzer in einer parallelen Ausführung unterstützen kann. Drittens zeigen wir eine Methode für das Konstruieren effizienter Zero-Knowledge-Protokolle, die gleichzeitig aus algebraischen und arithmetischen Teilen bestehen. Zuletzt zeigen wir ein Framework für das Konstruieren effizienter Single-Leader-Election-Protokolle, was kürzlich als ein wichtiger Bestandteil in den Proof-of-Stake Kryptowährungen erkannt worden ist.

Abstract

Advances in information technologies gave a rise to powerful ubiquitous computing devices, and digital networks have enabled new ways of fast communication, which immediately found tons of applications and resulted in large amounts of data being transmitted. For decades, cryptographic schemes and privacy-preserving protocols have been studied and researched in order to offer end users privacy of their data and implement useful functionalities at the same time, often trading security properties for cryptographic assumptions and efficiency. In this plethora of cryptographic constructions, anonymity properties play a special role, as they are important in many real-life scenarios. However, many useful cryptographic primitives lack anonymity properties or imply prohibitive costs to achieve them.

In this thesis, we expand the territory of cryptographic primitives with anonymity in mind. First, we define Anonymous RAM, a generalization of a single-user Oblivious RAM to multiple mistrusted users, and present two constructions thereof with different trade-offs between assumptions and efficiency. Second, we define an encryption scheme that allows to establish chains of ciphertexts anonymously and verify their integrity. Furthermore, the aggregatable version of the scheme allows to build a Parallel Anonymous RAM, which enhances Anonymous RAM by supporting concurrent users. Third, we show our technique for constructing efficient non-interactive zero-knowledge proofs for statements that consist of both algebraic and arithmetic statements. Finally, we show our framework for constructing efficient single secret leader election protocols, which have been recently identified as an important component in proof-of-stake cryptocurrencies.

Background of this Dissertation

This dissertation is based on four projects, two of which are published in peer-reviewed conferences and two are under submission. I contributed to all projects as one of the main authors.

The initial idea for our first work, Anonymous RAM [P1], originated during a joint discussion of Ivan Pryvalov, Michael Backes, Aniket Kate, and Amir Herzberg at Saarland University. Michael Backes shaped the theoretical foundation for the anonymous RAM problem, and all authors discussed and reviewed the first (linear) construction. Ivan Pryvalov came up with the two-server polylogarithmic solution to the problem. Aniket Kate, Amir Herzberg, and Michael Backes contributed in writing and improving the presentation of the paper. All authors reviewed the paper.

Our second work, Randomize-or-Change encryption [P2], emerged as a follow-up project to identify a basic primitive that could be used for building a secure Anonymous RAM. All authors contributed to shaping the security properties of the cryptographic primitive and discussed the discrete log construction. Aniket Kate proposed a technique for saving one exponentiation in the discrete log construction. Amir Herzberg proposed an idea for the hybrid construction, which was later formalized by Ivan Pryvalov. Aniket Kate, Amir Herzberg, and Michael Backes contributed in writing and improving the presentation of the paper. The preliminary version of this work was presented at [P3]. Further, Ivan Pryvalov came up with the aggregatable extension. All authors reviewed the paper.

Our third work about non-interactive zero-knowledge proofs [P4] emerged first as a part of the hybrid construction in the Randomize-or-Change project. It was later identified as a missing spot in the literature and formalized as a separate independent contribution. Lucjan Hanzlik joined the project at a later stage and contributed in writing and improving the security proofs. All authors reviewed the paper.

Finally, the problem statement for secret leader election [P5] was first discussed by Ivan Pryvalov and Pascal Berrang during their meeting at NDSS'2019 in San-Diego. Ivan Pryvalov first came up with the idea of randomly swapping commitments and then formalized the solution. Lucjan Hanzlik found a security problem in an earlier version of the draft, which was later fixed by Ivan Pryvalov. Ivan Pryvalov conducted experimental evaluation of the proposed framework. In one of the instantiations, Ivan Pryvalov re-used the source code from another publication [S1], also presented at [S3, S2], for which he was responsible for implementing the MPC algorithms. All authors reviewed the paper.

[P1] Backes, M., Herzberg, A., Kate, A., and Pryvalov, I. Anonymous ram. In: *European Symposium on Research in Computer Security–ESORICS 2016*. Springer. 2016, 344–362.

[P2] Backes, M., Herzberg, A., Kate, A., and Pryvalov, I. Randomize-or-Change Encryption. In: *Under submission.*

[P3] Backes, M., Herzberg, A., Kate, A., and Pryvalov, I. Touch-or-change: multi-user privacy and integrity in universally re-randomizable encryption. In: *Grande Region Security and Reliability Day (GRSRD)*. 2017.

[P4] Backes, M., Hanzlik, L., Herzberg, A., Kate, A., and Pryvalov, I. Efficient non-interactive zero-knowledge proofs in cross-domains without trusted setup. In: *IACR International Workshop on Public Key Cryptography*. Springer. 2019, 286–313.

[P5] Backes, M., Berrang, P., Hanzlik, L., and Pryvalov, I. A framework for constructing Single Secret Leader Election from MPC. In: *European Symposium on Research in Computer Security–ESORICS 2022*. 2022, to appear.

Further Contributions of the Author

[S1] Eigner, F., Kate, A., Maffei, M., Pampaloni, F., and Pryvalov, I. Differentially private data aggregation with optimal utility. In: *Proceedings of the 30th Annual Computer Security Applications Conference*. 2014, 316–325.

[S2] Eigner, F., Kate, A., Maffei, M., Pampaloni, F., and Pryvalov, I. Privacy-preserving data aggregation with optimal utility using arithmetic smc. In: *Workshop on Usable and Efficient Secure Multiparty Computation (UaESMC)*. 2014.

[S3] Eigner, F., Kate, A., Maffei, M., Pampaloni, F., and Pryvalov, I. Privada: a generic framework for privacy-preserving data aggregation. In: *Grande Region Security and Reliability Day (GRSRD)*. 2014.

Acknowledgments

First of all, I would like to thank Michael Backes for giving me the opportunity to be his student. I first contacted him back in 2012 regarding open PhD positions. At the time I was a member of the Graduate School of Computer Science in Saarbrücken and had recently completed the preparatory phase. However, I was still exploring potential fields for pursuing my PhD and information security was the only major field for which I hadn't taken any courses during the preparatory phase. So Michael had suggested that I take his upcoming cryptography core course so that he could judge my application based on my performance in the course. That was the departing point into a world of magic cryptographic numbers. Not only had we learned in the class, say, why encryption schemes are secure according to the modern security standards but also we were introduced some remarkable primitives, one of which – zero-knowledge proofs – blew my mind. In 2014, I joined Michael's Information Security and Cryptography group and I am very thankful to Michael for his support and supervision, especially in the first years, when his help was most needed. Moreover, I would like to thank Michael for his efforts in promoting research and, together with all people involved, transforming a handful of research groups co-located at Saarbrücken campus into the Helmholtz Center, which is the biggest academic success one can have in Germany. Thanks to this transformation, our working environment has greatly improved and I extended my academic network.

I would like to thank Aniket Kate for our long-term collaboration, which started prior to my PhD. Not only has our collaboration served me as a bridge between the cryptography course and the doctoral studies but also it continued further and resulted in three co-authored conference publications, two of which are included into this thesis. I am thankful for Aniket's support throughout the entire time of my journey into the cryptographic world.

I would like to thank Amir Herzberg for the opportunity to be his teaching assistant in a seminar when he was visiting Saarland University in 2014; for his thorough and detailed feedback in our joint projects; for helping me with my first conference presentation at ESORICS'2016 in Greece; and for being supportive throughout my PhD life. Our collaboration has resulted in two co-authored conference publications, which are included into this thesis.

I would like to thank Lucjan Hanzlik for his help and support during the final years of my PhD. Our collaboration has resulted in one co-authored conference publication. Besides, he contributed to our recent research project with Pascal Berrang. Thanks to Lucjan's expertise, we discovered and fixed a security flaw in an earlier draft of the paper.

I would like to thank Pascal Berrang for our recent collaboration, which started with a meeting at NDSS'2019 in San-Diego when he shared a research question for which he was looking a solution. The findings from this project are included into this thesis.

I would like to thank Nico Döttling and Amir Herzberg for agreeing to review this thesis.

Regarding thesis writing, I would like to thank Ágnes Kiss, for giving me a high-level feedback on the thesis, and Pascal Berrang, for helping me with translating the abstract into German.

During my PhD time, I shared office with many people and I would like to thank everyone of them for being a nice officemate: Ahmed Salem, Fabian Bendun, Jonas Schneider-Bensch, Kamil Kluczniak, Kathrin Grosse, Lucjan Hanzlik, Min Chen, Sebastian Meiser, Sebastian Weisgerber, and Tahleen Rahman. Separately, I would like to thank Muhammad Rizwan Asghar, with whom I shared office in 2014 when he was a postdoc at Michael's chair, for his help in writing my first review for a conference paper and for numerous discussions about academic life. After he left CISPA to join University of Oakland, New Zealand, we have been keeping a semi-regular contact over the Internet; I consider Rizwan my external mentor in navigating my academic path.

I would like to thank everyone at CISPA, who I met during my PhD time, for making life full of interesting events, for simplifying any bureaucratic procedures, and for making the working environment exceptionally good. I would like to thank the Saarbrücken Graduate School of Computer Science for offering international students a stipend program, which I have been part of.

Last but not least, I would like to thank Alina, for being nearby and supporting me during these long almost 7 years of my PhD, and to our now-four-month-old son Adam, for constantly reminding me that there are better alternatives to writing a thesis. Finally, I would like to thank my parents and my brother for being supportive throughout my doctoral studies.

Contents

1 Introduction **1**

2 Anonymous RAM **9**

2.1 Introduction . . . 11

2.2 AnonRAM Definitions . . . 13

2.3 Linear-complexity AnonRAM . . . 16

2.3.1 Seemingly Natural but Flawed Approaches . . . 16

2.3.2 $\mathsf{AnonRAM}_{\mathsf{lin}}$ and its Security Against HbC Adversaries . . . 17

2.3.3 $\mathsf{AnonRAM}^{\mathbf{M}}_{\mathsf{lin}}$ and its Security Against Malicious Users . . . 19

2.4 Polylogarithmic-complexity AnonRAM . . . 21

2.4.1 Cryptographic Building Blocks . . . 21

2.4.2 $\mathsf{AnonRAM}_{\mathsf{polylog}}$ Data Structure . . . 23

2.4.3 $\mathsf{AnonRAM}_{\mathsf{polylog}}$ Protocol Overview . . . 24

2.4.4 User Algorithm . . . 27

2.4.5 Reshuffle . . . 28

2.4.6 Complexity and Security Analysis . . . 30

2.4.7 $\mathsf{AnonRAM}^{\mathbf{M}}_{\mathsf{polylog}}$ Secure Against Malicious Users . . . 31

2.5 (Partially Key-Homomorphic) Oblivious PRF . . . 32

2.6 Detailed Description of $\mathsf{AnonRAM}^{\mathbf{M}}_{\mathsf{polylog}}$ . . . 35

2.7 Postponed Proofs . . . 37

2.7.1 Proof of $\mathsf{AnonRAM}_{\mathsf{polylog}}$ . . . 37

2.7.2 Proof of $\mathsf{AnonRAM}^{\mathbf{M}}_{\mathsf{polylog}}$ . . . 42

2.8 Conclusion . . . 43

3 Randomize-or-Change Encryption **45**

3.1 Introduction . . . 47

3.2 Overview . . . 50

3.2.1 Variants of RoC . . . 50

3.2.2 Security properties . . . 51

3.2.3 Applications . . . 53

3.3 (Basic) Randomize-or-Change Encryption . . . 55

3.3.1 Definition . . . 56

3.3.2 Discrete log based construction $\mathsf{RoC}_{\mathsf{dlog}}$ . . . 59

3.4 Aggregatable Randomize-or-Change Encryption 60
3.4.1 Definition 61
3.4.2 Aggregatable same-key RoC construction 63
3.4.3 Aggregatable any-key RoC construction 65
3.5 Keyed Randomize-or-Change Encryption 69
3.5.1 Definition 70
3.5.2 Hybrid construction $\mathsf{RoC}_{\mathsf{hyb}}$ 71
3.5.3 Security analysis of $\mathsf{RoC}_{\mathsf{hyb}}^{\mathbf{HbC}}$ 74
3.5.4 Integrity of plaintext in $\mathsf{RoC}_{\mathsf{hyb}}^{\mathbf{M}}$ 76
3.6 Parallel Anonymous RAM 78
3.6.1 Definition 78
3.6.2 Construction 80
3.7 Group payment system 82
3.8 Related Work 83
3.9 Postponed proofs 85
3.10 Conclusion 87

4 Non-interactive Zero-Knowledge Proofs in Cross-Domains 89
4.1 Introduction 91
4.2 Preliminaries 96
4.2.1 Homomorphic Commitment Schemes 96
4.2.2 Zero-Knowledge Proofs 98
4.2.3 ZKBoo/ZKB++ 100
4.3 Combining ZKB++ with Algebraic Commitments 100
4.3.1 Our Technique - First Approach 101
4.3.2 Improved Version 107
4.3.3 Security analysis 109
4.3.4 Optimization for large input space 114
4.3.5 Efficiency 114
4.4 NIZK OR-proofs in cross-domains 115
4.5 Conclusion 116

5 Single Secret Leader Election from MPC 119
5.1 Introduction 121
5.1.1 Our contribution 122
5.1.2 Related work 122
5.1.3 Comparison with Boneh et al. [18] 123
5.1.4 On the practicality of our SSLE framework 126
5.2 Definitions 126
5.2.1 Preliminaries 126
5.2.2 Single Secret Leader Election 128
5.3 (Non-secret) single leader election constructions 131
5.3.1 Naive attempts 131

5.3.2 Two-party Oblivious Select 132
5.3.3 Leader Election based on Two-party Oblivious Select . . . 134
5.4 Our SSLE from DDH 135
5.4.1 Upgrading to secret leader 135
5.4.2 Full construction 138
5.5 Our SSLE based on garbled circuits 140
5.5.1 Preliminaries 140
5.5.2 Construction 142
5.6 Practical considerations 142
5.6.1 Non-uniform distributions 144
5.6.2 Extensions to DDH-based MPC 145
5.7 Evaluation 145
5.7.1 Experimental setup 145
5.7.2 Construction 5.4.1 (Section 5.4.2) 145
5.7.3 Construction 5.5.1 (Section 5.5.2) 147
5.8 Postponed definitions 149
5.9 Security analysis 154
5.10 Conclusion 158

6 Conclusion **159**

List of Figures

2.1 User algorithm of $\mathsf{AnonRAM}_{\mathsf{lin}}$ with access request (j, α, m) for user U_i. 18

2.2 Flow of User algorithm in $\mathsf{AnonRAM}_{\mathsf{polylog}}$ for user U_i, cell j, and level ℓ: 1) U_i asks the tag server T for a dummy pre-tag. 2) T runs an OPRF protocol with the storage server S such that T learns the dummy pre-tag and S learns nothing. 3) T sends the dummy pre-tag to U_i. 4) U_i runs OPRF with S to learn a pre-tag for her cell j obliviously. 5) Depending on whether cell j is found in the previous levels or not, U_i selects one of the two pre-tags to compute a tag and sends the tag to S. 6) S re-randomizes and sends the block(s) associated with the user's tag. 7) U_i re-randomizes or updates the block(s), and possibly learns the value of cell j. If $\ell = 1$, steps 4) and 5) are skipped, and in step 6) S sends all blocks from that level. 26

2.3 Our (partially key-homomorphic) OPRF construction. 33

2.4 Part of zero-knowledge proof system for $\mathsf{AnonRAM}^{\mathbf{M}}_{\mathsf{polylog}}$ relationship between blocks B_k and $\hat{B}_k$ at level ℓ, represented as ciphertexts (γ_k, v_k) and $(\hat{\gamma}_k, \hat{v}_k)$ respectively, and a new block $\hat{B}_0$, represented as $(\hat{\gamma}_0, \hat{v}_0)$. Encryption of pre-tag for the dummy block associated with $\hat{B}_k$ is denoted as $\gamma_{\mathcal{D}_\ell}$. Encryption of the constant dummy value is denoted as $v_{\mathcal{D}}$. 36

3.1 Parallel Anonymous RAM from aggregatable RoC. Example for 3 users, two of them sending write-requests to the same location at the server via anonymous channels. 54

3.2 Definition of IND-CPA and ANON games. 58

3.3 Definition of ANON-INT-R and ANON-EXT-R games. 71

3.4 Ciphertext structure in $\mathsf{RoC}^{\mathbf{HbC}}_{\mathsf{hyb}}$. User U_i is the recipient of ciphertext, $\mathcal{G}$ denotes a PRG, $\mathsf{e}_i(\cdot)$ denotes the encryption algorithm of a (traditional) re-randomizable CPA-secure encryption scheme, m is the encrypted message. Relative sizes of ciphertext components are kept, γ is supposed to be the largest component, while all remaining are of equal size. 73

3.5 Group payment system from ARoC-ak. Example for 3 users, two of them sending transactions to the server via anonymous channels. Server's state consists of 4 units that belong to 3 users. User U_1 owns two units and sends one unit to U_2. User U_2 owns one unit and sends it to U_1. For each unit, the server aggregates a new unit value, forming a new server's state. 83

4.1 Description of Cross-ZKB++ (First Attempt) Prove algorithm for function $F(x) = 1$ with a committed input $C_x = \mathsf{Com}(x, r)$, made non-interactive using the Fiat-Shamir transformation and with t rounds of ZKB++. 103
4.2 Description of Cross-ZKB++ (First Attempt) Verify algorithm for function $F(x) = 1$ with a committed input $C_x = \mathsf{Com}(x, r)$, made non-interactive using the Fiat-Shamir transformation and with t rounds of ZKB++. 104
4.3 Description of Cross-ZKB++ Prove algorithm for function $F(x) = 1$ with a committed input $C_x = \mathsf{Com}(x, r)$, made non-interactive using the Fiat-Shamir transformation and with t rounds of ZKB++. . 110
4.4 Description of Cross-ZKB++ Verify algorithm for function $F(x) = 1$ with a committed input $C_x = \mathsf{Com}(x, r)$, made non-interactive using the Fiat-Shamir transformation and with t rounds of ZKB++. . 111
4.5 Non-interactive ZKB++ [37]. 117

5.1 Ideal functionality $\mathcal{F}_{\mathsf{SSLE}}^{N,\ell,c}$ 130
5.2 Two-party oblivious select $\mathsf{OSelect}$ protocol between Alice and Bob. 133
5.3 $\mathsf{OSelect}_N$: Extension of two-party $\mathsf{OSelect}$ to N users. Example for $N = 4$ and party $\mathcal{P}_1$ as the operational leader. 134
5.4 The intermediate Secret Leader Election protocol. 138
5.5 $\mathsf{OSelectM}$: Oblivious Select in the MPC setting. 139
5.6 $\mathsf{OSelectM_N}$: Extension $\mathsf{OSelectM}$ to N inputs. Example for $N = 4$. 139
5.7 Single Secret Leader Election construction SSLE instantiated with $\mathsf{OSelectM_N}$. 141
5.8 Elect algorithm and Oblivious Select instantiated in the MPC framework by Wang et al. [129] 143
5.9 Tree optimization technique. Example $\mathsf{OSelectM_N}$ for $\mathcal{P}_1$ and $\mathcal{P}_2$ with stakes $(1, 3)$. 144
5.10 Comparison of timings for Oblivious Select in Construction 5.5.1 in the LAN and WAN settings. 146

List of Tables

3.1 Permitted operations (randomize, change) for a given ciphertext c to obtain c' in randomize-or-change encryption, depending on the user's role. 49

3.2 Security properties in UREnc (as defined in [133]) and randomize-or-change encryption. We use $\mathsf{E}, \mathsf{R}, \mathsf{Ch}$ to denote the encryption, re-randomization, and change algorithms, respectively. "+" ("–") means that a respective security property can (cannot) be achieved. 52

3.3 Computing expected values $(\widetilde{c}_{[1]}, \widetilde{c}_{[3]}, \widetilde{c}_{[6]}, \widetilde{c}_{[8]})$ from $(\widetilde{c}_{[2]}, \widetilde{c}_{[4]}, \widetilde{c}_{[5]}, \widetilde{c}_{[7]})$ in ARoC-ak construction. Indices in the circles indicate to the values computed by re-randomizers in the first round. 68

4.1 Comparison of ZK proof systems in cross-domains for a circuit F with an algebraically committed input x, where $|F|$ denotes the circuit size, $|x|$ the number of input bits. We denote by λ the security parameter, by pub a public-key operation, by sym a symmetric-key operation. 96

5.1 Comparison of SSLE protocols, assuming all N users participate in election, amortized per one election 124

5.2 Experimenal results for Construction 5.4.1 in the honest-but-curious and malicious adversary models. 148

5.3 Experimental results for Construction 5.5.1 in the malicious adversary model. 149

1 Introduction

Anonymity is essential in many real-life scenarios. The most prominent examples are anonymous communication, electronic voting (e-voting), and payment systems. Anonymous communication is crucial in countries where the government censors their citizens by blocking unwanted content that may reveal or point out to violations of human rights or corruption, so exposing identity of the source of such information may put that person's life in danger. Anonymous communication is an active research field, and there have been many systems proposed in the literature, perhaps the most known to the general public is Tor Browser. Likewise, e-voting protocols are often seen as a secure alternative to the traditional paper-like in-person voting. The main benefit of e-voting systems is that the voter can vote from her electronic device at any point of time at any location during the election while hiding the vote and the fact of voting, i.e. anonymity of voting. Even a coerced user, who voted against her wish, should be able to privately and securely cast again a legitimate vote and only that vote would be counted. As of today, only a few countries have adopted e-voting for elections, mostly on a local level. Therefore, an important step to defend democracy against authoritarian regimes is to propose a provably secure e-voting system that would be widely accepted and would withstand all known threats, including – but not limited to – physical coercion, installed malware on electronic devices, or side channels. Finally, privacy and anonymity are essential in payment systems. Paying in cash resembles the essentials of a perfect payment system: banknotes are freely circulating in the system (serial numbers written on them are usually not tracked during most of the transactions); it is not known how much money left in the pocket; and banknotes do not reveal the payer's identity (unless the payer is explicitly asked to present it). Users would like to reveal as little as possible about their account details and identity when performing transactions. Nowadays, banks keep users' accounts in a digital form and allow online payments with certain privacy guarantees; however, users have to trust these banks. The past decade has seen the emergence of cryptocurrencies, with the main goal to remove the trust assumption by admitting other, more acceptable, assumptions and at the same time to guarantee certain privacy and anonymity properties. This is an active ongoing research topic.

Although anonymous communication, e-voting, and payment systems are important topics in themselves, the focus of this thesis is studying more basic cryptographic primitives that either lack foundation with respect to anonymity properties or have such properties while lacking in efficiency. Basic cryptographic building blocks with anonymity in mind can be used for constructing more complex systems with anonymity guarantees, without sacrificing efficiency and the achieved security properties in those systems. Depending on which primitive is studied, anonymity properties may have different meanings and forms. We first explain on a high level, which anonymity properties are addressed in the studied primitives, and then detail the respective contributions:

- In our first work, Anonymous RAM, we generalize the problem of hiding client's access patterns to the external storage, known as Oblivious RAM (ORAM) in the literature, to multiple mistrusting users. Hence, in addition to the security properties of a single-user ORAM primitive, we add user's anonymity property. As we show in this work, solving the Anonymous RAM problem is not trivial.

- In our second work , we define Randomize-or-Change (RoC), a public-key encryption scheme, which allows chaining of ciphertexts, while protecting integrity of plaintexts. For a pair of ciphertexts, called input and output ciphertexts, an external adversary should not be able to distinguish whether the output ciphertext was created by the recipient of the input ciphertext, or the output ciphertext is the re-randomized version of the input ciphertext created by some other user. The ciphertexts in the chain are anonymous, i.e. they are not associated with public keys explicitly. To ensure the integrity of plaintexts, a pair of ciphertexts can be publicly verified. The starting point of this work is universally re-randomizable encryption schemes (UREnc), which allow anyone to re-randomize ciphertexts without knowing the corresponding public keys. However, adding the integrity of plaintexts propeprty alone diverges RoC from UREnc in terms of the achieved security properties and applications.

- In our third work, we improve efficiency in so-called non-interactive zero-knowledge proofs. Zero-knowledge protocols allow a prover to prove a statement to a verifier such that the verifier at the end of the protocol is convinced that the statement is true, without learning anything else, in particular, without learning the prover's secret. Protocols for zero-knowledge proofs can be used via anonymous channels, hence constitute an important building block for constructing more complex systems with anonymity guarantees.

- Finally, in our fourth work, we present a framework for constructing efficient protocols for Single Secret Leader Election (SSLE). In SSLE, the participating parties would like to randomly select a leader such that at the end of the protocol only the leader knows that it is elected, i.e. anonymity of the leader is achieved. At some point later, the leader presents a proof of leadership and some payload, which depends on the application, in which an SSLE protocol in used.

Anonymous RAM In [P1], we define the concept of and present provably secure constructions for *Anonymous RAM (AnonRAM)*, a novel multi-user storage primitive that offers strong privacy and integrity guarantees. AnonRAM combines privacy features of anonymous communication and oblivious RAM (ORAM) schemes, allowing it to protect, simultaneously, the *privacy of content, access* *patterns* and *user's identity*, from curious servers and from other (even adversarial) users. AnonRAM further protects *integrity*, i.e., it prevents malicious users from corrupting data of other users.

We present two secure AnonRAM schemes, differing in design and time complexity. The first scheme has a simpler design; like efficient ORAM schemes, its time complexity is poly-logarithmic in the number of cells (per user); however, it is *linear* in the number of users. The second AnonRAM scheme reduces the overall complexity to poly-logarithmic in the total number of cells (of all users) at the cost of requiring two non-colluding servers.

Randomize-or-Change Encryption In [P2], we introduce *randomize-or-change encryption (RoC)*, an encryption scheme which allows any party to *randomize* ciphertexts, and where the *recipient* may *change* the plaintext, i.e., output encryption of a new message. On the one hand, RoC ensures *operation indistinguishability*, i.e., an eavesdropper cannot tell if an outgoing ciphertext is a randomization of the incoming ciphertext, or if the ciphertext is the encryption of a different message (by the recipient). On the other hand, RoC also ensures *universal verifiability*, i.e., anyone can validate that the output is, indeed, either the randomization of the incoming ciphertext *or* encryption of a new message (by the recipient), i.e., valid 'randomize-or-change encryption'. This property protects the integrity of re-randomized ciphertexts, together with *recipient anonymity*. Furthermore, we introduce *aggregated* RoC, which allows to publicly aggregate multiple ciphertexts linked to the same source ciphertext. We show three applications for RoC: parallel anonymous RAM, group payment system, and anonymous communication. For the first application, we significantly improve latency over our anonymous RAM construction [P1].

We present a class of RoC constructions, which extends universal re-encryption construction by Golle et al. [77]. To improve scalability, we then propose a class of hybrid RoC constructions, that composes a RoC with a pseudorandom generator, in a novel manner such that RoC can be employed for large messages. The hybrid constructions trade off scalability for security, that is, the recipient of randomized ciphertexts can identify randomizers. Note that existing universally re-randomizable encryption (UREnc) schemes provide neither verifiability nor operation indistinguishability.

Non-Interactive Zero-Knowledge Proofs in Cross-Domains With the recent emergence of efficient zero-knowledge (ZK) proofs for general circuits, while efficient zero-knowledge proofs of algebraic statements have existed for decades, a natural challenge arose to combine algebraic and non-algebraic statements. Chase et al. [38] proposed an interactive ZK proof system for this cross-domain problem. As a use case they show that their system can be used to prove knowledge of a RSA/DSA signature on a message m with respect to a publicly known Pedersen commitment. One drawback of their system is that it requires interaction between the prover and the verifier. This is due to the interactive nature of garbled circuits, which are used in their construction. Subsequently, Agrawal et al. [2] proposed an efficient non-interactive ZK (NIZK) proof system for cross-domains based on succinct non-interactive arguments of knowledge (SNARKs), which, however, require a trusted setup assumption.

In [P4], we propose a NIZK proof system for cross-domains that requires no trusted setup and is efficient both for the prover and the verifier. Our system constitutes a combination of Schnorr-based ZK proofs and ZK proofs for general circuits by Giacomelli et al. [71]. The proof size and the running time of our system are comparable to the approach by Chase et al. Compared to Bulletproofs [25], a recent NIZK proofs system on committed inputs, our techniques achieve asymptotically better performance on prover and verifier, thus presenting a different trade-off between the proof size and the running time.

Single Secret Leader Election from Multi-Party Computation The emergence of distributed digital currencies has raised the need for a reliable consensus mechanism. In proof-of-stake cryptocurrencies, the participants periodically choose a closed set of validators, who can vote and append transactions to the blockchain. Each validator can become a leader with the probability proportional to its stake. Keeping the leader private yet unique until it publishes a new block can significantly reduce the attack vector of an adversary and improve the throughput of the network. The problem of Single Secret Leader Election (SSLE) was first formally defined by Boneh et al. [18] in 2020.

In [**backes2021ssle:inProgress**], we propose a novel framework for constructing SSLE protocols, which relies on secure multi-party computation (MPC) and satisfies the desired security properties. Our framework does not use any shuffle or sort operations and has a computational cost for N parties as low as $O(N)$ of basic MPC operations per party. Moreover, our SSLE scheme efficiently handles weighted elections. That is, for a total weight S of N parties, the associated costs are only increased by a factor of $\log S$. When the MPC layer is instantiated with techniques based on Shamir's secret-sharing, our SSLE has a communication cost of $O(N^2)$ which is spread over $O(\log N)$ rounds, can tolerate up to $t < N/2$ faulty nodes without restarting the protocol, and its security relies on the decisional Diffie-Hellman assumption in the random oracle model. When the MPC layer is instantiated with more efficient techniques based on garbled circuits, our SSLE requires all parties to participate, up to $N - 1$ of which can be malicious, and its security is based on the random oracle model. Our results show that 128 parties can execute our SSLE protocol in under 7 minutes in a practical setup.

The rest of the thesis is organized as follows. In Chapter 2, we define AnonRAM schemes and present two constructions thereof. In Chapter 3, we define randomize-

or-change encryption (RoC) schemes in three flavors – basic, aggregatable, and keyed RoC – and present constructions for each of them; we then extend AnonRAM to parallel AnonRAM schemes and show how to build them from aggregatable RoC. Chapter 4 is devoted to efficient non-interactive zero-knowledge proofs in cross-domains. Finally, in Chapter 5 we present our framework for constructing efficient protocols for Single Secret Leader Election. Chapter 6 concludes.

2 Anonymous RAM

2.1 Introduction

The advent of cloud-based outsourcing services has been accompanied by a growing interest in *security and privacy*, striving to prevent exposure and abuse of sensitive information by adversarial cloud service providers and users. This includes, in particular, the tasks of *data privacy*, i.e., hiding users' data from overly curious entities such as the provider, as well as *access privacy*, i.e., hiding information about data-access patterns such as *which* data element is being accessed and *how* (read/write?). The underlying rationale is that exposure of data access patterns may often lead to a deep exposure of *what* the user intends to do. An extensive line of research has produced impressive results and tools for achieving both data and access privacy. In particular, oblivious RAM (ORAM) schemes, first introduced by Goldreich and Ostrovsky [105], have been extensively investigated in the last few years, yielding a multitude of elegant and increasingly efficient results [121, 94, 96, 108, 126, 45, 116, 80].

Another important privacy goal is to hide *who* is accessing the data, i.e., conceal the *identity* of the user to ensure anonymity. This area spawned extensive research and multiple protocols and systems for anonymous communication [40, 51, 41, 55]. The Tor network [124] currently constitutes the most widely used representative of these works.

We focus on the combination of these two goals: hiding content and access patterns as offered by ORAM schemes, but also concealing the user identities as offered by anonymous communication protocols. Experts in the relevant areas may not be completely surprised to find that designing this primitive is quite challenging. In particular, the privacy guarantees cannot be constructed by solely combining both approaches: the naïve idea to achieve these privacy properties simultaneously is to maintain separate ORAM data structures for each user and have users access the system using the anonymous communication protocol. However, this construction does not hide the access patterns, since the server can determine if the same data structure is accessed twice, and thereby trivially link two accesses made by the same anonymous user. Instead of multiple ORAMs, one could try to use a single ORAM as a black-box with data of all users contained in it. However, this does not work either, as inherently, the users have to share the same key, and the privacy properties immediately fail in the presence of curious adversaries. (See Section 2.3 for more details.) Supporting multiple, potentially malicious (or even 'just curious') users is significantly harder than supporting multiple cooperating clients (e.g., devices of the same user), as in [81, 130, 97, 63].

Furthermore, when considering an adversarial environment and, in particular, malicious users, *integrity*, i.e., preventing one user from corrupting data of other users, is also critical. Notice that the (popular) 'honest-but-curious' model is easier to justify for servers (e.g., running ORAM) than for clients; handling (also) malicious client is very important. Note also that ensuring integrity is fairly straightforward, when users can be identified securely; however, this conflicts with the goals of anonymity and, even more, with the desire for oblivious access, i.e., hiding even the pattern of access to data. As often happens in security, the mechanisms for the different goals do not seem to nicely combine, resulting in a rather challenging problem, to which we offer the first – but definitely not final – pair of solutions, albeit with significant limitations and room for improvement.

Our Contributions We define *Anonymous RAM* (AnonRAM) schemes and present two constructions that are provably secure in the random oracle model. AnonRAM schemes support multiple users, each user owning multiple memory cells. AnonRAM schemes simultaneously hide data content, access patterns, and the users' identities against honest-but-curious servers and against malicious users of the same service while ensuring that data can only be modified by the legitimate owner.

The first scheme, called $\mathsf{AnonRAM}_{\mathsf{lin}}$, realizes a conceptually simple transformation that turns any secure single-user ORAM scheme into a secure AnonRAM scheme (that supports multiple users). The key idea here is to convert every single-user ORAM cell to a multi-cell having a cell for each user, and to employ re-randomizable encryption such that a user can hide her identity by re-randomizing all other cells in a multi-cell while updating her own cell. The drawback of $\mathsf{AnonRAM}_{\mathsf{lin}}$, however, is that its complexity is linear in the number of users (although poly-logarithmic in the number of cells per users). This linear complexity stems from the requirement that a user has to touch one cell of each user when accessing her own cell.

The second scheme, called $\mathsf{AnonRAM}_{\mathsf{polylog}}$, reduces the overall complexity to poly-logarithmic in the number of users. This comes at the cost of requiring two non-colluding servers S and T. Server S maintains all user data in encrypted form using a *universal* re-encryption scheme, thereby disallowing S and other users to establish a mapping between a user and her data blocks. Essentially, $\mathsf{AnonRAM}_{\mathsf{polylog}}$ constitutes an extension of hierarchical ORAM designs, e.g., by Goldreich-Ostrovsky [74], where the reshuffle operation and mapping to 'dummy' blocks are performed by the dedicated server T. This prevents user deanonymization by the server S or by other users. Furthermore, mappings to specific buckets are achieved by means of a specific oblivious PRF.

For the sake of exposition, we first describe simplified variants of both schemes in the presence of honest-but-curious users. We subsequently show how to extend both constructions to handle malicious users as well. The extension mainly involves adding an integrity element to the employed (universal) re-encryption, such that any user can only re-encrypt data of other users, but not corrupt it.

Finally, we consider it an important contribution that we present a rigorous model and a definition for this challenging problem of AnonRAM, and show their suitability by providing provable security protocol instantiations.

Related Work Several multi-*client* ORAM solutions have been proposed in literature. Goodrich et al. [81] observe that stateless ORAM schemes, in which no state is carried from one access action to another, are suitable for a group of trusted clients. [23, 44] address the concurrent accesses by multiple client devices of the same user in the synchronous model, while [130, 120, 15, 112] deal with asynchronous concurrent accesses.

Franz et al. [63] introduce the concept of delegatable ORAM, where a (trusted) database owner can delegate access rights to other users and periodically performs reshuffling to protect the privacy of their accesses. [97] allows a storage owner to share a server-side ORAM structure among a group of users, but assumes that all users share the same symmetric key which none of them is going to provide to the server. These works, however, do not protect privacy of a client from malicious or 'curious' clients.

AnonRAM schemes avoid the strong non-collusion assumption between the users and the storage server. In other words, we consider the problem of anonymously accessing the server by multiple users, where the server (cooperating with some users) should not be able to learn which honest user accessed which cell over the server. Notably, we achieve our stronger privacy guarantees against a stronger adversary without requiring any communication among the users.

The only other multi-*user* ORAM scheme has been proposed by Zhang et al. [89]. Their scheme uses a set of intermediate nodes to convert a user's query to an ORAM query to the server. Privacy of the scheme is, however, analyzed only for individual non-anonymous user accesses and not for multi-user anonymous access patterns. Furthermore, their scheme does not provide integrity protection against malicious users. Moreover, their work lacks both definitions and proofs; as the reader will see in our work, the definitions and proofs, we found necessary to claim security of our schemes, are non-trivial.

Chapter Outline The rest of the chapter is organized as follows. In Section 2.2, we introduce and define AnonRAM schemes. Next, we present two AnonRAM schemes: in Section 2.3 our linear AnonRAM and in Section 2.4 our polylogarithmic AnonRAM ($\mathsf{AnonRAM}_{\mathsf{polylog}}$). In Section 2.5, we present an OPRF construction that is used in $\mathsf{AnonRAM}_{\mathsf{polylog}}$. In Section 2.6, we show the details for making $\mathsf{AnonRAM}_{\mathsf{polylog}}$ secure against malicious adversaries. In Section 2.7, we complete our security analysis. Section 2.8 concludes.

2.2 AnonRAM Definitions

We consider a set of N users $\mathcal{U} = \{\mathsf{U}_1, \dots, \mathsf{U}_N\}$, a set of η servers $\mathcal{S} = \{\mathsf{S}_1, \dots, \mathsf{S}_\eta\}$, a set Σ of messages, and we let M denote the number of data cells available to each user. All protocols are parametrized by the security parameter 1^λ. Before defining the class of AnonRAM schemes, we provide the definitions of access requests and access patterns.

Definition 2.2.1 (Access Requests). *An* access request *AR is a tuple* $(j, \alpha, m) \in [1, M] \times \{\mathsf{Read}, \mathsf{Write}\} \times \Sigma$*. Here j is called the* (cell) index *of AR, α the* access type*, and m the* input message*.*

Intuitively, an access request (j, α, m) will denote that m should be written into cell j (if $\alpha = \mathsf{Write}$), or that the content of cell j should be read (if $\alpha = \mathsf{Read}$; in this case m is ignored and we often just write $(j, \alpha, *)$).

Definition 2.2.2 (Access Patterns). *An* access pattern *is a series of tuples (i, AR_i) where $i \in [1, N]$ is a user identifier and AR_i is an access request.*

For notational simplicity, we will write (i, j, α, m) instead of $(i, (j, \alpha, m))$ for the individual elements of access patterns.

We next define AnonRAM schemes. In this work, we consider sequential schemes where one participant is active at any point in time.

Definition 2.2.3 (AnonRAM Schemes). *An* AnonRAM *scheme is a tuple* $(\mathsf{Setup}, \mathsf{User}, \mathsf{Server}_1, \ldots, \mathsf{Server}_\eta)$ *of $\eta + 2$ PPT algorithms, where:*

- *The* initialization algorithm Setup *maps a security parameter 1^λ and an identifier id, to an initial state, where $id \in \{0, 1, \ldots, \eta\}$ identifies one of the servers (for $id > 0$) or the user (for $id = 0$).*
- *The* user algorithm User *processes two kinds of inputs: (a) access requests (from the user) and (b) pairs (l, m) where $l \in [1, \eta]$ denotes a server and m a message from server S_l. User maps the current state and input to a new state and to either a response provided to the user or a pair (l, m) with $l \in [1, \eta]$ denoting a server and m being a message for S_l.*
- *The* server algorithm Server_l *for server S_l maps the current server state and input (message from user or from another server) to a new server state and a message either to the user or to another server.*

Adversarial Models and Protocol Execution We consider two different adversarial models: (i) *honest-but-curious* (HbC) adversaries that learn the state of one server S^* and of a subset $\mathcal{U}^*$ of users, and (ii) *malicious users* ($\mathsf{Mal\_Users}$) adversaries that learn the state of one server (as before) and additionally control a subset $\mathcal{U}^*$ of users. In both models, the adversary can additionally eavesdrop on all messages sent on the network, i.e., between users and servers, and between two servers.

We now define the sequential execution $\mathsf{Exec}(\mathcal{AR}, \mathsf{Adv}, AP, \zeta)$ of an AnonRAM scheme $\mathcal{AR}$ in the presence of an adversary Adv and a given access pattern AP assuming an adversarial model $\zeta \in \{\mathsf{HbC}, \mathsf{Mal\_Users}\}$.

Definition 2.2.4 (Execution)**.** *Let $\mathcal{AR}$ be an AnonRAM scheme* $(\mathsf{Setup}, \mathsf{User}, \mathsf{Server}_1, \ldots, \mathsf{Server}_\eta)$*,* Adv *be a PPT algorithm,* $\zeta \in \{\mathsf{HbC}, \mathsf{Mal\_Users}\}$ *and AP be an access pattern. The* execution $\mathsf{Exec}(\mathcal{AR}, \mathsf{Adv}, AP, \zeta)$ *is the following randomized process:*

1. All parties are initialized using Setup*, resulting in initial states σ_{U_i} for each user* U_i*, and σ_{S_l} for each server* S_l*.*

2. Adv *selects a server* S^* *and a strict subset $\mathcal{U}^* \subset \mathcal{U}$.*

3. Let $(i, j, \alpha, m_{i,j})$ be the first element of AP; if AP is empty, terminate.

4. If $\mathsf{U}_i \in \mathcal{U}^*$ *and* $\zeta = \mathsf{Mal\_Users}$*, then let (l, m) be the output of* Adv *on input $(i, j, \alpha, m_{i,j})$. Otherwise, let (l, m) be the output of* User *on input $(j, \alpha, m_{i,j})$, with state σ_{U_i}, and update σ_{U_i} accordingly.*

5. Invoke S_l *with (input) message m. The server* S_l *may call other servers (possibly recursively) and finally produces an (output) message m'.*

6. If $\mathsf{U}_i \in \mathcal{U}^*$ *and* $\zeta = \mathsf{Mal\_Users}$*, provide the message m' to* Adv*. Otherwise, provide m' to user* U_i*.* U_i *(*Adv *if* $\mathsf{U}_i \in \mathcal{U}^*$ *and* $\zeta = \mathsf{Mal\_Users}$*) may repeat sending messages to any servers. Eventually,* U_i *(*Adv*) terminates.*

7. Repeat the loop (from step 3) with the next element of AP (until empty).

Throughout the execution, the adversary learns the internal states of S^* *and of all users in $\mathcal{U}^*$, as well as all messages sent on the network.*

A trace *is the random variable defined by an execution, using uniformly random coin-tosses for all parties. The trace includes the sequence of messages in the execution corresponding to access requests and the final state of the adversary. Let $\Theta(x)$ denote the trace of execution x.*

Privacy and Integrity of AnonRAM schemes To define privacy for AnonRAM schemes, we consider an additional PPT adversary $\mathcal{D}$ called the distinguisher. $\mathcal{D}$ outputs two arbitrary access patterns of the same finite length, which differ only in inputs to unobserved users. We then randomly select and execute one of these two patterns. The distinguisher's goal is to identify which pattern was used. Since these two accesses may differ in user, cell, operation, or value, this definition encompasses all relevant privacy properties in this setting including anonymity (identity privacy), confidentiality (value privacy), and obliviousness (cell and operation privacy). We call an adversary Adv *compliant* with a pair of access patterns (AP_0, AP_1) if Adv only outputs sets $\mathcal{U}^*$ of users in Step 2) of $\mathsf{Exec}(\mathcal{AR}, \mathsf{Adv}, AP_0, \zeta)$ and $\mathsf{Exec}(\mathcal{AR}, \mathsf{Adv}, AP_1, \zeta)$ such that AP_0 and AP_1 are identical when restricted to users in $\mathcal{U}^*$.

Definition 2.2.5 (Privacy of AnonRAM)**.** *An AnonRAM scheme $\mathcal{AR}$ preserves* privacy *in adversarial model* $\zeta \in \{\mathsf{HbC}, \mathsf{Mal\_Users}\}$*; if for every pair of (same finite length) access patterns (AP_0, AP_1) and for every pair of PPT algorithms* $(\mathsf{Adv}, \mathcal{D})$ *s.t.* Adv *is compliant with (AP_0, AP_1), we have that*

$$\left|\Pr\left[b^* = b \colon b^* \leftarrow \mathcal{D}\left(\Theta\left(\mathsf{Exec}(\mathcal{AR}, \mathsf{Adv}, AP_b, \zeta)\right)\right)\right] - \frac{1}{2}\right|$$

is negligible in 1^λ, where the probability is taken over uniform coin tosses by all parties, and $b \leftarrow_R \{0,1\}$.

Note that when all-but-one (i.e., $N-1$) users are observed and $\zeta = \mathsf{HbC}$, our privacy property corresponds to the standard ORAM access privacy definition [74]. ORAM is hence a special case of AnonRAM with a single user ($N = 1$).

AnonRAM should ensure *integrity* to prevent invalid executions caused by parties deviating from the protocol. Informally, a trace is invalid if a value read from a cell does not correspond to the most recently written value to the cell.

Definition 2.2.6 (Integrity of AnonRAM)**.** *Let ϑ be a trace of execution with access pattern AP, and let $AR = (j, \mathsf{Read}, *)$ with $(i, AR_i) \in AP$ be a read request for cell j of user U_i, returning a value x. Let $AR' = (j, \mathsf{Write}, x')$ be the most recent previous write request to cell j of user U_i in AP, or $\perp$ if there was no such previous write request. If $x \neq x'$, we say that this read request is* invalid*. If any read request in the trace is invalid, then the trace is invalid.*

An AnonRAM scheme $\mathcal{AR}$ preserves integrity *if there is negligible (in 1^λ) probability of invalid traces when the traces are constrained to the view of the honest users (all $\mathsf{U}_i \in \mathcal{U}$ in the* HbC *model, and all users $\mathsf{U}_i \in \mathcal{U}/\mathcal{U}^*$ in the* Mal_Users *model) for any PPT adversary and any access pattern AP.*

2.3 Linear-complexity AnonRAM

In this section, we present our first AnonRAM constructions and prove them secure in the underlying model. For the sake of exposition, we start with a few seemingly natural but flawed approaches to construct AnonRAM schemes.

2.3.1 Seemingly Natural but Flawed Approaches

A first natural idea to design an AnonRAM scheme is to maintain all the $M \cdot N$ cells in an encrypted form on the server and to only access them via an anonymous channel such as Tor [124]. However, this approach fails to achieve AnonRAM privacy, since the adversary can simply observe all memory accesses on the server and thereby determine how often the same cell j of a user is accessed. One may try to overcome this problem using a shared $(M \cdot N)$-cell *stateless* ORAM [81] containing M cells for each of N users and assuming that every user executes her ORAM requests via an anonymous channel. In this case, all users will have to use the *same* private key in the symmetric encryption scheme employed in the ORAM protocol to hide their cells from the server. However, this allows Eve, an HbC user, to break privacy of honest users, by observing the values in cells (allocated to honest users) which she downloads and decrypts as part of her legitimate ORAM requests.

Another natural design would be to use a separate ORAM for the M cells of each user and rely on anonymous access to hide user identities. This use would hide the users' individual access patterns, but the server can identify all accesses by the same user and thereby violate the AnonRAM privacy requirement.

The AnonRAM schemes presented in this work overcome such problems by having users re-randomize cells belonging to other users as well whenever their own cells are being accessed in addition to encrypting the user's own cells.

2.3.2 $\mathsf{AnonRAM}_{\mathsf{lin}}$ and its Security Against HbC Adversaries

We now present the $\mathsf{AnonRAM}_{\mathsf{lin}}$ construction and prove it secure in the HbC adversarial model. $\mathsf{AnonRAM}_{\mathsf{lin}}$ uses an anonymous communication channel [124] and the (single-user, single-server) Path ORAM [121] or other ORAM scheme satisfying a property identified below.

In Path ORAM, the user's cells are stored on the server RAM as a set of encrypted data *blocks* such that each block consists of a single ciphertext and all blocks are encrypted with the same key known to the user's ORAM client. A block encrypts either a user's cell, or auxiliary information used by the User algorithm. To access a cell, the ORAM client *reads* (and decrypts) a fixed number of blocks from the server, and *writes* encrypted values (cells or some special messages) in a fixed number of blocks. The server's duty is to execute these user's read and write requests.

$\mathsf{AnonRAM}_{\mathsf{lin}}$ employs N instances (one per user) of Path ORAM for M cells each while requiring a single server.[1] To encrypt data as required in the ORAM scheme, $\mathsf{AnonRAM}_{\mathsf{lin}}$ uses a semantically secure re-randomizable encryption (RE) scheme $(\mathsf{E}, \mathsf{R}, \mathsf{D})$ (e.g., ElGamal encryption), where E, R, and D are respectively encryption, re-randomization, and decryption operations. The $\mathsf{AnonRAM}_{\mathsf{lin}}$ client of user U_i, has access to her private key sk_i and to the public keys $(pk_1, \ldots, pk_N)$ of all users. In $\mathsf{AnonRAM}_{\mathsf{lin}}$, the ORAM scheme uses this RE scheme $(\mathsf{E}, \mathsf{R}, \mathsf{D})$ instead of the (symmetric) encryption scheme used in 'regular' Path ORAM.

Intuitively, an $\mathsf{AnonRAM}_{\mathsf{lin}}$ client internally runs an ORAM client and mediates its communication with the server. Whenever the ORAM client reads or writes a specific block, the $\mathsf{AnonRAM}_{\mathsf{lin}}$ client performs corresponding read or write operations *for all users*, without divulging the user identity to the server at the network level, as follows: Reading a block of another user can be trivially achieved, since the block is encrypted for the owner's key, but the contents are not used (our goal is only to create indistinguishable accesses for all users). Writing a block belonging to other user's ORAM must not corrupt the data inside and is hence achieved by re-randomizing the blocks of other users.

The Setup and Server algorithms of $\mathsf{AnonRAM}_{\mathsf{lin}}$ are simply N instances of the

[1] $\mathsf{AnonRAM}_{\mathsf{lin}}$ can also use an ORAM scheme that uses multiple servers. In this case, $\mathsf{AnonRAM}_{\mathsf{lin}}$ will use the same number of servers.

```
upon access request $(j, \alpha, m)$ from user $\mathsf{U}_i$ :
     Invoke the ORAM client $\mathcal{O}_c$ with access request $(j, \alpha, m)$.
upon read request from $\mathcal{O}_c$ for block $j'$ :
    for $k \in [1, N]$ do Let $B[j', k] \leftarrow$ Read block $j'$ of user $\mathsf{U}_k$ kept by the server.
    Return $B[j', i]$ to $\mathcal{O}_c$.
upon write request from $\mathcal{O}_c$, for value (ciphertext) $B$ in block $j'$ :
    $B[j', i] \leftarrow B$
    for $k \in [1, N] | k \neq i$ do $B[j', k] \leftarrow \mathsf{R}_k(B[j', k])$
    for $k \in [1, N]$ do
        Write block $B[j', k]$ to position $j'$ of user $\mathsf{U}_k$ and release it from memory.
upon Receiving a result $res$ from $\mathcal{O}_c$ :
    Return $res$ to $\mathsf{U}_i$.
```

Figure 2.1: User algorithm of AnonRAM$_{\mathsf{lin}}$ with access request (j, α, m) for user U_i.

corresponding algorithm of the underlying ORAM scheme (e.g., Path ORAM). Namely, the Setup initializes state for N copies of the ORAM (one per user) and the Server receives a 'user identifier' i together with each request, and runs the ORAM's Server algorithm using the i^{th} state over the request. The Server algorithm for the AnonRAM$_{\mathsf{lin}}$ scheme simply processes Read/Write requests sent by the users as in the ORAM scheme, e.g. the server returns the content of the requested block for Read requests or overrides the content of the requested block with the new value for Write requests.

We finally describe the User algorithm of AnonRAM$_{\mathsf{lin}}$ using pseudocode in Fig. 2.1 to increase readability. It relies on an oracle $\mathcal{O}_c$ for the ORAM client, and an RE scheme $(\mathsf{E}, \mathsf{R}, \mathsf{D})$. We write $(\mathsf{E}_i, \mathsf{R}_i, \mathsf{D}_i)$ for the corresponding encryption, re-encryption and decryption operations using the corresponding keys for user U_i. The pseudocode depicts which operations are performed for an individual access request (j, α, m) of user U_i. Its execution starts with invoking user U_i's local ORAM client $\mathcal{O}_c$ with the access request (j, α, m), and ends with a **Return** message to U_i. The process involves multiple instances of Read and Write requests from $\mathcal{O}_c$ for specified blocks kept by the server. These requests to Read and Write blocks kept by the server should not be confused with access requests (j, α, m), where $\alpha \in \{\mathsf{Read}, \mathsf{Write}\}$ for ORAM cells.

So far, we selected Path ORAM as a specific ORAM instantiation. However, any other ORAM scheme is equally applicable, provided that it exhibits an additional property: individual accesses have to be indistinguishable, i.e., the adversary observing just one access request from an access pattern should not be able to recognise how many accesses the honest user performed so far. We call this property *indistinguishability of individual accesses*, and it is trivially satisfied by Path ORAM. Hierarchical ORAMs (e.g., [74, 81, 94, 96]), however, do not achieve indistinguishability of individual accesses, as the runtime of individual accesses depends on the number of accesses performed so far; in particular, the client has to reshuffle periodically a variable amount of data.

Theorem 2.3.1. $\mathsf{AnonRAM}_{\mathsf{lin}}$ *preserves access privacy in the adversarial model* HbC*, when using a secure ORAM scheme* $\mathcal{O}$ *that satisfies indistinguishability of individual accesses, and a semantically secure re-randomizable encryption scheme* $(\mathsf{E}, \mathsf{R}, \mathsf{D})$.

Proof. Assume to the contrary that some PPT HbC adversary $\mathcal{D}$ can efficiently distinguish, with a non-negligible advantage, between a pair of access-patterns $AP = \{(i_u, j_u, \alpha_u, m_u)\}, AP' = \{(i'_u, j'_u, \alpha'_u, m'_u)\}_{u\in[1,len]}$ of length len.

Let $AP_v = \{(i^*_u, j^*_u, \alpha^*_u, m^*_u)\}_{u\in[1,len]}$ be a 'hybrid' access pattern, where $(i^*_u, j^*_u, \alpha^*_u, m^*_u) = (i_u, j_u, \alpha_u, m_u)$ for $u \leq v$, and $(i^*_u, j^*_u, \alpha^*_u, m^*_u) = (i'_u, j'_u, \alpha'_u, m'_u)$ for $u > v$. In fact, let v be the smallest such value, where some adversary (say $\mathcal{D}$) can distinguish between AP_{v-1} and AP_v, and such $v > 0$ exists by the standard 'hybrid argument' as AP and AP' differ at least in one access.

If $i_v = i'_v$ (i.e., for the same user), the executions only differ in the ORAM client $\mathcal{O}_c$ Read/Write blocks for U_{i_v}; however, this immediately contradicts the privacy of the underlying ORAM scheme. Notice that a user does not decrypt or modify the other users' data during her accesses.

Therefore, assume $i_v \neq i'_v$. Since we expect our ORAM client $\mathcal{O}_c$ to satisfy indistinguishability of individual accesses, the difference between these two patterns is only between the encryption of the blocks output by $\mathcal{O}_c$ and the re-encryption of the blocks received anonymously by the ORAM server. However, ability to distinguish between these, contradicts the indistinguishability property of the semantically secure re-randomizable encryption scheme $(\mathsf{E}, \mathsf{R}, \mathsf{D})$. □

Let c_S and c_B denote the amortized costs of client-side storage and communication complexity of the underlying ORAM protocol. Then, the respective amortized costs of $\mathsf{AnonRAM}_{\mathsf{lin}}$ are $N \cdot c_S$ and $N \cdot c_B$. For example, using Path ORAM, the client-side storage and communication complexity costs of $\mathsf{AnonRAM}_{\mathsf{lin}}$ become $O(N \log M)$ and $O(N \log^2 M)$.

2.3.3 $\mathsf{AnonRAM}^{\mathrm{M}}_{\mathsf{lin}}$ and its Security Against Malicious Users

When some users are malicious, we need to ensure that only a user knowing the private key associated with a block can update the value inside the block, while other users should only be able to re-randomize it. Leveraging the security of $\mathsf{AnonRAM}_{\mathsf{lin}}$ to the adversarial model of malicious users, we require a semantically secure encryption primitive such that a ciphertext C' can replace a ciphertext C if C' is a re-randomization of C, or if the encryptor knows the encryption key for C. Whenever a block is written, the user attaches a zero-knowledge proof showing *either* that the ciphertext is re-encryption of the previous ciphertext *or* that the user has the (secret) encryption key. The server verifies the proof before updating the block in its RAM memory. This ensures indistinguishability of re-encryption from new encryptions, while ensuring that one user cannot corrupt or modify any value of another user. We denote the resulting scheme as $\mathsf{AnonRAM}_{\mathsf{lin}}^{\mathbf{M}}$.

The required zero-knowledge (ZK) proofs are standard. For the re-randomizable CPA-secure ElGamal encryption scheme, this will involve a ZK proof of knowledge of discrete logarithm [114] and a ZK proof of equality of the discrete logarithm of two pairs of group elements [43] composed in such a way that a user proves validity of one of the statements, without releaving to the server which statement has been proven [49, 53] (see also Example 3 of [30]). Following the formal notation from [87] and extending it for proving "one-out-of-several" statements, the required proof is

$$PoK\{x_i |\ pk_i = g^{x_i}\} \ \vee \ P\{\exists r \mid (c_1', c_2') = (c_1^r, c_2^r)\}, \tag{2.1}$$

where P stands for proof, PoK for proof of knowledge, $\vee$ denotes a disjunction of several parts of the proof system where one part has to be proven; g is a generator of a group of prime order q, $(pk_i, c_1, c_2, c_1', c_2')$ are group elements, and (x_i, r) are elements in $\mathbb{Z}_q$. Let (C_1, C_2) be the input ciphertext encrypted using public key pk_i, then a re-randomized ciphertext is computed as $(C_1', C_2') \leftarrow (C_1 \cdot g^r, C_2 \cdot pk_i^r)$. After re-arranging terms, we get $(c_1, c_2, c_1', c_2') = (g, pk_i, C_1'/C_1, C_2'/C_2)$. For the sake of exposition, we will use c-notation throughout this section.

Theorem 2.3.2. $\mathsf{AnonRAM}_{\mathsf{lin}}^{\mathbf{M}}$ *based on a secure ORAM scheme $\mathcal{O}$ that satisfies indistinguishability of individual accesses, CPA-secure public-key encryption scheme (e.g., ElGamal), and a ZK proof defined above, preserves integrity and privacy in the adversarial model* Mal_Users*.*

Proof. The integrity argument is simple: the use of ZK proofs for proving one-out-of-two statements effectively reduces the adversarial abilities to the ones as in the HbC model. The P-part of (2.1) corresponds to the honest behavior; to break integrity the adversary must prove the PoK-part (we consider the integrity is broken if the P-part does not hold). We can use an adversary that, given (c_1, c_2), produces with non-negligible probability a valid proof for (c_1, c_2, c_1', c_2'), where $(c_1', c_2') \neq (c_1^r, c_2^r)$ for any $r > 0$, to solve the discrete log problem with non-negligible probability. Consider two games G_0 and G_1, where G_0 is the normal execution of the AnonRAM protocol, and G_1 is same as G_0, expect that the challenger replaces public key g^{x_i} for user $\mathsf{U}_i \in \mathcal{U}/\mathcal{U}^*$ with X^a , where X is a group element with unknown discrete logarithm and a is drawn randomly; elements (c_1, c_2) are computed as $(c_1, c_2) \leftarrow (g^r, X^{a \cdot r})$. Games G_0 and G_1 are indistinguishable to the adversary, as the public key for user U_i is a random group element in either game. Upon receiving a valid proof in G_1 from the adversary, the challenger runs the extractor to learn the discrete log of X^a and substracts a to obtain the discrete log of X with non-negligible probability, which contradicts the hardness of the discrete log assumption.

The privacy properties are also preserved similarly to $\mathsf{AnonRAM}_{\mathsf{lin}}$ as the disjunctive nature of the included ZK proof does not allow the server to determine which of N cells is modified by an honest user, while privacy of the accessed cell-index as well as the access type is maintained by the employed ORAM scheme. □

2.4 Polylogarithmic-complexity AnonRAM

The $\mathsf{AnonRAM}_{\mathsf{lin}}$ scheme exhibits acceptable performance for a small number of users, but the linear overhead renders it prohibitively expensive as the number of users increases. In this section, we present $\mathsf{AnonRAM}_{\mathsf{polylog}}$, an AnonRAM scheme whose overhead is poly-logarithmic in the number of users.

$\mathsf{AnonRAM}_{\mathsf{polylog}}$ is conceptually based on the hierarchical Goldreich-Ostrovsky ORAM (GO-ORAM) construction [74], where a user periodically reshuffles her cells maintained over a *storage server* S. To reshuffle cells belonging to multiple users, we introduce in $\mathsf{AnonRAM}_{\mathsf{polylog}}$ an additional server, the so-called *tag server* T. The tag server reshuffles data on the users' behalf, without knowing the data elements, and thereby maintains user privacy from the storage server S as well as from the other users. The tag server only requires constant-size storage to perform this reshuffling, and we show that, similarly to the storage server, it cannot violate (on its own or with colluding users) the privacy requirements of AnonRAM schemes.[2]

In what follows, we first describe the employed cryptographic tools, and then present the $\mathsf{AnonRAM}_{\mathsf{polylog}}$ construction and its complexity and security analysis, first for the honest-but-curious case, and after that its extension, $\mathsf{AnonRAM}^{\mathbf{M}}_{\mathsf{polylog}}$, to cope with malicious users.

2.4.1 Cryptographic Building Blocks

Universally Re-randomizable Encryption A universally re-randomizable encryption (UREnc) scheme [77, 109] allows to re-randomize given ciphertexts without requiring access to the encryption key. We use the construction of Golle et al. [77]: for a generator g of a multiplicative group G_q of prime order q and a private/public key pair (x_i, g^{x_i}) for party i with $x_i \in \mathbb{Z}_q^*$, the encryption $C = \mathsf{E}_i^*(m)$ of a message m is computed as an El-Gamal encryption of m together with an El-Gamal encryption of the identity $1 \in G_q$; i.e., $C = (g^a, g^{ax_i} \cdot m, g^b, g^{bx_i} \cdot 1)$ for $a, b \in \mathbb{Z}_q^*$. The ciphertext C can be re-randomized, denoted $\mathsf{R}^*(C)$ by selecting $a', b' \leftarrow_R \mathbb{Z}_q^*$ and outputting $(g^a \cdot (g^b)^{a'}, g^{ax_i} \cdot m \cdot (g^{bx_i})^{a'}, (g^b)^{b'}, (g^{bx_i})^{b'})$ as the new ciphertext. Note that this scheme is also multiplicatively homomorphic.

[2]Adhering to our adversarial model from Section 2.2, we only consider the corruption of a single server, and hence assume non-colluding servers S and T.

We employ a distributed version of the UREnc scheme, where the private key is shared between two servers such that both have to be involved in decryption.

(Partially Key-Homomorphic) Oblivious PRF An oblivious pseudo-random function (OPRF) [64, 87] enables a party holding an input tag μ to obtain an output $F_s(\mu)$ of a PRF $F_s(\cdot)$ from another party holding the key s without the latter party learning any information about the input tag μ.

We use the Jarecki-Liu OPRF construction [87] as our starting point. Here, the underlying PRF $f_s(\cdot)$ is a variant of the Dodis-Yampolskiy PRF construction [57] such that $f_s(\mu) := \mathsf{g}^{1/(s+\mu)}$ is defined over a composite-order group of order $n = p_1 p_2$ for safe primes p_1 and p_2. This function constitutes a PRF if factoring safe RSA moduli is hard and the Decisional q-Diffie-Hellman Inversion assumption holds on a suitable group family G_n [87].

To securely realize pre-tag randomization in our Reshuffle algorithm (explained later), we propose a modification of the Jarecki-Liu OPRF where a second key $\hat{s}$ is used to define a new PRF $f_{s,\hat{s}}(\mu) := \mathsf{g}^{\hat{s}/(s+\mu)}$. We call such a PRF *partially key-homomorphic* as $(f_{s,\hat{s}}(\mu))^\delta = f_{s,(\hat{s}\cdot\delta)}(\mu)$ holds for it. For unlinkability of PRF values of the same input μ with updated δ, we expect the Composite DDH assumption [3] [35] to hold in G_n. We denote our OPRF construction as $\mathsf{OPRF}^{\mathcal{A},\mathcal{B}}_{s,\hat{s}}(\mu)$, where $\mathcal{A}$ denotes a party with input μ, and $\mathcal{B}$ denotes a server possessing the keys s and $\hat{s}$. Our OPRF protocol makes only minor changes to the Jarecki-Liu OPRF, and we postpone its full description and security analysis to Section 2.5.

Multiplicatively Homomorphic Encryption For appropriately computing on our OPRF outputs that are elements of a group of order n generated by g, we need a suitable multiplicatively homomorphic encryption scheme whose decryption is shared between our two servers. To this end, we employ a semantically-secure ElGamal encryption scheme, whose security relies on the DDH assumption in the underlying group. Here, an encryption $\mathsf{E}^{+*}_{\mathsf{pk}}(m)$ denotes a message $m \in \mathsf{G}_n$ encrypted under a public key $\mathsf{pk} = \mathsf{g}^{\mathsf{sk}}$, where g is a generator of group G_n of size n, and the private key sk belongs to $\mathbb{Z}^*_{n/4}$.

On the one hand, ElGamal is a multiplicatively homomorphic encryption scheme; on the other hand, the message space matches the output space of our OPRF. Therefore the scheme is additively homomorphic w.r.t. OPRF inputs: $\mathsf{E}^{+*}_{\mathsf{pk}}(m) \cdot \mathsf{E}^{+*}_{\mathsf{pk}}(m') = \mathsf{E}^{+*}_{\mathsf{pk}}(m \cdot m')$ for any $m, m' \in \mathsf{G}_n$, and $\mathsf{E}^{+*}_{\mathsf{pk}}(m)^\delta = \mathsf{E}^{+*}_{\mathsf{pk}}(m^\delta)$ for any $\delta \in \mathbb{Z}^*_n$. This scheme, moreover, allows shared decryption; i.e., given public/private key pairs $(\mathsf{pk}_\mathcal{A}, \mathsf{sk}_\mathcal{A})$ and $(\mathsf{pk}_\mathcal{B}, \mathsf{sk}_\mathcal{B})$ of parties $\mathcal{A}$ and $\mathcal{B}$ and the joint public key $\mathsf{pk} = \mathsf{pk}_\mathcal{A} \cdot \mathsf{pk}_\mathcal{B}$, parties $\mathcal{A}$ and $\mathcal{B}$ can jointly decrypt a ciphertext $\mathsf{E}^{+*}_{\mathsf{pk}}(m)$ for a receiver using their private keys $\mathsf{sk}_\mathcal{A}$ and $\mathsf{sk}_\mathcal{B}$. In our construction,

[3] Composite Decisional Diffie-Hellman assumption [35] is a variant of the standard DDH assumption [16], but defined over a composite order group.

given a ciphertext encrypted under the joint public key of servers S and T, they jointly decrypt the ciphertext such that the plaintext message is available to T.

Oblivious Sort In oblivious sort (OSort), one party (in our case, S) holds an encrypted data array and the other party (T) operates on the data array such that the data array becomes sorted according to some comparison criteria, and S learns nothing about the array (hence, the name "oblivious" sort). OSort can be instantiated, e.g., by the *randomized ShellSort* algorithm [79], which runs in $O(z\log(z))$ for z elements.

2.4.2 $\mathsf{AnonRAM_{polylog}}$ Data Structure

$\mathsf{AnonRAM_{polylog}}$ caters N independent users ($\mathsf{U}_1, \ldots, \mathsf{U}_N$) with their $M \cdot N$ cells (i.e., M cells per user) using a storage server S and a tag server T. Similarly to other hierarchical schemes [74, 108, 94, 96], blocks are organized in $L = \lceil \log(M \cdot N) \rceil + 1$ levels, where each level $\ell \in [1, L]$ contents 2^ℓ buckets. Each bucket contains $\beta := \lceil \mathbf{c}_\beta \log(M \cdot N) \rceil$ blocks, where $\mathbf{c}_\beta$ is a (small) constant.

Similarly to GO-ORAM, during each access the user reads a pseudo-randomly chosen (entire) bucket from each level such that server S cannot learn anything by observing the bucket access patterns. $\mathsf{AnonRAM_{polylog}}$ adopts a recent improvement to GO-ORAM proposed in [108, 94, 96] to avoid duplicate user blocks in the server-side (RAM) storage at any point in time. To achieve this, on every access, the user has to write a 'dummy' block into the location where it finds the data such that S cannot distinguish between the added 'dummy' block and the 'real' data block. These user-added dummy blocks are periodically removed to avoid RAM memory expansion, and the rest of the blocks are periodically reshuffled to allow users to access the same cell multiple times.

In existing single-user single-server GO-ORAM designs [74, 108, 94], this reshuffling is performed by the user. In $\mathsf{AnonRAM_{polylog}}$, reshuffling operations involve blocks of different users, and it cannot be performed by one or more users without interacting with all other users. As we want to avoid interaction among the users, reshuffling in $\mathsf{AnonRAM_{polylog}}$ is jointly performed by two *non-colluding* servers (the storage-server S and the tag-server T) without exposing users' data or access patterns to either server.

Block Types Each block in $\mathsf{AnonRAM_{polylog}}$ consists of two parts: a ElGamal-encrypted OPRF output called *pre-tag* part and a UREnc-encrypted *value* part. We consider three types of blocks: *real*, *empty*, and *dummy* blocks.

A *real* block is of the form $\langle \mathsf{E}^{+*}_{\mathsf{TS}}(\theta_i), \mathsf{E}^*_{\mathsf{U}_i}(j, m_{i,j}) \rangle$. Here, the value part contains the j^{th} cell of user U_i with value $m_{i,j}$ encrypted with UREnc for U_i, while the pre-tag part contains a pre-tag θ_i computed using OPRF for some secret input of U_i and encrypted using ElGamal for a joint public key of T and S. The pre-tag θ_i

is computed by U_i with help from the storage server S using OPRF and is used to map the block to a particular bucket on a given level. Given a pre-tag θ, for a level $\ell \in [1, L]$, the bucket index (or tag) is computed by applying a random oracle hash function, $h_\ell : \{0,1\}^* \to \mathbb{Z}_{2^\ell}$. The mapping changes after 2^ℓ accesses, which we refer to as an *epoch*.

Empty blocks are padding blocks that are used to form buckets of the required size β on the storage server S. An empty block is of the form $\langle \mathsf{E}^{+*}_{\mathsf{TS}}(1), \mathsf{E}^{*}_{\mathsf{TS}}(\text{“}\textit{empty}\text{”})\rangle$, where “*empty*” is a constant in the UREnc message space. An empty block will be encrypted similarly to other types of blocks to ensure privacy against the storage server S, and the server should not be able to determine whether a user fetched an empty block or a real block. The first part of the empty block is an encryption of unity 1; it allows the tag server T to determine if a block is empty during the reshuffle.

Finally, similarly to most ORAM algorithms, we use *dummy* blocks to hide locations of the real blocks. Once a real block with a specific index is found at some level, it is moved to a new bucket at the first level and is replaced with a dummy block in its old location. A dummy block is of the form $\langle \mathsf{E}^{+*}_{\mathsf{TS}}(\theta_{\mathcal{D}}), \mathsf{E}^{*}_{\mathsf{TS}}(\text{“}\textit{dummy}\text{”})\rangle$, where the pre-tag $\theta_{\mathcal{D}}$ is computed using OPRF on the number (t) of accesses made by the users so far and a secret input $\mu_{\mathcal{D}}$ known only to server T, and “*dummy*” is a constant in the UREnc message space.

Note that different blocks are completely indistinguishable to non-colluding servers S and T individually. Nevertheless, during the reshuffle operations, when necessary, server T can determine the type of a block with the help of server S.

2.4.3 AnonRAM$_{\text{polylog}}$ Protocol Overview

Initialization We need to initialize UREnc, ElGamal, and OPRF. For the security parameter 1^λ, we choose a multiplicative group G_q of an appropriate prime order q for UREnc, and a multiplicative group G_n of order equal to an appropriate safe RSA modulus n for ElGamal and OPRF. Let g and g be generators of groups G_q and G_n, respectively.

Given this setup, every user generates her UREnc key from $\mathbb{Z}_q^*$. The two servers select their individual shared private keys for both UREnc and ElGamal and publish the corresponding combined public key for ElGamal; we do not need UREnc public key for the two servers. We represent these encryptions as follows: $\mathsf{E}^{*}_{\mathsf{U}_i}(\cdot)$ represents a UREnc encryption for user U_i; $\mathsf{E}^{*}_{\mathsf{TS}}(\cdot)$ and $\mathsf{E}^{+*}_{\mathsf{TS}}(\cdot)$ respectively represent shared UREnc and ElGamal encryptions for the servers S and T. The servers make an encrypted empty block $\mathsf{E}^{*}_{\mathsf{TS}}(\text{“}\textit{empty}\text{”})$ and an encrypted dummy block $\mathsf{E}^{*}_{\mathsf{TS}}(\text{“}\textit{dummy}\text{”})$ public to all users.

Similarly to all existing hierarchical ORAM constructions, all levels in the AnonRAM$_{\text{polylog}}$ data structure on S are initially empty. In particular, the complete first level is filled up with empty blocks, while the rest of the levels are not yet allocated. The users write $M \cdot N$ cells initialized to some default value, one by one, at the first level such that, at the end of the initialization procedure, $M \cdot N$ users' cells will be stored at level L and the remaining levels will be empty (w.l.o.g. we assume that $M \cdot N$ is a power of 2). Let t denote the access counter, which is made available publicly by the servers. Each level ℓ has an epoch counter $\xi(t, \ell)$ that increments after every $2^{\ell-1}$ accesses. In other words, for level ℓ and t accesses, the epoch counter is $\xi(t, \ell) = \lfloor t/2^{\ell-1} \rfloor$.

Recall that our OPRF employs two keys. S generates the first (and fixed) OPRF key $s \leftarrow_R \mathbb{Z}_n^*$, and then a series of second OPRF keys $\hat{s}[\ell, \xi(t, \ell)] \leftarrow_R \mathbb{Z}_n^*$, for each level $\ell \in [1, L]$ and the current access t. User U_i generates independently a secret PRF input $\mu_i \in \mathbb{Z}_n$ and computes a pre-tag θ for her block j using μ_i by performing OPRF with S. Similarly, the tag server T generates a secret input $\mu_{\mathcal{D}}$ for dummy blocks. To tag blocks, the construction uses a hash function family $\{h_\ell\}$ domain $[0, 2^\ell - 1]$, for each level $\ell \in [1, L]$. In particular, a tag (or bucket index) for a pre-tag θ is computed as $h_\ell(\xi(t, \ell)||\theta)$, where $||$ represents string concatenation.

Protocol Flow Similar to our constructions in Section 2.3, users have to communicate with the servers via anonymous channels. To access a cell j during the t^{th} access, user U_i first computes the associated pre-tags for all levels θ_i using OPRF with server S on her secret inputs μ_i and j. She also obtains $\theta_{\mathcal{D}}$ pre-tags from server T for all levels for the current value of access counter t. Here, T computes pre-tags for dummy blocks by interacting with S and sends those to the users, as the users cannot locally compute them. These pseudorandom pre-tag values depend on the level and the current epoch through the PRF keys used by S. Due to the oblivious nature of OPRF and secret inputs μ_i for U_i and $\mu_{\mathcal{D}}$ for T, server S does not learn the pre-tag values.

Once pre-tags are computed, the user maps each of those to a bucket index (or *tag*) in their level ℓ using h_ℓ. Now, she starts searching for her cell j from level 1 using tags computed using a pre-tag θ_i. Similarly to other hierarchical schemes, after obtaining her cell she searches for the remaining levels with tags computed using $\theta_{\mathcal{D}}$ values. The updated cell j is added back to level 1. During this process, a pre-tag θ associated with the user's cell changes to another value θ' indistinguishable from random. Fig. 2.2 shows the main sub-flow of User algorithm executed by U_i in cooperation with servers S and T. In User flow, this sub-flow is repeated once for each level. Finally, at the end of User, the user computes a new pre-tag for possibly updated cell j, and computes and stores a block with them at the first level.

Although dummy pre-tags and tags are computed by and known to T, it cannot learn the tag employed by the user while requesting blocks from S, as communication between the user and server S is encrypted. Neither can T learn this information based on the content of blocks of specific tags retrieved by

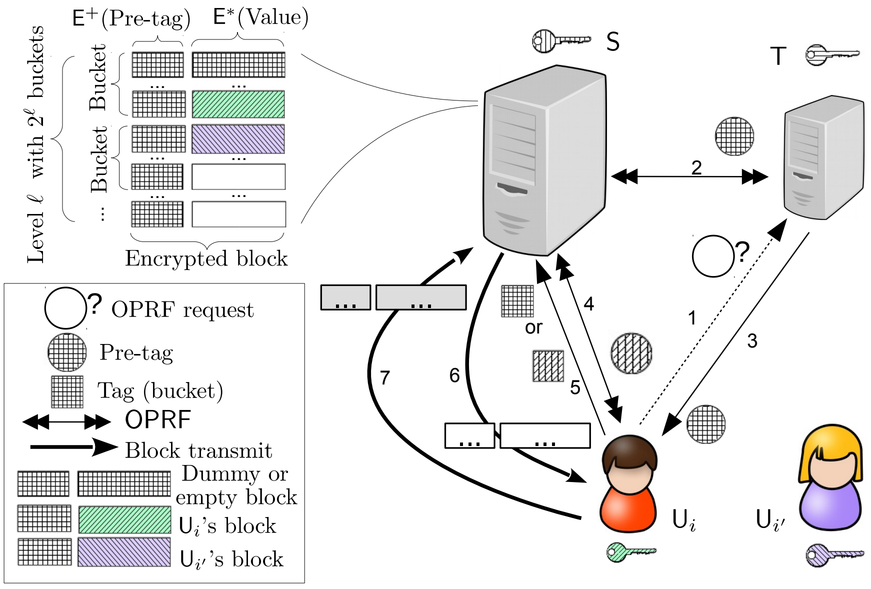

Figure 2.2: Flow of User algorithm in $\mathsf{AnonRAM}_{\mathsf{polylog}}$ for user U_i, cell j, and level ℓ: 1) U_i asks the tag server T for a dummy pre-tag. 2) T runs an OPRF protocol with the storage server S such that T learns the dummy pre-tag and S learns nothing. 3) T sends the dummy pre-tag to U_i. 4) U_i runs OPRF with S to learn a pre-tag for her cell j obliviously. 5) Depending on whether cell j is found in the previous levels or not, U_i selects one of the two pre-tags to compute a tag and sends the tag to S. 6) S re-randomizes and sends the block(s) associated with the user's tag. 7) U_i re-randomizes or updates the block(s), and possibly learns the value of cell j. If $\ell = 1$, steps 4) and 5) are skipped, and in step 6) S sends all blocks from that level.

observed users, since S *re-randomizes* blocks before sending them to users.

The main task of T is to reshuffle the blocks *without* involving users. In the Reshuffle protocol, while reshuffling levels 1 to ℓ into level $\ell+1$, server T copies, re-randomizes or changes blocks from levels 1 to ℓ, and then sorts them using oblivious-sorting (OSort) such that the users can obtain their required cells over level $\ell+1$ by procuring the appropriate pre-tag values from server S. This step requires server S helping server T to decrypt the randomized version of pre-tags in the blocks. Here, for every second access, T performs reshuffle of level 1 into level 2 on S to empty level 1. For every fourth access, all the real blocks at levels 1 and 2 will be moved to level 3, and so on.

The crucial property is that, while reshuffling, server T should not learn any information about user's data from pre-tags. To prevent T from identifying users' cells by pre-tags, S proactively shuffles all blocks that T will access during Reshuffle and updates the pre-tags associated with the blocks. Here, S utilizes homomorphic properties of OPRF: in particular, for some pre-tag $\theta = f_{s,\hat{s}}(\mu)$ for server S's OPRF keys $s, \hat{s}$, the server computes $\theta^{\delta} = f_{s,(\hat{s}\cdot\delta)}(\mu)$ for some random δ. Although pre-tags in the blocks are stored in the encrypted form and cannot be decrypted by S alone, the homomorphic properties of ElGamal allow S to apply the aforementioned trick to ciphertexts *without* knowing pre-tags in plain. Finally, S partially decrypts the pre-tags of the blocks that have to be reshuffled by T and moves these blocks to a temporary array.

After the pre-processing by server S, server T decrypts pre-tags of the blocks and reshuffles non-empty blocks to arrange them into buckets based on the pre-tags. This process is essentially the same as the Oblivious-Hash step in GO-ORAM [74] except for de-duplication of blocks [108]. Specifically, while reshuffling blocks from levels 1 to ℓ into level $\ell+1$, T first adds 2^ℓ *forward* dummy blocks that can potentially be accessed by a user in subsequent accesses. It then assigns tags to non-empty blocks using hash function $h_{\ell+1}$ and ensures that no tag gets assigned to more than β blocks. Finally, T pads the temporary array with the tagged empty blocks such that exactly β blocks have the same tag, replaces forward dummy blocks with empty ones, and moves all these blocks to level $\ell+1$ on server S. Here, T cannot link the pre-tags seen in the current Reshuffle execution to those observed during previous reshuffles, as the value δ chosen by S is unknown to T.

2.4.4 User Algorithm

In the User algorithm, a user searches in all levels for a block containing her cell j. Once the block is found, it is moved to a new location at the first level after a possible update (in case of Write operation), and a dummy block is instead added to the old location.

User algorithm for user U_i on input (j, α, m) consists of the following steps:

1. Allocate local space to hold a single encrypted block value res and initialize it to $\mathsf{E}^*_{\mathsf{TS}}(\text{“}dummy\text{”})$. Set boolean variable **found** to $false$.

2. Receive from T the pre-tag $\theta_{\mathcal{D}} := \mathsf{OPRF}^{\mathsf{T},\mathsf{S}}_{\hat{s}[1,\xi(t,1)]}(\mu_{\mathcal{D}} + t)$ computed by T after performing OPRF with S. Read all blocks at level 1. Let B denote a current block at level 1, *re-randomized* and sent by S to U_i. Parse block B into its two components (B_1, B_2), where the first part is an ElGamal ciphertext and the second part is a UREnc ciphertext. User U_i deciphers B_2 using her UREnc private key. If the block with cell index j is found, then the user sets **found** to $true$, copies B_2 to res, and replaces B with a dummy block $\langle \mathsf{E}^{+*}_{\mathsf{TS}}(\theta_{\mathcal{D}}), \mathsf{R}^*(\mathsf{E}^*_{\mathsf{TS}}(\text{“}dummy\text{”}))\rangle$. Otherwise, U_i replaces the block B with its re-randomized version.

3. For each level ℓ from 2 to L:
   (a) Compute a pre-tag $\theta_i \leftarrow \mathsf{OPRF}^{\mathsf{U}_i,\mathsf{S}}_{\hat{s}[\ell,\xi(t,\ell)]}(\mu_i + j)$ by interacting with S, and receive a pre-tag $\theta_{\mathcal{D}} \leftarrow \mathsf{OPRF}^{\mathsf{T},\mathsf{S}}_{\hat{s}[\ell,\xi(t,\ell)]}(\mu_{\mathcal{D}} + t)$ from T. If **found**, then compute tag $\tau := h_\ell(\xi(t,\ell)||\theta_{\mathcal{D}})$, else $\tau := h_\ell(\xi(t,\ell)||\theta_i)$.
   (b) Read all blocks of bucket τ at level ℓ. Let B denote a current block of bucket τ at level ℓ, *re-randomized* and sent by S to U_i. If B is U_i's real block with index j, the user sets **found** to $true$, copies the value of B to res, and replaces B with a dummy block $\langle \mathsf{E}^{+*}_{\mathsf{TS}}(\theta_{\mathcal{D}}), \mathsf{R}^*(\mathsf{E}^*_{\mathsf{TS}}(\text{“}dummy\text{”}))\rangle$. Otherwise, i.e., if B is not U_i's real block with index j, U_i replaces block B with a re-randomized

version of B.

4. If $\alpha = \mathsf{Write}$, update res to the new value $\mathsf{E}^*_{\mathsf{U}_i}(m)$.

5. Re-randomize and send $\langle \mathsf{E}^{+*}_{\mathsf{TS}}(\theta_i), res\rangle$ to S, where $\theta_i := \mathsf{OPRF}^{\mathsf{U}_i,\mathsf{S}}_{\hat{s}[1,\xi(t,\ell)]}(\mu_i + j)$. Server S writes the block to the first available empty slot at level 1.

6. If $\alpha = \mathsf{Read}$, return res.

2.4.5 Reshuffle

Reshuffle of every level ℓ into a higher level is performed every 2^ℓ accesses, when the number of non-empty blocks (real or dummy) at level ℓ reaches 2^ℓ. We reshuffle *all* blocks, from levels 1 to ℓ into level $\ell + 1$, and there are no duplicates.

Recall that after User, the number of non-empty blocks at the first level increases by 1. After two accesses, all blocks from the first level will be reshuffled into the second level. After two more accesses, all blocks from first level should be reshuffled into the second level. This event triggers the reshuffle of all blocks from the first two levels into the third level. After this point, there will be four non-empty blocks at level 3, and levels 1 and 2 will be empty.

For the current value of t, let $\ell_m := \max\{\ell > 0 \text{ s.t. } 2^\ell \text{ divides } t\}$. Then, *before* the reshuffle is performed, level 1 has two non-empty blocks, and each level $\ell \in [2, \ell_m]$ has $2^{\ell-1}$ non-empty blocks. As we show in Lemma 2.7.2 later, level $\ell_m + 1$ is empty. Hence, the $\mathsf{Reshuffle}$ procedure takes all elements from levels 1 up to ℓ_m and moves them into level $\ell_m + 1$. The total number of non-empty blocks in levels 1 to ℓ_m is $2 + \sum_{\ell=2}^{\ell_m} 2^{\ell-1} = 2^{\ell_m}$, so 2^{ℓ_m} real or dummy blocks will be added into level $\ell_m + 1$. The array is sparse; there are $(2^{\ell_m+1} - 2)\beta$ blocks at levels 1 to ℓ_m including empty ones, and among them only 2^{ℓ_m} dummy or real ones. A $\mathsf{Reshuffle}$ protocol between S and T requires two operations performed by T on data stored at S: Scan and OSort.

A generic $\mathsf{Reshuffle}$ algorithm for levels 1 up to ℓ_m into level $\ell_m + 1$ is given below; steps 1-3 are performed by S, and remaining steps 4-13 by T:

1. S allocates space for a temporary array A to hold $2^{\ell_m+2} \cdot \beta$ blocks.

2. For each level ℓ from 1 to ℓ_m:

(a) S moves all $2^\ell \cdot \beta$ blocks from level ℓ into a temporary array A', and fills the level ℓ with empty blocks. Each new empty block is just a re-randomization of the public empty block $\langle \mathsf{E}^{+*}_{\mathsf{TS}}(1), \mathsf{E}^*_{\mathsf{TS}}(\text{“}\mathit{empty}\text{”})\rangle$.

(b) Let C denote the encrypted pre-tag of some block in A', and $C = \mathsf{E}^{+*}_{\mathsf{TS}}\Big(\theta_{\hat{s}[\ell,\xi(t,\ell)-1]}\Big)$, where $\theta_{\hat{s}}$ denotes the value of PRF $f_{s,\hat{s}}(\mu)$ for some input $\mu \in \mathbb{Z}_n$. For each block in A', S replaces C with $C' \leftarrow C^{\hat{s}[\ell+1,\xi(t,\ell+1)]/\hat{s}[\ell,\xi(t,\ell)-1]}$. Thanks to the properties of f and homomorphic properties of ElGamal, $C' = \mathsf{E}^{+*}_{\mathsf{TS}}\left(\theta_{\hat{s}[\ell,\xi(t,\ell+1)]}\right)$.

(c) S moves all blocks from array A' to array A.

3. S pads A with empty blocks and partially decrypts pre-tags of all blocks. Finally, S permutes A.

4. *Decrypting pre-tags*: Decrypt pre-tags using partial decryption of S and attach (plaintext) pre-tags to the blocks using (fast) local encryption scheme.

5. *Adding forward dummies:* Add 2^{ℓ_m} dummy blocks $\langle \mathsf{E}^{+*}_{\mathsf{TS}}(\theta_0), \mathsf{E}^{*}_{\mathsf{TS}}(\text{“}\mathit{dummy}\text{”})\rangle$, $\ldots$, $\langle \mathsf{E}^{+*}_{\mathsf{TS}}(\theta_{2^{\ell_m}-1}), \mathsf{E}^{*}_{\mathsf{TS}}(\text{“}\mathit{dummy}\text{”})\rangle$, where T computes pre-tags interacting with S as $\theta_k := \mathsf{OPRF}^{\mathsf{T},\mathsf{S}}_{\hat{s}[\ell_m,\xi(t+k,\ell_m+1)]}(\ \mu_{\mathcal{D}} + (t+k))$, for $k \in [0, 2^{\ell_m} - 1]$. Array A now contains 2^{ℓ_m+1} non-empty blocks. Scan A and add temporary encryption of flags to the blocks: value 1 to the forward dummy blocks, otherwise value 0.

6. *Mapping to buckets*: Scan A and use hash function h_{ℓ_m+1} to assign tags to non-empty blocks. Since there are no duplicates, each non-empty block has (with overwhelming probability) a unique input to the hash function. Specifically, T attaches the tag $h_{\ell_m+1}(\xi(t, \ell_m + 1)||\theta)$ to a block, where θ denotes the pre-tag of that block.

7. Obliviously-sort A using the OSort protocol, according to the following criteria: (a) non-empty blocks before empty ones, (b) lower tags first.

8. *Checking if there is no bucket overflow*: Scan A and check that no single tag was given to more than β blocks. If there is such a tag, this is considered as an overflow. It can happen only with low probability; in this case, choose another hash-family $h_{\ell+1}$, and go to step 6.

9. Scan A and assign tags to $2^{\ell_m+1}\beta$ *untagged* empty blocks, one tag per β blocks. This step ensures that each tag $0, \ldots, 2^{\ell_m+1} - 1$ is represented by *at least* β blocks.

10. OSort A according to the following criteria: (a) tagged blocks before untagged, (b) lower tags first, (c) non-empty blocks before empty ones (among the blocks with the same tag).

11. Scan A and make sure that exactly β blocks have the same tag, erasing excessive blocks. Note that all excessive blocks are empty.

12. *Prepare buckets for level* $(\ell_m + 1)$: OSort A according to the following criteria: (a) tagged blocks before untagged, (b) lower tags first.

13. Scan A to replace the dummy blocks introduced in Step 5 with empty ones; these blocks have encrypted flag 1. Scan A to erase tags, temporary encryptions of pre-tags (attached to the blocks in Step 4), and flags for forward dummy blocks. Move the $2^{\ell_m+1}\beta$ prefix of A into level $\ell_m + 1$, one by one in the same order as the blocks appear in A.

The result of $\mathsf{Reshuffle}$ is $2^{\ell_m+1}\beta$ blocks is stored at level $\ell_m + 1$. The first β blocks are tagged 0, the next β blocks tagged 1, etc. This layout corresponds to putting β blocks with same tag to one bucket, since the storage at server S is organized in buckets of size β.

Reshuffle algorithm ensures, using forward dummy blocks, that buckets do not overflow in the follow-up accesses made by users until the next reshuffle. Specifically, there are no buckets accessed more than β times at some level ℓ within one epoch. Note that if instead of accessing dummy blocks the user chooses random buckets, it can lead to a small, but *not* negligible probability of distinguishing specific access patterns.

Last Reshuffle After reshuffle into level $L+1$, $M \cdot N$ real and $M \cdot N$ dummy blocks are located at that level. T and S eliminate dummy blocks by jointly decrypting the block value: if a block is dummy, the decryption succeeds. Finally, $M \cdot N$ real blocks from level $L+1$ are reshuffled into level L, thus achieving the state after the initial setup.

2.4.6 Complexity and Security Analysis

Computational and communication complexity of User is $O(\log^2(M{\cdot}N))$ since there are $L = O(\log(M{\cdot}N))$ levels, and for each level the user performs $\beta = O(\log(M{\cdot}N))$ encryptions, decryptions, and OPRF evaluations. Each of these operations requires $O(1)$ exponentiations.

Computational and communication complexity of $\mathsf{Reshuffle}$ depends on parameter t. Consider the state after Setup and the state after $M{\cdot}N$ subsequent accesses. They are identical, as all the real blocks are located at level L. Hence, it suffices to analyze the aforementioned interval. Let $\mathsf{Reshuffle}(\ell)$ denote the reshuffle from levels 1 to ℓ into level $\ell{+}1$, and let $\rho(\ell)$ denote the complexity thereof. In $\mathsf{Reshuffle}(\ell)$, the number of blocks involved is $2^{\ell+1}\beta$, hence $\rho(\ell) = O(2^{\ell+1} \cdot \beta \cdot \log(2^{\ell+1} \cdot \beta))$ due to the cost of OSort. Then, within $M \cdot N$ accesses, there is one $\mathsf{Reshuffle}(L)$, none of $\mathsf{Reshuffle}(L-1)$ (since level L initially already contains $M \cdot N$ elements), one $\mathsf{Reshuffle}(L-2)$, two $\mathsf{Reshuffle}(L-3)$, four $\mathsf{Reshuffle}(L-4)$, etc. Thus, the total complexity of all reshuffles made within $M \cdot N$ accesses is

$$
\begin{aligned}
&\Big(\sum_{\ell=1}^{L-2} 2^{L-2-\ell} \cdot \rho(\ell)\Big) + \rho(L) \\
&= \Big(\sum_{\ell=1}^{L-2} 2^{L-2-\ell} \cdot O(2^{\ell+1} \cdot \beta \cdot \log(2^{\ell+1} \cdot \beta))\Big) + O(2^{L+1} \cdot \beta \cdot \log(2^{L+1} \cdot \beta)) \\
&= \Big(2^{L-1} \cdot \beta \cdot \sum_{\ell=1}^{L-2} O(\log(2^{\ell+1} \cdot \beta))\Big) + O(M \cdot N \cdot \log(M \cdot N) \cdot \log(2^{L+1} \cdot \beta)) \\
&= O(M \cdot N) \cdot \beta \cdot O(L^2) + O(M \cdot N \cdot \log^2(M \cdot N)) = O(M \cdot N \cdot \log^3(M \cdot N)).
\end{aligned}
$$

Hence, the amortized cost of $\mathsf{Reshuffle}$ is $\tilde{O}(\log^3(M \cdot N))$.

Theorem 2.4.1. $\mathsf{AnonRAM}_{\mathsf{polylog}}$ *preserves access privacy against HbC adversaries in the random oracle model, when instantiated with semantically secure universally re-randomizable encryption (UREnc) and multiplicatively homomorphic encryption schemes, and a secure (partially key-homomorphic) oblivious PRF scheme for appropriate compatible domains.*

The proof of Theorem 2.4.1 is postponed to Section 2.7.1.

2.4.7 $\mathsf{AnonRAM}^{\mathbf{M}}_{\mathsf{polylog}}$ Secure Against Malicious Users

In $\mathsf{AnonRAM}_{\mathsf{polylog}}$ we need to constrain users to avoid any tampering by the malicious adversaries. Integrity and privacy of an honest user U_i can be achieved if a malicious user can neither change real blocks of other users, nor introduce new blocks encrypted using U_i's key. In this section, we present $\mathsf{AnonRAM}^{\mathbf{M}}_{\mathsf{polylog}}$, a modification to $\mathsf{AnonRAM}_{\mathsf{polylog}}$, and we will prove it is secure against malicious users.

Observing that blocks are written to the first level in a pre-defined manner, we say, without loss of generality, that User for tuple (i, j, α, m), parameterized with t, does the following:

1. User U_i reads (t mod 2) blocks at level 1, and β blocks at each level $\ell \in [2, L]$.
2. Among these blocks there is at least one block belonging to U_i, and exactly one block matches the target index j.
3. User U_i replaces one of the blocks that belongs to her with the dummy block.
4. The replaced in 3) real block is moved to a new pre-defined location at level 1, replacing an empty block.

It is important to enforce both 3) and 4), otherwise a malicious user could either remove the real block of an honest user or introduce a new real block of an unobserved user, thus violating integrity of honest users. We elaborate on appropriate zero-knowledge proofs for UREnc and ElGamal encryption, which enforce 3) and 4).

Recall that the ZK proof system, required in $\mathsf{AnonRAM}^{\mathbf{M}}_{\mathsf{lin}}$ for a pair of old and new ciphertexts, ensures that the new ciphertext is either a re-randomization of the old ciphertext, or the user knows the associated with the old ciphertext secret key. Re-randomization of ciphertext can be split into two parts: a) ciphertexts are encrypted under the same key, and b) encoded messages are the same. To achieve integrity in $\mathsf{AnonRAM}^{\mathbf{M}}_{\mathsf{polylog}}$, we require the former component, too, i.e. a proof that two ciphertexts are encrypted under the same key.

Summarizing, we have the following types of proof systems w.r.t. a single ciphertext $C = (C_1, \ldots, C_4)$, or relations between two ciphertexts C and C':

- the user knows the decryption key of UREnc ciphertext C; it requires a ZK proof of knowledge of discrete logarithm $PoK\{x_i | C_4 = C_3^{x_i}\}$.
- ciphertext C' is encrypted under the same key as ciphertext C; the required proof is a ZK proof of equality of the discrete logarithm of two pairs of group elements $P\{\exists r \mid (C'_3, C'_4) = (C_3^r, C_4^r)\}$.
- ciphertext C' is a re-randomization of ciphertext C; the required proof is conjunction of two ZK proofs of equality of the discrete logarithm of two pairs of group elements if C is a UREnc ciphertext, or just one such ZK proof if C is an ElGamal ciphertext (in this case, $C = (C_1, C_2)$).

We refer to Section 2.6 for the detailed description of changes in $\mathsf{AnonRAM}_{\mathsf{polylog}}$ in order to obtain $\mathsf{AnonRAM}^{\mathbf{M}}_{\mathsf{polylog}}$ secure against malicious users.

Theorem 2.4.2. $\mathsf{AnonRAM}^{\mathbf{M}}_{\mathsf{polylog}}$ *in the random oracle model, when instantiated with semantically secure universally re-randomizable encryption (UREnc) and multiplicatively homomorphic encryption schemes, a secure (partially key-homomorphic) oblivious PRF scheme for appropriate compatible domains, and augmented with ZK proof system defined above, preserves integrity and privacy in the adversarial model* $\mathsf{Mal\_Users}$.

The proof of Theorem 2.4.2 is postponed to Section 2.7.2.

2.5 (Partially Key-Homomorphic) Oblivious PRF

We next present the oblivious PRF (OPRF) construction, based on [87], and mentioned in Section 2.4.1.

Definition 2.5.1 (Oblivious PRF—OPRF [64]). *A 2-party protocol π is an OPRF scheme if there exists some PRF family F_s such that π privately realizes the following functionality:* **Input:** *Client holds an evaluation point μ; Server S holds a key s.* **Output:** *Client outputs $F_s(\mu)$; Server outputs nothing.*

This can be denoted as a secure computation protocol for functionality $\mathcal{F}_{\mathsf{OPRF}} : (s, \mu) \to (\perp, F_s(\mu))$.

Our construction is based on the Jarecki-Liu OPRF [87] and presumes a malicious adversary, but a simpler version without zero-knowledge proof systems is suitable against an HbC adversary.

Similar to the original OPRF, we need an encryption scheme to satisfy additive homomorphism, verifiable encryption, and verifiable decryption properties as defined in [87, Sec. 2.2]. The encryption scheme satisfying these properties can be instantiated with a semantically-secure variant [87] of the Camenisch-Shoup encryption (CSEnc) [28] scheme, accompanied with suitable zero-knowledge proof systems, as specified in [87]. Here, we denote K^+, E^+, and D^+ the key generation, the encryption, and the decryption algorithms of CSEnc, respectively.

CSEnc is an additively homomorphic encryption scheme such that $\mathsf{E}^+_{pk}(m) \cdot \mathsf{E}^+_{pk}(m') = \mathsf{E}^+_{pk}(m + m')$ for any $m, m' \in \mathbb{Z}_n$, and $\mathsf{E}^+_{pk}(m)^\delta = \mathsf{E}^+_{pk}(m \cdot \delta)$ for any $\delta \in \mathbb{Z}^*_n$, where pk denotes a CSEnc public key. Moreover, CSEnc allows for shared decryption. Denoting $C_m = \mathsf{E}^+_{pk_{\mathsf{SU}}}(m)$, $C^{(\mathsf{S})}_m = \mathsf{E}^+_{pk_{\mathsf{S}}}(m)$, and $C^{(\mathsf{U})}_m = \mathsf{E}^+_{pk_{\mathsf{U}}}(m)$, decryption of C_m using the U's private key gives $C^{(\mathsf{S})}_m$, and decryption of C_m using the S's private key gives $C^{(\mathsf{U})}_m$.

Theorem 2.5.1. *Assuming hardness of factoring of safe RSA moduli, a semantically secure encryption scheme on $\mathbb{Z}_n$ which satisfies properties listed above and assuming that each proof (of knowledge) system in Fig. 2.3 is zero-knowledge and simulation-sound, the protocol in Fig. 2.3 is a secure computation protocol for functionality $\mathcal{F}_{\mathsf{OPRF}}$.*

Proof. Our proof is similar to the proof of the original OPRF [87].
Constructing an ideal-world server SIM_S *from a malicious real server* S^**:* SIM_S interacts with S^* and $\mathcal{F}_\mathsf{OPRF}$, and does the following:

– If S^* succeeds in the proof π_1, then SIM_S runs the extraction algorithm for π_1 with S^* to extract $(s, \hat{s})$, s.t. $\mathsf{pk}_1 = \mathsf{g}^s$, $\mathsf{pk}_2 = \mathsf{g}^{\hat{s}}$.

– SIM_S simulates the real-world user U as follows:

  1. $(pk_\mathsf{U}, sk_\mathsf{U}) \leftarrow \mathsf{K}^+$.
  2. $r \leftarrow_R \mathbb{Z}_n^*$, $C_r^{(\mathsf{U})} \leftarrow \mathsf{E}^+_{pk_\mathsf{U}}(r)$.
  3. $a \leftarrow_R \mathbb{Z}_n^*$, $C_a^{(\mathsf{S})} \leftarrow \mathsf{E}^+_{pk_\mathsf{S}}(a)$.
  4. $C_a^{(\mathsf{S})} \leftarrow \mathsf{E}^+_{pk_\mathsf{S}}(a)$.
  5. Send $(pk_\mathsf{U}, C_r^{(\mathsf{U})}, C_a^{(\mathsf{S})})$ and simulate the proof π_2.

– If the proof π_3 verifies, SIM_S sends s, $\hat{s}$ to $\mathcal{F}_\mathsf{OPRF}$. On SIM_S's inputs $(s, \hat{s})$ and ideal world user $\bar{\mathsf{U}}$'s input μ, $\mathcal{F}_\mathsf{OPRF}$ outputs $f_{s,\hat{s}}(\mu)$ to $\bar{\mathsf{U}}$.

Let $\mathcal{D}$ be a distinguisher that controls the server S^*, chooses the input of the user U, and also observes the output of U. We argue that $\mathcal{D}$'s view in the real

Common input: $(n, \mathsf{g}, \mathsf{pk}_1, \mathsf{pk}_2)$. S's private input: s, $\hat{s}$, s.t. $\mathsf{g}^s = \mathsf{pk}_1$, $\mathsf{g}^{\hat{s}} = \mathsf{pk}_2$. U's private input: μ.
Step 1 (S). $(pk_\mathsf{S}, sk_\mathsf{S}) \leftarrow \mathsf{K}^+$, $C_s^{(\mathsf{S})} \leftarrow \mathsf{E}^+_{pk_\mathsf{S}}(s)$, $\pi_1 \leftarrow PoK\{s, \hat{s} \| C_s^{(\mathsf{S})} \in \mathsf{E}^+_{pk_\mathsf{S}}(s),\ \mathsf{pk}_1 = \mathsf{g}^s,\ \mathsf{pk}_2 = \mathsf{g}^{\hat{s}}\}$. Send $\left(pk_\mathsf{S}, C_s^{(\mathsf{S})}\right), \pi_1$ to U.
Step 2 (U). If π_1 verifies, then $(pk_\mathsf{U}, sk_\mathsf{U}) \leftarrow \mathsf{K}^+$, $r \leftarrow_R \mathbb{Z}_n^*$, $C_r^{(\mathsf{U})} \leftarrow \mathsf{E}^+_{pk_\mathsf{U}}(r)$, $C_a^{(\mathsf{S})} \leftarrow \left(C_s^{(\mathsf{S})} \cdot \mathsf{E}^+_{pk_\mathsf{S}}(\mu)\right)^r$, $\pi_2 \leftarrow PoK\left\{\mu \| \exists r, \text{s.t. } C_r^{(\mathsf{U})} \in \mathsf{E}^+_{pk_\mathsf{U}}(r), C_a^{(\mathsf{S})} \in \left(C_s^{(\mathsf{S})} \cdot \mathsf{E}^+_{pk_\mathsf{S}}(\mu)\right)^r\right\}$. Send $\left(pk_\mathsf{U}, C_r^{(\mathsf{U})}, C_a^{(\mathsf{S})}, \pi_2\right)$ to S.
Step 3 (S). If π_2 verifies, then $a \leftarrow \mathsf{D}^+_{sk_\mathsf{S}}(C_a^{(\mathsf{S})})$. If $gcd(n, a) \neq 1$, send $\perp$ to U and abort. $b \leftarrow \hat{s} \cdot (a)^{-1} \bmod n$, $\varphi_\mathsf{S} \leftarrow_R \mathbb{Z}_n$, $v_\mathsf{S} \leftarrow \mathsf{g}^{\varphi_\mathsf{S}}$, $C_{\varphi_\mathsf{U}}^{(\mathsf{U})} \leftarrow \left(C_r^{(\mathsf{U})}\right)^b \cdot \mathsf{E}^+_{pk_\mathsf{U}}(-\varphi_\mathsf{S})$, $\pi_3 \leftarrow P\left\{\begin{array}{c} \exists a, \varphi_\mathsf{S}, b, sk_\mathsf{S} \text{ s.t. } a = \mathsf{D}^+_{sk_\mathsf{S}}\left(C_a^{(\mathsf{S})}\right), (pk_\mathsf{S}, sk_\mathsf{S}) \in \mathsf{KeyVal} \\ C_{\varphi_\mathsf{U}}^{(\mathsf{U})} \in \left(C_r^{(\mathsf{U})}\right)^b \cdot \mathsf{E}^+_{pk_\mathsf{U}}(-\varphi_\mathsf{S}),\ v_\mathsf{S} = \mathsf{g}^{\varphi_\mathsf{S}}, a \cdot b = \hat{s} \bmod n,\ \mathsf{pk}_2 = \mathsf{g}^{\hat{s}} \end{array}\right\}$. Send $\left(v_\mathsf{S}, C_{\varphi_\mathsf{U}}^{(\mathsf{U})}, \pi_3\right)$ to U.
Step 4 (U). Output $\perp$ if receiving $\perp$ from S, or if π_3 fails. $\varphi_\mathsf{U} \leftarrow \mathsf{D}^+_{sk_\mathsf{U}}\left(C_{\varphi_\mathsf{U}}^{(\mathsf{U})}\right)$, $v_\mathsf{U} \leftarrow \mathsf{g}^{\varphi_\mathsf{U}}$. Output $v_\mathsf{S} \cdot v_\mathsf{U}$.

Figure 2.3: Our (partially key-homomorphic) OPRF construction.

world (S^*'s view + U's output) and its view in the ideal world (S^*'s view + ideal user $\bar{\mathsf{U}}$'s output) are indistinguishable. To show this, we introduce a series of games $\mathsf{G}_0, \ldots, \mathsf{G}_6$ where G_0 is the real world experiment (S^* interacting with the real user U), G_6 is the ideal world experiment (S^* interacting with $\mathsf{SIM_S}$), and arguing that the views in G_i and G_{i+1} are indistinguishable.

G_1: Same as G_0 except that instead of proving π_2, U simulates it. By zero-knowledge of the π_2, $\mathcal{D}$'s views in G_0 and G_1 are indistinguishable.

G_2: Same as G_1 except that if S^* succeeds in the proof π_1, G_2 runs the extractor algorithm for π_1 with S^* to extract s, $\hat{s}$. By simulation soundness of π_1, G_2 extracts s, $\hat{s}$ with non-negligible probability.

G_3: Same as G_2 except that if the proof π_3 verifies, then G_3 outputs $f_{s,\hat{s}}(\mu) = \mathsf{g}^{\hat{s}/(s+\mu)}$ as the final output (or $\perp$ if $gcd(s+\mu, n) \neq 1$). By simulation soundness of π_3, $\mathcal{D}$'s views in G_2 and G_3 are indistinguishable.

G_4: Same as G_3 except that as long as $gcd(s+\mu, n) = 1$, G_4 does the following: 1. $(pk_\mathsf{U}, sk_\mathsf{U}) \leftarrow \mathsf{K}^+$. 2. $a \leftarrow_R \mathbb{Z}_n^*$, $C_a^{(\mathsf{S})} \leftarrow \mathsf{E}^+_{pk_\mathsf{S}}(a)$. 3. $r \leftarrow a \cdot \hat{s}/(s+\mu)$, $C_r^{(\mathsf{U})} \leftarrow \mathsf{E}^+_{pk_\mathsf{U}}(r)$. 4. Simulate the proof π_2. The probability that $gcd(s+\mu, n) \neq 1$ is negligible assuming factoring safe RSA moduli is hard. If $gcd(s+\mu, n) = 1$, then the tuple $(pk_\mathsf{U}, C_r^{(\mathsf{U})}, C_a^{(\mathsf{S})})$ is distributed identically in G_3 and G_4, and so $\mathcal{D}$'s views in these games are indistinguishable.

G_5: Same as G_4 except that value r is replaced by random $r' \in \mathbb{Z}_n^*$. By semantic security of the encryption scheme, $\mathcal{D}$'s views in G_4 and G_5 are indistinguishable. (See G_4 in the first part of the proof of Theorem 1 in [87] for a reduction.)

G_6: G_6 is the ideal world game between $\mathsf{SIM_S}$ (with access to S^*), $\mathcal{F}_\mathsf{OPRF}$, and the ideal world user $\bar{\mathsf{U}}$. Instead of computing Step 4. in G_5, the simulator $\mathsf{SIM_S}$ sends $(s, \hat{s})$ to $\mathcal{F}_\mathsf{OPRF}$ in G_6. On inputs $(s, \hat{s})$ from $\mathsf{SIM_S}$ and μ from $\bar{\mathsf{U}}$, $\mathcal{F}_\mathsf{OPRF}$ computes and sends to $\bar{\mathsf{U}}$ the value $f_{s,\hat{s}}(\mu)$. We have that $\mathcal{D}$'s views in G_5 and G_6 are indistinguishable.

Constructing an ideal-world user $\mathsf{SIM_U}$ *from a malicious real-world user* U^**:* $\mathsf{SIM_U}$ interacts with U^* and $\mathcal{F}_\mathsf{OPRF}$ and does the following:

– $\mathsf{SIM_U}$ picks $(pk_\mathsf{S}, sk_\mathsf{S}) \leftarrow_R \mathsf{K}^+$, $s' \leftarrow_R \mathbb{Z}_n^*$, computes $C_s^\mathsf{S} \leftarrow \mathsf{E}^+_{pk_\mathsf{S}}(s')$, sends pk_S and $C_s^{(\mathsf{S})}$ to U^*, and simulates the proof π_1.

– If the proof π_2 verifies, $\mathsf{SIM_U}$ runs the extractor algorithm of π_2 with U^* to extract μ and sends it to $\mathcal{F}_\mathsf{OPRF}$.

– Getting $v = f_{s,\hat{s}}(\mu)$ from $\mathcal{F}_\mathsf{OPRF}$, which computes it on ideal-world server $\bar{\mathsf{S}}$'s inputs $(s, \hat{s})$ and $\mathsf{SIM_U}$'s input μ, $\mathsf{SIM_U}$ does the following:

1. If $f_{s,\hat{s}}(\mu) = 1$, then $\mathsf{SIM_U}$ sends $\perp$ to U^* and aborts.
2. $\varphi_\mathsf{U} \leftarrow_R \mathbb{Z}_n$.
3. $v_\mathsf{S} \leftarrow v/\mathsf{g}^{\varphi_\mathsf{U}}$.
4. $C_{\varphi_r}^{(\mathsf{U})} \leftarrow \mathsf{E}^+_{pk_\mathsf{U}}(\varphi_\mathsf{U})$.
5. Send $(v_\mathsf{S}, C_{\varphi_r}^{(\mathsf{U})})$ and simulate the proof π_3.

Let $\mathcal{D}$ be a distinguisher that controls the user U^*, chooses the input of the server S, and also observes the output of S. We show a series of games $\mathsf{G}_0, \dots, \mathsf{G}_4$, where G_0 is the real world experiment, G_4 is the ideal world experiment, and argue that the views in G_i and G_{i+1} are indistinguishable.

G_1: same as G_0 except that S simulates the proofs π_1 and π_3. By zero-knowledge of these proof systems, $\mathcal{D}$'s views in G_0 and G_1 are indistinguishable.

G_2: same as G_1 except that if the proof π_2 verifies, G_2 runs the extractor algorithm for π_2 with U^* to extract μ. By simulation soundness of π_2, G_2 extracts μ with non-negligible probability.

G_3: same as G_2 except that it does the following after extracting μ: 1. $v = f_{s,\hat{s}}(\mu)$; if $v = 1$, send $\perp$ to U^* and abort. 2. $\varphi_{\mathsf{U}} \leftarrow_R \mathbb{Z}_n$, $v_{\mathsf{S}} \leftarrow v/\mathsf{g}^{\varphi_r}$. 3. $C^{(\mathsf{U})}_{\varphi_{\mathsf{U}}} \leftarrow \mathsf{E}^+_{pk_{\mathsf{U}}}(\varphi_{\mathsf{U}})$. 4. Send $\left(v_{\mathsf{S}}, C^{(\mathsf{U})}_{\varphi_{\mathsf{U}}}\right)$ and simulate the proof π_3.

For the same arguments made in G_3 in the second part of the proof of Theorem 1 in [87], $\mathcal{D}$'s views in G_2 and G_3 are indistinguishable.

G_4: same as G_3 except that when v is to be computed, $\mathsf{SIM_U}$ sends the extracted μ to $\mathcal{F}_{\mathsf{OPRF}}$ (which also gets inputs $(s, \hat{s})$ from the ideal world server $\bar{\mathsf{S}}$) and gets the value $v = f_{s,\hat{s}}(\mu)$ from the ideal functionality instead. We have that the views in G_3 and G_4 are indistinguishable. □

2.6 Detailed Description of AnonRAM$^{\mathbf{M}}_{\mathsf{polylog}}$

In the following, we present the required changes to make $\mathsf{AnonRAM_{polylog}}$ secure against malicious users. The resulting construction is called $\mathsf{AnonRAM}^{\mathbf{M}}_{\mathsf{polylog}}$.

Changes in User We elaborate on a ZK proof system, in which a user proves specific relations between the old (stored at S at the moment before the user started User) and the new ciphertexts (sent from the user to S during User).

We collect all the blocks read and written by U_i into the array of $w := (t \bmod 2) + (L-1)\beta$ blocks, $B_1, \dots, B_w$, the modified array of w blocks, $\hat{B}_1, \dots, \hat{B}_w$, and an extra new block $\hat{B}_0$. Each block B_k is the pair (γ_k, v_k), where $\gamma_k := (\gamma_k[1], \gamma_k[2])$ denotes encryption of the pre-tag, and $v_k := (v_k[1], v_k[2], v_k[3], v_k[4])$ encryption of the value. During User, the tag server T computes L pre-tags for the dummy blocks for each level, and the user computes a pre-tag for the new block (in the proof, we are interested only in aforementioned $L{+}1$ pre-tags). These pre-tags are sent (computed) to (by) the user in Steps 2 and 3a (Step 5) of User. In the modified version of User, the encryptions of dummy pre-tags are also sent to S, so that S could verify ZK proofs sent by the user to S at the end of User. Let $\gamma_{\mathcal{D}_\ell}$ denote the encrypted pre-tag for the dummy block computed for level $\ell \in [1, L]$, and γ_0 denote the dummy pre-tag computed in Step 5 of User. Let $v_{\mathcal{D}} := \mathsf{E}^*_{\mathsf{TS}}(\text{“}\mathit{dummy}\text{”})$ denote the public ciphertext initialized in Setup.

The proof consists of $w + 1$ individual parts. For each $k \in \{1, \dots, w\}$, the

$$P\begin{Bmatrix}\exists\ r_1, r_2, r_3 \ \text{ s.t. } \hat{\gamma}_k = \Big(\gamma_k[1]\cdot \mathsf{g}^{r_1}, \gamma_k[2]\cdot(\mathsf{pk}_{\mathsf{TS}})^{r_1}\Big), & 1\\ \hat{v}_k = \Big(v_k[1]\cdot(v_k[3])^{r_2}, v_k[2]\cdot(v_k[4])^{r_2}, (v_k[3])^{r_3}, (v_k[4])^{r_3}\Big) & 2\end{Bmatrix}$$

$$\vee$$

$$PoK\begin{Bmatrix} x|\ \exists\ r_4, r_5, r_6, r_7 \ \text{ s.t. } v_k[4] = (v_k[3])^x, & 3\\ \hat{\gamma}_k = (\gamma_{\mathcal{D}_\ell}[1]\cdot \mathsf{g}^{r_4}, \gamma_{\mathcal{D}_\ell}[2]\cdot(\mathsf{pk}_{\mathsf{TS}})^{r_4}), & 4\\ \hat{v}_k = \Big(v_{\mathcal{D}}[1]\cdot(v_{\mathcal{D}}[3])^{r_5}, v_{\mathcal{D}}[2]\cdot(v_{\mathcal{D}}[4])^{r_5}, (v_{\mathcal{D}}[3])^{r_6}, (v_{\mathcal{D}}[4])^{r_6}\Big), & 5\\ (\hat{v}_0[3], \hat{v}_0[4]) = \Big((v_k[3])^{r_7}, (v_k[4])^{r_7}\Big) & 6\end{Bmatrix}$$

Figure 2.4: Part of zero-knowledge proof system for $\mathsf{AnonRAM}^{\mathbf{M}}_{\mathsf{polylog}}$ relationship between blocks B_k and $\hat{B}_k$ at level ℓ, represented as ciphertexts (γ_k, v_k) and $(\hat{\gamma}_k, \hat{v}_k)$ respectively, and a new block $\hat{B}_0$, represented as $(\hat{\gamma}_0, \hat{v}_0)$. Encryption of pre-tag for the dummy block associated with $\hat{B}_k$ is denoted as $\gamma_{\mathcal{D}_\ell}$. Encryption of the constant dummy value is denoted as $v_{\mathcal{D}}$.

individual k-th part looks as described in Figure 2.4. The ZK proof for the individual part consists of six components. The first two components in the proof correspond to re-randomization of the block B_k:

– component 1 states that $\hat{B}_k$ encrypts the same pre-tag as B_k;

– component 2 states that $\hat{B}_k$ and B_k encrypt the same block value under the same key.

The next four components correspond to the case when a real block is found by the user. The user then replaces that block with the dummy block and stores an updated value of the real block as $\hat{B}_0$. Specifically:

– component 3 states that the user knows the secret key, using which she can decrypt the value part of B_k;

– component 4 states that $\hat{B}_k$ encrypts the same pre-tag as $\gamma_{\mathcal{D}_\ell}$;

– component 5 states that the block value of $\hat{B}_k$ is a re-randomization of $v_{\mathcal{D}}$;

– component 6 states that the value of $\hat{B}_0$ is encrypted under the same key as the user's ciphertext v_k.

Note that we do not restrict the value of $\hat{B}_0$, since the user may want to update the value of her cell to some other value, different from the value of B_k.

The last, $(w+1)$-th, part of ZK proof system is

$$P\Big\{\bigvee_{k=0}^{w} \exists r_k \text{ s.t. } (\hat{v}_k[3], \hat{v}_k[4]) = ((v_{\mathcal{D}}[3])^{r_k}, (v_{\mathcal{D}}[4])^{r_k})\Big\}.$$

It states that the value in at least one of the new blocks is encrypted under the same key as the dummy value $v_{\mathcal{D}}$. This part of ZK proof system tolerates a malicious user who just re-randomizes all blocks she has read (and thus proving only components 1-2 in Figure 2.4) and introduces a new block of an honest user.

In this case, the newly introduced block $\hat{B}_0$ has to be encrypted using the shared key $\mathsf{pk}_{\mathsf{TS}}$.

The user sends $w+1$ proofs to the server S at the end of User command. The server rejects if at least one of the proofs does not verify.

Changes in Reshuffle Reshuffle has to be modified as follows: if T observes two or more blocks with same pre-tag, it replaces them with empty blocks. To this end, T performs an additional preparation step: OSort array A by pre-tags in the ascending order, then Scan A and replace any consecutive blocks having the same pre-tag with empty blocks. This modification to Reshuffle circumvents a malicious scenario in which too many blocks have the same pre-tag (such blocks would be mapped to the same tag and cause a bucket overflow with high probability).

2.7 Postponed Proofs

2.7.1 Proof of $\mathsf{AnonRAM}_{\mathsf{polylog}}$

Theorem 2.7.1. $\mathsf{AnonRAM}_{\mathsf{polylog}}$ *provides access privacy against HbC adversaries controlling* S^* *in the random oracle model, when instantiated with a semantically secure universally re-randomizable encryption (UREnc) scheme, and a semantically secure multiplicatively homomorphic encryption and a secure (partially key-homomorphic) oblivious PRF schemes for appropriate compatible domains.*

Before proving Theorem 2.7.1, we state several facts and lemmas.

Fact 2.7.1. *During* User*, a user* U_i *modifies at most one block at level* $\ell > 1$*, and this block is a real block belonging to* U_i *and it is replaced with a dummy block by* U_i*.*

Based on Fact 2.7.1, after User is executed, the total number of dummy and real blocks at level $\ell > 1$ remains unchanged and these blocks are located at the same locations.

Fact 2.7.2. *After* User *is executed, the number of dummy or real blocks is increased by one at level 1.*

Lemma 2.7.1. *The number of non-empty blocks at any level* ℓ *is determined only by the current value* t*, and not by access pattern.*

Proof. It follows from Facts 2.7.1 and 2.7.2, and the invariant of Reshuffle from levels 1 to ℓ into $\ell+1$ for some ℓ: all non-empty blocks from levels 1 to ℓ are moved to level $\ell+1$, and Reshuffle does not introduce any new dummy or real blocks. □

Lemma 2.7.2. *Before reshuffle levels* 1 *to* ℓ *into level* $\ell+1$*, level* $\ell+1$ *is empty.*

Proof. Analogously to the proof of Lemma 1 in [96]. □

Proof of Theorem 2.7.1. We argue that locations of U_i's real blocks and dummy locations used by U_i are not known to S^* (and therefore cannot be distinguished). First, the adversary cannot decrypt any part of block except for the block value of the blocks belonging to $\mathcal{U}^*$, so S^* merely observes the access locations used by U_i in User and can detect whether the real blocks of corrupted used has been changed by U_i. Since U_i does not change the blocks of other users while executing User (Fact 2.7.1), the only thing that could help the adversary is the locations accessed by U_i during User. And second, after $\mathsf{Reshuffle}$, the server S^* does not know the dummy locations, and the adversary learns dummy locations used by the compromised users. Dummy locations used by U_i in User, on the other hand, are known only to U_i itself and the tag server T, since these locations are computed by T and sent to U_i in User, and they depend on private input $(\mu_{\mathcal{D}} + t)$ known only to T.

We define a series of games $\mathsf{G}_1, \ldots, \mathsf{G}_{10}$, where G_1 denotes an execution $\mathsf{Exec}(\mathcal{AR}, \mathsf{Adv}, AP_0, \mathsf{HbC})$, and G_{10} an execution $\mathsf{Exec}(\mathcal{AR}, \mathsf{Adv}, AP_1, \mathsf{HbC})$ for any two HbC-compliant access patterns AP_0, AP_1, and show that the views of the adversary in these games are indistinguishable.

We briefly describe the games. G_1 is the real experiment with AP_0. In G_2, an uncorrupted user U_i does not overwrite a found real block with the dummy block, but just re-randomizes the real block and writes to the first level the dummy block with the pre-tag computed for that level. In other words, the user re-randomizes *all* blocks read in Steps 2-4 of User. In G_3, the user always sets τ to the value corresponding to the dummy pre-tag in Step 3b of User. In G_4, the user sends random inputs to OPRF evaluation in Step 3a. In G_5, there is unique user $\mathsf{U}' \notin \mathcal{U}^*$ who performs accesses instead of any other user $\mathsf{U}_i \notin \mathcal{U}^*$. The remaining games $\mathsf{G}_6, \ldots, \mathsf{G}_{10}$ form a counter part for AP_1 (in the reversed order) such that $\mathsf{G}_5 = \mathsf{G}_6$. Below, we present reductions from G_1 to G_{10}.

G_2 - same as G_1 except for the changes in User presented below. We define series of games $\mathsf{G}_1^0, \ldots, \mathsf{G}_1^{|AP|}$ with $\mathsf{G}_1 := \mathsf{G}_1^{|AP|}$ and $\mathsf{G}_2 := \mathsf{G}_1^0$, where the games $\mathsf{G}_1^{\mathsf{i}+1}$ and $\mathsf{G}_1^{\mathsf{i}}$ differ in the following: if the $(\mathsf{i}+1)$-th access is made by some uncorrupted U_i, do not change the location and content of the real block in $\mathsf{G}_1^{\mathsf{i}}$, and instead add a dummy block to the first level. We can show that the views of the adversary in these games are indistinguishable by a reduction to DDH. For that we need to introduce additional intermediate games as tools that look as follows:

a. Change the encrypted value $(g^a, m \cdot g^{x_i a}, g^b, g^{x_i b})$ in a game G to some random value as $(g^a, \tilde{m} \cdot g^{x_i a}, g^b, g^{x_i b})$ for some random $\tilde{m} \in G_q$ in a game G'. We show that the views of the adversary in the games G and G' are indistinguishable by reduction to DDH: given a DDH tuple $(X, Y, Z) = (g^{\mathsf{x}}, g^{\mathsf{y}}, g^{\mathsf{z}} \textit{ or } g^{\mathsf{xy}})$, we construct a distinguisher $\mathcal{D}$ against DDH as follows. $\mathcal{D}$ runs $\mathsf{Setup}(1^\lambda)$ of $\mathsf{AnonRAM}_{\mathsf{polylog}}$ where instead of generating the U_i's secret key x_i for UREnc, it sets $g^{x_i} = X$, and simulates the uncorrupted user U_i without x_i based on the second part of

UREnc ciphertext. Then $\mathcal{D}$ replaces a challenge ciphertext with $(Y, Z \cdot m, g^b, X^b)$. If (X, Y, Z) is a true DH tuple, then the game proceeds as G, otherwise, i.e. if (X, Y, Z) is a random tuple, the game proceeds as in G'.

b. Change the ownership of a ciphertext $(g^a, m \cdot g^{xa}, g^b, g^{xb})$ under the key $x \in \mathbb{Z}_q$ in game G to some random key $\tilde{x} \in \mathbb{Z}_q$ and a random message $\tilde{m} \in G_q$ as $(g^a, \tilde{m} \cdot g^{\tilde{x}a}, g^b, g^{\tilde{x}b})$ in a game G'. We show that the views of the adversary in these two games are indistinguishable by a reduction to DDH: given a DDH tuple $(X, Y, Z) = (g^{\mathsf{x}}, g^{\mathsf{y}}, g^{\mathsf{z}} \textit{ or } g^{\mathsf{xy}})$, we construct a distinguisher $\mathcal{D}$ against DDH as follows. $\mathcal{D}$ runs $\mathsf{Setup}(1^\lambda)$ of $\mathsf{AnonRAM}_{\mathsf{polylog}}$ and instead of generating U_i's secret key x_i for UREnc, it sets $X = g^{x_i}$ and simulates the uncorrupted user U_i without x_i. Then, $\mathcal{D}$ replaces the challenge ciphertext with $(g^a, X^a \cdot m, Y, Z)$. If (X, Y, Z) is a true DH tuple, then the game proceeds as G, otherwise, i.e. if (X, Y, Z) is a random tuple, the game proceeds as in G'.

These tools are applied to the block value, since they are encrypted using UREnc. We just mention that we also need the first tool (for changing the encrypted value) for the pre-tag part, which is encrypted using the semantically secure encryption scheme on $\mathbb{Z}_n$. Having all these tools in place, we define intermediate games between $\mathsf{G}_1^{\mathbf{i}+1}$ and $\mathsf{G}_1^{\mathbf{i}}$, so that the views in these games are indistinguishable for the adversary.

G_3 - same as G_2, except that pre-tags in OPRF evaluations performed by U_i are replaced with dummy pre-tags computed and sent by T to U_i. Specifically, we define series of games $\mathsf{G}_2^0, \dots, \mathsf{G}_2^{|AP_0| \cdot L}$ with $\mathsf{G}_2 := \mathsf{G}_2^{|AP_0| \cdot L}$ and $\mathsf{G}_3 := \mathsf{G}_2^0$, where games $\mathsf{G}_2^{\mathbf{i}+1}$ and $\mathsf{G}_2^{\mathbf{i}}$ differ in the following: if the $(\mathbf{i}+1)$-th OPRF evaluation is performed by U_i, set τ corresponding to the dummy pre-tag computed in Step 3a, in $\mathsf{G}_2^{\mathbf{i}}$.

Let E be the event that there exist two OPRF evaluations with at least one of them belonging to an honest user U_i, such that the OPRF is evaluated using the same input and the same key. We show that the probability of this event is negligible. In $\mathsf{G}_2^{\mathbf{i}+1}$, pre-tag is evaluated via OPRF using a storage server's key $(s, \hat{s})$ and some input $(\mu_{\mathcal{D}} + t)$ or $(\mu_i + j)$. In the former case, t is incremented every new access, therefore the OPRF evaluation will be pseudo-random (the special property of OPRF w.r.t. the second key applies only if the inputs to OPRF are the same), and so will be the tag computed via h_ℓ. In the latter case, i.e. if OPRF is evaluated using input $(\mu_i + j)$, the inputs to OPRF at specific level ℓ can be the same, however the keys used by S^* will be different with overwhelming probability. The reason lies in the mechanics of User and $\mathsf{Reshuffle}$. Fix level $\ell > 1$. Assume $(\mu_i + j)$ was used as input to OPRF at level ℓ for some t. This only could happen if the real block was located at the level ℓ or at one of the next levels. Regardless of that level, once read, the real block will be moved by U_i to the first level. In any subsequent accesses, U_i will send $(\mu_i + j)$ as input to OPRF for level ℓ only if the real block is located at level ℓ or one of the next levels; denote the time (access counter) when this event happened as $t' > t$. The real block can be moved from lower levels to one of the next levels only by performing Reshuffle. In particular, there should have been Reshuffle of all levels from 1 to $\ell - 1$ into level ℓ at time $t^* < t'$ and $t^* \geq t$. But when level ℓ is involved into Reshuffle, the storage server S^* generates a fresh second key for OPRF. The probability to draw some $\hat{s}'$ that already has been used as the second key to OPRF in the past is negligible, so is the probability of E. The specific property of OPRF is "neglected" since the adversary observes the tags, i.e. the output of h, which is modelled as a random oracle. Hence, the views of the adversary in games $\mathsf{G}_2^{\mathbf{i}+1}$ and $\mathsf{G}_2^{\mathbf{i}}$ are indistinguishable.

G_4 - same as G_3, except that uncorrupted users use random inputs to OPRF. Specifically, we define intermediate games $\mathsf{G}_3^0, \ldots, \mathsf{G}_3^{|AP_0|\cdot L}$ with $\mathsf{G}_3 := \mathsf{G}_3^{|AP_0|\cdot L}$ and $\mathsf{G}_4 := \mathsf{G}_3^0$, where $\mathsf{G}_3^{\mathbf{i}+1}$ and $\mathsf{G}_3^{\mathbf{i}}$ differ in the following: if the $(\mathbf{i}+1)$-th OPRF evaluation is performed by U_i, the input to OPRF is replaced by a random value. The output of OPRF is not used subsequently by U_i in $\mathsf{G}_3^{\mathbf{i}+1}$. If Adv can distinguish between these two games, it can break the underlying assumptions of OPRF.

G_5 - same as G_4, except that all accesses performed by uncorrupted users are replaced with "equivalent" accesses performed by some fixed uncorrupted user U'. Note that uncorrupted users in G_4 do not use their secret inputs in OPRF evaluation. The uncorrupted users re-randomize the blocks they have read, and introduce the dummy block to the first level. The views of the adversary in these two games are indistinguishable provided that communication channels between users and servers are anonymous.

The remaining games, $\mathsf{G}_6, \ldots, \mathsf{G}_{10}$ are defined analogously to $\mathsf{G}_1, \ldots, \mathsf{G}_5$, in reversed order, so that G_{10} corresponds to an experiment with AP_1, and whenever AP_0 is mentioned in $\mathsf{G}_1 - \mathsf{G}_5$, it should be replaced with AP_1 in $\mathsf{G}_{10} - \mathsf{G}_6$. We have $\mathsf{G}_5 = \mathsf{G}_6$. Hence the views in games G_1 and G_{10} are indistinguishable.

Theorem 2.7.2. $\mathsf{AnonRAM}_{\mathsf{polylog}}$ *provides access privacy against HbC adversaries controlling* T^* *in the random oracle model, when instantiated with a semantically secure universally re-randomizable encryption (UREnc) scheme, and a semantically secure additively homomorphic encryption and a secure (partially key-homomorphic) oblivious PRF schemes for appropriate compatible domains.*

Proof. There are several arguments for the proof. First, while accessing S, a corrupted user from $\mathcal{U}^*$ reads and writes some blocks. Since S re-randomizes the block before sending it to users, the adversary cannot detect whether a particular bucket was touched or not, by an uncorrupted user. So, Adv has no information about tags used by an uncorrupted user, even though T^* knows pre-tags for dummy blocks in User.

Second, in Reshuffle, T^* can observe pre-tags and identify empty blocks, but only with help of S since pre-tags are encrypted under the joint key TS, while the users' data (the value of block) are protected by semantic security of underlying UREnc. Before pre-tags become accessible to T^* in Reshuffle, S preliminarily changes pre-tags (Step 2 of Reshuffle) and shuffles array A. Intuitively, T^* should not learn the link between pre-tags observed in any two reshuffles, since the storage server S draws a new random second key to OPRF for the level ℓ, into which all the blocks from levels below are moved, every time $\mathsf{Reshuffle}$ is performed.

Finally, after pre-tags become accessible, T^* can identify empty cells, however, the number of empty blocks is array A in $\mathsf{Reshuffle}$ does not depend on the access pattern (Lemma 2.7.1). Let Adv denote an adversary who corrupts T^* and a subset of users $\mathcal{U}^*$. The goal is to show that Adv cannot distinguish between $\mathsf{Exec}(\mathcal{AR}, \mathsf{Adv}, AP_0, \mathsf{HbC})$, and $\mathsf{Exec}(\mathcal{AR}, \mathsf{Adv}, AP_1, \mathsf{HbC})$ for any two compliant access patterns AP_0, AP_1 significantly better than pure guessing.

We define a series of games $\mathsf{G}_1, \dots, \mathsf{G}_{10}$, where G_1 denotes an execution $\mathsf{Exec}(\mathcal{AR}, \mathsf{Adv}, AP_0, \mathsf{HbC})$, and G_{10} an execution $\mathsf{Exec}(\mathcal{AR}, \mathsf{Adv}, AP_1, \mathsf{HbC})$ for any two compliant access patterns AP_0, AP_1, and show that the views of the adversary in these games are indistinguishable. The games are defined in the same way as in the proof of Theorem 2.7.1, however reductions from G_1 to G_2 and from G_2 to G_3 proceed differently, while other reductions remain without changes. Below, we present reductions from G_1 to G_{10}.

G_2 - same as G_1 except for the changes in User: if the access is made by an uncorrupted user U_i, then in Steps 2 and 3b, re-randomize the found block, and write the dummy block in Step 5.

We introduce an intermediate game $\bar{\mathsf{G}}_1$ to apply the changes made in the game G_2 into two steps. In the first step (in $\bar{\mathsf{G}}_1$ compared to G_1), the changes affect only the block value, while the pre-tag remains as in G_1. And in the second step (in G_2 compared to $\bar{\mathsf{G}}_1$), pre-tags are changed, i.e. if User is performed by an uncorrupted user, then send $\mathsf{E}^{+*}_{\mathcal{D}}(\theta_{\mathcal{D}})$ in Step 5 for $\theta_{\mathcal{D}}$ computed in Step 2, and re-randomize the pre-tag in Step 3b regardless of the value of **found**. The reason for this split is the fact that T^* observes pre-tags in $\mathsf{Reshuffle}$.

Reduction from G_1 to $\bar{\mathsf{G}}_1$ is done similarly to the reduction from G_1 to G_2 in the proof of Theorem 2.7.1 (except that pre-tags are not altered). For reduction from $\bar{\mathsf{G}}_1$ to G_2, assume w.l.o.g. that there is a list of all initial $M \cdot N$ real pre-tags and $M \cdot N$ dummy pre-tags. Each pre-tag is an element of G_n, so we can represent pre-tags as a list of $2M \cdot N$ distinct (probability of collision is negligible) group elements: $(g_1, \dots, g_{2M \cdot N})$. The $\mathbf{i}$-th element of the list corresponds to a specific input to OPRF: $\mu_{\mathcal{D}} + (\mathbf{i} - N \cdot M)$ if $\mathbf{i} \geq M \cdot N$, and $\mu_{1+\mathbf{i}/M} + (\mathbf{i} \bmod M)$ otherwise. In Step 2 of $\mathsf{Reshuffle}$, w.l.o.g. we may assume that the list is changed by S to $(g_1^r, \dots, g_{2M \cdot N}^r)$ for some random r, and T^* observes a subset of it ordered randomly. There are $|AP|/2$ such lists, and we refer to them as to the lists of pre-tags.

We define series of games $\bar{\mathsf{G}}_1^{0,0}, \dots, \bar{\mathsf{G}}_1^{0,|AP|/2}$, $\bar{\mathsf{G}}_1^{1,0}, \dots, \bar{\mathsf{G}}_1^{1,|AP|/2}, \dots$, $\bar{\mathsf{G}}_1^{2M \cdot N,0}$, $\dots$, $\bar{\mathsf{G}}_1^{2M \cdot N,|AP|/2}$ with $\bar{\mathsf{G}}_1 := \bar{\mathsf{G}}_1^{2M \cdot N,|AP|/2}$ and $\mathsf{G}_2 := \bar{\mathsf{G}}_1^{0,0}$, where the games $\bar{\mathsf{G}}_1^{\mathbf{i},\mathbf{j}+1}$ and $\bar{\mathsf{G}}_1^{\mathbf{i},\mathbf{j}}$ differ in the following: for the $\mathbf{i}$-th element of the $\mathbf{j}$-th list of pre-tags, if this element corresponds to the dummy input or to the input of uncorrupted user, replace the corresponding value with a random value, and related OPRF evaluations (for $t = 2\mathbf{j}$ and $t = 2\mathbf{j} + 1$ and inputs w.r.t. $\mathbf{i}$-th element of the list) are responded by S with random values. We show that views of in $\bar{\mathsf{G}}_1^{\mathbf{i},\mathbf{j}+1}$ and $\bar{\mathsf{G}}_1^{\mathbf{i},\mathbf{j}}$ are indistinguishable to the adversary by a reduction to composite DDH. Let $\hat{s}_{\mathbf{j}}$ denote the second OPRF key used in the $\mathbf{j}$-th list of pre-tags. Given $(X, Y, Z) = (\mathsf{g}^x, \mathsf{g}^y, \mathsf{g}^z)$, we construct a distinguisher $\mathcal{D}$ as follows: on input μ corresponding to the $\mathbf{i}$-th element of the $\mathbf{j}$-th list, set $\mathsf{g}^{1/(s+\mu)} = X$, so that $f_{s,\hat{s}}(\mu) = X^{\hat{s}}$ for any $\hat{s} \neq \hat{s}_{\mathbf{j}}$, and $f_{s,\hat{s}_{\mathbf{j}}}(\mu) = Z$, where $\mathsf{g}^{\hat{s}_{\mathbf{j}}} = Y$. For other inputs $\mu' \neq \mu$, $f_{s,\hat{s}_{\mathbf{j}}}(\mu') = Y^{1/(\hat{s}+\mu')}$ (respectively, OPRF is simulated as $f_{1,\hat{s}_{\mathbf{j}}}(\mu')$ with $\mathsf{g} \leftarrow Y$). If (X, Y, Z) is a DH tuple, then the game proceeds as $\bar{\mathsf{G}}_1^{\mathbf{i},\mathbf{j}+1}$, otherwise, if (X, Y, Z) is a random tuple, the game proceeds as $\bar{\mathsf{G}}_1^{\mathbf{i},\mathbf{j}}$. Since dummy pre-tags and the pre-tags of uncorrupted users observed by T^* are replaced with random pre-tags, the views of the adversary in $\bar{\mathsf{G}}_1$ and G_2 are indistinguishable.

G_3: same as G_2, except that pre-tags in OPRF evaluations performed by an uncorrupted user U_i are replaced with dummy pre-tags computed and sent by T^* to U_i. Since the adversary does not learn tags used by uncorrupted users in User, the views in these games are indistinguishable.

Reductions from G_3 to G_5 are the same as in the proof of Theorem 2.7.1. Games $\mathsf{G}_6, \ldots, \mathsf{G}_{10}$ are defined analogously to $\mathsf{G}_1, \ldots, \mathsf{G}_5$, in reversed order, with the change that whenever AP_0 is mentioned, it is replaced with AP_1. We have that $\mathsf{G}_5 = \mathsf{G}_6$, and so the views of the adversary in games G_1 and G_{10} are indistinguishable. □

Proof of Theorem 2.4.1 . We have to show that $\mathsf{AnonRAM}_{\mathsf{polylog}}$ construction provides indistinguishability of access patterns in the two following cases: a) collusion of honest-but-curious subset of users $\mathcal{U}^*$ and the storage server, b) collusion of honest-but-curious subset of users $\mathcal{U}^*$ and the tag server T^*. In either case, we have to construct a simulator that simulates the execution of an access pattern (for uncorrupted users) without knowing it, such that the adversary is not able to distinguish between the real or simulated execution. Note that the probability of overflow in Step 8 of $\mathsf{Reshuffle}$ algorithm is determined by n balls to n bins experiment and therefore is small ($\leq 1/n$); the overflow itself does not help the adversary, it only affects its running time. The theorem follows from Theorems 2.7.1 and 2.7.2. □

2.7.2 Proof of $\mathsf{AnonRAM}^{\mathbf{M}}_{\mathsf{polylog}}$

Proof Sketch of Theorem 2.4.2. The argument for integrity is based on ZK proof systems introduced in Section 2.4.7. They ensure that a user can modify the block value only if she knows the secret key which is required for decryption of the value. Since all the users generate their keys independently, the probability that a malicious user from $\mathcal{U}^*$ knows the secret key of any uncorrupted user is negligible. The employed ZK proof system also ensures that the introduced to the first level block is encrypted either under the same key as one of the blocks, for which the user has proven the knowledge of the secret key, or under the joint servers' key. Thus, the probability that the adversary changes during User any block belonging to an honest user, or introduces a new block of an honest user, is negligible. Finally, the privacy properties are preserved based on the security analysis of $\mathsf{AnonRAM}_{\mathsf{polylog}}$ against HbC adversaries. □

2.8 Conclusion

We have defined the concept of *Anonymous RAM (AnonRAM)* and presented two provably secure constructions. AnonRAM simultaneously provides privacy of content, access patterns and the user identities, while additionally ensuring the integrity of the user's data. It hence constitutes a natural extension of the concept of oblivious RAM (ORAM) to a domain with multiple, mutually distrusting users. Our first construction exhibits an access complexity linear in the number of users, while the second one improves the complexity to an amortized access cost that is poly-logarithmic in the total number of cells of all users, at the cost of requiring two non-colluding servers. Both constructions have two versions: a simpler one that assumes honest-but-curious users and a version secure against malicious users.

Several challenges still remain. In particular, it will be interesting to design a single server AnonRAM scheme with a poly-logarithmic access complexity. It will be also interesting to manage concurrent accesses to the server by the users. As a folow-up work, we address the latter challenge in Chapter 3.

3 Randomize-or-Change Encryption

3.1 Introduction

Encryption schemes that allow re-randomization of ciphertexts are important tools for applications related to privacy and anonymity, especially ‘universal’ mechanisms which allow re-randomization without public key (or any identifier of the recipient) [77, 33, 82, P1]. We will refer to such schemes as to *universally re-randomizable* encryption schemes (UREnc) throughout the chapter. We illustrate the core idea of UREnc [77]. The construction uses homomorphic properties of El-Gamal encryption. For a generator g of a prime-order group G_q of order q and a private and public key pair $x_i \in \mathbb{Z}_q$ and $g^{x_i} \in G_q$ for party i, encryption $\mathsf{E}_i^*(m)$ of message m is computed as an El-Gamal encryption of m together with an El-Gamal encryption of the value 1; i.e., $(g^r, g^{rx_i} \cdot m, g^t, g^{tx_i} \cdot 1)$. Re-randomization of a ciphertext c is possible using two random exponents r', t' such that the re-randomized ciphertext is equal to $(g^r \cdot (g^t)^{r'}, g^{rx_i} \cdot m \cdot (g^{tx_i})^{r'}, (g^t)^{t'}, (g^{tx_i})^{t'})$.

Backes et al. [P1] used UREnc as a building block in anonymous RAMs, which extend oblivious RAM (ORAM) [74] to the setting of mutually untrusted users. In anonymous RAM, users store and access their data on a server in encrypted form. The server should not learn users’ access patterns or users’ identities. A user, who reads a data block of another user, overwrites it with a re-randomization of that block. Respectively, a user, who reads a data block of itself, can overwrite it arbitrarily. This example shows two possible ways of ‘chaining’ ciphertexts (connecting one ciphertext with another one), for which there is a lack of foundational work since only re-randomization as a way of chaining ciphertexts is captured in the definition of UREnc.

In this work, we initiate a foundational effort for defining and analyzing a cryptographic primitive that allows chaining of ciphertexts.

Our contribution

1. We formally define *randomize-or-change encryption* (RoC), which allows chaining of ciphertexts by either re-randomizing them or encrypting new ciphertexts. RoC schemes ensure *operation-privacy*, i.e., given a ciphertext c', it is infeasible for an adversary to determine whether c' is a randomization of another ciphertext c, or a freshly constructed by the recipient of c. RoC schemes additionally ensure data *integrity*, which informally means that an adversary who is not the recipient of c *cannot* produce a ciphertext c' such that verification succeeds and the ciphertexts (c, c') decrypt to different plaintexts using the same decryption key. We refer to Table 3.1 for allowed operations for a given ciphertext w.r.t. the user’s role. The definitional part includes semantic security properties: we define *message indistinguishability* properties for the algorithms that output ciphertext, which are the encryption, the change, and the re-randomization algorithms. Next, we define *key anonymity* properties for the algorithms that output ciphertext, which accounts for the ability to relate ciphertexts under different keys. Finally,

we distinguish *same-key* and *any-key* RoC schemes. In same-key RoC, the recipient of a ciphertext is allowed to encrypt another message that can be decrypted using the same key (hence, same-key), whereas in any-key RoC this constraint is relaxed. We present a discrete log based *same-key* RoC construction, which is based on UREnc by Golle et al. [77], assisted with algebraic zero-knowledge proofs. With a small modification in the zero-knowledge proof system, the construction can be turned into *any-key*.

2. We define an extension to RoC, which we call *aggregatable* RoC. In aggregatable RoC, parties can simultaneously (in parallel) randomize or change a given ciphertext to a pre-ciphertext, and these pre-ciphertexts are then publicly combined to obtain the output ciphertext. The aggregation algorithm should be semantically correct as long as at most one pre-ciphertext is the result of the change operation. If an aggregatable RoC requires one or more rounds of synchronization between the users, we call such a scheme *interactive*, otherwise it is *non-interactive*. We present two discrete log based *aggregated* RoC constructions, which extend our discrete log based RoC. The first one is *same-key* and *non-interactive*, the second is *any-key* and requires one round of interaction.

3. We define a restricted variant of RoC, which we call *keyed* RoC. In keyed RoC, re-randomizers need to use keys in order to re-randomize ciphertexts. To protect these keys, we define *anonymity of re-randomization* property, which ensures that the recipient of ciphertext (or at least other re-randomizers and external observers) cannot relate re-randomizers to identities (public keys). We present a *hybrid* keyed RoC construction, which is designed for a pre-defined set of re-randomizers and based on both symmetric-key and public-key operations. Its public-key complexity depends on the number of users in the system and does not depend on message space, which offers a trade-off between these two parameters. The complexity factor is inherent to the construction and stems from the requirement that a user, while computing c' from c, has to update in the ciphertext the respective parts of all other users. To achieve integrity in the hybrid construction, we use verifier-efficient zero-knowledge proofs for proving composite statements, consisting of algebraic and arithmetic parts [2]. Plausible cost w.r.t. the message space comes at a price that anonymity of re-randomization is achieved only against other re-randomizers and external observers.

4. We show three applications to RoC and aggregated RoC: (parallel) anonymous RAM, group payment system, and anonymous communication. With the help of the aggregate functionality, we allow concurrent access from the users in specific scenarios. In particular, aggregated RoC facilitates practically instant responsiveness in our parallel anonymous RAM compared to the non-concurrent anonymous RAM [P1].

Comparison with UREnc

UREnc and RoC are public key encryption schemes that allow anyone to re-randomize ciphertexts without knowing the recipient's

Table 3.1: Permitted operations (randomize, change) for a given ciphertext c to obtain c' in randomize-or-change encryption, depending on the user's role.

	Recipient of c	Others
Randomize	Yes	Yes
Change	Yes	No

key. Security goals of both primitives, however, are partly orthogonal. By design, UREnc prevents correlation between the original and randomized ciphertexts. RoC schemes, on contrary, require an explicit link between ciphertexts in order to ensure integrity.

Basic RoC schemes are equipped with two more algorithms that are not present in UREnc: the *change* and the *verification* algorithms. The *change* algorithm allows the recipient of a ciphertext c to encrypt a new plaintext of her choice as c'. An adversary, observing a pair of ciphertexts (c, c'), should not be able to distinguish whether c' was a randomization of c or a new encryption computed by the recipient of c. We refer to this security notion as *indistinguishability of operation*.

The public *verification* algorithm allows to protect integrity of ciphertexts against a malicious adversary, who may try to change c to an arbitrary ciphertext c' while not being the intended recipient of c. Such an adversary is left with the only possible option in RoC: to re-randomize c.

In RoC, message indistinguishability and key anonymity properties defined for the change and re-randomization algorithms no longer hold, since a pair of ciphertexts can be verified (hence linked) in RoC by definition. If the malicious behaviour is not expected, i.e. an adversary is honest-but-curious, a trivial verification (returning always true) can be used in a RoC scheme, in which case message indistinguishability and key anonymity properties do hold, and we call such a scheme $\text{RoC}^{\mathbf{HbC}}$; conversely, we call a RoC scheme secure against malicious adversaries $\text{RoC}^{\mathbf{M}}$. In Table 3.2, we summarize the security properties defined for RoC and compare them to UREnc [133].

Chapter Outline The rest of the chapter is organized as follows. In Section 3.2, we briefly discuss different variants of RoC and their applications. In Section 3.3, we define randomize-or-change encryption (RoC), its security properties, and present $\mathsf{RoC}^{\mathbf{HbC}}_{\mathsf{dlog}}$ and $\mathsf{RoC}^{\mathbf{M}}_{\mathsf{dlog}}$ constructions in the discrete log setting. In Section 3.4, we define aggregatable RoC and present two ARoC constructions in the discrete log setting: the first is non-interactive ARoC-sk, the second is ARoC-ak and it requires one round of interaction. In Section 3.5, we define keyed RoC and present our hybrid keyed construction. Next, we provide more details on the applications of RoC: we define parallel anonymous RAM schemes and present a construction thereof in Section 3.6, then we elaborate on our group payment system Section 3.7.

In Section 3.8, we discuss related work. In Section 3.9, we complete the security analysis. Section 3.10 concludes.

3.2 Overview

In this section, we give an informal overview on different variants of randomize-or-change encryption, their security properties and applications.

3.2.1 Variants of RoC

RoC schemes enable chaining of ciphertexts. We say a party *changes* a ciphertext if it applies the change algorithm for a ciphertext and obtains another ciphertext. Conversely, we say a party *randomizes* a ciphertext if it applies the randomize algorithm.

Let a pair of ciphertexts be in relation $\mathcal{R}$ if they are chained. We distinguish the following cases:

1. Any party can re-randomize a given ciphertext c that encrypts message m without knowing the corresponding public key. The resulting ciphertext c_1 can be verified against c and thereby is in the chain relation, i.e. $(c, c_1) \in \mathcal{R}$.
2. The recipient of c (a party that knows the corresponding secret key to decrypt c) is allowed to encrypt any message m_2 as c_2 such that $(c, c_2) \in \mathcal{R}$.
3. Any party that is not the recipient of c is not allowed to find a ciphertext c_3 encrypting some message $m_3 \neq m$, which would extend the chain. For such a pair the relation does not hold, i.e. $(c, c_3) \notin \mathcal{R}$.

The adversary model Integrity of plaintext property ensures that in all three cases the verification algorithm outputs the correct boolean value, which indicates whether a pair of ciphertext is in the chain relation or not. The first two cases represent honest behaviour of parties, while the third case represents a malicious behaviour. When considering honest behaviour only, the verification is trivial (not enforced). The choice of the adversary model impacts not only the verification algorithm, but also the security properties of a RoC scheme, which is foreseen in Table 3.2 and we will consider this distinction when formally defining RoC schemes and presenting our RoC constructions.

Recipient constraints for RoC Depending on whether the recipient is restricted to encrypt c_2 under the same key, or is allowed to encrypt under a different key, we distinguish two variants of RoC schemes: *same-key-RoC* (RoC-sk) and *any-key-RoC* (RoC-ak).

Aggregatable RoC Let for some given ciphertext c there be a tuple $(\widetilde{c}_1, \ldots, \widetilde{c}_\ell)$ such that $(c, \widetilde{c}_i) \in \mathcal{R}$ for all $i \in [1, \ell]$. A RoC scheme allows for verification of multiple pairs $(c, \widetilde{c}_i)$, to check whether or not they are in the relation $\mathcal{R}$. An *aggregatable* RoC (ARoC) additionally allows to publicly combine (aggregate) $(c, \widetilde{c}_1, \ldots, \widetilde{c}_\ell)$ and compute a new ciphertext c', such that the semantic properties of the scheme are preserved. We allow up to one non-rerandomizer among the pairs $(c, \widetilde{c}_i)$. Since the output ciphertext c' is publicly computed from $(c, \widetilde{c}_1, \ldots, \widetilde{c}_\ell)$, the values $\widetilde{c}_i$ are hence treated as intermediate and may not necessarily represent valid ciphertexts. ARoC may require interaction between users and the aggregation server, in which case the scheme is called *interactive* (otherwise, *non-interactive*). We call a RoC scheme *basic* if it does not implement the aggregate functionality.

Keyed RoC The main motivation for UREnc and RoC is that the ciphertexts cannot be publicly linked to the recipient's public key. Similar to UREnc, the re-randomization algorithm in RoC does not use the recipient's public key. The re-randomization algorithm in UREnc requires only a ciphertext as input. In RoC, we will refer to this feature as to *keyless randomization*. We will define a complementary variant of RoC, called *keyed* RoC, which requires the re-randomization algorithm to take a *randomization key* as an additional input.

3.2.2 Security properties

Since RoC is a functional extension of UREnc (basic RoC defines two more algorithms that are not present in UREnc), our starting point is security games defined in [133]. In all definitions, we consider a polynomial-time adversary.

Message indistinguishability The encryption algorithm provides message indistinguishability if an adversary cannot find out (significantly better than pure guessing) which message out of two provided by the adversary, was encrypted by the challenger. The re-radnomization algorithm provides message indistinguishability if an adversary cannot find out which ciphertext out of two provided by the adversary was re-randomized by the challenger. In addition to these two properties, already defined in [133], we formulate an indistinguishability game for the change algorithm. The change algorithm provides message indistinguishability if an adversary cannot find out which ciphertext out of two provided by the adversary was changed by the challenger.

Key anonymity Ciphertext anonymity [22] refers to the property of ciphertext to hide identities of the sender (i.e. who created the ciphertext) and the recipient (who can read the ciphertext). The encryption algorithm provides key anonymity if an adversary cannot find out (significantly better than pure guessing) which key, out of two public keys generated by the challenger, was used to encrypt an

Table 3.2: Security properties in UREnc (as defined in [133]) and randomize-or-change encryption. We use E, R, Ch to denote the encryption, re-randomization, and change algorithms, respectively. "+" ("−") means that a respective security property can (cannot) be achieved.

Security property		UREnc	$\mathrm{RoC}^{\mathbf{HbC}}$	$\mathrm{RoC}^{\mathbf{M}}$
Integrity of plaintext				+
Indistinguishability of operation	for R and Ch		+	+
Message indistinguishability	for E	+	+	+
	for R	+	+	−
	for Ch		+	−
Key anonymity	for E	+	+	+
	for R	+	+	−
	for Ch		+	−
Anonymity of re-randomization (for keyed RoC)	external		+	+
	internal		+ or −	+ or −

adversarial message. The re-randomization algorithm provides key anonymity if an adversary cannot find out which key, out of two public keys generated by the challenger, was used to encrypt a message chosen by the adversary to re-randomize it. Here, the adversary provides a message and randomness to the challenger, hence can reconstruct possible ciphertexts, one of which is used by the challenger as input to the re-randomize algorithm. In addition to these two key anonymity properties, already defined in [133], we add a key anonymity game for the change algorithm. The change algorithm provides key anonymity if an adversary cannot find out which key, out of two public keys generated by the challenger, was used to encrypt a message chosen by the adversary to change it. Similarly to the previous property, the adversary provides a message and randomness to the challenger, hence can reconstruct two potential inputs the change algorithm.

Anonymity of re-randomization As the re-randomization algorithm in keyed RoC requires randomization keys as input, we need to ensure that re-randomizers cannot be identified. To this end, we define *anonymity of re-randomization* properties. We distinguish two cases: anonymity of re-randomization against the recipient of ciphertext (internal anonymity) and against re-randomizers (external anonymity). In both cases the challenger generates randomization keys for two users and picks one of them to randomize an adversarial ciphertext. The adversary plays a role of either the recipient of the ciphertext or another re-randomizer. A

RoC schemes achieves anonymity of re-randomization if the adversary cannot find out which user re-randomized the ciphertext. Note that RoC schemes with keyless randomization achieve both properties by definition, as there is no key material used in the algorithm which could allow an adversary to differentiate re-randomizers.

Indistinguishability of operation Using the re-randomization and the change operations, users can create chains of ciphertexts, in which two neighboring ciphertexts verify. The initial ciphertext in such a chain is created using the encryption algorithm. Regardless of the type of operation performed to append a chain, it should not be revealed to an external observer.

Integrity of plaintext Intuitively, RoC permits ciphertext chaining as follows: for a given ciphertext c, the recipient (i.e. a party who holds the secret key to decrypt c) can produce a new ciphertext, possibly encrypting another value, while any other party can only re-randomize c. In either case, the resulting ciphertext c' should verify against c. Integrity of plaintext property ensures that no party but the recipient can do more than re-randomization, i.e. it protects integrity and prevents malicious adversaries from corrupting users' data in the chains of ciphertexts. We refer to Table 3.2 for a comparison of security properties in UREnc and RoC.

3.2.3 Applications

The traditional application for UREnc are mixnets [77]. A mixnet is a set of mixes, whose task is to re-randomize incoming messages and output them in a random order. Interestingly enough, RoC with integrity property cannot protect against malicious mixes. More specifically, RoC does not break the link between re-randomized ciphertexts; on contrary, to verify a randomized ciphertext the original ciphertext is required, and so an external observer could recover the original order of messages processed by a mix server. Nonetheless, we identify several applications for RoC, which we discuss below.

Parallel Anonymous RAM In anonymous RAM [P1], a set of users anonymously operates on the joint external storage, while hiding from the server everything except for the fact that someone has accesses some data. The linear anonymous RAM construction [P1] assumes a single server, whose storage is encrypted and equally divided among the users. The users have to use anonymous communication to access the server. A user can hide her access pattern by using any secure ORAM [74], and her identity by accessing the same ORAM's locations in other users' storages, simply re-randomizing them.

We observe that we can build an anonymous RAM using RoC-sk, as this primitive captures the necessary properties by definition. First, the user can *change* ciphertexts in her part of the storage, which corresponds to a 'write' operation in anonymous RAM. Second, the user can *randomize* ciphertexts that are not in her part of the storage, which corresponds to a 'read' operation. Finally, indistinguishability of operation property ensures that the change and the randomize operations are not distinguishable to the storage server. Resulting anonymous RAM is keyless, as the RoC ciphertexts do not require knowledge of public keys to randomize them and asymptotically the construction is the same as the linear anonymous RAM [P1].

In anonymous RAM, by definition only one user can access the storage at a time, which may lead to unwanted waiting if multiple users would like to access the storage. If we allow concurrent access to the server, using aggregatable RoC, we can achieve anonymous RAM and at the same time eliminate waiting time for users. We call the resulting construction *parallel anonymous RAM*. In Fig. 3.1,

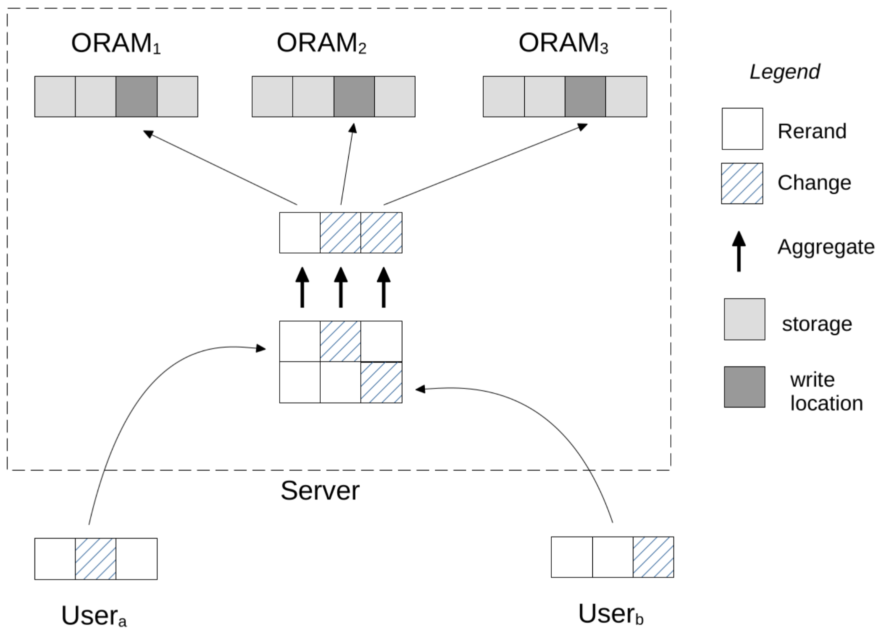

Figure 3.1: Parallel Anonymous RAM from aggregatable RoC. Example for 3 users, two of them sending write-requests to the same location at the server via anonymous channels.

we show on a high level how aggregatable RoC can be used to construct a parallel anonymous RAM scheme. The resulting construction is fully based on aggregatable RoC and requires anonymous communication channel for the users to communicate with the server. We refer to Section 3.6 for the definitions and security analysis.

Group payment system We consider the following scenario: a small group of users would like to manage micro-transactions among themselves. All users' coins are stored at the server in encrypted form. The recipient of ciphertext is the owner of the coin. Users may send (anonymous) requests to the server to change a coin c to another value c'. The server verifies the validity of the request and updates the coin to a new value. The server's storage is organized as any-key-RoC ciphertexts. User U_a can send her coin c to another user U_b by *changing* it to c', so that it is decryptable by U_b. To hide from the server, which coin has been sent, U_a *randomizes* the remaining coins stored on the server. Thanks to indistinguishability of operation property of RoC, the server cannot distinguish, whether the coin was indeed sent or randomized. If we allow concurrent access to the server, using aggregatable any-key-RoC we can handle user's transactions as they arrive to the server. We refer to Section 3.7 for further details.

A blockchain/broadcast protocol for anonymous pairwise communication within a group We use ideas from the group payment system to construct anonymous communication protocol resilient to malicious mixes (users). Previously, we were interested only in the ownership of ciphertexts (coins). In the communication protocol, we also interested in the content of ciphertexts.

The protocol is based on periodical publishing (by broadcast or in a blockchain) new values of a block of ciphertexts. Each time, each party reencrypts all ciphertexts sent to others, and may decrypt and replace the ciphertext sent to it (using her secret key). It is verifiable so parties cannot simply corrupt entries of other parties. This is a bit similar to mixnets, and resilient against malicious mixes (participants), except that the messages are only between the participants. The protocol provides strong anonymity: one cannot identify the sender and recipient, i.e. it provides *unobservability.*

3.3 (Basic) Randomize-or-Change Encryption

In this section, we define randomize-or-change encryption (RoC) schemes, its security properties, and present a RoC construction in the discrete log setting.

Notation By $c_{[i,j]}$ we denote an array of elements $(c_i, c_{i+1}, \ldots, c_j)$. To represent the distinguishable symbol $\perp$ in the groups, we use the identity element $1 \in G$. Multiplication and exponentiation of tuples is performed component-wise: $(c_1, c_2) \cdot (c_3, c_4) = (c_1 \cdot c_3, c_2 \cdot c_4)$ and $(c_1, c_2)^x = (c_1^x, c_2^x)$. We denote ElGamal encryption by $\mathfrak{e}_i(m; r) := (g^r, g^{rx_i} \cdot m) \in (G_q)^2$, where G_q is a group of a prime order q, $x_i \in \mathbb{Z}_q$ is a private key. The following homomorphic properties hold: $\mathfrak{e}_i(m_1; r_1) \cdot \mathfrak{e}_i(m_2; r_2) = \mathfrak{e}_i(m_1 \cdot m_2; r_1 + r_2)$, $\mathfrak{e}_i(m; r)^s = \mathfrak{e}_i(m^s; r^s)$. We may write $(\mathfrak{e}_i(m_1), \mathfrak{e}_i(m_2))$ instead of $(\mathfrak{e}_i(m_1; r_1), \mathfrak{e}_i(m_2; r_2))$, for some r_1, r_2 drawn independently. In particular,

$(\mathfrak{c}_1, \mathfrak{c}_2) \leftarrow (\mathfrak{e}_i(m), \mathfrak{e}_i(m))$ does not mean that the two ElGamal ciphertexts are equal, since it is a short-cut for $(\mathfrak{e}_i(m; r_1), \mathfrak{e}_i(m; r_2))$ for some r_1, r_2.

3.3.1 Definition

Definition 3.3.1 (RoC)**.** *A randomize-or-change encryption (RoC) scheme with key spaces* ($\mathcal{SK}$, $\mathcal{PK}$)*, message space* $\mathcal{M}$*, and ciphertext space* $\mathcal{C}$ *is a tuple of algorithms* $(\mathsf{KG}, \mathsf{E}, \mathsf{D}, \mathsf{R}, \mathsf{Ch}, \mathsf{V})$ *such that:*

- *The* key generation algorithm KG *is a randomized algorithm that takes a security parameter* 1^λ *as input and returns a key pair* $sk \in \mathcal{SK}$ *and* $pk \in \mathcal{PK}$*.*
- *The* encryption algorithm E *is a randomized algorithm that takes a public key* $pk \in \mathcal{PK}$ *and a plaintext* $m \in \mathcal{M}$ *as input and returns a ciphertext* $c' \in \mathcal{C}$*.*
- *The* decryption algorithm D *is a deterministic algorithm that takes a secret key* $sk \in \mathcal{SK}$ *and a ciphertext* $c \in \mathcal{C}$ *as input and returns a plaintext* $m \in \mathcal{M} \cup \{\perp\}$*, where* $\perp$ *is a distinguishable error symbol.*
- *The* re-randomization algorithm R *is a randomized algorithm that takes a ciphertext* $c \in \mathcal{C}$ *as input and returns a ciphertext* $c' \in \mathcal{C}$*.*
- *The* change algorithm Ch *is a randomized algorithm that takes a secret key* $sk \in \mathcal{SK}$*, a plaintext* $m \in \mathcal{M}$*, and a ciphertext* $c \in \mathcal{C}$*, as input and returns a ciphertext* $c' \in \mathcal{C}$*.*
- *The* verification *algorithm* V *is a deterministic algorithm that takes a pair of ciphertexts* $(c, c') \in \mathcal{C}^2$ *as input, and returns a boolean value.*

The above algorithms should satisfy the following correctness properties:

1. *Correctness of the encryption-decryption — Let* $(sk, pk) \leftarrow \mathsf{KG}(1^\lambda)$*. Then,* $\mathsf{D}_{sk}(\mathsf{E}_{pk}(m)) = m$ *for any* $m \in \mathcal{M}$*.*
2. *Correctness of the re-randomization — Let* $(sk, pk) \leftarrow \mathsf{KG}(1^\lambda)$*, and* $c \in \mathcal{C}$ *such that* $\mathsf{D}_{sk}(c) = m$ *for some* $m \in \mathcal{M}$*. Then for* $c' \leftarrow \mathsf{R}(c)$*,* $\mathsf{D}_{sk}(c') = m$*.*
3. *Correctness of the change — Let* $(sk, pk) \leftarrow \mathsf{KG}(1^\lambda)$*. Then* $\mathsf{D}_{sk}(\mathsf{Ch}_{sk}(m, c)) = m$ *for any* $m \in \mathcal{M}$ *and* $c \in \mathcal{C}$*.*
4. *Correctness of verification of re-randomization — Assume* $(c_1, c_2) \in \mathcal{C}^2$ *such that* $c_2 \leftarrow \mathsf{R}(c_1)$*. Then,* $\mathsf{V}(c_1, c_2) = true$*.*
5. *Correctness of verification of encryption — Assume* $(sk, pk) \leftarrow \mathsf{KG}(1^\lambda)$*, and* $(c_1, c_2) \in \mathcal{C}^2$ *such that* $c_2 \leftarrow \mathsf{Ch}_{sk}(m, c_1)$ *for some* $m \in \mathcal{M}$*. Then,* $\mathsf{V}(c_1, c_2) = true$*.*

Definition 3.3.1 is in flavor of RoC-sk, i.e., the change operation uses the recipient's key to encrypt a new message. The RoC-ak version of Definition 3.3.1 requires additionally the target public key as input in the change algorithm Ch and appropriate changes in the correctness properties involving Ch.

For the rest of this section, when defining games let $\Pi = (\mathsf{KG}, \mathsf{E}, \mathsf{D}, \mathsf{R}, \mathsf{Ch}, \mathsf{V})$ be a RoC-scheme with message space $\mathcal{M}$, ciphertext space $\mathcal{C}$, and let a game for Π between the challenger and a PPT adversary Adv be defined as shown in subsequent definitions. Since RoC schemes offer two options for chaining ciphertexts (using R and using Ch), we define a property that captures indistinguishability of operation.

Definition 3.3.2 (IND-OP game)**.** Π *provides* indistinguishability of operation *(or is* IND-OP secure*) if* $|\Pr[\text{IND-OP}^{\mathsf{Adv}}_{\Pi}(\lambda) = 1] - 1/2|$ *is negligible in* λ*. The game is defined as follows:* $b \leftarrow \{0,1\}$, $(sk, pk) \leftarrow \mathsf{KG}(1^\lambda)$, $(\mathsf{state}, m, m^*, r^*) \leftarrow \mathsf{Adv}(1^\lambda, pk)$, $c \leftarrow \mathsf{E}_{pk}(m^*; r^*)$, $(\hat{c}_0, \hat{c}_1) \leftarrow (\mathsf{R}(c), \mathsf{Ch}_{sk}(m, c))$, $b' \leftarrow \mathsf{Adv}(1^\lambda, \hat{c}_b, \mathsf{state})$, *return* $b = b'$.

In the RoC-ak version of the definition, the adversary additionally provides the target public key to the challenger. This adjustment has to be done in subsequent game definitions whenever the adversary provides a ciphertext to the challenger which is supposed to be used by the challenger as input to the change operation Ch.

Definition 3.3.3 (Message indistinguishability)**.** E *(respectively,* Ch*,* R*) provides* message indistinguishability *if* $|\Pr[\text{IND-CPA-E}^{\mathsf{Adv}}_{\Pi}(\lambda) = 1] - 1/2|$ *(respectively,* $|\Pr[\text{IND-CPA-Ch}^{\mathsf{Adv}}_{\Pi}(\lambda) = 1] - 1/2|$, $|\Pr[\text{IND-CPA-R}^{\mathsf{Adv}}_{\Pi}(\lambda) = 1] - 1/2|$ *) is negligible in* λ.

We show a trivial example of a scheme that is IND-CPA-Ch secure, but not IND-OP secure. Let an IND-CPA-Ch secure RoC scheme's ciphertextspace be extended by a single bit. If a ciphertext was computed using Ch, then set the bit to 1, otherwise, if a ciphertext was computed using R, set the bit to 0. Clearly, an adversary can easily win IND-OP game with the probability 1.

Definition 3.3.4 (Key anonymity)**.** E *(respectively,* Ch*,* R*) provides* key anonymity *if* $|\Pr[\text{ANON-E}^{\mathsf{Adv}}_{\Pi}(\lambda) = 1] - 1/2|$ *(respectively,* $|\Pr[\text{ANON-Ch}^{\mathsf{Adv}}_{\Pi}(\lambda) = 1] - 1/2|$, $|\Pr[\text{ANON-R}^{\mathsf{Adv}}_{\Pi}(\lambda) = 1] - 1/2|$ *) is negligible in* λ.

Definition 3.3.5 (Integrity of plaintext game)**.** Π *provides* integrity of plaintext *if* $\Pr[\text{INT-PTXT}^{\mathsf{Adv}}_{\Pi}(\lambda) = 1]$ *is negligible in* λ*. The game is defined as follows:* $(sk, pk) \leftarrow \mathsf{KG}(1^\lambda)$, $(\mathsf{state}, c_0, c_1) \leftarrow \mathsf{Adv}(1^\lambda, pk)$, *return* $\mathsf{V}(c_0, c_1) =$ *true* ***and*** $\mathsf{D}_{sk}(c_0) \neq \mathsf{D}_{sk}(c_1)$.

We call a RoC $\Pi = (\mathsf{KG}, \mathsf{E}, \mathsf{D}, \mathsf{R}, \mathsf{Ch}, \mathsf{V})$ *secure* if Π provides message indistinguishability for E and indistinguishability of operation, Ch provides key anonymity and satisfies integrity of plaintext.

Relation between security properties Integrity of plaintext property in RoC negates some previously defined security properties just by the fact that a pair of ciphertexts can be verified by anyone including the adversary. Specifically, message-indistinguishability and key anonymity of the change and of the re-randomization operations no holder hold. We explain it in more detail.

Key anonymity ensures that, in particular, two arbitrary ciphertexts cannot be linked to a single public key by an external adversary, however, this does not hold for all ciphertexts from a single chain as they are encrypted under the same public key. If a RoC scheme Π provides integrity of plaintext, then key anonymity of the

```
1 : $b \leftarrow \{0,1\}$
2 : $(sk, pk) \leftarrow \mathsf{KG}(1^\lambda)$

    IND-CPA-E$^{\mathsf{Adv}}_{\Pi}(\lambda)$
    3 : $(\mathsf{state}, m_0, m_1) \leftarrow \mathsf{Adv}(1^\lambda, pk)$
    4 : $\hat{c} \leftarrow \mathsf{E}_{pk}(m_b)$

    IND-CPA-Ch$^{\mathsf{Adv}}_{\Pi}(\lambda)$
    3 : $(\mathsf{state}, m_0, m_1, m^*, r^*) \leftarrow \mathsf{Adv}(1^\lambda, pk)$
    4 : $c \leftarrow \mathsf{E}_{pk}(m^*; r^*)$
    5 : $\hat{c} \leftarrow \mathsf{Ch}_{sk}(m_b, c)$

    IND-CPA-R$^{\mathsf{Adv}}_{\Pi}(\lambda)$
    3 : $(\mathsf{state}, m_0^*, r_0^*, m_1^*, r_1^*) \leftarrow \mathsf{Adv}(1^\lambda, pk)$
    4 : if $m_0^* = m_1^*$   return $\perp$
    5 : $c_0, c_1 \leftarrow \mathsf{E}_{pk}(m_0^*; r_0^*), \mathsf{E}_{pk}(m_1^*; r_1^*)$
    6 : $\hat{c} \leftarrow \mathsf{R}(c_b)$
```

```
1 : $b \leftarrow \{0,1\}$
2 : $(sk_i, pk_i)_{(i=0,1)} \leftarrow (\mathsf{KG}(1^\lambda))^2$

    ANON-E$^{\mathsf{Adv}}_{\Pi}(\lambda)$
    3 : $(\mathsf{state}, m) \leftarrow \mathsf{Adv}(1^\lambda, pk_0, pk_1)$
    4 : $\hat{c} \leftarrow \mathsf{E}_{pk_b}(m)$

    ANON-Ch$^{\mathsf{Adv}}_{\Pi}(\lambda)$
    3 : $(\mathsf{state}, m, m^*, r^*) \leftarrow \mathsf{Adv}(1^\lambda, pk_0, pk_1)$
    4 : $c \leftarrow \mathsf{E}_{pk_b}(m^*; r^*)$
    5 : $\hat{c} \leftarrow \mathsf{Ch}_{sk_b}(m, c)$

    ANON-R$^{\mathsf{Adv}}_{\Pi}(\lambda)$
    3 : $(\mathsf{state}, m^*, r^*) \leftarrow \mathsf{Adv}(1^\lambda, pk_0, pk_1)$
    4 : $c \leftarrow \mathsf{E}_{pk_b}(m^*; r^*)$
    5 : $\hat{c} \leftarrow \mathsf{R}(c)$
```

```
$b' \leftarrow \mathsf{Adv}(1^\lambda, \hat{c}, \mathsf{state})$
return $b = b'$
```

Figure 3.2: Definition of IND-CPA and ANON games.

change operation *does not* hold for Π. Indeed, in the key anonymity experiment for anonymity of the change operation, the adversary chooses inputs (message and randomness) to the encryption algorithm. The challenger then generates a ciphertext by plugging in these inputs and using one of the public keys. The adversary can generate both possible ciphertexts computed by the challenger and then verify the challenge ciphertext c' against each of them individually. Verification will succeed on the one chosen by the challenger. The same logic applies for the remaining cases, marked as 'No' in Table 3.2.

Theorem 3.3.1. *Let* $\Pi = (\mathsf{KG}, \mathsf{E}, \mathsf{D}, \mathsf{R}, \mathsf{Ch}, \mathsf{V})$ *be a RoC. If* Ch *provides message indistinguishability and the distributions of the change operation* $\mathsf{Ch}_{sk}(m, \cdot)$ *and re-randomization* $\mathsf{R}(c)$*, where* $\mathsf{D}_{sk}(c) = m$ *and a key pair* (sk, pk) *is generated by* KG*, are indistinguishable, then* Π *is IND-OP secure.*

Proof of Theorem 3.3.1 is postponed to Section 3.9.

3.3.2 Discrete log based construction $\mathsf{RoC}_{\mathsf{dlog}}$

We present $\mathsf{RoC}_{\mathsf{dlog}}$, a family of RoC schemes based on the UREnc construction by Golle et al. [77]. As the requirement in RoC, $\mathsf{RoC}_{\mathsf{dlog}}$ has the change and verification algorithms. We first present the construction in the HbC setting.

Construction 3.3.1 ($\mathsf{RoC}_{\mathsf{dlog}}^{\mathbf{HbC}}$)**.** *Let q be a prime and g be a generator of group G_q of order q. $\mathsf{RoC}_{\mathsf{dlog}}^{\mathbf{HbC}}$ is a RoC scheme with private key space $\mathcal{SK} := \mathbb{Z}_q$, public key space $\mathcal{PK} := G_q$, message space $\mathcal{M} := G_q$, ciphertext space $\mathcal{C} := (G_q)^4$, and the following algorithms:*

- *The key generation algorithm KG: on input security parameter λ, generate public parameters q, g, pick the secret key $sk \leftarrow_R \mathbb{Z}_q$, compute the public key $pk \leftarrow g^{(sk)}$, and return (sk, pk).*
- *The encryption algorithm E: on input $pk \in \mathcal{PK}$ and $m \in \mathcal{M}$, pick $(r,t) \leftarrow_R \mathbb{Z}_q$, compute $c_{[1,4]} \leftarrow (g^r, (pk)^r m, g^t, (pk)^t)$, and return $c_{[1,4]}$.*
- *The decryption algorithm D: on input $sk \in \mathcal{SK}$ and $c \in \mathcal{C}$, parse c as $c_{[1,4]}$. If $(c_3)^{sk} \neq c_4$, then return $\perp$, otherwise return $c_2 \cdot (c_1)^{-sk}$.*
- *The re-randomization algorithm R: on input $c \in \mathcal{C}$, parse c as $c_{[1,4]}$, pick $(r,t) \in (\mathbb{Z}_q)^2$, compute $c'_{[1,4]} \leftarrow (c_1 \cdot c_3^r, c_2 \cdot c_4^r, c_3^t, c_4^t)$, and return $c'_{[1,4]}$.*
- *The change algorithm Ch: on input $sk \in \mathcal{SK}$, $m \in \mathcal{M}$, $c \in \mathcal{C}$, pick $(h_1, h_2) \leftarrow_R (G_q)^2$, compute $c'_{[1,4]} \leftarrow (h_1, (h_1)^{sk} m, h_2, (h_2)^{sk})$, and return $c'_{[1,4]}$.*
- *The verification algorithm V: on input $(c, c') \in \mathcal{C}^2$, return true.*

In the RoC-ak version, the change algorithm Ch takes one more parameter, the target public key pk', which is used to encrypt message m via a call to the encryption algorithm with this key: $\mathsf{E}_{pk'}(m)$.

Theorem 3.3.2. *Let Π be a $\mathsf{RoC}_{\mathsf{dlog}}^{\mathbf{HbC}}$ as defined in Construction 3.3.1 in a group G_q where DDH is hard. Then the encryption algorithm, the re-randomization algorithm, and the change algorithm provide message indistinguishability and key anonymity, Π provides indistinguishability of operation and anonymity of re-randomization.*

Remark 3.3.1. *Since its invention, UREnc has been understood as a public key primitive. We stress than it can also be defined as a* private *key primitive. $\mathsf{RoC}_{\mathsf{dlog}}^{\mathbf{HbC}}$ can be easily turned into such a primitive.*

Integrity of plaintext in $\mathsf{RoC}_{\mathsf{dlog}}^{\mathbf{M}}$ Here, we complete the presentation of $\mathsf{RoC}_{\mathsf{dlog}}$ by actually implementing the verification algorithm with the help of appropriate zero-knowledge proofs and proving that the construction achieves integrity of plaintext.

$\mathsf{RoC}_{\mathsf{dlog}}^{\mathbf{M}}$'s proof system is based on proving the knowledge of discrete logarithm [114] and its generalizations (proving the equality of discrete logarithms [43]).

These proofs are one of the most efficient and popular classes of ZK proofs. Combined proof systems (proof that all statements hold, proof that some of the statements hold) can be efficiently done (see, e.g., [30]).

Construction 3.3.2 ($\mathsf{RoC}^{\mathbf{M}}_{\mathsf{dlog}}$)**.** *In Construction 3.3.1, extend ciphertext space to* $\mathcal{C} := (G_q)^4 \times (\mathbb{Z}_q)^5$, *and update/add the following algorithms:*

- *The encryption algorithm* E*: on input* $pk \in \mathcal{PK}$ *and* $m \in \mathcal{M}$*, compute* ${c'}_{[1,4]}$ *as before and return* $({c'}_{[1,4]}, (1)^5)$*.*
- *The re-randomization algorithm* R*: on input* $(c_{[1,4]}, \sigma) \in \mathcal{C}$*, pick* $(r, t) \in (\mathbb{Z}_q)^2$*, compute* ${c'}_{[1,4]}$ *as before, then execute the algorithm* $\mathsf{P}(c_{[1,4]}, {c'}_{[1,4]}, r, t)$ *to obtain* $\sigma' \in (\mathbb{Z}_q)^5$*, and return* $({c'}_{[1,4]}, \sigma')$*.*
- *The change algorithm* Ch*: on input* $sk \in \mathcal{SK}$*,* $m \in \mathcal{M}$*,* $(c_{[1,4]}, \sigma) \in \mathcal{C}$*, compute* ${c'}_{[1,4]}$ *as before, then execute the prove algorithm* $\mathsf{P}(c_{[1,4]}, {c'}_{[1,4]}, sk)$ *to obtain* $\sigma' \in (\mathbb{Z}_q)^5$ *and return* $({c'}_{[1,4]}, \sigma')$*.*
- *The prove algorithm* P*: on input* $(c_{[1,4]}, {c'}_{[1,4]}) \in (G_q)^8$ *and either* $sk \in \mathbb{Z}_q$ *or* $(r, t) \in (\mathbb{Z}_q)^2$*, execute the steps necessary to prove the following statement about* $(c_{[1,4]}, {c'}_{[1,4]})$*:*

$$\begin{aligned}\Pi_{DL} := PoK\{sk \mid (c_4, c'_4) = (c_3^{sk}, (c'_3)^{sk})\} \vee \\ P\{\exists r, t \ \text{ s.t. } (c'_1, c'_2, c'_3, c'_4) = (c_1 \cdot c_3^r, c_2 \cdot c_4^r, c_3^t, c_4^t)\},\end{aligned}$$

and return the result as $\sigma \in (\mathbb{Z}_q)^5$*.*

- *The verification algorithm* V*: on input* $((c_{[1,4]}, \sigma), ({c'}_{[1,4]}, \sigma')) \in \mathcal{C}^2$*, execute the steps required to verify* $(c_{[1,4]}, {c'}_{[1,4]}, \sigma')$ *in* Π_{DL}*, and return the result as true or* $false$*.*

We note that the prove algorithm P is used as a subroutine in the change Ch and the re-randomization R algorithms and is not a part of the syntax. The component $c'_4 = (c'_3)^{sk}$ of Π_{DL} prevents the recipient from changing the associate public key of the output ciphertext. To allow the change of the public key, in the RoC-ak version of $\mathsf{RoC}^{\mathbf{M}}_{\mathsf{dlog}}$ we remove that component from the proof system.

Lemma 3.3.1. $\mathsf{RoC}^{\mathbf{M}}_{\mathsf{dlog}}$ *in the groups* G_q *where DDH is hard and assuming the random oracle model provides integrity of plaintext.*

Proof of Theorem 3.3.1 is postponed to Section 3.9.

3.4 Aggregatable Randomize-or-Change Encryption

In this section we discuss an extension to RoC, which allows for simultaneous verification and aggregation of multiple ciphertexts created by randomizing or changing the same ciphertext.

3.4.1 Definition

For a given ciphertext c, which encrypts m for user U_a, we consider an array of ciphertexts $(c_1, \ldots, c_\ell)$, where each element is computed either as $\mathsf{R}(c)$ or $\mathsf{Ch}_{sk_a}(m', c)$, and the goal is to "squash" ℓ elements into one. To make the aggregation of ciphertexts meaningful and unambiguous, we allow the number of elements in the array that are computed using Ch to be less or equal one. We assume that no user knows in advance how many ciphertexts will be aggregated. Simultaneous participation of users requires coordination. To this end, we distinguish the aggregator entity, whose sole task is to aggregate incoming messages in a well-defined manner (e.g., deterministically). Since the users may not know each others, we facilitate a star-like communication model between them and the aggregation server.

Definition 3.4.1 (Aggregatable RoC)**.** *An aggregatable randomize-or-change encryption (ARoC) scheme with key spaces* $(\mathcal{SK}, \mathcal{PK})$*, message space* $\mathcal{M}$*, ciphertext space* $\mathcal{C}$*, pre-ciphertext space* $\widetilde{\mathcal{C}}$*, and a polynomial function* ℓ *is a tuple of algorithms* $(\mathsf{KG}, \mathsf{E}, \mathsf{D}, \mathsf{R}, \mathsf{Ch}, \mathsf{Ag}, \mathsf{V})$ *such that:*

- *The* key generation algorithm KG *is a randomized algorithm that takes a security parameter* 1^λ *as input and returns a key pair* $sk \in \mathcal{SK}$ *and* $pk \in \mathcal{PK}$*.*
- *The* encryption algorithm E *is a randomized algorithm that takes a public key* $pk \in \mathcal{PK}$ *and a plaintext* $m \in \mathcal{M}$ *as input and returns a ciphertext* $c' \in \mathcal{C}$*.*
- *The* decryption algorithm D *is a deterministic algorithm that takes a secret key* $sk \in \mathcal{SK}$ *and a ciphertext* $c \in \mathcal{C}$ *as input and returns a plaintext* $m \in \mathcal{M} \cup \{\perp\}$*, where* $\perp$ *is a distinguishable error symbol.*
- *The* re-randomization algorithm R *is a randomized algorithm that takes a ciphertext* $c \in \mathcal{C}$*, a randomness* r [1]*, and possibly an auxilliary string* $\mathsf{aux} \in \{0,1\}^{\ell(\lambda)}$ *as input and returns a pre-ciphertext* $\widetilde{c} \in \widetilde{\mathcal{C}}$*.*
- *The* change algorithm Ch *is a randomized algorithm that takes a secret key* $sk \in \mathcal{SK}$*, a public key* $pk \in \mathcal{PK}$*, a plaintext* $m \in \mathcal{M}$*, a ciphertext* $c \in \mathcal{C}$*, a randomness* r*, and possibly an auxilliary string* $\mathsf{aux} \in \{0,1\}^{\ell(\lambda)}$ *as input and returns a pre-ciphertext* $\widetilde{c} \in \widetilde{\mathcal{C}}$*.*
- *The* aggregate algorithm Ag *is a deterministic algorithm that takes a ciphertext* $c \in \mathcal{C}$ *and pre-ciphertexts* $(\widetilde{c}_1, \ldots, \widetilde{c}_\ell) \in \widetilde{\mathcal{C}}^\ell$ *as input and returns a ciphertext* $c' \in \mathcal{C} \cup \perp$ *and possibly a string* $\mathsf{aux} \in \{0,1\}^{\ell(\lambda)}$*. We call* $(c, \widetilde{c}_1, \ldots, \widetilde{c}_\ell)$ *an* aggregation tuple*. We will omit* aux *if it is not returned by the algorithm.*
- *The* verification *algorithm* V *is a deterministic algorithm that takes a pair of ciphertexts* $(c, c') \in \mathcal{C}^2$ *and pre-ciphertexts* $\widetilde{c}_1, \ldots, \widetilde{c}_\ell$ *as input and returns a boolean value.*

[1]Here, we use randomness explicitly as input to avoid maintaining a state between subsequent calls to the algorithm in the case when the re-randomization and change operations are interactive w.r.t. the aggregate algorithm. If it is clear from the context, we may omit randomness r.

Let $(sk, pk) \leftarrow \mathsf{KG}(1^\lambda)$, $(sk', pk') \in (\mathcal{SK}, \mathcal{PK})$ be a key pair, $c \in \mathcal{C}$, $m \in \mathcal{M}$, $(\widetilde{c}_1, \ldots, \widetilde{c}_\ell) \in \widetilde{\mathcal{C}}^\ell$, where $\widetilde{c}_i \leftarrow \mathsf{R}(c, \mathsf{aux})$ or $\widetilde{c}_i, \leftarrow \mathsf{Ch}_{sk}(pk', m, c, \mathsf{aux})$, for some $\mathsf{aux} \in \{0,1\}^{\ell(\lambda)} \cup \perp$ for all $i \in [1, \ell]$ such that Ch *is used at most once. We call such $(c, \widetilde{c}_1, ..., \widetilde{c}_\ell)$ a* valid aggregation tuple*. If the last ℓ elements of a valid aggregation tuple are computed using* R*, we say it is a* randomize tuple*. Otherwise, we say a* change tuple*. The above algorithms should satisfy the following correctness properties:*

1. Correctness of the encryption — Let $(sk, pk) \leftarrow \mathsf{KG}(1^\lambda)$. Then, $\mathsf{D}_{sk}(\mathsf{E}_{pk}(m)) = m$ for any $m \in \mathcal{M}$.

2. Unordered aggregation — Let $(c, \widetilde{c}_1, ..., \widetilde{c}_\ell)$ be an aggregation tuple. Then, for any permutation $\mathbf{\Pi}$ of ℓ elements, $\mathsf{Ag}(c, \widetilde{c}_{\mathbf{\Pi}_1}, ..., \widetilde{c}_{\mathbf{\Pi}_\ell}) = \mathsf{Ag}(c, \widetilde{c}_1, ..., \widetilde{c}_\ell)$.

3. Correctness of the re-randomization — Let $(sk, pk) \leftarrow \mathsf{KG}(1^\lambda)$, and let $c \in \mathcal{C}$ such that $\mathsf{D}_{sk}(c) = m$ for some $m \in \mathcal{M}$. Let $(c, \widetilde{c}_1, ..., \widetilde{c}_\ell)$ be a valid randomize aggregation tuple, and let $c' \leftarrow \mathsf{Ag}(c, \widetilde{c}_1, ..., \widetilde{c}_\ell)$, such that $c' \neq \perp$. Then, $\mathsf{D}_{sk}(c') = m$.

4. Correctness of the change — Let $(sk, pk) \leftarrow \mathsf{KG}(1^\lambda)$, and let $(c, \widetilde{c}_1, ..., \widetilde{c}_\ell)$ be a valid change aggregation tuple, in which one of the pre-ciphertexts is computed as $\mathsf{Ch}_{sk}(pk', m, c, \mathsf{aux})$ for some $m \in \mathcal{M}$ and $\mathsf{aux} \in \{0,1\}^\ell \cup \perp$, and $c' \leftarrow \mathsf{Ag}(c, \widetilde{c}_1, ..., \widetilde{c}_\ell)$, such that $c' \neq \perp$. Then, $\mathsf{D}_{sk'}(c') = m$.

5. Correctness of verification of re-randomization — Let $c \in \mathcal{C}$ be a ciphertext and $(c, \widetilde{c}_1, ..., \widetilde{c}_\ell)$ a valid randomize aggregation tuple, and let $c' \leftarrow \mathsf{Ag}(c, \widetilde{c}_1, ..., \widetilde{c}_\ell)$ such that $c' \neq \perp$. Then, $\mathsf{V}(c, c', \widetilde{c}_1, ..., \widetilde{c}_\ell) = true$.

6. Correctness of verification of change — Let $c \in \mathcal{C}$ be a ciphertext and $(c, \widetilde{c}_1, ..., \widetilde{c}_\ell)$ a valid randomize change tuple, and let $c' \leftarrow \mathsf{Ag}(c, \widetilde{c}_1, ..., \widetilde{c}_\ell)$ such that $c' \neq \perp$. Then, $\mathsf{V}(c, c', \widetilde{c}_1, ..., \widetilde{c}_\ell) = true$. If the change algorithm is restricted to the case $(sk', pk') = (sk, pk)$, we call the scheme same-key*, otherwise* any-key*. We may omit pk' from the notation for a same-key scheme.*

The randomization R and change Ch algorithms may additionally use an auxiliary input $\mathsf{aux} \in \{0,1\}^*$ and output an intermediate result; thus, to ensure that the aggregation leads to a meaningful result, we define *interactive aggregation.*

Definition 3.4.2 (Interactive aggregation)**.** *Let $(sk, pk) \leftarrow \mathsf{KG}(1^\lambda)$, $(sk', pk') \in (\mathcal{SK}, \mathcal{PK})$ be a key pair, $c \in \mathcal{C}$ such that c can be decrypted using sk, and $m' \in \mathcal{M}$. The interactive aggregation for ℓ parties in k rounds w.r.t. ciphertext c consists of the following steps:*

```
1 : $\mathsf{aux}_0 = \perp$    // For each $u \in [1, \ell]$ fix the algorithm, either R or Ch.
2 : for $i \in [0, k]$
3 :     for $u \in [1, \ell]$   $\widetilde{c}_u \leftarrow \mathsf{R}(c, \mathsf{aux}_i; r_u)$ or $\mathsf{Ch}_{sk}(pk', m', c, \mathsf{aux}_i; r_u)$
4 :     if $i < k$     $(\perp, \mathsf{aux}_{i+1}) \leftarrow \mathsf{Ag}(c, \widetilde{c}_1, ..., \widetilde{c}_\ell)$
5 :     else           $(c', \perp) \leftarrow \mathsf{Ag}(c, \widetilde{c}_1, ..., \widetilde{c}_\ell)$
6 : return $c'$
```

We call an interaction aggregation complete *if $c' \neq \perp$.*

3.4.2 Aggregatable same-key RoC construction

To implement the aggregation functionality, we utilize homomorphic properties of our discrete log RoC scheme (Section 3.3.2). If all elements of array $(c_1, \dots, c_\ell)$ are randomizations of c, we can directly use the scheme, multiply the ciphertexts, raise the message part to $(\ell)^{-1}$ and obtain a new valid randomization of c. However, we have ensure the validity of the aggregation in the remaining two cases: a) when one of the ciphertexts encrypts another message m' for U_a (RoC-sk), and b) when one of the ciphertexts encrypts some message m^* for some other user U_b (RoC-ak). In this section, we address the former case and in Section 3.4.3, the latter. The aggregate functionality does not require users to form valid ciphertexts, they need to compute *pre-ciphertexts* $(\widetilde{c}_1, \dots, \widetilde{c}_\ell)$, which may or may not be decryptable on its own and will be aggregated into a valid ciphertext.

We start with the RoC-sk construction in the HbC model. Recall that in Construction 3.3.1 a ciphertext is a tuple $(\mathfrak{e}_a(m), \mathfrak{e}_a(1))$, where $\mathfrak{e}_i(m; r) := (g^r, g^{rx_i} \cdot m)$. The re-randomization algorithm returns $(\mathfrak{e}_a(m), \mathfrak{e}_a(1))$ and the change algorithm $(\mathfrak{e}_a(m'), \mathfrak{e}_a(1))$, while the randomness is fresh. To account for the unknown number of aggregated ciphertexts, we extend the ciphertext space by one more element to $(\mathfrak{e}_a(m), \mathfrak{e}_a(1), \perp)$ and modify the algorithms as follows: the re-randomize operation returns $(\mathfrak{e}_a(m), \mathfrak{e}_a(1), \mathfrak{e}_a(1))$, and the change operation $(\mathfrak{e}_a(m), \mathfrak{e}_a(1), \mathfrak{e}_a(m'/m))$. Unlike in Construction 3.3.1, the result of the change operation is *not* a valid encryption of m', the final result will be computed by the aggregator for a vector of (pre-)ciphertexts. The aggregator multiplies c and the vector of ciphertexts componentwise, ignoring the bottom symbol $\perp$, to obtain $(\mathfrak{e}_a(m^{\ell+1}), \mathfrak{e}_a(1), \mathfrak{e}_a(m'/m))$. By raising the first component to $(\ell + 1)^{-1}$ and multiplying with the third, one obtains $\mathfrak{e}_a(m')$. Finally, the aggregator outputs $(\mathfrak{e}_a(m'), \mathfrak{e}_a(1), \perp)$, which is a valid ciphertext.

To achieve integrity of plaintext, we use appropriate zero-knowledge proofs, similar to Construction 3.3.2.

Construction 3.4.1 (ARoC-sk). *Let q be a prime and g be a generator of group G_q of order q. ARoC-sk is an aggregatable RoC scheme with private key space $\mathcal{SK} := \mathbb{Z}_q$, public key space $\mathcal{PK} := G_q$, message space $\mathcal{M} := G_q$, ciphertext space $\mathcal{C} := (G_q)^4$, pre-ciphertext space $\widetilde{\mathcal{C}} := (G_q)^6 \times (\mathbb{Z}_q)^6$, and the following algorithms:*

- *The key generation algorithm KG: on input security parameter λ, generate public parameters q, g, pick the secret key $sk \leftarrow_R \mathbb{Z}_q$, compute the public key $pk \leftarrow g^{sk}$, and return (sk, pk).*
- *The encryption algorithm E: on input $pk \in \mathcal{PK}$ and $m \in \mathcal{M}$, pick $(r, t) \leftarrow_R \mathbb{Z}_q$, compute $c \leftarrow (g^r, (pk)^r \cdot m, g^t, (pk)^t)$, and return c.*
- *The decryption algorithm D: on input $sk \in \mathcal{SK}$ and $c_{[1,4]} \in \mathcal{C}$, if $(c_{[3]})^{sk} \neq c_{[4]}$, then return $\perp$, otherwise return $c_{[2]} \cdot (c_{[1]})^{-sk}$.*
- *The re-randomization algorithm R: on input $c_{[1,4]} \in \mathcal{C}$, pick $(r, t_1, t_2) \in (\mathbb{Z}_q)^3$, compute $c'_{[1,6]} \leftarrow (c_{[1]} \cdot c^r_{[3]}, c_{[2]} \cdot c^r_{[4]}, c^{t_1}_{[3]}, c^{t_1}_{[4]}, c^{t_2}_{[3]}, c^{t_2}_{[4]})$, then execute the algorithm*

$\mathsf{P}^*(c, c'_{[1,6]}, r, t_1, t_2)$ *to obtain* $\sigma \in (\mathbb{Z}_q)^6$*, and return* $(c'_{[1,6]}, \sigma)$*.*

- *The change algorithm* Ch*: on input* $sk \in \mathcal{SK}$*,* $m' \in \mathcal{M}$*, and* $c_{[1,4]} \in \mathcal{C}$*, compute* $m \leftarrow \mathsf{D}_{sk}(c)$*. If* $m = \perp$*, terminate. Otherwise, pick* $(h_1, h_2, h_3) \leftarrow_R (G_q)^3$*, compute* $c'_{[1,6]} \leftarrow (h_1, (h_1)^{sk} \cdot m, h_2, (h_2)^{sk}, h_3, (h_3)^{sk} \cdot m'/m)$*, then execute the prove algorithm* $\mathsf{P}^*(c, c', sk)$ *to obtain* $\sigma' \in (\mathbb{Z}_q)^6$ *and return* (c', σ')*.*
- *The aggregate algorithm* Ag*: on input* $c \in \mathcal{C}$ *and* $(\widetilde{c}_1, ..., \widetilde{c}_\ell) \in \widetilde{\mathcal{C}}^\ell$*, verify that* $\mathsf{V}^*(c, \widetilde{c}_u) = true$ *for all* $u \in [1, \ell]$*. Terminate, if any of the verification results is* $false$*. Otherwise, proceed to compute* $\widetilde{C}_{[1,6]} \leftarrow (c_{[1,4]}, 1, 1) \cdot \prod_{u=1}^{\ell} (\widetilde{c}_u)_{[1,6]}$ *and output* $((\widetilde{C}_{[1]})^{(\ell+1)^{-1}} \cdot \widetilde{C}_{[5]}, (\widetilde{C}_{[2]})^{(\ell+1)^{-1}} \cdot \widetilde{C}_{[6]}, \widetilde{C}_{[3]}, \widetilde{C}_{[4]})$*.*
- *The prove algorithm* P^**: on input* $(c_{[1,4]}, c'_{[1,6]}) \in (G_q)^{10}$ *and either* $sk \in \mathbb{Z}_q$ *or* $(r, t_1, t_2) \in (\mathbb{Z}_q)^3$*, execute the steps necessary to prove the following statement about* (c, c')*:*

$$\Pi_{DL} := PoK\left\{sk \mid (c_{[4]}, c'_{[4]}) = \left(c_{[3]}^{sk}, (c'_{[3]})^{sk}\right)\right\} \vee \\ P\{\exists r, t_1, t_2 \mid c'_{[1,6]} = (c_{[1]} \cdot c_{[3]}^{r}, c_{[2]} \cdot c_{[4]}^{r}, c_{[3]}^{t_1}, c_{[4]}^{t_1}, c_{[3]}^{t_2}, c_{[4]}^{t_2})\},$$

and return the result as $\sigma \in (\mathbb{Z}_q)^6$*.*

- *The verification algorithm* V^**: on input* $(c, (\widetilde{c}, \sigma')) \in (\mathcal{C}, \widetilde{\mathcal{C}})$*, execute the steps required to verify* $(c, \widetilde{c}, \sigma')$ *in* Π_{DL}*, and return the result as true or* $false$*.*
- *The verification algorithm* V*: on input* $(c, c', \widetilde{c}_1, \ldots, \widetilde{c}_\ell) \in (\mathcal{C}^2, \widetilde{\mathcal{C}}^\ell)$*, return* $false$ *if any of* $\mathsf{V}^*(c, \widetilde{c}_u)$ *for* $u \in [1, \ell]$ *returned* $false$*. Otherwise, compute* $\hat{c} \leftarrow \mathsf{Ag}(c, \widetilde{c}_1, \ldots, \widetilde{c}_\ell)$ *and return the result of comparison* $\hat{c} = c'$ *as true or false.*

Lemma 3.4.1. *Construction 3.4.1 satisfies correctness properties, as defined in Definition 3.4.1.*

Proof. Correctness of encryption follows from its correctness in Construction 3.3.1, and unordered aggregation follows from the commutativity of multiplication.

Correctness of the re-randomization — Let encryption of message m with randomness (r_0, t_0) be $c = (\mathfrak{e}(m; r_0), \mathfrak{e}(1; t_0))$. Applying R on c, we get a pre-ciphertext $(\mathfrak{e}(m; r_0{+}t_0{\cdot}r), \mathfrak{e}(1; t_0{\cdot}t_1), \mathfrak{e}(1; t_0{\cdot}t_2))$, where (r, t_1, t_2) is fresh randomness each time $\mathsf{R}(c)$ is executed. Let $\widetilde{c}_u \leftarrow \mathsf{R}(c)$ for all $u \in [1, \ell]$. In the aggregation we have $\prod_{u=1}^{\ell} (\widetilde{c}_u)_{[1,6]} = (\mathfrak{e}(m^\ell), \mathfrak{e}(1), \mathfrak{e}(1))$, let us denote it as $\mathfrak{c}_{[1,3]}$. Then, $\widetilde{C}_{[1,6]} = (c_{[1,4]}, 1, 1) \cdot \mathfrak{c}_{[1,3]} = (\mathfrak{e}(m), \mathfrak{e}(1), (1, 1)) \cdot \mathfrak{c}_{[1,3]} = (\mathfrak{e}(m^{\ell+1}), \mathfrak{e}(1), \mathfrak{e}(1))$ and let us denote it as $\mathfrak{c}'_{[1,3]}$. Finally, the aggregation returns $((\mathfrak{c}'_1)^{(\ell+1)^{-1}} \cdot \mathfrak{c}'_3, \mathfrak{c}'_2)$, which is $(\mathfrak{e}(m), \mathfrak{e}(1))$.

Correctness of the change — Since there is only one pre-ciphertext that is not a randomization, we can w.l.o.g. rewrite the aggregation as follows: $\widetilde{C}_{[1,6]} = (c_{[1,4]}, 1, 1) \cdot (\mathfrak{e}(m^{\ell-1}), \mathfrak{e}(1), \mathfrak{e}(1)) \cdot \mathsf{Ch}_{sk}(c) = (\mathfrak{e}(m^\ell), \mathfrak{e}(1), \mathfrak{e}(1)) \cdot (\mathfrak{e}(m), \mathfrak{e}(1), \mathfrak{e}(m'/m))$ $= (\mathfrak{e}(m^{\ell+1}), \mathfrak{e}(1), \mathfrak{e}(m'/m))$ and let us denote it as $\mathfrak{c}'_{[1,3]}$. Finally, the aggregation returns $((\mathfrak{c}'_1)^{(\ell+1)^{-1}} \cdot \mathfrak{c}'_3, \mathfrak{c}'_2)$, which is $(\mathfrak{e}(m'), \mathfrak{e}(1))$.

Correctness of verification of re-randomization and of change follows from correctness of these algorithms and the fact that the proof system Π_{DL} returns true if computations are done honestly. □

Theorem 3.4.1. *Construction 3.4.1 provides indistinguishability of operation, integrity of plaintext, message indistinguishability and key anonymity for encryption algorithm* E.

Proof. Indistinguishability of operation – Since the aggregation is non-interactive, we simply have to prove that an adversary cannot distinguish between a pre-ciphertext computed as $\mathsf{R}(c)$ or as $\mathsf{Ch}_{sk}(m', c)$ for some key pair (sk, pk), message m' and ciphertext $c = \mathsf{E}_{pk}(m)$. Following the scheme, $\mathsf{R}(c)$ is computed as $(\mathfrak{e}(m), \mathfrak{e}(1), \mathfrak{e}(1))$, and $\mathsf{Ch}_{sk}(m, c)$ as $(\mathfrak{e}(m), \mathfrak{e}(1), \mathfrak{e}(m'/m))$. The third component $\mathfrak{e}(m'/m)$ is indistinguishable from $\mathfrak{e}(1)$ by semantic security of ElGamal encryption. We can show it by defining a series of games $\mathsf{G}_0, ..., \mathsf{G}_2, \mathsf{G}_2$ and proving indistinguishability between them. In G_0, the challenger computes re-randomization $(\mathfrak{e}(m), \mathfrak{e}(1), \mathfrak{e}(1))$. In G_1, the challenger outputs $(\mathfrak{e}(m), \mathfrak{e}(1), \widetilde{c}_{[5]}, \widetilde{c}_{[6]})$, where $\widetilde{c}_{[5,6]}$ are random elements. Finally in G_2, the challenger computes $(\mathfrak{e}(m), \mathfrak{e}(1), \mathfrak{e}(m'/m))$.

We show that $\mathsf{G}_0 \approx \mathsf{G}_1$ (G_0 and G_1 are computationally indistinguishable) via a reduction to DDH. Let (X, Y, Z) be a DDH-tuple. Key generation is replaced with $pk \leftarrow Y$, and the challenger outputs $(g^r, Y^r \cdot m, g^{t_1}, Y^{t_1}, X^{t_2}, Z^{t_2})$ for some random (r, t_1, t_2). If (X, Y, Z) is a true DDH tuple, the game is identical to G_0. If (X, Y, Z) is a random DDH tuple, the game is identical to G_1. From the DDH assumption it follows that the adversary can distinguish between G_0 and G_1 with at most a negligible probability. Similarly, we prove that $\mathsf{G}_1 \approx \mathsf{G}_2$. In the reduction, the challenger outputs $(g^r, Y^r \cdot m, g^{t_1}, Y^{t_1}, X^{t_2}, Z^{t_2} \cdot m'/m)$. We conclude that $\mathsf{G}_0 \approx \mathsf{G}_2$.

Message indistinguishability and key anonymity for E immediately follows from message indistinguishability and key anonymity in Construction 3.3.1. *Integrity of plaintext* – Similar to the proof of Lemma 3.3.1. □

In our construction, the aggregate functionality is *non-interactive*, meaning that users send a single message to the aggregator in order to chain a specific ciphertext.

3.4.3 Aggregatable any-key RoC construction

We further develop our same-key ARoC construction in Section 3.4.2 to achieve any-key ARoC. The goal is to allow the aggregation of a ciphertext $(\mathfrak{e}_b(m'), \mathfrak{e}_b(1))$ from the input ciphertext $(\mathfrak{e}_a(m), \mathfrak{e}_a(1))$ under the constraint that at most one change operation is permitted in the aggregation. The change algorithm in our any-key construction is *interactive* and requires 2 rounds of communication. The basic idea is to fix the target ciphertext τ and learn the randomness used by the re-randomizers in the first round, and then prepare a pre-ciphertext in the second round in such a way that the aggregation will output τ.

We will first focus on the identity part $\mathfrak{e}_a(1)$ and $\mathfrak{e}_b(1)$, and later extend our technique to the entire ciphertext. The input encryption of the identity is a pair $(g^r, g^{sk_a \cdot r})$, and the output of the aggregation should be $(g^{r'}, g^{sk_b \cdot r'})$ for some random r'. Since the secret key sk_b is not known to the owner U_a of the input ciphertext, we simply use $\mathfrak{c} = \mathfrak{e}_b(1)$ to encrypt the identity of U_b: to this end, user U_a raises raises $\mathfrak{c}$ to a random element from $\mathbb{Z}_q$.

Assume for a moment that U_a is the last party to send messages to the aggregator and U_a can read all messages that other parties send to the aggregator. Assume each re-randomizer send $(g^r, g^{sk_a \cdot r})^{r_i}$ for $i \in [1, \ell - 1]$, and let U_a send $(X, Y) \in (G_q)^2$. Multiplying the elements in the pairs with the input ciphertext, we get $(X \cdot g^{\hat{r}}, Y \cdot g^{sk_a \cdot \hat{r}})$, where $\hat{r} = r \cdot (1 + \sum_{i=1}^{\ell-1} r_i)$, which can be rewritten as $(X \cdot g^{\hat{r}}, Y/X \cdot (X \cdot g^{\hat{r}})^{sk_a})$. We observe that if U_a learns $g^{\hat{r}}$ before sending X, she can compute X and Y appropriately without knowing any other users' messages. The remaining challenge is to ensure that $X \cdot g^{\hat{r}} = g^{r'}$. Once we have it, we can remove the assumption that we temporarily made about U_a.

The technique for choosing (X, Y) discussed above suggests two rounds of communication: in the first round, the re-randomizers send $(g^r)^{r_i}$; in the second $(g^{sk_a \cdot r})^{r_i}$. If we change the order of rounds, U_a will be able to compute X and Y appropriately after learning $g^{sk_a \cdot \hat{r}}$. By careful combining these two techniques, we allow U_a to learn the necessary input from the re-randomizers in the first round, and then use it to correctly compute (X, Y) in the second round, so that the aggregation will output $(g^{r'}, g^{sk_b \cdot r'})$ chosen by U_a. The re-randomizers draw random numbers $(r_i, r_i') \in (\mathbb{Z}_q)^2$, compute $(\widetilde{c}_i)_{[1,4]} \leftarrow ((g^r, g^{sk_a \cdot r})^{r_i}, (g^r, g^{sk_a \cdot r})^{r_i'})$, send $(\widetilde{c}_i)_{[2]}$, $(\widetilde{c}_i)_{[3]}$ in the first round, and $(\widetilde{c}_i)_{[1]}$, $(\widetilde{c}_i)_{[4]}$ in the second round. The aggregator simply multiplies the input ciphertext with all pairs received from the users in the first round, and then in the second round the output will be $c \cdot \prod_{i=1}^{\ell} (\widetilde{c}_i)_{[1,2]} \cdot \prod_{i=1}^{\ell} (\widetilde{c}_i)_{[3,4]}$, which is the resulting ciphertext (encryption of the identity).

We have shown how U_a can ensure that $\mathfrak{e}_a(1)$ is aggregated into $\mathfrak{e}_b(1)$. We apply the same techniques to ensure that $\mathfrak{e}_a(m)$ is aggregated into $\mathfrak{e}_b(m')$. The owner has to cancel out the term that encrypts the input message, the rest is identical. Next, we present the full construction.

Construction 3.4.2 (ARoC-ak). *Extend Construction 3.4.1 by setting pre-ciphertext space to* $\widetilde{\mathcal{C}} := (G_q)^8 \times (\mathbb{Z}_q)^7$, *adding* $(\mathsf{P}_1, \mathsf{V}_1)$ *and modifying the following algorithms:*

- *The re-randomization algorithm* R*: on input* $c_{[1,4]} \in \mathcal{C}$*, randomness* r*, and an auxiliary string* $\mathtt{aux} \in \{0,1\}^{\ell(\lambda)}$*, parse* r *as* $(r_1, r_2, t_1, t_2) \in (\mathbb{Z}_q)^4$*, compute*

$$\widetilde{c}_{[1,8]} \leftarrow (c_{[1]} \cdot c_{[3]}^{r_1}, c_{[2]} \cdot c_{[4]}^{r_1}, c_{[3]}^{t_1}, c_{[4]}^{t_1}, c_{[1]} \cdot c_{[3]}^{r_2}, c_{[2]} \cdot c_{[4]}^{r_2}, c_{[3]}^{t_2}, c_{[4]}^{t_2}),$$

then proceed as follows. If $\mathtt{aux} = \perp$*, execute the algorithm* $\mathsf{P}_1(c, \widetilde{c}_{[2]}, \widetilde{c}_{[4,5]}, \widetilde{c}_{[7]}, r_1, t_1, r_2, t_2)$ *to obtain* $\sigma_1 \in (\mathbb{Z}_q)^5$ *and return* $(\widetilde{c}_{[2]}, \widetilde{c}_{[4,5]}, \widetilde{c}_{[7]}, \sigma_1)$*. Otherwise, ignore* $\mathtt{aux}$*, execute the algorithm* $\mathsf{P}_2(c, \widetilde{c}_{[1,8]}, r_1, t_1, r_2, t_2)$ *to obtain* $\sigma_2 \in (\mathbb{Z}_q)^7$ *and return* $(\widetilde{c}_{[1,8]}, \sigma_2)$*.*

- *The change algorithm* Ch*: on input* $sk \in \mathcal{SK}$, $pk' \in \mathcal{PK}$, $m' \in \mathcal{M}$, $c \in \mathcal{C}$, *randomness* r, *and an auxiliary string* $\mathsf{aux} \in \{0,1\}^{\ell(\lambda)}$, *compute* $m \leftarrow \mathsf{D}_{sk}(c)$*. If* $m = \perp$*, terminate. Parse* r *as* $(r', t', r_1, r_2, t_1, t_2) \in (\mathbb{Z}_q)^6$*, compute*

$$(\widetilde{c}_{[2]}, \widetilde{c}_{[4,5]}, \widetilde{c}_{[7]}) \leftarrow (c_{[2]} \cdot c_{[4]}^{r_1}, c_{[4]}^{t_1}, c_{[1]} \cdot c_{[3]}^{r_2}, c_{[3]}^{t_2}).$$

If $\mathsf{aux} = \perp$*, execute the algorithm* $\mathsf{P}_1(c_{[1,4]}, \widetilde{c}_{[2]}, \widetilde{c}_{[4,5]}, \widetilde{c}_{[7]}, r_1, t_1, r_2, t_2)$ *to obtain* $\sigma_1 \in (\mathbb{Z}_q)^5$ *and return* $(\widetilde{c}_{[2]}, \widetilde{c}_{[4,5]}, \widetilde{c}_{[7]}, \sigma_1)$*. Otherwise, parse* aux *as* $(\widetilde{C}_{[2]}, \widetilde{C}_{[4,5]}, \widetilde{C}_{[7]})$*, compute* $\bar{c}_{[1,4]} \leftarrow (g^{r'}, (pk')^{r'} \cdot m', g^{t'}, (pk')^{t'})$,

$$\widetilde{C}_{[1]} \leftarrow \left(\widetilde{C}_{[2]} \cdot m^{-\ell}\right)^{1/sk}, \widetilde{C}_{[3]} \leftarrow (\widetilde{C}_{[4]})^{1/sk}, \widetilde{C}_{[6]} \leftarrow \left(\widetilde{C}_{[5]} \cdot m^{\ell}\right)^{sk}, \widetilde{C}_{[8]} \leftarrow (\widetilde{C}_{[7]})^{sk},$$

$$\begin{aligned}
\widetilde{c}_{[1]} &\leftarrow (\bar{c}_{[1]})^{2\ell+1} \cdot (\widetilde{c}_{[2]}/m)^{1/sk} \cdot (\widetilde{C}_{[1]} \cdot \widetilde{C}_{[5]} \cdot c_{[1]})^{-1}, \\
\widetilde{c}_{[3]} &\leftarrow (\bar{c}_{[3]}) \cdot (\widetilde{c}_{[4]})^{1/sk} \cdot (\widetilde{C}_{[3]} \cdot \widetilde{C}_{[7]} \cdot c_{[3]})^{-1}, \\
\widetilde{c}_{[6]} &\leftarrow (\bar{c}_{[2]})^{2\ell+1} \cdot ((\widetilde{c}_{[5]})^{sk} \cdot m) \cdot (\widetilde{C}_{[2]} \cdot \widetilde{C}_{[6]} \cdot c_{[2]})^{-1}, \\
\widetilde{c}_{[8]} &\leftarrow (\bar{c}_{[4]}) \cdot (\widetilde{c}_{[7]})^{sk} \cdot (\widetilde{C}_{[4]} \cdot \widetilde{C}_{[8]} \cdot c_{[4]})^{-1},
\end{aligned}$$

then execute the prove algorithm $\mathsf{P}_2(c, \widetilde{c}_{[1,8]}, sk)$ *to obtain* $\sigma_2 \in (\mathbb{Z}_q)^7$ *and return* $(\widetilde{c}_{[1,8]}, \sigma_2)$*.*

- *The aggregate algorithm* Ag*: on input ciphertext* $c \in \mathcal{C}$ *and pre-ciphertexts* $(\widetilde{c}_1, ..., \widetilde{c}_\ell) \in \widetilde{\mathcal{C}}^\ell$*, re-order pre-ciphertexts by their* second *element and check if a tuple* $(c, (\widetilde{c}_1)_{[2]}, ..., (\widetilde{c}_\ell)_{[2]})$ *was previously stored. If not, parse* $\widetilde{c}_u$ *as four group elements* $d := ((\widetilde{c}_u)_{[2]}, (\widetilde{c}_u)_{[4,5]}, (\widetilde{c}_u)_{[7]})$ *and a proof* σ *and verify that* $\mathsf{V}_1(c, d, \sigma) = true$ *for all* $u \in [1, \ell]$*, then store the tuple, compute* $\widetilde{C}_{[\rho]} \leftarrow \prod_{u=1}^{\ell} (\widetilde{c}_u)_{[\rho]}$ *for* $\rho \in \{2, 4, 5, 7\}$ *and return* $(\widetilde{C}_{[2]}, \widetilde{C}_{[4,5]}, \widetilde{C}_{[7]})$*. Otherwise, proceed as follows. Abort, if there exist* $u \in [1, \ell]$ *such that* $(\widetilde{c}_u)_{[2]}$ *does not match the stored tuple. Verify that* $\mathsf{V}^*(c, \widetilde{c}_u) = true$ *for all* $u \in [1, \ell]$*. Terminate, if any of the verification results is* $false$*. Finally, compute* $C_{[1,4]} \leftarrow c \cdot \prod_{u=1}^{\ell} ((\widetilde{c}_u)_{[1,4]} \cdot (\widetilde{c}_u)_{[5,8]})$ *and output* $((C_{[1]})^{(2\ell+1)^{-1}}, (C_{[2]})^{(2\ell+1)^{-1}}, C_{[3]}, C_{[4]})$*.*

- *The prove algorithm* P_1*: on input* $(c_{[1,4]}, d_{[1,4]}) \in (G_q)^8$ *and* $(r_1, t_1, r_2, t_2) \in (\mathbb{Z}_q)^4$*, execute the steps necessary to prove the following statement about* $(c_{[1,4]}, d_{[1,4]})$*:*

$$\Pi_1 := PoK\{r_1, t_1, r_2, t_2 \mid d_{[1,4]} = (c_{[2]} \cdot c_{[4]}^{r_1}, c_{[4]}^{t_1}, c_{[1]} \cdot c_{[3]}^{r_2}, c_{[3]}^{t_2})\},$$

and return the result as $\sigma \in (\mathbb{Z}_q)^5$*.*

- *The verification algorithm* V_1*: on input* $(c_{[1,4]}, d_{[1,4]}, \sigma) \in (G_q)^8 \times (\mathbb{Z}_q)^5$*, execute the steps required to verify* $(c_{[1,4]}, d_{[1,4]}, \sigma)$ *in* Π_1*, and return the result as* $true$ *or* $false$*.*

Table 3.3: Computing expected values $(\widetilde{c}_{[1]}, \widetilde{c}_{[3]}, \widetilde{c}_{[6]}, \widetilde{c}_{[8]})$ from $(\widetilde{c}_{[2]}, \widetilde{c}_{[4]}, \widetilde{c}_{[5]}, \widetilde{c}_{[7]})$ in ARoC-ak construction. Indices in the circles indicate to the values computed by re-randomizers in the first round.

i	1	$\textcircled{2}$	3	$\textcircled{4}$
$\widetilde{c}_{[i]}$	$(\widetilde{c}_{[2]}/m)^{1/sk}$	$\widetilde{c}_{[2]}$	$(\widetilde{c}_{[4]})^{1/sk}$	$\widetilde{c}_{[4]}$
i	$\textcircled{5}$	6	$\textcircled{7}$	8
$\widetilde{c}_{[i]}$	$\widetilde{c}_{[5]}$	$(\widetilde{c}_{[5]})^{sk} \cdot m$	$\widetilde{c}_{[7]}$	$(\widetilde{c}_{[7]})^{sk}$

- *The prove algorithm* P^**: on input* $(c_{[1,4]}, \widetilde{c}_{[1,8]}) \in (G_q)^{12}$ *and either* $sk \in \mathbb{Z}_q$ *or* $(r_1, t_1, r_2, t_2) \in (\mathbb{Z}_q)^4$*, execute the steps necessary to prove the following statement about* $(c_{[1,4]}, \widetilde{c}_{[1,8]})$*:*

$$\begin{aligned}\Pi_2 := PoK\{sk \mid c_{[4]} = c_{[3]}^{sk}\} \vee \\ P\{\exists r_1, t_1, r_2, t_2 \mid c'_{[1,4]} &= (c_{[1]} \cdot c_{[3]}^{r_1}, c_{[2]} \cdot c_{[4]}^{r_1}, c_{[3]}^{t_1}, c_{[4]}^{t_1}), \\ c'_{[5,8]} &= (c_{[1]} \cdot c_{[3]}^{r_2}, c_{[2]} \cdot c_{[4]}^{r_2}, c_{[3]}^{t_2}, c_{[4]}^{t_2})\},\end{aligned}$$

and return the result as $\sigma \in (\mathbb{Z}_q)^7$.

- *The verification algorithm* V^**: on input* $((c, \sigma), (\widetilde{c}, \sigma')) \in (\mathcal{C}, \widetilde{\mathcal{C}})$*, execute the steps required to verify* $(c, \widetilde{c}, \sigma')$ *in* Π_2*, and return the result as* $true$ *or* $false$*.*

Lemma 3.4.2. *Construction 3.4.2 satisfies correctness properties, as defined in Definition 3.4.1.*

Proof. Correctness of encryption follows from its correctness in Construction 3.4.1, and unordered aggregation follows from the commutativity of multiplication.

Correctness of the re-randomization — Since $\mathtt{aux}$ is not used in R, we can focus on the case $\mathtt{aux} \neq \perp$ in the algorithm. Let encryption of message m with randomness (r_0, t_0) be $c = (\mathfrak{e}(m; r_0), \mathfrak{e}(1; t_0))$. Applying R on c, we get a pre-ciphertext $(\mathfrak{e}(m; r_0 + t_0 \cdot r_1), \mathfrak{e}(1; t_0 \cdot t_1), \mathfrak{e}(m; r_0 + t_0 \cdot r_2), \mathfrak{e}(1; t_0 \cdot t_2))$, where (r_1, r_2, t_1, t_2) is fresh randomness each time $\mathsf{R}(c)$ is executed. Let $\widetilde{c}_u \leftarrow \mathsf{R}(c)$ for all $u \in [1, \ell]$. In the aggregation we have $\prod_{u=1}^{\ell} (\widetilde{c}_u)_{[1,8]} = (\mathfrak{e}(m^\ell), \mathfrak{e}(1), \mathfrak{e}(m^\ell), \mathfrak{e}(1))$, let us denote it as $\mathfrak{c}_{[1,4]}$. Then, $C_{[1,4]} = c_{[1,4]} \cdot \mathfrak{c}_{[1,2]} \cdot \mathfrak{c}_{[3,4]} = (\mathfrak{e}(m^{2\ell+1}), \mathfrak{e}(1))$ and let us denote it as $\mathfrak{c}'_{[1,2]}$. Finally, the aggregation returns $((\mathfrak{c}'_1)^{(2\ell+1)^{-1}}, \mathfrak{c}'_2)$, which is $(\mathfrak{e}(m), \mathfrak{e}(1))$.

Correctness of the change — In the first round, the parties compute partial pre-ciphertexts as $(c_{[2]} \cdot c_{[4]}^{r_1}, c_{[4]}^{t_1}, c_{[1]} \cdot c_{[3]}^{r_2}, c_{[3]}^{t_2})$ for some random (r_1, r_2, t_1, t_2). Using the secret key sk the owner can pre-compute the values computed by re-randomizers in the second round, as shown in Table 3.3. Let $\widetilde{C}_u = \prod_{u=1}^{\ell} \widetilde{c}_u$ for all $u \in [1, \ell]$ be expected re-randomizations (including the owner's part). Then, the expected aggregated output is $C_{[1,2]} = \left(c_{[1,2]} \cdot \widetilde{C}_{[1,2]} \cdot \widetilde{C}_{[5,6]}\right)^{(2\ell+1)^{-1}}$

and $C_{[3,4]} = \left(c_{[3,4]} \cdot \widetilde{C}_{[3,4]} \cdot \widetilde{C}_{[7,8]}\right)$. The owner of c fixes the target ciphertext as $\bar{c}_{[1,4]} \leftarrow (g^{r'}, (pk')^{r'} \cdot m', g^{t'}, (pk')^{t'})$ for some random (r', t'). By manipulating the owner's values $(\tilde{c}_{[1]}, \tilde{c}_{[3]}, \tilde{c}_{[6]}, \tilde{c}_{[8]})$ sent in the second round, she ensures that $C_{[1,4]}$ matches $\bar{c}_{[1,4]}$. We first show it for $C_{[1]}$. We have that $(\bar{c}_{[1]})^{2\ell+1} = c_{[1]} \cdot \widetilde{C}_{[1]} \cdot \widetilde{C}_{[5]} = c_{[1]} \cdot \frac{\widetilde{C}_{[1]} \cdot \tilde{c}_{[1]}}{(\tilde{c}_{[2]}/m)^{1/sk}} \cdot \widetilde{C}_{[5]}$. Rewriting terms, we get $\tilde{c}_{[1]} = (\bar{c}_{[1]})^{2\ell+1} \cdot (\tilde{c}_{[2]}/m)^{1/sk} \cdot (\widetilde{C}_{[1]} \cdot \widetilde{C}_{[5]} \cdot c_{[1]})^{-1}$. Analogously, we obtain $\tilde{c}_{[3]}, \tilde{c}_{[6]}, \tilde{c}_{[8]}$.

Correctness of verification of re-randomization follows similar to the proof of Lemma 3.4.1. □

Theorem 3.4.2. *Construction 3.4.2 provides indistinguishability of operation, integrity of plaintext, message indistinguishability and key anonnimity for encryption algorithm* E*.*

Proof. Indistinguishability of operation – Following the scheme, $\mathsf{R}(c)$ is computed as $(\mathfrak{e}_a(m), \mathfrak{e}_a(1), \mathfrak{e}_a(m), \mathfrak{e}_a(1))$, and $\mathsf{Ch}_{sk_a}(pk_b, m, c)$ as a specially crafted value $\mathfrak{c}_{[1,4]}$ that allows to aggregate pre-ciphertexts to a pre-defined $(\mathfrak{e}_b(m'), \mathfrak{e}_b(1))$. We can view $\mathfrak{c}_{[1,4]}$ as $(\mathfrak{c}_{[1,4]}) = (\mathfrak{e}_A(m_1), \mathfrak{e}_A(1),\ \mathfrak{e}_B(m_2), \mathfrak{e}_B(1))$ for some unknown A, B. By semantic security of ElGamal encryption, the component $\mathfrak{e}_b(m')$ is indistinguishable from encryption of a random message $\mathfrak{e}_b(m^*)$, and by key anonymity $\mathfrak{e}_A(m^*)$ is indistinguishable from $\mathfrak{e}_a(m^*)$. We define a series of games as follows. In G_0, the challenger computes $(\mathfrak{e}_a(m), \mathfrak{e}_a(1), \mathfrak{e}_a(m), \mathfrak{e}_a(1))$. In G_1, $(\mathfrak{e}_a(m), \mathfrak{e}_R(1), \mathfrak{e}_a(m), \mathfrak{e}_a(1))$ for some random pk_R. We show that $\mathsf{G}_0 \approx \mathsf{G}_1$ via a reduction to DDH. Given a DDH tuple (X, Y, Z), the challenger generates a public key as $pk_a = Y^r$ and computes (X^r, Z^r) instead of $\mathfrak{e}_a(1)$. If (X, Y, Z) is a true DDH tuple, the computation is identical to G_0. If (X, Y, Z) is a random tuple, the computation is identical to G_1. We have that $\mathsf{G}_0 \approx \mathsf{G}_1$ due to the hardness of the DDH assumption. The remaining games are constructed using the technique above and as in the proof of Theorem 3.4.1.

Message indistinguishability and key anonymity for E immediately follows from message indistinguishability and key anonymity in Construction 3.3.1.

Integrity of plaintext – The proof system used in the construction ensures that messages from the second round of aggregation are bound to the first round, hence we can focus on the second round. The rest is similar to the proof of Lemma 3.3.1. □

3.5 Keyed Randomize-or-Change Encryption

In this section, we define *keyed* RoC schemes, which extend basic RoC schemes (Section 3.3) by including re-randomization keys that are used in the randomize algorithm. With inclusion of the additional keys, we define a new security game to characterize the ability of an adversary to distinguish re-randomization keys that are used by the challenger. We then present our hybrid keyed construction, which makes use of both symmetric-key and public-key operations.

3.5.1 Definition

Definition 3.5.1 (Keyed RoC)**.** *A keyed RoC scheme with key spaces* ($\mathcal{SK}$, $\mathcal{RK}$, $\mathcal{PK}$, $\mathcal{PRK}$)*, message space* $\mathcal{M}$*, and ciphertext space* $\mathcal{C}$ *is a tuple of algorithms* $(\mathsf{KG}, \mathsf{E}, \mathsf{D}, \mathsf{R}, \mathsf{Ch}, \mathsf{V})$ *such that:*

- *The* key generation algorithm KG *is a randomized algorithm that takes a security parameter* 1^λ *as input and returns a key pair* $sk \in \mathcal{SK}$ *and* $pk \in \mathcal{PK}$*.*
- *The* randomization key generation algorithm RG *is a randomized algorithm that takes a security parameter* 1^λ *as input and returns a key pair* $rk \in \mathcal{RK}$ *and* $prk \in \mathcal{PRK}$*. Public keys* prk *are added to public parameters* pub*.*
- *The* encryption algorithm E *is a randomized algorithm that takes a public key* $pk \in \mathcal{PK}$ *and a plaintext* $m \in \mathcal{M}$ *as input and returns a ciphertext* $c' \in \mathcal{C}$*.*
- *The* decryption algorithm D *is a deterministic algorithm that takes a secret key* $sk \in \mathcal{SK}$ *and a ciphertext* $c \in \mathcal{C}$ *as input and returns a plaintext* $m \in \mathcal{M} \cup \{\perp\}$*, where* $\perp$ *is a distinguishable error symbol.*
- *The* re-randomization algorithm R *is a randomized algorithm that takes a ciphertext* $c \in \mathcal{C}$ *and a randomization key* $rk \in \mathcal{RK}$ *as input and returns a ciphertext* $c' \in \mathcal{C}$*.*
- *The* change algorithm Ch *is a randomized algorithm that takes a secret key* $sk \in \mathcal{SK}$*, a plaintext* $m \in \mathcal{M}$*, and a ciphertext* $c \in \mathcal{C}$*, as input and returns a ciphertext* $c' \in \mathcal{C}$*.*
- *The* verification *algorithm* V *is a deterministic algorithm that takes a pair of ciphertexts* $(c, c') \in \mathcal{C}^2$ *as input, and returns a boolean value.*

The above algorithms should satisfy the following correctness properties:

1. *Correctness of the encryption-decryption — Let* $(sk, pk) \leftarrow \mathsf{KG}(1^\lambda)$*,* $(rk_i,\ prk_i)_{i=[1,\rho]} \leftarrow \mathsf{RG}(1^\lambda)^\rho$*. Then,* $\mathsf{D}_{sk}(\mathsf{E}_{pk}(m)) = m$ *for any* $m \in \mathcal{M}$*.*
2. *Correctness of the re-randomization — Let* $(sk, pk) \leftarrow \mathsf{KG}(1^\lambda)$*,* $(rk_i,\ prk_i)_{i=[1,\rho]} \leftarrow \mathsf{RG}(1^\lambda)^\rho$*, and* $c \in \mathcal{C}$ *such that* $\mathsf{D}_{sk}(c) = m$ *for some* $m \in \mathcal{M}$*. Then for* $c' \leftarrow \mathsf{R}_{rk_i}(c)$*,* $\mathsf{D}_{sk}(c') = m$*.*
3. *Correctness of the change — Let* $(sk, pk) \leftarrow \mathsf{KG}(1^\lambda)$*,* $(rk_i,\ prk_i)_{i=[1,\rho]} \leftarrow \mathsf{RG}(1^\lambda)^\rho$*. Then* $\mathsf{D}_{sk}(\mathsf{Ch}_{sk}(m, c)) = m$ *for any* $m \in \mathcal{M}$ *and* $c \in \mathcal{C}$*.*
4. *Correctness of verification of re-randomization — Assume* $(c_1, c_2) \in \mathcal{C}^2$ *such that* $c_2 \leftarrow \mathsf{R}_{rk_i}(c_1)$*. Then,* $\mathsf{V}(c_1, c_2) = true$*.*
5. *Correctness of verification of encryption — Assume* $(sk, pk) \leftarrow \mathsf{KG}(1^\lambda)$*,* $(rk_i,\ prk_i)_{i=[1,\rho]} \leftarrow \mathsf{RG}(1^\lambda)^\rho$*, and* $(c_1, c_2) \in \mathcal{C}^2$ *such that* $c_2 \leftarrow \mathsf{Ch}_{sk}(m, c_1)$ *for some* $m \in \mathcal{M}$*. Then,* $\mathsf{V}(c_1, c_2) = true$*.*

```
1 : $b \leftarrow \{0,1\}$
2 : $(rk_i, prk_i)_{i=0,1} \leftarrow \mathsf{RG}(1^\lambda)^2$
```

```
ANON-INT-R$_\Pi^{\mathsf{Adv}}(\lambda)$
3 : $\mathsf{pub} \leftarrow (prk_i)_{i=[0,1]}$
4 : $(\mathsf{state}, pk, m^*, r^*) \leftarrow \mathsf{Adv}(1^\lambda, \mathsf{pub})$
```

```
ANON-EXT-R$_\Pi^{\mathsf{Adv}}(\lambda)$
3 : $(sk, pk) \leftarrow \mathsf{KG}(1^\lambda)$
4 : $(\mathsf{state}, prk_2, m^*, r^*) \leftarrow \mathsf{Adv}(1^\lambda, pk, prk_0, prk_1)$
5 : $\mathsf{pub} \leftarrow (prk_i)_{i=[0,2]}$
```

```
$c \leftarrow \mathsf{E}_{pk}(m^*; r^*)$
$\hat{c} \leftarrow \mathsf{R}_{rk_b}(c)$
$b' \leftarrow \mathsf{Adv}(1^\lambda, \hat{c}, \mathsf{state})$
return $b = b'$
```

Figure 3.3: Definition of ANON-INT-R and ANON-EXT-R games.

Anonymity of re-randomization We define the notion of *anonymity of re-randomization* to characterize the ability to determine the identity of the user who performed re-randomization of ciphertext. Depending on the adversarial knowledge, we distinguish two instances of the security game: against the recipient of ciphertext (internal anonymity) and against an external observer (external anonymity). Clearly, we require that external anonymity hold for any RoC scheme.

Definition 3.5.2 (Internal and external anonymity of R)**.** Π *provides* internal (respectively, external) anonymity of randomization *if* $|\Pr[\text{ANON-INT-R}_\Pi^{\mathsf{Adv}}(1^\lambda) = 1] - 1/2|$ *(respectively,* $|\Pr[\text{ANON-EXT-R}_\Pi^{\mathsf{Adv}}(1^\lambda) = 1] - 1/2|$*) is negligible in* 1^λ.

3.5.2 Hybrid construction $\mathsf{RoC}_{\mathsf{hyb}}$

In this section, we present the generic hybrid randomize-or-change encryption ($\mathsf{RoC}_{\mathsf{hyb}}$). $\mathsf{RoC}_{\mathsf{hyb}}$ is based on hybrid encryption, which allows to handle large message spaces efficiently. The construction pays off when the number of re-randomizers is small and especially in the honest-but-curious adversary model, for which a trivial verification suffices.

We first describe the idea of hybrid encryption and then present $\mathsf{RoC}_{\mathsf{hyb}}$. Hybrid encryption works as follows. Assume there is a pair of private and public keys (sk, pk). To encrypt a long message MSG, one picks a random key k for a symmetric encryption scheme, encrypts the message as $C \leftarrow \mathsf{E}_k(MSG)$, and then encrypts k using the public key as $c \leftarrow \mathsf{E}_{pk}(k)$. To decrypt ciphertext (C, c), one decrypts c using sk to obtain k, and then decrypts C using k to obtain MSG. Because of relative efficiency of private-key encryption schemes over public-key ones, the hybrid encryption pays-off when message space is large. Golle et al. [77] describe the hybrid variant of universal mixing, in which the number of re-encryptions applied to a specific ciphertext is limited. In $\mathsf{RoC}_{\mathsf{hyb}}$ we circumvent the mentioned limitation for a fixed set of re-randomizers.

We introduce $\mathsf{RoC}_{\mathsf{hyb}}$ in two steps. First we will define $\mathsf{RoC}_{\mathsf{hyb}}^{\mathbf{HbC}}$ with a trivial verification and prove its security against HbC adversaries, then in Section 3.5.4 we will show the full construction $\mathsf{RoC}_{\mathsf{hyb}}^{\mathbf{M}}$ and prove its integrity of plaintext. Unlike $\mathsf{RoC}_{\mathsf{dlog}}$, $\mathsf{RoC}_{\mathsf{hyb}}$ does not provide internal anonymity of re-randomization.

Generic description In $\mathsf{RoC}_{\mathsf{hyb}}^{\mathbf{HbC}}$, we presume that there are N static users who are at the same time re-randomizers $\mathcal{U} = \{\mathsf{U}_1, \ldots, \mathsf{U}_N\}$, and each user U_i has a pair of private and public keys (sk_i, pk_i). For the same of presentation, we will use private keys as randomization keys. Encryption using pk_i is denoted as $\mathsf{e}_i(\cdot)$, decryption using sk_i as $\mathsf{d}_i(\cdot)$. We also assume that the encryption scheme is partially homomorphic and allows re-randomization; we denote re-randomization using pk_i as $\mathsf{re}_i(\cdot)$. For may explicitly specify the randomness r in the algorithm by $\mathsf{re}_i(c; r)$, where c is a ciphertext.

A ciphertext is associated with some user (the recipient U_i), and encryption of an identity under the recipient's key is attached to the ciphertext, so that no one can learn this accosiation, expect for the recipient itself. The ciphertext $(\alpha_{[1,N]}, \beta_{[1,N]}, \gamma, \delta)$ is constructed using N seeds $s_1, \ldots, s_N$. These seeds are encrypted into two arrays $\alpha_1, \ldots, \alpha_N$ and $\beta_1, \ldots, \beta_N$. Element j in these arrays corresponds to seed s_j. Array $\alpha_{[1,N]}$ contains encryptions of seeds $s_1, \ldots, s_N$ using the public key pk_i of the recipient $\mathsf{U}_i \in \mathcal{U}$, i.e. $\mathsf{e}_i(s_j)$ for all $j \in [1, N]$, and array $\beta_{[1,N]}$ contains encryptions of the same seeds using the public keys of the corresponding users, i.e. $\mathsf{e}_j(s_j)$ for all $j \in [1, N]$. The payload is stored in γ, and δ is encryption of an identity under the recipient's public key. Array $\alpha_{[1,N]}$ allows the recipient to obtain a plaintext message m from γ, and array $\beta_{[1,N]}$ and component δ allow to perform re-randomization of γ. The structure of ciphertexts in $\mathsf{RoC}_{\mathsf{hyb}}^{\mathbf{HbC}}$ is presented in Fig. 3.4.

We describe next the encryption, the decryption, the change, and the re-randomization algorithms for user U_i, who holds the key pair (sk_i, pk_i).

To encrypt a message m, user U_i generates N seeds s_j for all $j \in [1, N]$ and initializes arrays α and β with $(\mathsf{e}_i(s_j), \mathsf{e}_j(s_j))$, respectively. Then, the user computes the payload as $\gamma \leftarrow m \oplus_{j=1}^{N} \mathcal{G}(s_j)$, where $\mathcal{G}$ is a pseudo-random generator. Finally, the user encrypts the identity $\delta \leftarrow \mathsf{e}_i(1)$. To decrypt a ciphertext $c = (\alpha_{[1,N]}, \beta_{[1,N]}, \gamma, \delta)$, user U_i first decrypts all values from array $\alpha_{[1,N]}$ using sk_i to obtain $s_1, \ldots, s_N$ and then computes the plaintext message as $m = \gamma \oplus_{j=1}^{N} \mathcal{G}(s_j)$.

Assume U_i is the recipient of a ciphertext $c = (\alpha_{[1,N]}, \beta_{[1,N]}, \gamma, \delta)$, i.e. can decrypt it. To encrypt a message m' using the change operation, user U_i performs the following. First, the user decrypts c to obtain the plaintext m and the seed $s_i \leftarrow \mathsf{d}_i(\alpha_i)$. Next, he generates a fresh seed s'_i and computes the new payload as $\gamma' = (m' \oplus m) \oplus \gamma \oplus \mathcal{G}(s_i) \oplus \mathcal{G}(s'_i)$. Finally, the user encrypts s'_i into (α_i, β_i)

Users: U_1 *with* (pk_1, sk_1), ..., U_N *with* (pk_N, sk_N).
Public information: pk_1, ..., pk_N.
Ciphertext for U_i*:*

$\gamma = m \oplus \mathcal{G}(s_1) \oplus \cdots \oplus \mathcal{G}(s_N)$		
$\alpha_1 = \mathsf{e}_i(s_1)$	$\dots$	$\alpha_N = \mathsf{e}_i(s_N)$
$\beta_1 = \mathsf{e}_1(s_1)$	$\dots$	$\beta_N = \mathsf{e}_N(s_N)$
$\delta = \mathsf{e}_i(1)$		

Figure 3.4: Ciphertext structure in $\mathsf{RoC}^{\mathbf{HbC}}_{\mathsf{hyb}}$. User U_i is the recipient of ciphertext, $\mathcal{G}$ denotes a PRG, $\mathsf{e}_i(\cdot)$ denotes the encryption algorithm of a (traditional) re-randomizable CPA-secure encryption scheme, m is the encrypted message. Relative sizes of ciphertext components are kept, γ is supposed to be the largest component, while all remaining are of equal size.

and re-randomizes the remaining seed entries (for all valid $j \neq i$) of arrays $\alpha_{[1,N]}$ and $\beta_{[1,N]}$ and re-randomizes the encryption of an identity, i.e. δ. Note that the recipient should not change (α_j, β_j) plaintext values of any other user U_j, otherwise U_j could distinguish between the change operation and re-randomization based on the observed s_j value.

Assume U_j is the recipient of a ciphertext $c = (\alpha_{[1,N]}, \beta_{[1,N]}, \gamma, \delta)$. To re-randomize c, user U_i needs to replace seed s_i of the ciphertext with a newly generated s'_i while preserving the encrypted message m. Specifically, user U_i computes $s_i \leftarrow \mathsf{d}_i(\beta_i)$, generates a fresh seed s'_i, replaces the payload γ with $\gamma \oplus \mathcal{G}(s_i) \oplus \mathcal{G}(s'_i)$, and updates the associated seed entries as $\alpha_i = \mathsf{e}_j(s'_i)$ and $\beta_i \leftarrow \mathsf{e}_i(s'_i)$. Entry α_i cannot be computed using the recipient's public key pk_j, since the recipient is not known to U_i, instead U_i computes this value using δ and homomorphic properties of the underlying encryption scheme. Finally, the user re-randomizes δ and the remaining elements of arrays $\alpha_{[1,N]}$ and $\beta_{[1,N]}$.

Construction 3.5.1 ($\mathsf{RoC}^{\mathbf{HbC}}_{\mathsf{hyb}}$). *Let N be the number of users, $\mathcal{G} : \{0,1\}^{s(\lambda)} \to \{0,1\}^{l(\lambda)}$ be an efficient function, and $\widetilde{\mathcal{E}} = (\mathsf{kg}, \mathsf{e}, \mathsf{d}, \mathsf{re})$ be ElGamal encryption with key space $(\widetilde{\mathcal{SK}}, \widetilde{\mathcal{PK}})$, message space $\widetilde{\mathcal{M}} \subseteq \{0,1\}^{s(\lambda)}$, and ciphertext space $\widetilde{\mathcal{C}}$. $\mathsf{RoC}^{\mathbf{HbC}}_{\mathsf{hyb}}$ is a RoC scheme with secret key space $\mathcal{SK} = \mathcal{RK} := \widetilde{\mathcal{SK}}$, public key space $\mathcal{PK} = \mathcal{PRK} := \widetilde{\mathcal{PK}}$, message space $\mathcal{M} := \{0,1\}^{l(\lambda)}$, ciphertext space $\mathcal{C} := \widetilde{\mathcal{C}}^N \times \widetilde{\mathcal{C}}^N \times \{0,1\}^{l(\lambda)} \times \widetilde{\mathcal{C}}$, and the following algorithms:*

- *The key generation algorithm* KG *(which is also used as the randomization key algorithm* RG*): on security parameter λ as input, generate a pair of secret and public keys $(sk, pk) \leftarrow \mathsf{kg}(1^\lambda)$ and return (sk, pk). Once generated, public keys of N users become publicly available.*
- *The encryption algorithm* E *for user* U_i*: on input a public key $pk_i \in \mathcal{PK}$ and a message $m \in \mathcal{M}$, perform the following steps:*

```
1 : for $j \in [1, N]$ do
2 :     $s_j \leftarrow_R \{0,1\}^{s(\lambda)}$,
3 :     $\alpha_j \leftarrow \mathsf{e}_i(s_j), \quad \beta_j \leftarrow \mathsf{e}_j(s_j)$
4 : endfor
5 : $\gamma \leftarrow m \oplus_{j=1}^{N} \mathcal{G}(s_j), \quad \delta \leftarrow \mathsf{e}_i(1)$
6 : return $(\alpha_{[1,N]}, \beta_{[1,N]}, \gamma, \delta)$
```

- *The decryption algorithm* D *for user* U_i*: on input a private key* $sk_i \in \mathcal{SK}$ *and a ciphertext* $(\alpha_{[1,N]}, \beta_{[1,N]}, \gamma, \delta) \in \mathcal{C}$*, compute* $s_j \leftarrow \mathsf{d}_i(\alpha_j)$ *for all* $j \in [1, N]$*, then compute* $m \leftarrow \gamma \oplus_{j=1}^{N} \mathcal{G}(s_j)$*, and output* m*.*
- *The re-randomization algorithm* R *for user* U_i*: on input a ciphertext* $(\alpha_{[1,N]}, \beta_{[1,N]}, \gamma, \delta) \in \mathcal{C}$ *and a randomization key* sk_i *perform the following steps:*

```
1 : $s_i \leftarrow \mathsf{d}_i(\beta_i), \quad s'_i \leftarrow \{0,1\}^{s(\lambda)}$,
2 : $\alpha'_i \leftarrow \mathsf{re}(\delta) \cdot s'_i, \quad \beta'_i \leftarrow \mathsf{e}_i(s'_i)$
3 : for $j \in [1, N] \backslash i$ do
4 :     $\alpha'_j \leftarrow \mathsf{re}(\alpha_j), \quad \beta'_j \leftarrow \mathsf{re}(\beta_j)$
5 : endfor
6 : $\gamma' \leftarrow \gamma \oplus \mathcal{G}(s_i) \oplus \mathcal{G}(s'_i), \quad \delta' \leftarrow \mathsf{re}(\delta)$
7 : return $({\alpha'}_{[1,N]}, {\beta'}_{[1,N]}, \gamma', \delta')$
```

- *The change algorithm* Ch *for user* U_i*: on input a secret key* $sk_i \in \mathcal{SK}$*, a message* $m' \in \mathcal{M}$*, and a ciphertext* $c := (\alpha_{[1,N]}, \beta_{[1,N]}, \gamma, \delta) \in \mathcal{C}$*, perform the following steps:*

```
1 : $m \leftarrow \mathsf{D}_i(c), \quad s_i \leftarrow \mathsf{d}_i(\alpha_i), \quad s'_i \leftarrow \{0,1\}^{s(\lambda)}$
2 : $\alpha'_i \leftarrow \mathsf{e}_i(s'_i), \quad \beta'_i \leftarrow \mathsf{e}_i(s'_i)$
3 : for $j \in [1, N] \backslash i$ do
4 :     $\alpha'_j \leftarrow \mathsf{re}(\alpha_j), \quad \beta'_j \leftarrow \mathsf{re}(\beta_j)$
5 : endfor
6 : $\gamma' \leftarrow m' \oplus m \oplus \mathcal{G}(s_i) \oplus \mathcal{G}(s'_i), \quad \delta' \leftarrow \mathsf{re}(\delta)$
7 : return $({\alpha'}_{[1,N]}, {\beta'}_{[1,N]}, \gamma', \delta')$
```

- *The verification algorithm* V*: on input* $(c, c') \in \mathcal{C}^2$*, return true.*

In the RoC-ak version, the change algorithm Ch additionally takes the target public key pk_τ as input, which requires to appropriately update the algorithm.

3.5.3 Security analysis of $\mathsf{RoC}^{\mathbf{HbC}}_{\mathsf{hyb}}$

Definition 3.5.3 (PRG)**.** *A function* $\mathcal{G} : \{0,1\}^\lambda \to \{0,1\}^{l(\lambda)}$ *is called pseudo-random generator, in short PRG, if no PPT adversary* Adv *can distinguish the output of* $\mathcal{G}(s)$ *for some* $s \leftarrow_R \{0,1\}^\lambda$ *and a truly random sequence of* $l(\lambda)$ *bits significantly better than pure guessing.*

Theorem 3.5.1. *Let* $\Pi = (\mathsf{KG}, \mathsf{E}, \mathsf{D}, \mathsf{R}, \mathsf{Ch}, \mathsf{V})$ *be a* $\mathsf{RoC}_{\mathsf{hyb}}^{\mathbf{HbC}}$ *as defined in Construction 3.5.1,* $\mathcal{G}$ *be a PRG, and* $\mathcal{E} = (\mathsf{kg}, \mathsf{e}, \mathsf{d}, \mathsf{re})$ *be a semantically secure re-randomizable homomorphic encryption scheme (ElGamal encryption in the groups where DDH assumption holds), and* e *provides key anonymity. Then the encryption algorithm* E*, the change algorithm* Ch*, and the re-randomization algorithm* R *provide message indistinguishability and key anonymity,* Π *provides indistinguishability of operation and external anonymity of re-randomization.*

Lemma 3.5.1. *Let* $\Pi = (\mathsf{KG}, \mathsf{E}, \mathsf{D}, \mathsf{R}, \mathsf{Ch}, \mathsf{V})$ *be a* $\mathsf{RoC}_{\mathsf{hyb}}^{\mathbf{HbC}}$ *as defined in Construction 3.5.1,* $\mathcal{G}$ *be a PRG, and* $\mathcal{E} = (\mathsf{kg}, \mathsf{e}, \mathsf{d}, \mathsf{re})$ *be a semantically secure re-randomizable homomorphic encryption scheme. Then the encryption algorithm* E*, the change algorithm* Ch*, and the re-randomization algorithm* R *provide message indistinguishability.*

Proof. Assume that an adversary corrupts all users expect for the challenger U_i. The adversary can decrypt and obtain all seeds except for s_i. The encryption, the change, and the re-randomization algorithms are reduced to the encryption algorithm of the textbook hybrid public encryption, with the only difference that is the seed s_i is encrypted twice. It is well known that a CPA-secure hybrid public encryption scheme can be achieved by combining a CPA-secure public key encryption and semantically secure private key encryption schemes, see Theorem 10.13 of [90]. A semantically secure private encryption scheme can be instantiated using PRG, see Theorem 3.17 of [90]. $\square$

Lemma 3.5.2. *Let* $\Pi = (\mathsf{KG}, \mathsf{E}, \mathsf{D}, \mathsf{R}, \mathsf{Ch}, \mathsf{V})$ *be a* $\mathsf{RoC}_{\mathsf{hyb}}^{\mathbf{HbC}}$ *as defined in Construction 3.5.1,* $\mathcal{G}$ *be a PRG, and* $\mathcal{E} = (\mathsf{kg}, \mathsf{e}, \mathsf{d}, \mathsf{re})$ *be a semantically secure re-randomizable homomorphic encryption scheme. Then* Π *provides indistinguishability of operation.*

Proof. Re-randomization R and the change operation Ch performed by user U_j have the following in common: 1) s'_j remains the same for all users $j' \neq j$, 2) α'_j and β'_j will be re-randomized for all users $j' \neq j$, and 3) α_j and β will encrypt the new value s'_j.

The only difference between R and Ch is that, in addition to $\mathcal{G}(s'_j)$ for a fresh s'_j, the value $m' \oplus m$ is added. Given an adversary Adv that wins IND-OP game with a non-negligible probability, we construct a distinguisher $\mathcal{D}$ that distinguishes output of $\mathcal{G}$ and random bits.

Let G_0 be the IND-OP game against $\mathsf{RoC}_{\mathsf{hyb}}^{\mathbf{HbC}}$. Let G_1 be the same as G_0 except that message space of α and β is increased from $s(\lambda)$ to $l(\lambda)$ bits and only the first λ bits are used as input to $\mathcal{G}$ in E, D, Ch, and R, while the rest are ignored. The probability of the adversary success in G_1 is the same as in G_0.

Let G_2 be the same as G_1 except that $l(\lambda)$ bits of seeds are used in E, D, Ch, and R, instead of using the first λ bits as input to $\mathcal{G}$. We show that the success probabilities in these games differ by a negligible factor by a reduction to PRG:

given an adversary Adv against IND-OP, we construct a distinguisher $\mathcal{D}$ that uses Adv to distinguish between PRG and random bits. First, $\mathcal{D}$ generates a random bit $b \leftarrow_R \{0,1\}$ and then sends to Adv the challenger's output in IND-OP game against G_1 if $b = 0$, or against G_2 if $b = 1$. Since $\mathcal{G}$ used in $\mathsf{RoC}^{\mathbf{HbC}}_{\mathsf{hyb}}$ is a PRG, the probability of Adv guessing correctly the value of b is negligible, and so is the probability to distinguish between G_1 and G_2.

The success probability of an adversary in IND-OP against G_2 is $1/2$, since the bits used in Ch and R are random. □

Lemma 3.5.3. *Let* $\Pi = (\mathsf{KG}, \mathsf{E}, \mathsf{D}, \mathsf{R}, \mathsf{Ch}, \mathsf{V})$ *be a* $\mathsf{RoC}^{\mathbf{HbC}}_{\mathsf{hyb}}$ *as defined in Construction 3.5.1,* $\mathcal{G}$ *be a PRG, and* $\mathcal{E} = (\mathsf{kg}, \mathsf{e}, \mathsf{d}, \mathsf{re})$ *be a semantically secure re-randomizable homomorphic encryption scheme. Then encryption algorithm* E*, the change algorithm* Ch*, and re-randomization algorithm* R *provide key anonymity, if so does the underlying public key encryption scheme* $\mathcal{E}$*.*

Proof. Let $\mathcal{E} = (\mathsf{kg}, \mathsf{e}, \mathsf{d}, \mathsf{re})$ by the underlying public key encryption scheme of $\mathsf{RoC}^{\mathbf{HbC}}_{\mathsf{hyb}}$. Let $c = (\alpha_{[1,N]}, \beta_{[1,N]}, \gamma, \delta)$ be a ciphertext produced by the challenger using E in the game. The only component that differs depending on which public key has been used by the challenger is encryption of an identity, i.e. δ.

To show that Ch and R provide key anonymity, we use a similar argument, as in the proof of Lemma 3.5.2. Lemma follows given that e provides key anonymity. □

Lemma 3.5.4. *Let* $\Pi = (\mathsf{KG}, \mathsf{E}, \mathsf{D}, \mathsf{R}, \mathsf{Ch}, \mathsf{V})$ *be a* $\mathsf{RoC}^{\mathbf{HbC}}_{\mathsf{hyb}}$ *as defined in Construction 3.5.1,* $\mathcal{G}$ *be a PRG, and* $\mathcal{E} = (\mathsf{kg}, \mathsf{e}, \mathsf{d}, \mathsf{re})$ *be a semantically secure re-randomizable homomorphic encryption scheme, and* e *provides key anonymity. Then* Π *provides external anonymity of re-randomization.*

Proof. We use a similar argument, as in the proof of Lemma 3.5.2. □

Theorem 3.5.1 follows from Lemmas 3.5.1 to 3.5.4.

Remark 3.5.1. $\mathsf{RoC}^{\mathbf{HbC}}_{\mathsf{hyb}}$ *does not provide internal anonymity of re-randomization: the recipient can decrypt the ciphertext and obtain all the seeds and find out which seeds have been changed during re-randomization.*

3.5.4 Integrity of plaintext in $\mathsf{RoC}^{\mathbf{M}}_{\mathsf{hyb}}$

In this section, we extend the hybrid construction $\mathsf{RoC}^{\mathbf{HbC}}_{\mathsf{hyb}}$ to tolerate any malicious behavior of users by actually implementing the verification algorithm with the help of zero-knowledge proofs for composite statements [2]. The resulting construction is called $\mathsf{RoC}^{\mathbf{M}}_{\mathsf{hyb}}$.

NIZK proof system in $\mathsf{RoC}^{\mathbf{M}}_{\mathsf{hyb}}$ To achieve integrity in $\mathsf{RoC}^{\mathbf{M}}_{\mathsf{hyb}}$, one has to prove statements composed of both algebraic and arithmetic parts by means of non-interactive zero-knowledge proofs. Efficient zero-knowledge proofs are well known for algebraic structures [114, 41] and there are recent practical constructions for arithmetic circuits [88, 71]. Moreover, there exist efficient zero-knowledge proofs for composed statements [38, 25, 2, P4]. To date, zero-knowledge proofs by Agrawal et al. [2] are the most verifier efficient and compact, but require a trusted setup assumption.

Using zero-knowledge proofs for composite statements, we construct a proof system for the hybrid construction. To initialize or extend a ciphertext chain that ends with ciphertext c, a user computes c' and proves that c' was correctly computed according to the NIZK system. The verify algorithm takes this pair of ciphertexts (c, c') and returns a boolean value. We distinguish the following two cases.

Case 1 (the change operation). Recall that the recipient of ciphertext $(\alpha, \beta, \gamma, \delta)$, say U_i, can decrypt array α and component δ. It suffices for the prover to prove the knowledge of the decryption key sk for δ. We show the proof system for a pair of ciphertexts $(\alpha, \beta, \gamma, \delta)$ and $(\alpha', \beta', \gamma', \delta')$:

$$PoK\{sk|\ \delta = \mathsf{e}_i(1)\} \ \wedge\ P\{\exists\ r_\delta|\ \delta' = \mathsf{re}(\delta; r_\delta)\}$$

The component δ' prevents the recipient from changing the associate public key in $(\alpha', \beta', \gamma', \delta')$. To allow the change of the public key, in any-key-RoC version of $\mathsf{RoC}^{\mathbf{M}}_{\mathsf{hyb}}$ we need to remove this component from the proof system.

Case 2 (the randomize operation). We have to bind the algebraic and arithmetic expressions, which can be done by means of commitment schemes. We expect that commitment scheme Π_{Com} used in the proof system allows for homomorphic operations and is instantiated via an encryption scheme. In the description of the proof system, we will use Φ to denote the commit operation of Π_{Com}. Thanks to the homomorphic properties, from commitments to individual bits $\Phi(x_0)\ldots\Phi(x_{\ell-1})$ one can publicly construct a new commitment $\Phi(x)$, where $x = x_{\ell-1}||\ldots||x_0$ is a concatenation of individual bits. We will denote by $\Phi(x_0)||_{hom}...||_{hom}\Phi(x_{\ell-1})$ computing $\Phi(x)$ from commitments to the individual bits. Finally, Π_{Com} should allow re-randomizations.

In the following, for convenience we will omit the randomness used for the proof of re-randomization and proof of correct commitment Φ. We distinguish three instances of the commit operation: Φ for committing the individual bits, Φ_{pk_i} for committing using the public key pk_i, and Φ_δ for committing using encryption of the identity δ. Let $f(s||s') = \mathcal{G}(s) \oplus \mathcal{G}(s')$. The proof system for $(\alpha, \beta, \gamma, \delta)$ and

$(\alpha', \beta', \gamma', \delta')$ looks as follows:

$$\begin{aligned} P\{f(\chi) = \gamma' \oplus \gamma \quad \wedge \quad \Phi(\chi) = (\Phi(s)||_{hom}\Phi(s')) \quad \wedge \quad \delta' = \mathsf{re}(\delta) \\ \wedge \vee_{i\in[1,N]} [(\beta_i, \alpha'_i, \beta'_i) = (\Phi_{pk_i}(s), \Phi_\delta(s'), \Phi_{pk_i}(s')) \\ \wedge_{j\in[1,N]\setminus i} [(\alpha'_j, \beta'_j) = (\mathsf{re}(\alpha_j), \mathsf{re}(\beta_j))]]\} \end{aligned}$$

The first line of the proof system essentially enforces that $\gamma' = \gamma \oplus \mathcal{G}(s) \oplus \mathcal{G}(s')$ for some seeds (s, s') and that δ was correctly re-randomized. The second and the third lines ensure that for at least one index i the seeds s, s' are bound to $(\beta_i, \alpha'_i, \beta'_i)$ via commitments (re-randomizer does not use α_i at all), and that for all remaining indices $j \neq i$, (α_j, β_j) are correctly re-randomized to (α'_j, β'_j) (third line). The resulting proof is an OR-proof of the two cases. Let Π_{Hy} denote the corresponding proof system.

Construction 3.5.2 ($\mathsf{RoC}^{\mathbf{M}}_{\mathsf{hyb}}$)**.** *Implement in* $\mathsf{RoC}^{\mathbf{HbC}}_{\mathsf{hyb}}$ *(Construction 3.5.1) the verification algorithm and update the change and re-randomization algorithms appropriately using NIZK proof system* Π_{Hy} *defined in this section.*

Lemma 3.5.5. $\mathsf{RoC}^{\mathbf{M}}_{\mathsf{hyb}}$ *with assumptions defined in Theorem 3.5.1 and appropriate non-interactive zero-knowledge proofs, as defined in Section 3.5.4, provides integrity of plaintext.*

Proof Sketch. Assume to the contrary that a PPT adversary Adv wins integrity of plaintext game with a non-negligible probability. Since Π_{Hy} is a zero-knowledge proof system, there is a knowledge extractor for sk. We use the fact that semantically secure $\mathcal{E}$ is instantiated with ElGamal encryption, which requires DDH assumption. We use Adv to construct a distinguisher D of DH tuples and random tuples piggybacked into $\mathcal{E}$ encryption algorithm. The success probability of D is non-negligible, which contradicts the DDH assumption. $\square$

3.6 Parallel Anonymous RAM

In this section, we define parallel anonymous RAM schemes and present our construction thereof, which makes use of aggregatable RoC.

3.6.1 Definition

We extend the definition of anonymous RAM (Section 2.2) to support concurrent requests.

Definition 3.6.1 (Parallel Anonymous RAM)**.** *Parallel Anonymous RAM is a tuple of algorithms* $(\mathsf{Setup}, \mathsf{Server}, \mathsf{User}_1, ..., \mathsf{User}_\ell)$*, where:*

- *The* initialization algorithm Setup *maps a security parameter* 1^λ *and an identifier id, to an initial state, where id* $\in \{0, 1, ..., \eta\}$ *identifies one of the servers (for id* > 0*) or the user (for id* $= 0$*).*
- *The user algorithm* User *processes two kinds of inputs: (a) access requests (from the user) and (b) pairs* (l, m) *where* $l \in [1, \eta]$ *denotes a server and* m *a message from server* Server_l *. User maps the current state and input to a new state and to either a response provided to the user or a pair* (l, m) *with* $l \in [1, \eta]$ *denoting a server and* m *being a message for* Server_l *.*
- *The server algorithm* Server_l *for server* Server_l *maps the current server state and input (*$k \leq \ell$ *messages from* k *users or from another server) to a new server state and a message either to* k *users or to another server.*

Definition 3.6.2 (Access requests)**.** *An access request AR is a tuple* $(i, j, \alpha, m) \in [1, M] \times \{\mathsf{Read}, \mathsf{Write}\} \times \Sigma$*. Here* $i \in [1, N]$ *is a user identifier,* j *is called the (cell) index of AR,* α *the access type, and* m *the input message.*

Definition 3.6.3 (Access pattern)**.** *An access pattern is a series of tuples* $(AR_1, ..., AR_k)$ *where* AR_i *is an access request and all access requests in a tuple belong to different users.*

We define the protocol execution in two different adversarial models: (i) honest-but-curious HbC adversaries that learn the state of one server S^* and a subset $\mathcal{U}^*$ of users, and (ii) malicious users $\mathsf{Mal\_Users}$ that learn the state of one of the server (as before) and control a subset $\mathcal{U}^*$ of users. In both models, the adversary can eavesdrop on all messages sent on the network.

Definition 3.6.4 (Execution)**.** *Let* $\mathcal{PAR}$ *be a Parallel Anonymous RAM scheme* $(\mathsf{Setup}, \mathsf{User}, \mathsf{Server}_1, \ldots, \mathsf{Server}_\eta)$*,* Adv *be a PPT algorithm,* $\zeta \in \{\mathsf{HbC}, \mathsf{Mal\_Users}\}$ *and AP be an access pattern. The* execution $\mathsf{Exec}(\mathcal{PAR}, \mathsf{Adv}, AP, \zeta)$ *is the following randomized process:*

1. All parties are initialized using Setup*, resulting in initial states* σ_{U_i} *for each user* U_i*, and* σ_{S_l} *for each server* S_l*.*

2. Adv *selects a server* S^* *and a strict subset* $\mathcal{U}^* \subset \mathcal{U}$*.*

3. Let $(i, j, \alpha, m_{i,j})$ *be the first element of the first tuple of AP; if AP is empty, terminate.*

4. If $\mathsf{U}_i \in \mathcal{U}^*$ *and* $\zeta = \mathsf{Mal\_Users}$*, then let* (l, m_i) *be the output of* Adv *on input* $(i, j, \alpha, m_{i,j})$*. Otherwise, let* (l, m_i) *be the output of* User *on input* $(j, \alpha, m_{i,j})$*, with state* σ_{U_i}*, and update* σ_{U_i} *accordingly.*

5. Repeat the loop (from step 3) with the next element of the tuple (until empty).

6. Invoke S_l *with (input) messages* (l, m_i)*. The server* S_l *may call other servers (possibly recursively) and finally produces (output) messages* m'_i*.*

7. For each m'_i*, if* $\mathsf{U}_i \in \mathcal{U}^*$ *and* $\zeta = \mathsf{Mal\_Users}$*, provide the message* m' *to* Adv*. Otherwise, provide* m'_i *to user* U_i*.* U_i *(*Adv *if* $\mathsf{U}_i \in \mathcal{U}^*$ *and* $\zeta = \mathsf{Mal\_Users}$*) may repeat sending messages to any servers. Eventually,* U_i *(*Adv*) terminates.*

8. Repeat the loop (from step 3) with the next tuple of AP (until empty).

Throughout the execution, the adversary learns the internal states of S^* *and of all users in* $\mathcal{U}^*$*, as well as all messages sent on the network.*

A trace *is the random variable defined by an execution, using uniformly random coin-tosses for all parties. The trace includes the sequence of messages in the execution corresponding to access requests and the final state of the adversary. Let* $\Theta(x)$ *denote the trace of execution* x*.*

We call an adversary Adv compliant with a pair of access patterns (AP_0, AP_1) if Adv only outputs sets $\mathcal{U}^*$ of users in Step 2) of $\mathsf{Exec}(\mathcal{PAR}, \mathsf{Adv}, AP_0, \zeta)$ and $\mathsf{Exec}(\mathcal{PAR}, \mathsf{Adv}, AP_1, \zeta)$ such that AP_0 and AP_1 are identical when restricted to users in $\mathcal{U}^*$.

Definition 3.6.5 (Privacy of Parallel Anonymous RAM)**.** *A parallel anonymous RAM scheme* $\mathcal{PAR}$ *preserves* privacy *in adversarial model* $\zeta \in \{\mathsf{HbC}, \mathsf{Mal\_Users}\}$*; if for every pair of (same finite length) access patterns* (AP_0, AP_1) *and for every pair of PPT algorithms* $(\mathsf{Adv}, \mathcal{D})$ *s.t.* Adv *is compliant with* (AP_0, AP_1)*, we have that*

$$\left|\mathsf{Pr}\left[b^* = b \colon b^* \leftarrow \mathcal{D}\left(\Theta\left(\mathsf{Exec}(\mathcal{PAR}, \mathsf{Adv}, AP_b, \zeta)\right)\right)\right] - \frac{1}{2}\right|$$

is negligible in 1^λ*, where the probability is taken over uniform coin tosses by all parties, and* $b \leftarrow_R \{0,1\}$*.*

Parallel anonymous RAM should ensure *integrity* to prevent invalid executions caused by parties deviating from the protocol. Informally, a trace is invalid if a value read from a cell does not correspond to the most recently written value to the cell.

Definition 3.6.6 (Integrity of Parallel Anonymous RAM)**.** *Let* ϑ *be a trace of execution with access pattern AP, and let* $AR = (j, \mathsf{Read}, *)$ *with* $(i, AR_i) \in AP$ *be a read request for cell* j *of user* U_i*, returning a value* x*. Let* $AR' = (i, j, \mathsf{Write}, x')$ *be the most recent previous write request to cell* j *of user* U_i *in AP, or* $\perp$ *if there was no such previous write request. If* $x \neq x'$*, we say that this read request is* invalid*. If any read request in the trace is invalid, then the trace is invalid.*

A parallel anonymous RAM scheme $\mathcal{PAR}$ *preserves* integrity *if there is negligible (in* 1^λ*) probability of invalid traces when the traces are constrained to the view of the honest users (all* $\mathsf{U}_i \in \mathcal{U}$ *in the* HbC *model, and all users* $\mathsf{U}_i \in \mathcal{U}/\mathcal{U}^*$ *in the* $\mathsf{Mal\_Users}$ *model) for any PPT adversary and any access pattern AP.*

3.6.2 Construction

Our parallel anonymous RAM construction from ARoC-sk, $\mathsf{ParAnonRAM}$ is based on the linear AnonRAM construction $\mathsf{AnonRAM}_{\mathsf{lin}}$ (Section 2.3). The main difference is that the users may send simultaneous requests to the server and the server now has a cryptographic tool to handle simultaneous requests, which is ARoC-sk.

We now present ParAnonRAM. Similar to AnonRAM$_{\mathsf{lin}}$, ParAnonRAM uses an anonymous communication channel [124] and the (single-user, single-server) Path ORAM [121] or other ORAM scheme satisfying *indistinguishability of individual accesses.* The property requires that the adversary observing just one access request from an access pattern should not be able to recognise how many accesses the honest user performed so far. We will use Path ORAM in the description of our scheme. We refer to Section 2.3.2 for a high-level description of Path ORAM.

ParAnonRAM employs N instances (one per user) of Path ORAM for M cells each while requiring a single server. [2] To encrypt data as required in the ORAM scheme, ParAnonRAM uses ARoC-sk (Section 3.4). The ParAnonRAM client of user U_i, has access to her private key sk_i. In ParAnonRAM, the ORAM scheme uses our ARoC-sk instead of the (symmetric) encryption scheme used in 'regular' Path ORAM.

Intuitively, an ParAnonRAM client internally runs an ORAM client and mediates its communication with the server. Whenever the ORAM client reads or writes a specific block, the ParAnonRAM client performs corresponding read or write operations *for all users*, without divulging the user identity to the server at the network level, as follows: Reading a block belonging to other user's ORAM can be trivially achieved, since the contents are not used. Writing a block belonging to other user's ORAM must not corrupt the data inside and is hence achieved by re-randomizing the blocks of other users.

The Setup and Server algorithms of ParAnonRAM are simply N instances of the corresponding algorithm of the underlying ORAM scheme (e.g., Path ORAM). Namely, the Setup initializes state for N copies of the ORAM (one per user) and the Server receives a 'user identifier' i together with each request, and runs the ORAM's Server algorithm using the i^{th} state over the request. The Server algorithm for the ParAnonRAM scheme simply processes Read/Write requests sent by the users as in the ORAM scheme, e.g. the server returns the content of the requested block for Read requests or overrides the content of the requested block with the new value for Write requests.

Since we allow simultaneous requests from the users, it may happen that two or more users ask the server to update the same block with different values. In this case, the server uses the aggregate functionality of ARoC-sk and computes the aggregated value which is then written to the cell.

Theorem 3.6.1. ParAnonRAM *preserves preserves integrity and privacy in the adversarial model* Mal_Users*, when using a secure ORAM scheme that satisfies indistinguishability of individual accesses, and a secure ARoC-sk.*

Proof. The proof is essentially similar to the proof of the linear anonymous RAM construction AnonRAM$_{\mathsf{lin}}$, with a few differences noted below. According to the

[2] ParAnonRAM can also use an ORAM scheme that uses multiple servers. In this case, ParAnonRAM will use the same number of servers.

defintion of parallel anonymous RAM, concurrent requests may come only from different users. Hence in each tuple of access requests, there is at most one owner who may update the block; all others are re-randomizers of that block. This observation matches the requirement for the correct aggregation.

The order in which the write requests arrive to the server does not change the aggregated value by the server, since a secure ARoC has unordered aggregation by definition.

Integrity is provided by integrity of plaintext property of ARoC, which prevents any user who does not know the correcponding secret key to break integrity of the input ciphertext. □

3.7 Group payment system

We elaborate on a simple privacy-preserving group payment system using any-key-RoC. Here, we consider a scenario, in which a small group of users would like to manage micro-transactions among themselves. There are N users $(\mathsf{U}_i)_{i\in[1,N]}$ and a server. Each user has a pair of public and secret keys and a fixed number of encrypted payment units, X. We say that a unit belongs to U_i if it is encrypted for U_i. In total, $N \cdot X$ units are stored at the server, encrypted using any-key-RoC. For simplicity, we assume that all units are of equal value. The server is honest-but-curious, while users can be malicious (we assume that, in particular, not all users know each others personally). The protocol is round-based. In each round, one of the users reads the whole storage and updates its each element. User U_i can send a specific number of units to user U_j by encrypting some of her tokens for U_j, and all remaining units are re-randomized. The server verifies each pair of ciphertexts sent by the user and rejects if there is a pair that does not verify. If accepted, the server updates the storage with the units sent by the user and reshuffles them.

We informally state the security properties of the described system. It is easy to see that each user can only send her own units to another user, and once units are sent the user loses control over them until someone sends them back. In the system no new units are generated, and a user cannot learn the details of transactions which she is not involved in. On the other hand, the server does not learn anything about transactions, since each round looks the same: a user updates all ciphertexts in the storage, and RoC hides content and association with user's public keys. Moreover, users cannot trace coins that they have sent, since the server reshuffles units in each round.

Complexity cost for each round is high: the user has to update $N \cdot X$ ciphertexts, although not in every round. In each round, only one user communicates with the server. Depending on the expected rate of micro-transactions, the group of users can adjust the time length of a round to achieve an appropriate latency.

ARoC-ak instead of RoC-ak The protocol above is instantiated with the help of RoC-ak and can process one user at a time. Using ARoC-ak, we can handle multiple users' requests simultaneously, thereby reducing waiting time. Our ARoC-ak construction requires one round of interaction (synchronization between active users sending transactions), which is handled by the server. In Fig. 3.5, we show a high-level example of our micro payment system based on ARoC-ak.

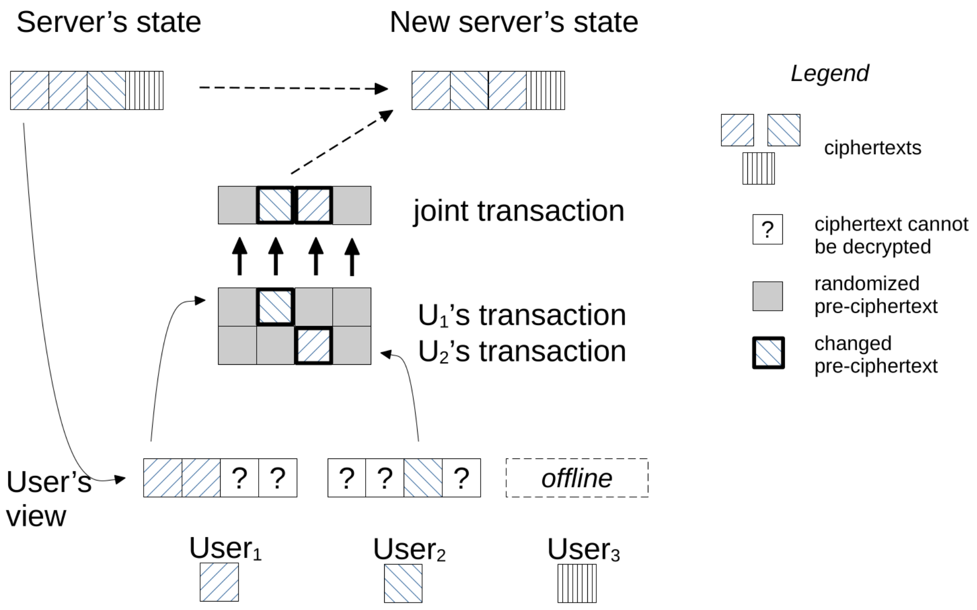

Figure 3.5: Group payment system from ARoC-ak. Example for 3 users, two of them sending transactions to the server via anonymous channels. Server's state consists of 4 units that belong to 3 users. User U_1 owns two units and sends one unit to U_2. User U_2 owns one unit and sends it to U_1. For each unit, the server aggregates a new unit value, forming a new server's state.

3.8 Related Work

Golle et al. [77] introduced universal re-encryption, a primitive that extends the standard CPA-secure ElGamal encryption scheme by adding a re-encrypt (re-randomize) operation that requires no decryption and no receiver's public key. Golle et al. proposed to use universal re-encryption in *universal* mixnets, which replace *re-encryption mixnets* used in voting protocols such as Helios [1] and Zeus [125]. Re-encryption mixnets require the mixing servers to keep public keys of the senders, in order to be able to re-randomize the ciphertexts [106, 101], and with the help of universal re-encryption, the mixing servers no longer require to hold public keys of the senders, as they can re-encrypt ciphertexts without any additional information beyond the actual ciphertexts.

Another application of universal re-encryption is the anonymization of radio-frequency identification (RFID) tags by universal mixnets. Golle et al. [77] proposed to use universal re-encryption in order to prevent tracking capabilities: universal mixnets re-randomizes a collection of RFID tags so that the original and re-randomized tags cannot be linked to each other. Unfortunately, an active adversary may still track an object with an RFID tag. He can try to mark the object by replacing the RFID tag with a ciphertext of choice, which he can decrypt, and universal mixnets will keep re-randomizing it without altering the plaintext. To address this kind of attack, Ateniese et al. [6] defined *insubvertible encryption* and proposed to use it in RFID tags. In insubvertible encryption, the rerandomizer can check whether a given ciphertext is encrypted under a pre-registered public key. If the check fails, the rerandomizer will set the RFID tag to a dummy "safe" value instead of actually re-randomizing it, thus limiting tracking capabilities of an active adversary. We note that, although insubvertible encryption [6] limits tracking capabilities of an active adversary, who tries to replace a ciphertext c with another ciphertext c' (in the RFID scenario), the primitive allows to check legitimacy of a single ciphertext and therefore does allow to replace a legitimate ciphertext of one user with another legitimate ciphertext of another user.

Fairbrother [59] improved space efficiency of universal re-encryption [77]. Klonowski et al. [92] proposed an extension to the original universal re-encryption, called universal re-encryption of signatures. In their construction, an RSA signature is attached to the ciphertext. Both the message and the signature can be then re-randomized by any party, while the signature remains valid during this transformation. However, as shown by Danezis [52], this construction is vulnerable to existential forgery. Young and Yung [133] define several security properties for UREnc: *key anonymity* (for the encryption algorithm, and for the re-randomize algorithm) and *message indistinguishability* (along with the classical property for the encryption algorithm, one for the re-randomize algorithm), and show that the original construction by Golle et al. [77] is secure under their revisited security definition. Universal re-encryption should not be confused with proxy re-encryption, where the idea of the latter is to transform a ciphertext under specific public key to a ciphertext under another public key [7, 32].

In the following line of works, CCA-like definitions have been researched for encryption schemes that allow re-randomization of ciphertexts. Canetti et al. [33] introduced a relaxation of the stardard CCA security notion called RCCA (R stands for Replayable), which captures scenarios that an attacker can generate different ciphertexts decrypting to the same plaintext. They called an encryption scheme *randomizable* if it allows anyone to convert a ciphertext c to another ciphertext c' that decrypts to the same plaintext, and require that c and c' should not be linkable in any way. Groth [82] presented a rerandomizable cryptosystem that achieved a weaker form of RCCA (WRCCA), and another construction that achieved RCCA in the generic groups model. Prabhakaran and Rosulek [109] presented a rerandomizable RCCA-secure cryptosystem, which extends the Cramer-Shoup public key cryptosystem. They also define RCCA *receiver-anonymity* and state that their construction does not achieve it and that it is an open problem.

In the following works, encryption schemes have some form of verifiability. Zheng [137] introduced *signcryption*, a public key scheme that combines simultaneously encryption and signing (the respective encryption/decryption algorithms are usually called signcrypt/unsigncrypt). The primitive improves on efficiency as compared to encrypt-then-sign/sign-them-encrypt schemes. Signcryption provides condifentiality and authenticity in the public-key setting. The symmetric-key analogue for signcryption is called *authenticated encryption*, introduced by Bellare et al. [12]. Our RoC extends the notion of authenticated encryption by adding *re-randomization* operation to the scheme definition. The difference is, however, in the way the verification proceeds: while in authenticated encryption authenticity of a single ciphertext can be verified by the recipient, in RoC it is done universally (i.e., by any party) for a pair of ciphertexts.

Malleable non-interactive zero-knowledge proofs, proposed by Chase et al. [39] allow anyone to produce new valid proofs of language membership based on some valid proofs using a controllable transformation. A proof created in such a way should be indistinguishable from a proof created from scratch using some witness. As we already discussed, verifiability in RoC is by design captured for a *pair* of ciphertexts.

3.9 Postponed proofs

Proof of Theorem 3.3.1. We show that an adversary Adv that breaks IND-OP can also break IND-CPA-Ch. To show this, we construct an algorithm $\mathcal{D}$ that given an oracle access to Adv can break IND-CPA-Ch. Let $\mathcal{O}_1$ be an oracle that responds IND-OP queries from Definition 3.3.2, i.e., given (c, m), it outputs either $\mathsf{R}(c)$ or $\mathsf{Ch}_{sk}(m, c)$ depending on the value of a random bit b_1. Let $\mathcal{O}_2$ be an oracle that responds IND-CPA-Ch queries, i.e. upon (m_0, m_1) and c' as input, it outputs $\mathsf{Ch}_{sk}(m_{b_2}, c')$ for a random bit b_2.

$\mathcal{D}$ simulates $\mathcal{O}_1$ as follows. Upon receiving a query (c, m), $\mathcal{D}$ decrypts c using sk to obtain m'. If $m' \neq m$, $\mathcal{D}$ sends (m', m) and c to $\mathcal{O}_2$, and upon receiving a response c_{b_2} from $\mathcal{O}_2$, $\mathcal{D}$ redirects the ciphertext to Adv. Otherwise, i.e. if $m' = m$, $\mathcal{D}$ computes and sends $\mathsf{Ch}_{sk}(m', c)$ to Adv. Finally, $\mathcal{D}$ receives a bit b' from Adv and outputs it as the result of the experiment.

In the case $m' = m$ it is easy to see that $\mathcal{D}$ and $\mathcal{O}_1$ produce identical distributions and that $Pr[\mathrm{RoC}^{\text{IND-OP}}_{\mathsf{Adv},\Pi}(\lambda) = 1] = 1/2$ because of identical distributions of re-randomization and encryption of the same plaintext. Here we use the fact that the distributions of $\mathsf{R}(c)$ and $\mathsf{Ch}_{sk}(m, c)$, for some m and c such that $\mathsf{D}(c) = m$, are identical.

Otherwise, $Pr[\mathrm{RoC}^{\text{IND-CPA-Ch}}_{\mathsf{Adv},\Pi}(\lambda) = 1] = Pr[\mathrm{RoC}^{\text{IND-OP}}_{\mathsf{Adv},\Pi}(\lambda) = 1]$, because of aforementioned property if $b_1 = 0$ and the same output distribution if $b_1 = 1$. □

Lemma 3.9.1. *Encryption algorithm, re-randomization algorithm, and the change algorithm of* $\mathsf{RoC}^{\mathbf{HbC}}_{\mathsf{dlog}}$ *as defined in Construction 3.3.1 in the groups* G_q *where DDH is hard provide message indistinguishability.*

Proof. For message indistinguishability of E and of R, we can directly use the proof from [133], since the encryption algorithm E and the re-randomization algorithm R do not use the public keys of other users. The proof relies on the hardness of decisional Diffie–Hellman (DDH) assumption [16] for group G_q.

Observing that the change algorithm Ch in fact does not use the input ciphertext c and that a call to Ch can be replaced with a call to E by plugging in $pk = g^{sk}$, IND-CPA-Ch security of $\mathsf{RoC}^{\mathbf{HbC}}_{\mathsf{dlog}}$ follows. □

Lemma 3.9.2. $\mathsf{RoC}^{\mathbf{HbC}}_{\mathsf{dlog}}$ *as defined in Construction 3.3.1 in the groups* G_q *where DDH is hard is IND-OP secure.*

Lemma 3.9.3. *Encryption algorithm, re-randomization algorithm, and the change algorithm of* $\mathsf{RoC}^{\mathbf{HbC}}_{\mathsf{dlog}}$ *as defined in Construction 3.3.1 in the groups* G_q *where DDH is hard provide key anonymity.*

Proof. Since E and R algorithms do not use the public keys of other users, anonymity of E and R follows from the proof of key anonymity in [133]. For the same reasons, mentioned in the proof of Lemma 3.9.1, Ch provides key anonymity. □

Lemma 3.9.4. $\mathsf{RoC}^{\mathbf{HbC}}_{\mathsf{dlog}}$ *as defined in Construction 3.3.1 provides external and internal anonymity of re-randomization.*

Proof. Immediately follows from the fact, that the re-randomizer does not use any public or private keys. □

Proof of Theorem 3.3.2. Follows from Lemmas 3.9.1 to 3.9.4. □

Proof of Lemma 3.3.1. Assume to the contrary that an adversary Adv wins integrity of plaintext game with a non-negligible probability. Since Π_{DL} is a zero-knowledge proof system with the proof of knowledge part, there is a knowledge extractor for the secret key sk. Let G_0 is the INT-PTXT game. Let G_1 be a modified game, for which the key generation is replaced with $pk \leftarrow X$, where X is a group element. The games are indistinguishable for the adversary. We use Adv to solve the discrete log problem for X by extracting the secret with non-negligible probability. This contradicts the assumption that solving the discrete log problem is hard. □

3.10 Conclusion

Universally re-randomizable encryption (UREnc) schemes allow any user to re-randomize ciphertexts without knowing the associated public key. They found its application in mixnets, protecting privacy of radio-frequency identification (RFID) tags. However, a malicious user, instead of re-randomizing, may corrupt data or even encrypt a chosen plaintext. In this work, we define randomize-or-change encryption (RoC) schemes which allow the recipient, in addition to re-randomization, to encrypt a new value ('change'). RoC provides indistinguishability of operation and protects the integrity of plaintext property against malicious re-randomizers.

We present two families of constructions, one in the discrete log (dlog) setting and another in the hybrid setting, which differ in their design and complexity. We expect the hybrid constructions perform better when the number of users is small and message space is large, while the dlog based constructions provide an extra property (anonymity of re-randomization). Furthermore, we devise the aggregation functionality in the dlog setting. The integrity of plaintext property binds a pair of ciphertexts and comes at a cost of sacrificing some anonymity properties, which is an unavoidable design side-effect in this case.

We propose several applications for RoC: anonymous communication, group payment system, and parallel anonymous RAM. It will be interesting to see how the ideas in this work can be extended to constructing *verifiable* proxy-reencryption with possibility of universal re-randomization. We leave this direction for future work.

4
Non-interactive Zero-Knowledge Proofs in Cross-Domains

4.1 Introduction

Zero-knowledge (ZK) proofs, introduced by Goldwasser, Micali, and Rackoff [76], are one of the central cryptographic building blocks, which allow a prover to convince a verifier that a statement is true without revealing any other information. Goldreich, Micali, and Wigderson showed that ZK proofs for NP-languages are possible [73], which opened up a number of new research directions in cryptography.

Zero-knowledge proof systems are an essential building block used in many privacy-preserving systems, e.g. anonymous credential systems [26] and voting protocols [83, 10]. Unfortunately, only a few systems have been used in practice. The main reason is that ZK proofs for general statements are usually inefficient. Thus, the research focus switched from general statements to interesting subclasses. In particular, a prover can efficiently prove knowledge of discrete logarithms in groups of known [42, 114] and unknown [29, 9] order. Those proofs were extended to allow other statements, e.g., equivalence of discrete logarithms, or knowledge of representation. The main advantage was that using the Fiat-Shamir transformation [61] one can make those systems non-interactive (NIZK), i.e. no interaction between the prover and the verifier is necessary to generate the proof, and transform honest-verifier ZK protocols into full ZK. Groth and Sahai [85, 58] further extended the class of efficient NIZK proofs to statements about pairing product equations. The common factor of those proofs is that they are restricted to algebraic groups and cannot be efficiently used to prove statements about non-algebraic structures, e.g., the SHA hash function or the AES encryption scheme.

The problem of efficient interactive ZK proofs for non-algebraic statements was solved by Jawurek et al. [88]. Their system allows to efficiently prove statements of the following form: "The prover knows an input x such that $y = SHA(x)$ for some public y". Unfortunately, one cannot apply the Fiat-Shamir transformation to make those proofs non-interactive. The system is based on garbled circuits [132], which are private coin protocols, which in turn makes the system inherently interactive. Giacomelli et al. [71] addressed this limitation and introduced ZKBoo, a non-interactive proof system for arithmetic circuits, based on the "MPC-in-the-head" technique [86]. In their system, the proof size depends linearly on the number of gates, input and output wires. This work was further improved by Chase et al. [37] with the introduction of the ZKB++ system. The authors were able to reduce the proof size by a constant factor and addressed post-quantum security of the construction. Ames et al. [4] proposed Ligero, a NIZK proof system based on the "MPC-in-the-head" technique, which has the proof size proportional to the square root of the verification circuit size.

An interesting line of research present succinct non-interactive zero-knowledge proofs (SNARKs) [84, 66, 13]. They allow compact proofs and very efficient verification, but require a complex trusted setup and the prover has to perform a number of public key operations (i.e. modular exponentiations or equivalently elliptic curve point multiplications) proportional to the circuit size. The setup algorithm can be executed by a trusted party or by the participants of the system using multi-party computation (MPC).

While there exist efficient proofs for algebraic and non-algebraic statements, it became a natural challenge to combine both worlds and create a proof system that would work efficiently in both, algebraic and non-algebraic, domains. Obviously, one can implement algebraic structure directly using non-algebraic statements by defining all group operations as functions. This approach introduces a significant overhead in size of the proven statement, which increases the size of the proof and the number of required computations. As noted by [2], depending on the group size, the circuit for computing a single exponentiation could be *thousands or millions* of gates. Alternatively, one can implement arithmetic circuit directly using algebraic statements by treating each gate in a circuit as an algebraic function and proving relations between gates. The prover's/verifier's work and the proof size would be linear in the number of gates, and in case of hash functions or block ciphers it could be *tens of thousands* of public key operations and group elements.

The first attempt to efficiently solve this cross-domain problem was the Crypto'16 work by Chase et al. [38]. Their system can be used e.g. to prove that a given algebraic commitment (e.g. Pedersen commitment) C is a commitment to x, where $F(x) = 1$ and F is expressed as a boolean circuit. The authors show that an efficient proof system for this statement can be used as a building block to construct more efficient proofs of knowledge of a signature and a committed message for RSA-FDH, DSA and EC-DSA signatures. Their system can be extended to a scenario, where we have k commitments to $x_1, \ldots, x_k$ and the input x is the concatenation $x_1||\ldots||x_k$ of values in those commitments.

Chase et al. propose two constructions of their proof system. For the first the number of public key operations is linear in the size of x and that of symmetric key operations is proportional to the number F_g of gates in F. The second construction reduced the number of public key operations to a number linear in the security parameter λ, but this comes at a cost of additional symmetric key operations which are proportional to $F_g + |x| \cdot \lambda$. Unfortunately, their approach is based on the ZK proofs from [88] and the proof system is therefore interactive by nature.

Bünz et al. [25] presented at S&P'18 efficient NIZK range proofs called Bulletproofs. Those proofs can also be used for proving statements expressed as arithmetic circuits with algebraically committed inputs. The proof technique relies on discrete log assumptions and the Fiat-Shamir transformation. While Bulletproofs produce relatively short proofs, the prover's work is still expensive, specifically, the prover has to perform a number of public key operations linear in the circuit size.

At Crypto'18, Agrawal, Ganesh, and Mohassel [2] presented non-iteractive zero-knowledge proofs for composite statements. Whereas the authors addressed the same problem of constructing zero-knowledge proofs in cross-domains, theirs and our proposals differ in the underlying cryptographic blocks that handle the arithmetic part of the proof system. Specifically, their proofs are based on Σ-protocols and SNARKs [66]. As already noted, SNARKs allow for short proofs and fast verification of arithmetic statements, however they require a trusted setup for generating the common reference string (CRS) for a particular circuit F. Typically, the CRS needs to be regenerated for a different circuit F'. This is not desirable in some applications such as ZCash, where an expensive MPC protocol has to be run to generate a CRS [135].

Our contribution In this work, we present an efficient (both for the prover and the verifier) non-interactive zero-knowledge proof system for cross-domains that requires no trusted setup assumption. Our system uses ZKB++ [37] as a building block, which is based on a technique called "MPC-in-the-head" [86]. The idea is that the prover represents the circuit F as a multi-party computation (MPC) and generates three shares $x_1 \oplus x_2 \oplus x_3 = x$, where x is the original input of the prover. The prover then performs the MPC computation using the values x_1, x_2, x_3 and given a challenge $e \in \{1, 2, 3\}$ returns the view of computations performed with inputs x_e and x_{e+1}. Executing these steps a number of times decreases the soundness error of the proof. What is more, we can apply the Fiat-Shamir transformation and allow the prover to compute this challenge itself, making the system non-interactive.

We extend this idea to allow algebraic statements. To illustrate our solution let us consider a simple example where the prover publishes $y = SHA(x)$ and a Pedersen commitment $C = g^x \cdot h^r$. In this case, the prover wants to convince the verifier that he knows x. To do so, he performs the "MPC-in-the-head" as in ZKB++ and computes Pedersen commitments to all bits of the values x_1, x_2, x_3. Upon receiving a challenge $e \in \{1, 2, 3\}$, additionally to the views of the MPC the prover opens all commitments to the bits of x_e and x_{e+1}. Finally, to bind the "MPC-in-the-head" part to the Pedersen commitment C, the prover computes commitments to the bits of the value $x_e \oplus x_{e+1} \oplus x_{e+2}$ and proves that these commitments contain the binary representation of the same value that is in C. As in ZKB++, this extended system has to be executed several times in order to decrease the probability of the prover cheating the verifier. However, in contrast to [38] we can apply the Fiat-Shamir transformation to get a NIZK system.

The number of public key operations in our system is proportional to $|x| \cdot \lambda$. This follows directly from the way we combine both domains. Each round we have to prove that the commitments to the bits of $x_e \oplus x_{e+1} \oplus x_{e+2}$ are the binary representation of x. We solve this obstacle by committing to full values of the ZKB++ share and not to its bits and show that we can still compute the XOR value of them because 2 out of 3 values are revealed by the ZKB++ protocol.

This unique technique allows us to further decrease the number of public key operations to $O(|x| + \lambda)$.

The contribution of this work can be summarized as follows. We are the first to present an efficient (both for the prover and the verifier) non-interactive zero-knowledge (NIZK) proof system for algebraic and non-algebraic domains (cross-domains) that requires no trusted setup. The solution is based on a combination of ZKBoo [71, 37] with standard Schnorr based proofs [114, 30]. Using our techniques, we obtain the efficient non-interactive proof of knowledge (proof of possession) of DSA/RSA signatures, without revealing the signature itself.

Applications A straightforward application of zero-knowledge proofs in cross-domains are anonymous credentials. Chase et al. [38] observed that many existing credential systems [24, 26, 27, 11, 8] rely on signature schemes that are tailored in a specific way to provide the desired properties of the system. The user proves that he knows a value x and a signature under this value. Using zero-knowledge proofs in cross-domains allows to use standard signature schemes like RSA-FDH or DSA for which there exist no efficient proof system in the standard algebraic setting. In contrast to the system by Chase et al. our proofs are non-interactive, which means that they can be used to construct round-optimal anonymous credential systems. This implies that using our techniques, one can create concurrently secure systems based on RSA and DSA signatures.

Another application of NIZK in cross-domains, mentioned in [2], are proofs of solvency for Bitcoin exchanges. In this scenario, an exchange wants to prove to its customers that it is solvent, i.e. that it has enough Bitcoins to cover its liabilities. To this end, the exchange would need to prove the control over some Bitcoin addresses. A certain Bitcoin address is a 160-bit hash of a public ECDSA key [123]. The corresponding proof is a proof of knowledge x such that $H(g^x) = y$, where H is a hash-function such as SHA-256. Here, only y is public, and the exchange would like to keep its public key part g^x hidden, otherwise an adversary could track the movement of exchanges associated with its public key. Since the Bitcoin network does not require a trusted setup assumption, proofs of solvency based on the approach by [2] would require a trusted CRS generation to be done. On the other hand, since our techniques do not require any trusted setup assumption, they can be used directly to prove solvency for Bitcoin exchanges. The proof system would additionally include a proof of equality of discrete logarithm of a committed value and another committed value. More specifically, a prover would need to prove knowledge of x such that $H(g^x) = y$ for some public y. Here the input to the circuit H is g^x. The prover has to commit to g^x as Com_{g^x} and to x as Com_x and use the proof of equality of discrete logarithm of a committed value and another committed value, for which we refer to [38].

Note that ours and the proof system by Chase et al. [38] or any other system cannot be post-quantum secure if the underlying security assumptions in the algebraic domain (integer factorization, discrete logarithms) can be broken by a quantum adversary [117].

Comparison with existing techniques We compare ours and prior work on zero-knowledge proofs in cross-domains in Table 4.1. We discuss the efficiency of the constructions based on a circuit F and a committed input x. For the algebraic part of the proof system, Σ-protocols are used in all ZK proof systems presented in the table. Σ-protocols require a constant number of public-key operations for a single algebraic statement and do not require any trusted setup assumption. The approach by Chase et al. [38] is the only interactive protocol in the table. In their first construction, the arithmetic part of the proof system is based on garbled circuits, whose prover's/verifier's cost amounts to $O(|F|)$ of symmetric-key operations. The number of public key operations is linear in the input size $|x|$. In the second construction, Chase et al. achieve the number of public key operations independent of $|x|$ at the cost of increasing the circuit that has to be garbled. Various techniques to reduce computation, communication, memory requirements of garbled circuits are available, e.g. [100, 134, 119]; in [93] XOR-gates can be garbled essentially at no cost. In Bulletproofs [25], the prover has to perform a constant number of public key operations for each multiplication gate of the circuit, while the verifier is more efficient due to the multi-exponentiation trick. The proof size in Bulletproofs is logarithmic in the number of multiplication gates in the arithmetic circuit for verifying the witness. The approach by Agrawal et al. [2], which is based on SNARKs, is the only protocol that requires a trusted setup assumption and produces constant proofs. Verifier's work does not depend on the circuit size, and the number of public key operations is linear in the input size. Prover's work requires a number of public key operations linear in the circuit size. We analyze efficiency of our Construction 4.3.2. As we show in Section 4.3.5, it requires $O(|x| + 1^\lambda)$ public key operations, while the number of symmetric-key operations is $O(|F| \cdot \lambda)$, since ZKB++ protocol has to be repeated to reduce the soundness error to a negligible value. Proof size is dominated by ZKB++ and amounts to $O((|F|\lambda + |x|)\lambda)$.

Chapter Outline The rest of the chapter is organized as follows. Section 4.2 contains preliminaries. In Section 4.3, we develop our solution for NIZK proofs in cross-domains. The section starts with the problem statement for NIZK proofs in cross-domains. Then, in Section 4.3.1 we present our first attempt cross-domain NIZK proof system based on ZKB++ followed by its security analysis. Next, in Section 4.3.2 we present an improved version and its security analysis in Section 4.3.3. Finally, in Section 4.3.4 we describe the optimization technique to reduce the number of public key operations and in Section 4.3.5 we perform efficiency analysis of our constructions. In Section 4.4, we complement NIZK

Table 4.1: Comparison of ZK proof systems in cross-domains for a circuit F with an algebraically committed input x, where $|F|$ denotes the circuit size, $|x|$ the number of input bits. We denote by λ the security parameter, by pub a public-key operation, by sym a symmetric-key operation.

	Non-interactive	Without trusted setup	Prover's work	Verifier's work	Communication/ Proof size
CGM16 [38] Constr.1	No	Yes	$O(\|x\|\cdot \text{pub}+\|F\|\cdot \text{sym})$	$O(\|x\|\cdot \text{pub}+\|F\|\cdot \text{sym})$	$O((\|F\|+\|x\|)\lambda)$
CGM16 [38] Constr.2	No	Yes	$O(\lambda\cdot \text{pub}+(\|F\|+\|x\|\lambda)\cdot \text{sym})$	$O(\lambda\cdot \text{pub}+(\|F\|+\|x\|\lambda)\cdot \text{sym})$	$O((\|F\|+\|x\|\lambda)\lambda)$
BBB+18 [25]	Yes	Yes	$O(\|F\|\cdot \text{pub})$	$O(\frac{\|F\|}{\log(\|F\|)}\cdot \text{pub})$	$O(\log(\|F\|)\lambda)$
AGM18 [2]	Yes	No	$O((\|F\|+\lambda)\cdot \text{pub})$	$O((\|x\|+\lambda)\cdot \text{pub})$	$O(\lambda)$
This work	Yes	Yes	$O((\|x\|+\lambda)\cdot \text{pub}+(\|F\|\cdot\lambda)\cdot \text{sym})$	$O((\|x\|+\lambda)\cdot \text{pub}+(\|F\|\cdot\lambda)\cdot \text{sym})$	$O((\|F\|\lambda+\|x\|)\lambda)$

proofs in cross-domains to allow OR-proofs. Section 4.5 concludes.

4.2 Preliminaries

In this section we recall the notions of commitment schemes, zero-knowledge and Σ-protocols. We also recall the details of the ZKBoo protocol introduced by Giacomelli et al. [71].

4.2.1 Homomorphic Commitment Schemes

Let us by $\mathcal{M}_{\mathsf{ck}}$ denote the message space of the commitment scheme and by $\mathcal{OI}_{\mathsf{ck}}$ the space of opening information (also called randomness).

Definition 4.2.1 (Commitment Scheme). *A commitment scheme consists of the following PPT algorithms* $(\mathsf{Gen}, \mathsf{Com}, \mathsf{Open})$*:*

- $\mathsf{Gen}(\lambda)$*: on input security parameter* λ*, this algorithm outputs a commitment key* ck*, which is an implicit input for the below algorithms.*
- $\mathsf{Com}(m, r)$*: on input message* m *and opening information* r*, this deterministic algorithm outputs a commitment* C_m*.*
- $\mathsf{Open}(C_m, m, r)$*: on input commitment* C_m*, message* m *and opening information* r*, this algorithm outputs a bit* $\{0, 1\}$*.*

Definition 4.2.2 (Perfect Hiding)**.** *A commitment scheme is perfectly hiding, if for all adversaries $\mathcal{A}$ we have:*

$$\Pr[\mathsf{ck} \leftarrow \mathsf{Gen}(\lambda), (m_0, m_1, \mathsf{st}) \leftarrow \mathcal{A}(\mathsf{ck}), r \overset{\$}{\leftarrow} \mathcal{OI}_{\mathsf{ck}}, C \leftarrow \mathsf{Com}(m_0, r) : \mathcal{A}(\mathsf{st}, C) = 1] =$$
$$\Pr[\mathsf{ck} \leftarrow \mathsf{Gen}(\lambda), (m_0, m_1, \mathsf{st}) \leftarrow \mathcal{A}(\mathsf{ck}), r \overset{\$}{\leftarrow} \mathcal{OI}_{\mathsf{ck}}, C \leftarrow \mathsf{Com}(m_1, r) : \mathcal{A}(\mathsf{st}, C) = 1].$$

Definition 4.2.3 (Computational Binding)**.** *A commitment scheme is computationally binding, if for all PPT adversaries $\mathcal{A}$ we have:*

$$|\Pr[(\mathsf{ck}) \leftarrow \mathsf{Gen}(\lambda), (m_0, r_0, m_1, r_1) \leftarrow \mathcal{A}(\mathsf{ck}) : m_0 \neq m_1 \ \wedge$$
$$\mathsf{Com}(m_0, r_0) = \mathsf{Com}(m_1, r_1)]| \leq \mathsf{Adv}^{\mathsf{binding}}_{\mathcal{A}}(\lambda),$$

where we require that $m_0, m_1 \in \mathcal{M}_{\mathsf{ck}}$, $r_0, r_1 \in \mathcal{OI}_{\mathsf{ck}}$ and $\mathsf{Adv}^{\mathsf{binding}}_{\mathcal{A}}(\lambda)$ is negligible in the security parameter λ.

Definition 4.2.4 (Equivocality)**.** *A commitment scheme is equivocal, if there exists an algorithm Eval and an alternative Gen' algorithm that additionally to the commitment key ck returns a trapdoor τ such that given a commitment $C = \mathsf{Com}(m, r)$ we have $C = \mathsf{Com}(m', \mathsf{Eval}(\tau, m', (C, m, r)))$ for any message m', i.e. Eval can be used to compute the randomness to open C to an arbitrary value.*

Moreover, we assume that there exists an efficient extraction algorithm $\mathsf{Extr}_{\mathsf{ck}}$ that given two openings of the same commitment, i.e. (m_1, r_1, m_2, r_2) where $\mathsf{Com}(m_1, r_1) = \mathsf{Com}(m_2, r_2)$ and $m_1 \neq m_2$, returns the trapdoor τ.

We require that a commitment scheme is binding, hiding and equivocal. Additionally, in this work we assume that the used commitment scheme has the following homomorphic property: for all $m_1, m_2 \in \mathcal{M}_{\mathsf{ck}}$ and $r_1, r_2 \in \mathcal{OI}_{\mathsf{ck}}$ we have: $\mathsf{Com}(m_1, r_1) \cdot \mathsf{Com}(m_2, r_2) = \mathsf{Com}(m_3, r_3)$, where $m_3 = m_1 + m_2$ and $r_3 = r_1 + r_2$. This homomorphism allows us to introduce multiplication by a known scalar, i.e. given $C = \mathsf{Com}(m, r)$ we can compute $C' = [k]C = \mathsf{Com}(k \cdot m, k \cdot r)$, where by $[k]C$ we denote multiplication of commitment C by a public scalar k. What is more, for a given commitment $C_b = \mathsf{Com}(b, r)$ to a bit b, we can easily compute the exclusive-or on this hidden value with a known bit α. If $\alpha = 0$, we leave C unchanged, otherwise, if $\alpha = 1$, we compute $C_{b \oplus \alpha} = C_1 / C_b = \mathsf{Com}(1 - b, -r)$, where $C_1 = \mathsf{Com}(1, 0)$ is formally a commitment to 1 with no randomness (instead of sampling it, it is set to 0). Note that for commitments $C_x = \mathsf{Com}(x, r) = \prod_{i \in [0, |x|-1]} [2^i] C_{x[i]} = \prod_{i \in [0, |x|-1]} [2^i] \mathsf{Com}(x[i], r_{x[i]})$, where $x[i]$ is the i-th bit of x, we can compute a commitment $C_{x \oplus \alpha}$ for a known α. To do so, we apply the above technique bitwise, i.e. to commitments $C_{x[i]}$ and using the new values we then compute the commitment $C_{x \oplus \alpha}$. Notice, that this operation changes the opening information, which is now $\sum_{i \in [0, |x|-1]} (-1)^{\alpha[i]} r_{x[i]}$, i.e. $C_{x \oplus \alpha} = \mathsf{Com}(x \oplus \alpha, \sum_{i \in [0, |x|-1]} (-1)^{\alpha[i]} r_{x[i]})$.

An example of a scheme that has those properties is the one introduced by Pedersen [107]. There, given a commitment key $\mathsf{ck} = (g, h, q, p)$, a message $m \in \mathbb{Z}_q$

and an opening information $r \in \mathbb{Z}_q^*$ the commitment is of the form $\mathsf{Com}(m, r) = g^m \cdot h^r \mod p$. Multiplying two commitments $\mathsf{Com}(m_1, r_1) \cdot \mathsf{Com}(m_2, r_2)$ we get $g^{m_1+m_2} h^{r_1+r_2}$, which is a commitment to message $m_1 + m_2 \mod q$ with opening information $r_1 + r_2 \mod q$, as required. Note that the commitment scheme is also equivocal and that there exists an extraction algorithm $\mathsf{Extr}_{\mathsf{ck}}$ (to this end, one simply needs to choose public parameters g, h with a known discrete logarithm).

4.2.2 Zero-Knowledge Proofs

Let $\mathcal{R} \subset \{0,1\}^* \times \{0,1\}^*$ be an efficiently computable binary relation, for which $\mathcal{R}(x, w) = 1 \iff (x, w) \in \mathcal{R}$. We call x a statement and w a witness. A very simple example of such a relation is $\mathcal{R} = \{(x, w) : x = SHA(w)\}$, where we are given a SHA value as part of the statement and the preimage is part of the witness. Obviously, given both values we can easily verify that $\mathcal{R}(x, w) = 1$ by computing the SHA value on w and comparing it with x. We will assume that $|w| \leq \mathsf{poly}(|x|)$, which means that the witness length should be polynomial in the statement length. We will denote by $L_{\mathcal{R}}$ the language consisting of true statements in $\mathcal{R}$, i.e. $L_{\mathcal{R}} = \{x | \exists w : (x, w) \in \mathcal{R}\}$.

We call a cryptographic protocol between two PPT parties, the prover $\mathcal{P}$ and the verifier $\mathcal{V}$ an *argument* for language $L_{\mathcal{R}}$ if it has the following properties. Using communication $\mathcal{P}$ wants to convince $\mathcal{V}$ that $x \in L_{\mathcal{R}}$, where x is a publicly known statement. Obviously, the prover has some extra private input, e.g. he knows a witness for which $\mathcal{R}(x, w) = 1$.

At the end of the protocol the verifier outputs accept if he is convinced and reject otherwise. The protocol is *complete* if for all $x \in L_{\mathcal{R}}$ an honest prover always convinces an honest verifier. We also require that if $x \notin L_{\mathcal{R}}$, then a cheating prover has only a small chance ϵ (called *soundess error*) to convince an honest verifier. This property should hold for all possible statements not in the language, i.e. for all $x \notin L_{\mathcal{R}}$ we have $\Pr[\mathcal{V}(x) = \mathsf{accept}] \leq \epsilon$. Finally, we require a property called *zero-knowledge* (ZK). Informally, this means that whatever strategy a verifier follows, he learns nothing besides whether $x \in L_{\mathcal{R}}$. It follows that he cannot get any information about the private input of the prover. A weaker notion of ZK is called honest-verifier ZK (HVZK). Zero-knowledge property in this case holds only for a verifier, who does not deviate from the protocol.

A special case of such arguments are Σ-protocols, which follow a specific communication pattern similar to the letter Σ. In the rest of the chapter, we will only consider this type of protocols.

Definition 4.2.5 (Σ-Protocol)**.** *A protocol $\Pi_{\mathcal{R}}$ between a prover $\mathcal{P}$ and a verifier $\mathcal{V}$ is a Σ-protocol for relation $\mathcal{R}$ if:*

- *The protocol consists of three phases:*

    1. *(Commit) $\mathcal{P}$ sends a message a to $\mathcal{V}$,*

2. *(Challenge) $\mathcal{V}$ picks a random e and sends it to $\mathcal{P}$,*
3. *(Response) $\mathcal{P}$ sends a second message z to $\mathcal{V}$.*

- $\Pi_{\mathcal{R}}$ *is complete - if both parties are honest, then for all $x \in L_{\mathcal{R}}$ we have* $\Pr[(\mathcal{P},\mathcal{V})(x) = 1] = 1$.
- $\Pi_{\mathcal{R}}$ *is s-special sound - for any x and any set of s accepting conversations $T = \{(a, e_i, z_i)\}_{i\in\{1,\ldots,s\}}$, where $e_i \neq e_j$ if $i \neq j$, there exists an efficient algorithm* Extr *that on input T outputs w such that $\mathcal{R}(x, w) = 1$.*
- $\Pi_{\mathcal{R}}$ *is a special honest-verifier ZK (HVZK) - there exists a PPT simulator $\mathcal{SIM}$ such that on input $x \in L_{\mathcal{R}}$ outputs a triple (a', e, z') with the same probability distribution of real conversations (a, e, z) of the protocol.*

The last property ensures only that Σ-protocols are ZK if the verifier is honest and does not base his challenge e on the first message of the prover. Σ-protocols have found many applications in the design of efficient identification and signature schemes. The main advantage of using those protocols is that using the Fiat-Shamir transformation [61], they can be made non-interactive in the random oracle model. What is more, using this technique the protocol is ZK even if the verifier is dishonest. Note that if the challenge e is chosen from a set of cardinality c, then s-special soundness implies that the soudness error is $(s-1)/c$.

Notation Given two commitments $C_x = \mathsf{Com}(x, r_x)$ and $C_y = \mathsf{Com}(y, r_y)$ we will denote by $\mathcal{P}\{(C_x \equiv C_y)\}$ the prover's part and by $\mathcal{V}\{(C_x \equiv C_y)\}$ the verifier's part of a Σ-protocol, where the prover tries to convince the verifier that it knows openings (x, r_x) and (y, r_y) of public commitments C_x and C_y, respectively, such that $x = y$. There exist very efficient Σ-protocols for the above mentioned Pedersen commitments. In such a case, the witness is composed of the committed value x and the opening informations r_x and r_y. We may sometimes append the notation to denote a subroutine algorithm such as *Commit*, *Response*, or *Reconstruct*. The *Commit* subroutine has a special output notation. We denote by (st, a) the result of *Commit* execution, where st denotes an internal state and a the output.

Notation for a bit commitment We will also use this notion for a special case, where the prover wants to show that the value committed in $C_x = \mathsf{Com}(x, r_x)$ is a bit, i.e. $x \in \{0, 1\}$. We will use $\mathcal{P}\{(C_x \equiv C_0) \vee (C_x \equiv C_1)\}$ to denote this special case. Note that we do not necessarily require the use of commitments to values 0 (C_0) and 1 (C_1), as there exist more efficient realizations, i.e. given a commitment $C = g^x \cdot h^r$ the prover simply shows that it knows the discrete logarithm of C or C/g to the base of h, and therefore C_0 and C_1 may be omitted. Moreover, we will use $\prod_{i=0}^{|x|-1} C_{2^i \cdot x[i]} = \mathsf{Com}(x, r)$ to denote a commitment to x, where $x = \sum_{i=0}^{|x|-1} 2^i x[i]$, $r = \sum_{i=0}^{|x|-1} 2^i r_{x[i]}$, and $C_{x[i]} = \mathsf{Com}(x[i], r_{x[i]})$.

4.2.3 ZKBoo/ZKB++

Giacomelli et al. [71] proposed ZKBoo, an efficient Σ-protocol based on the idea "MPC-in-the-head" [86]. Subsequently, Chase et al. [37] presented ZKB++, the successor of ZKBoo, which has more compact proofs. As both versions of the protocols differ primarily in technical aspects, our techniques can be applied to either version. The main advantage of this system over the one by Jawurek et al. [88] is that it can be made non-interactive using the Fiat-Shamir transformation.

ZKBoo/ZKB++ work for arithmetic functions F with prover's input x and the verifier holding no private input. Let y denote the output of function, i.e. $y = F(x)$. To create such a zero-knowledge proof of x, the prover splits the input into 3 shares (x_1, x_2, x_3) and for each pair x_i, x_{i+1} runs the function $F'(x_i, x_{i+1})$ to obtain y_i. F' is constructed in such a way that the correctness property of ZKBoo/ZKB++ ensures $y_1 \oplus y_2 \oplus y_3 = y$. The prover commits to all three views. The verifier sends a challenge $e \in \{1, 2, 3\}$, which can be replaced by the output of the random oracle applied on appropriate inputs. The prover opens input shares (x_e, x_{e+1}) and the randomness used in computing $F'(x_e, x_{e+1})$ in the corresponding two views. The verifier then checks whether y_e was correctly computed or not. Another property of F' is that two out of three views leak no information about the input x (the property is called 2-privacy; for more details we refer to Definition 3 in [71]). The protocol is 3-special sound and the soundness error of the protocol is 2/3. Therefore, to reduce the soundness error to a negligible value the prover runs multiple independent rounds of ZKBoo/ZKB++ protocol. In Fig. 4.5 in Appendix we present the non-interactive version of the ZKB++ protocol.

4.3 Combining ZKB++ with Algebraic Commitments

In this section we present our main contribution: a Σ-protocol for statements in cross-domains. Throughout this chapter we will consider the following statement as the main building block that can be composed to create proofs for more general statements.

Statement 4.3.1. *Prove that there exists x such that $F(x) = 1$ and x is committed to C_x, where x is a $|x|$-bit number, F is an arithmetic circuit, and the commitment scheme is based on the group structure of order larger than $2^{|x|}$ and allows some homomorphic operations.*

The naive approach to prove this statement is just to implement all algebraic operations as a part of the circuit F and execute the ZKB++ protocol. However, Chase et al. [38] already noticed that expressing modular exponentiation in a boolean circuit would be computationally too expensive and fairly inefficient. In particular, since the number of gates increases non-linearly in the size of the input, this also means that the proof size increases at the same rate and so does the time required to compute the proof. As we will show, there exists a more efficient way of realizing this kind of proofs.

4.3.1 Our Technique - First Approach

We propose the following technique, in which we take advantage of:

1) the fact that the ZKB++ protocol is a Σ-protocol,
2) the additive sharing of the prover's input x in the group $\mathbb{Z}_2$ in ZKB++,
3) that Σ-protocols can be executed in parallel,
4) a multiplicatively homomorphic commitment scheme in the group $\mathbb{Z}_q$; for simplicity we assume that $2^{|x|} < q$, the other case is addressed in Section 4.3.4.

The overall idea is to combine a ZKB++ round with zero-knowledge proofs that input bits of x are bound to the public commitment. This part involves ZK proofs for all individual bits of the ZKB++ input and the three exclusive-or based bit shares. In particular, we prove that the exclusive-or value of those shares is given in a commitment and is equal to the bit representation of x. We then prove that values in the commitments match the real shares by giving opening information for 2 out of 3 commitments, depending on the shares revealed by ZKB++. More details can be found in Construction 4.3.1.

Construction 4.3.1 (Cross-ZKB++ First Attempt). *Let $x[i]$ denote the i-th bit of input x, i.e. $x = (\ldots, x[1], x[0])$. In the following, we describe necessary steps to add to the ZKB++ protocol (Fig. 4.5) in order to realize the connection between the input bits $(\ldots, x[1], x[0])$ of the function F and the public commitment C_x, as defined in Statement 4.3.1.*

- *(Commit Phase) — The prover follows the steps specified by the ZKB++ protocol. Then, for each bit i of input x the prover commits to $x[i]$ and to the respective input shares $x[i]_1, x[i]_2, x[i]_3$ and gets $C_{x[i]}, C_{x[i]_1}, C_{x[i]_2}, C_{x[i]_3}$.*

  Again, for each bit i the prover executes the commit phase of a Σ-protocol (with challenge space $\{1, 2, 3\}$) for the following algebraic statement:

$$\begin{aligned}\mathcal{P}\{&\Big((C_{x[i]} \equiv C_0) \vee (C_{x[i]} \equiv C_1)\Big) \wedge \Big(\prod_{i\in[0,|x|-1]} C_{2^i \cdot x[i]} \equiv C_x\Big) \wedge \\ &\Big((C_{x[i]_1} \cdot C_{x[i]_2} \cdot C_{x[i]_3} \equiv C_{x[i]}) \vee (C_{x[i]_1} \cdot C_{x[i]_2} \cdot C_{x[i]_3} \equiv C_{2+x[i]})\Big)\}.\end{aligned} \tag{4.1}$$

  The prover sends commitments $\{C_{x[i]}, C_{x[i]_1}, C_{x[i]_2}, C_{x[i]_3}\}_{i\in[0,|x|-1]}$, and the commitments from the ZKB++ protocol and the Σ-protocol to prove the statement Eq. (4.1) to the verifier.

- *(Challenge Phase) — The verifier sends the challenge $e \in \{1, 2, 3\}$.*
- *(Response Phase) — The prover executes the last phase of the ZKB++ and the other proofs, and sends the result to the verifier. Additionally, he sends the opening information for commitments $C_{x[i]_e}$, $C_{x[i]_{e+1}}$, for all $i \in [0, |x| - 1]$.*

To verify the result the verifier follows the steps specified by the ZKB++ protocol and additionally performs the following checks: reject if the opening is wrong or the bits of the shares don't match the bits in the ZKB++ views, or if any of the additional algebraic proofs is invalid.

We present in Figs. 4.1 and 4.2 the detailed description of Construction 4.3.1, instantiated with t rounds of ZKB++ and made non-interactive using the Fiat-Shamir transformation.

Note that the proof system Eq. (4.1) does not explicitly enforce $C_{x[i]_1}, C_{x[i]_2}, C_{x[i]_3}$ to be commitments to bits. However, as we show in the proof of Theorem 4.3.1, it is the case.

4.3.1.1 Security analysis

Lemma 4.3.1. *Assuming the ZKB++ protocol is complete, the Σ-protocols for the algebraic statements are complete and the used commitment scheme is homomorphic, then Construction 4.3.1 is a complete Σ-protocol for the statement in Problem 4.3.1.*

Proof. Follows by inspection. □

Theorem 4.3.1. *Assuming the ZKB++ protocol is 3-special sound, the Σ-protocols for the algebraic statements are 2-special sound and the used commitment scheme is homomorphic and equivocal, then Construction 4.3.1 is a 3-special sound Σ-protocol for Statement 4.3.1.*

Proof. We will prove this theorem by constructing an efficient algorithm Extr_{Cross} that using 3 accepting tuples (a, e_1, z_1), (a, e_2, z_2) and (a, e_3, z_3) can compute $w^* = (x^*, r^*)$, such that $F(x^*) = 1$ and $C_x = \mathsf{Com}(x^*, r^*)$, which is a valid witness for the proven statement.

The algorithm works as follows:

1. First it uses the 3-special soundness of the ZKB++ protocol to extract a value x_{ZKB} for which $F(x_{ZKB}) = 1$.
2. It uses the 2-special soundness of the Σ-protocols for the algebraic statements to extract the values $x[i]$, for all $i \in [0, |x|-1]$, and the corresponding opening information $r_{x[i]}$.

```
$p \leftarrow \mathsf{Prove}(x, C_x = \mathsf{Com}(x, r))$
1 :     // ($Commit$ step)
2 : $(st_\zeta, a_\zeta) \leftarrow \mathsf{ZKB}_F.Commit(x)$
3 : foreach $i \in [0, |x| - 1]$ do
4 :     $C_{x[i]} = \mathsf{Com}(x[i], r_i)$
5 :     foreach $\rho \in [1, t]$ do
6 :         Extract shares $x[i]_1^{(\rho)}, x[i]_2^{(\rho)}, x[i]_3^{(\rho)}$ from $st_\zeta$
7 :         $C_{x[i]_1^{(\rho)}} = \mathsf{Com}(x[i]_1^{(\rho)}, r_{x[i]_1^{(\rho)}})$
8 :         $C_{x[i]_2^{(\rho)}} = \mathsf{Com}(x[i]_2^{(\rho)}, r_{x[i]_2^{(\rho)}})$
9 :         $C_{x[i]_3^{(\rho)}} = \mathsf{Com}(x[i]_3^{(\rho)}, r_{x[i]_3^{(\rho)}})$
10 : $(st_x, a_x) \leftarrow \mathcal{P}\{\prod_{i \in [0,|x|-1]} C_{2^i \cdot x[i]} \equiv C_x\}.Commit(x, \sum_{i=0}^{|x|-1} 2^i \cdot r_i, r)$
11 : foreach $i \in [0, |x| - 1]$ do
12 :     $(st_{x[i]}, a_{x[i]}) \leftarrow \mathcal{P}\{(C_{x[i]} \equiv C_0) \vee (C_{x[i]} \equiv C_1)\}.Commit(x[i], r_i)$
13 :     foreach $\rho \in [1, t]$ do
14 :         $C_{x[i]^{(\rho)}} = C_{x[i]_1^{(\rho)}} \cdot C_{x[i]_2^{(\rho)}} \cdot C_{x[i]_3^{(\rho)}}$
15 :         $(st_{x[i]^{(\rho)}}, a_{x[i]^{(\rho)}}) \leftarrow \mathcal{P}\{(C_{x[i]^{(\rho)}} \equiv C_{x[i]}) \vee (C_{x[i]^{(\rho)}} \equiv C_{2+x[i]})\}.Commit($
16 :             $x[i]_1^{(\rho)} + x[i]_2^{(\rho)} + x[i]_3^{(\rho)}, r_{x[i]_1^{(\rho)}} + r_{x[i]_2^{(\rho)}} + r_{x[i]_3^{(\rho)}}, r_i)$
17 : $a = (a_\zeta, (C_{x[i]})_{|x|}, (C_{x[i]_1^{(\rho)}})_{|x| \cdot t}, (C_{x[i]_2^{(\rho)}})_{|x| \cdot t}, (C_{x[i]_3^{(\rho)}})_{|x| \cdot t},$
18 :             $a_x, (a_{x[i]})_{|x|}, (a_{x[i]^{(\rho)}})_{|x| \cdot t})$   // output of (Commit step)
19 : $e \leftarrow H(a)$   // ($Challenge$ step)
20 :     // ($Response$ step)
21 : $\mathbf{r}_\zeta \leftarrow \mathsf{ZKB}_F.Response(e, st_\zeta)$
22 : $\mathbf{r}_x \leftarrow \mathcal{P}\{\prod_{i \in [0,|x|-1]} C_{2^i \cdot x[i]} \equiv C_x\}.Response(e, st_x)$
23 : foreach $i \in [0, |x| - 1]$ do
24 :     $\mathbf{r}_{x[i]} \leftarrow \mathcal{P}\{(C_{x[i]} \equiv C_0) \vee (C_{x[i]} \equiv C_1)\}.Response(e, st_{x[i]})$
25 :     foreach $\rho \in [1, t]$ do
26 :         $\mathbf{r}_{x[i]^{(\rho)}} \leftarrow \mathcal{P}\{(C_{x[i]^{(\rho)}} \equiv C_{x[i]}) \vee (C_{x[i]^{(\rho)}} \equiv C_{2+x[i]})\}.Response(e, st_{x[i]^{(\rho)}})$
27 : return $(e, a, \mathbf{r}_\zeta, \mathbf{r}_x, (\mathbf{r}_{x[i]})_{|x|}, (\mathbf{r}_{x[i]^{(\rho)}})_{|x| \cdot t},$
28 :             $(x[i]_e^{(\rho)}, r_{x[i]_e^{(\rho)}})_{|x| \cdot t}, (x[i]_{e+1}^{(\rho)}, r_{x[i]_{e+1}^{(\rho)}})_{|x| \cdot t})$
```

Figure 4.1: Description of Cross-ZKB++ (First Attempt) Prove algorithm for function $F(x) = 1$ with a committed input $C_x = \mathsf{Com}(x, r)$, made non-interactive using the Fiat-Shamir transformation and with t rounds of ZKB++.

```
$\{Reject, Accept\} \leftarrow \mathsf{Verify}(C_x, p)$
1 :   // $Reconstruct$ step
2 :   Parse $p$ as $(e, a, \mathbf{r}_\zeta, \mathbf{r}_x, (\mathbf{r}_{x[i]})_{|x|}, (\mathbf{r}_{x[i]^{(\rho)}})_{|x|\cdot t},$
3 :            $(x[i]_e^{(\rho)}, r_{x[i]_e^{(\rho)}})_{|x|\cdot t}, (x[i]_{e+1}^{(\rho)}, r_{x[i]_{e+1}^{(\rho)}})_{|x|\cdot t})$
4 :   Parse $a$ as $(a_\zeta, (C_{x[i]})_{|x|}, (C_{x[i]_1^{(\rho)}})_{|x|\cdot t}, (C_{x[i]_2^{(\rho)}})_{|x|\cdot t}, (C_{x[i]_3^{(\rho)}})_{|x|\cdot t},$
5 :            $a_x, (a_{x[i]})_{|x|}, (a_{x[i]^{(\rho)}})_{|x|\cdot t})$
6 :   foreach $i \in [0, |x|-1]$ do
7 :     foreach $\rho \in [1, t]$ do
8 :       $Reject$ if $C_{x[i]_e^{(\rho)}} \neq \mathsf{Com}(x[i]_e^{(\rho)}, r_{x[i]_e^{(\rho)}})$ or $C_{x[i]_{e+1}^{(\rho)}} \neq \mathsf{Com}(x[i]_{e+1}^{(\rho)}, r_{x[i]_{e+1}^{(\rho)}})$
9 :   $(st'_\zeta, a'_\zeta) \leftarrow \mathsf{ZKB}_F.Reconstruct(e, \mathbf{r}_\zeta)$
10 :  $Reject$ if $(x_e^{(\rho)})_t, (x_{e+1}^{(\rho)})_t$ do not match respective values in $st'_\zeta$
11 :  $a'_x \leftarrow \mathcal{V}\{\prod_{i\in[0,|x|-1]} C_{2^i\cdot x[i]} \equiv C_x\}.Reconstruct(e, \mathbf{r}_x)$
12 :  foreach $i \in [0, |x|-1]$ do
13 :    $a'_{x[i]} \leftarrow \mathcal{V}\{(C_{x[i]} \equiv C_0) \vee (C_{x[i]} \equiv C_1)\}.Reconstruct(e, \mathbf{r}_{x[i]})$
14 :    foreach $\rho \in [1, t]$ do
15 :      $C_{x[i]^{(\rho)}} = C_{x[i]_1^{(\rho)}} \cdot C_{x[i]_2^{(\rho)}} \cdot C_{x[i]_3^{(\rho)}}$
16 :      $a'_{x[i]^{(\rho)}} \leftarrow \mathcal{V}\{(C_{x[i]^{(\rho)}} \equiv C_{x[i]}) \vee (C_{x[i]^{(\rho)}} \equiv C_{2+x[i]})\}.Reconstruct(e, \mathbf{r}_{x[i]^{(\rho)}})$
17 :  $a' = (a'_\zeta, (C_{x[i]})_{|x|}, (C_{x[i]_1^{(\rho)}})_{|x|\cdot t}, (C_{x[i]_2^{(\rho)}})_{|x|\cdot t}, (C_{x[i]_3^{(\rho)}})_{|x|\cdot t},$
18 :           $a'_x, (a'_{x[i]})_{|x|}, (a'_{x[i]^{(\rho)}})_{|x|\cdot t})$
19 :  $e' \leftarrow H(a')$
20 :  $Accept$ if $e' = e$, otherwise $Reject$.
```

Figure 4.2: Description of Cross-ZKB++ (First Attempt) Verify algorithm for function $F(x) = 1$ with a committed input $C_x = \mathsf{Com}(x, r)$, made non-interactive using the Fiat-Shamir transformation and with t rounds of ZKB++.

We now show the rest of his steps. Without loss of generality, let us assume that $e_1 = 1$, $e_2 = 2$ and $e_3 = 3$. For all $i \in [0, |x|-1]$ let $w_2 = (x, r, x[i], r_{x[i]}, x[i]_1, r_{x[i]_1}, x[i]_2, r_{x[i]_2}, x[i]_3, r_{x[i]_3})$ be the witness extracted in step 2 and $w_1 = (x_{ZKB})$ be the witness extracted in step 1. Moreover, for $i \in \{1,2,3\}$ let $r_{x[i]_{e_i}}$ and $r_{x[i]_{e_i+1}}$ be the opening information to commitments $C_{x[i]_{e_i}}$ and $C_{x[i]_{e_i+1}}$, where we know that $C_{x[i]_{e_i}} = \mathsf{Com}(x[i]_{e_i}, r_{x[i]_{e_i}})$ and $C_{x[i]_{e_i+1}} = \mathsf{Com}(x[i]_{e_i+1}, r_{x[i]_{e_i+1}})$.

We now turn to the following observation. If at some point the algorithm Extr_{Cross} encounters two different opening information to one commitment, i.e. $\mathsf{Com}(a,b) = \mathsf{Com}(c,d)$ it can use (a,b,c,d) to compute the equivocal trapdoor and open any commitment to an arbitrarily value. In particular, it can use this trapdoor to open commitment C_x to the value x_{ZKB}, i.e. in case $x \neq x_{ZKB}$ we can use (x,r) and the equivocal trapdoor to compute $x^* = x_{ZKB}$ and the corresponding r^* such that $C_x = \mathsf{Com}(x^*, r^*)$, which would constitute a valid witness w^*.

We now proceed with the proof and notice that due to the verification done by the verifier and the extracted witness w_2, we know that

$$\begin{aligned}
C_{x[i]_1} &= \mathsf{Com}(x[i]_1, r_{x[i]_1}) = \mathsf{Com}(x[i]_{e_1}, r_{x[i]_{e_1}}) = \mathsf{Com}(x[i]_{e_3+1}, r_{x[i]_{e_3+1}}),\\
C_{x[i]_2} &= \mathsf{Com}(x[i]_2, r_{x[i]_2}) = \mathsf{Com}(x[i]_{e_2}, r_{x[i]_{e_2}}) = \mathsf{Com}(x[i]_{e_1+1}, r_{x[i]_{e_1+1}}),\\
C_{x[i]_3} &= \mathsf{Com}(x[i]_3, r_{x[i]_3}) = \mathsf{Com}(x[i]_{e_3}, r_{x[i]_{e_3}}) = \mathsf{Com}(x[i]_{e_2+1}, r_{x[i]_{e_2+1}}),
\end{aligned}$$

and that for $i \in \{1,2,3\}$ $x[i]_{e_i}$ are bits that correspond to disclosed views in the ZKB++ protocol. Thus, it follows that $x[i]_1 = x[i]_{e_1}$, $x[i]_2 = x[i]_{e_2}$ and $x[i]_3 = x[i]_{e_3}$ and in particular that $x_{ZKB}[i] = x[i]_1 \oplus x[i]_2 \oplus x[i]_3$ for all $i \in [0, |x|-1]$.

We will now argue that because of the soundness of the proof system used in step 2, for all $i \in [0, |x|-1]$ we have $x[i] = x[i]_1 \oplus x[i]_2 \oplus x[i]_3 = x_{ZKB}[i]$. Let us take a look at the following table.

$x[i]_1$	$x[i]_2$	$x[i]_3$	$x[i]_1 + x[i]_2 + x[i]_3$	$x[i]_1 + x[i]_2 + x[i]_3 - 2$	$x[i]_1 \oplus x[i]_2 \oplus x[i]_3$
0	0	0	0	-2	0
0	0	1	1	-1	1
0	1	0	1	-1	1
0	1	1	2	0	0
1	0	0	1	-1	1
1	0	1	2	0	0
1	1	0	2	0	0
1	1	1	3	1	1

The two rows $x[i]_1 + x[i]_2 + x[i]_3$ and $x[i]_1 + x[i]_2 + x[i]_3 - 2$ correspond to the value that the commitment $C_{x[i]} = \mathsf{Com}(x[i], r_{x[i]})$ can be opened to. However, due to the fact that the statement contains the additional constraint that the commitment opens to a bit, we conclude that for $(x[i], r_{x[i]})$ we have $x[i] = x_{ZKB}[i]$ (we used the coloured background to highlight the only way that witness w_2 can be correct).

Finally, we know that since the witness w_2 is correct, it follows that:

$$\sum_{i\in[0,|x|-1]} 2^i \cdot x[i] = x.$$

However, since $x[i]$ is the i-th bit of x_{ZKB} this means that $x_{ZKB} = x$ and the Extr_{Cross} can return $w^* = (x^*, r^*) = (x_{ZKB}, \sum_{i\in[0,|x|-1]} 2^i \cdot r_{x[i]})$, which is a valid opening for C_x, where $F(x^*) = 1$.

We conclude that the values returned by Extr_{Cross} are a valid witness for Statement 4.3.1. □

Theorem 4.3.2. *Assuming the ZKB++ protocol and the Σ-protocols for the algebraic statements are HVZK and the commitment scheme is perfectly hiding, then Construction 4.3.1 is a HVZK Σ-protocol for Statement 4.3.1.*

Proof. We will show how to construct a simulator $\mathcal{SIM}$ that on input in Statement 4.3.1, outputs a transcript (a, e, z). The simulator works as follows:

- It runs the simulator for ZKB++ receiving a transcript (a', e, z'), where z' contains all the bits $x[i]_e$ and $x[i]_{e+1}$. $\mathcal{SIM}$ chooses open information $r_{x[i]_e}$, $r_{x[i]_{e+1}}$ and computes commitments $C_{x[i]_e} = \mathsf{Com}(x[i]_e, r_{x[i]_e})$, $C_{x[i]_{e+1}} = \mathsf{Com}(x[i]_{e+1}, r_{x[i]_{e+1}})$. Note that the openings $r_{x[i]_e}, r_{x[i]_{e+1}}$ are part of the response z. Commitment to the bits $x[i]$ and the bits $x[i]_{e+2}$ are not opened, so the simulator can compute $C_{x[i]}$ and $C_{x[i]_{e+2}}$ as commitments to zero.
- $\mathcal{SIM}$ runs the simulator for the Σ-protocol for the algebraic statements receiving (a'', e'', z''). Note that since this simulator should work for all possible challenges, there is a non-negligible probability that $e'' = e$. Otherwise, $\mathcal{SIM}$ just restarts it.
- Finally, $\mathcal{SIM}$ sets $a = (a', a'', \{C_{x[i]}, C_{x[i]_1}, C_{x[i]_2}, C_{x[i]_3}\}_{i\in[0,|x|-1]})$ and $z = (z', z'', \{r_{x[i]_e}, r_{x[i]_{e+1}}\}_{i\in[0,|x|-1]})$

Since all the simulators used by $\mathcal{SIM}$ generate valid transcripts it remains to show that the commitments $C_{x[i]}$ and $C_{x[i]_{e+2}}$ generated by $\mathcal{SIM}$ are indistinguishable from values in real transcripts. However, this follows directly by the perfectly hiding property of the commitment scheme. □

Lemma 4.3.2. *The soundness error of the Σ-protocol presented in Construction 4.3.1 is $2/3$ and it has to be executed $\lambda/(\log_2(3) - 1)$ times/rounds to achieve a soundness error of $2^{-\lambda}$.*

Proof. The soundness error is implied directly from 3-special soundness of the protocol (Theorem 4.3.1) and the challenge space of cardinality 3. The number of rounds, let us denote it t, is simply the solution of equation $(2/3)^t = 2^{-\lambda}$. □

4.3.2 Improved Version

The main disadvantage of Construction 4.3.1 is that we have to compute $O(|x| \cdot t)$ commitments, which influences the number of public key operations we have to additionally compute. The $|x|$ factor is present because for each round $\rho \in [1, t]$ the relation $x[i]_1^{(\rho)} \oplus x[i]_2^{(\rho)} \oplus x[i]_3^{(\rho)} = x[i]^{(\rho)}$ is expressed as a conjunction of two possible statements and we commit to the bits of the input x in every round. In the following, we optimize Construction 4.3.1 to increase efficiency by decreasing the number of commitment to $O(|x| + t)$.

Firstly, we notice that we can use the same commitments to bits of x for every round that we repeat the protocol and instead of committing to the bits of the ZKB++ shares we actually compute commitment to the whole values, saving a lot of computations. Note that this idea will only work if the input to ZKB++ is smaller that the order of the algebraic group that we use, otherwise the bitwise exclusive-or of those values will not constitute a accepting input to the ZKB++ circuit (i.e. $x_1 \oplus x_2 \oplus x_3$ is not always equal to $(x_1 \mod q) \oplus (x_2 \mod q) \oplus (x_3 \mod q)$). However, in the next subsection we show how to make the protocol work for ZKB++ input without a size constraint.

Secondly, the bits $x[i]_{e^{(\rho)}}^{(\rho)}$ and $x[i]_{e^{(\rho)}+1}^{(\rho)}$ are revealed in the response step of the ZKB++ protocol (Fig. 4.5). Based on this observation, we can change the relation

$$x[i]_1^{(\rho)} \oplus x[i]_2^{(\rho)} \oplus x[i]_3^{(\rho)} = x[i]$$

and express the third share using the hidden value x, i.e.

$$x[i]_{e^{(\rho)}+2}^{(\rho)} = x[i] \oplus (x[i]_{e^{(\rho)}}^{(\rho)} \oplus x[i]_{e^{(\rho)}+1}^{(\rho)}).$$

We now take into account that this relation is constructed for known bits and that we can express $C_{a \oplus \alpha}$ for a given C_a and α using homomorphic properties of the commitment scheme. Thus, we can actually compute a commitment to $x[i]_{e^{(\rho)}+2}^{(\rho)}$ using the commitments to bits of x and the revealed bits of values $x[i]_{e^{(\rho)}}^{(\rho)}$ and $x[i]_{e^{(\rho)}+1}^{(\rho)}$. We use this commitment to bind the value $x[i]_1^{(\rho)} \oplus x[i]_2^{(\rho)} \oplus x[i]_3^{(\rho)}$ with the value x inside the commitment C_x.

In Construction 4.3.2 we describe those ideas in more detail. We will show a single round of the protocol, which only has a soundness error of 2/3 but below present the idea how decrease the soundness error efficiently. Our protocol is divided into four essential steps:

1. committing to bits of x,
2. proving using a Schnorr based Σ-protocol that those commitments contain a bit,
3. a ZKB++ proof that there exists a x_{ZKB} such that $F(x_{ZKB}) = 1$, and

4. constant number of commitments $C_{x_1}, C_{x_2}, C_{x_3}$, which ensure $x = x_{ZKB}$.

Thus, if one would run the protocol many times, this still would require the computation of $O(|x| \cdot t)$ commitments.

We solve this problem by taking advantage of the fact that Schnorr based Σ-protocols can use a larger challenge space that decreases the soundness error without repeating the protocol. Unfortunately, this does not apply for the ZKB++ part and for this to work we have to use a special kind of challenge. Let $e_1, \ldots, e_\rho$ be the challenges used for the ρ runs of the ZKB++ protocol, then we can use e.g. $e_\Sigma = \sum_{i \in [0,\rho-1]} 3^i \cdot e_{i+1}$ in step 2. In other words, we execute the first two steps once using the challenge e_Σ and simultaneously run the last two steps ρ-times, where each ZKB++ execution challenged respectively using $e_1, \ldots, e_\rho$.

This simple trick allows us to increase the efficiency of the proof. Now the prover only has to compute a constant number of commitments per round and commit to the bits of the input x only once.

Construction 4.3.2 (Cross-ZKB++). *In the following, we describe necessary steps to add to the ZKB++ protocol (Fig. 4.5) in order to realize the connection between the input bits $x = (\ldots, x[1], x[0])$ to the function F, where $x < q$ and the public commitment C_x in group of order q, as defined in Statement 4.3.1.*

- *(Commit Phase) — The prover executes the commit step of the ZKB++ protocol using input x, where $C_x = \mathsf{Com}(x, r)$. The prover chooses random opening informations $r_1, \ldots r_{|x|-1}$ and commits to the bits $x[i]$ by computing:*

  $$C_{x[i]} = \mathsf{Com}(x[i], r_i), \text{ for } i \in [1, |x| - 1].$$

  To compute the remaining commitment he uses the opening information $r_0 = r - \sum_{i \in [1,|x|-1]} 2^i \cdot r_i$. Note that because of the homomorphic properties of the commitment scheme this means that $C_x = \prod_{i \in [0,|x|-1]} [2^i] C_{x[i]} = \mathsf{Com}(2^i \cdot x[i], 2^i \cdot r_i)$. For each bit i of input x the prover executes the commit step of a Σ-protocol for the following algebraic statement:

  $$\mathcal{P}\{(C_{x[i]} \equiv C_0) \vee (C_{x[i]} \equiv C_1)\}. \tag{4.2}$$

  The next step is also different. In this protocol we commit to the full values of the respective input shares x_1, x_2, x_3 and get $C_{x_1}, C_{x_2}, C_{x_3}$, where $C_{x_1} = \mathsf{Com}(x_1, r_{x_1})$, $C_{x_2} = \mathsf{Com}(x_2, r_{x_2})$, $C_{x_3} = \mathsf{Com}(x_3, r_{x_3})$. The prover sends commitments $\{C_{x[i]}\}_{i \in [0,|x|-1]}$, $C_{x_1}, C_{x_2}, C_{x_3}$ and the commitments from the ZKB++ protocol and the Σ-protocol Eq. (4.2) *to the verifier.*

- *(Challenge Phase) — The verifier sends the challenge $e \in \{1, 2, 3\}$ to the prover.*

- *(Response Phase) — The prover executes the response step for ZKB++, the Σ-protocol and sends the result to the verifier. Knowing e, the prover computes $\alpha = x_e \oplus x_{e+1}$, where by $\alpha[i]$ we will denote its i-th bit. Using the homomorphic exclusive-or described in subsection 4.2.1, he then computes the commitment*
$$C_z = \prod_{i\in[0,|x|-1]} [2^i] C_{x[i]\oplus\alpha[i]},$$
which is
$$\mathsf{Com}(x_{e+2}, \sum_{i\in[0,|x|-1]} (-1)^{\alpha[i]} r_{x[i]}).$$
Finally, the prover sends the opening information r_{x_e}, $r_{x_{e+1}}$ for commitments C_{x_e}, $C_{x_{e+1}}$ and value $r_z = r_{x_{e+2}} - \prod_{i\in[0,|x|-1]}(-1)^{\alpha[i]} r_{x[i]}$.

To verify the result the verifier follows the steps specified by the ZKB++ protocol and additionally performs the following checks: reject if the opening is wrong or the shares in the commitments do not match the ones in the ZKB++ views, or if any of the additional algebraic proofs is invalid. The verifier aborts if $C_x \neq \prod_{i\in[0,|x|-1]}[2^i]C_{x[i]}$. Knowing the shares x_e, x_{e+1} and the openings r_{x_e}, $r_{x_{e+1}}$, the verifier also computes $C_z = \prod_{i\in[0,|x|-1]}[2^i]C_{x[i]\oplus\alpha_i}$ and checks that $C_z \cdot \mathsf{Com}(0, r_z) = C_{x_{e+2}}$.

We present in Figs. 4.3 and 4.4 the detailed description of Construction 4.3.2, instantiated with t rounds of ZKB++ and made non-interactive using the Fiat-Shamir transformation.

4.3.3 Security analysis

Lemma 4.3.3. *Assuming the ZKB++ protocol is complete, the Σ-protocols for the algebraic statements are complete and the used commitment scheme is homomorphic, then Construction 4.3.2 is a complete Σ-protocol for the statement in Problem 4.3.1.*

Proof. Follows by inspection. □

Theorem 4.3.3. *Assuming the ZKB++ protocol is* 3*-special sound the used Σ-protocols are* 2*-special sound and the used commitment scheme is homomorphic and equivocal, then Construction 4.3.2 is a* 3*-special sound Σ-protocol for Statement 4.3.1.*

Proof. As in the proof of Theorem 4.3.1 we will construct an efficient algorithm Extr_{Cross} that using 3 accepting tuples (a, e_1, z_1), (a, e_2, z_2) and (a, e_3, z_3) can compute a witness that the statement is true. The extraction algorithm will return a value x and an opening information r such that $F(x) = 1$ and $C_x = \mathsf{Com}(x, r)$, which is a valid witness for the proven statement. We will now describe the idea behind the algorithm Extr_{Cross}, which is as follows:

```
$\underline{p \leftarrow \mathsf{Prove}(x, C_x = \mathsf{Com}(x, r))}$
    // ($\mathit{Commit}$ step)
$(st_\zeta, a_\zeta) \leftarrow \mathsf{ZKB}_F.\mathit{Commit}(x)$
$\textbf{foreach } i \in [1, |x| - 1] \textbf{ do}$
    $C_{x[i]} = \mathsf{Com}(x[i], r_i)$
$r_0 = r - \sum_{i \in [1, |x|-1]} 2^i \cdot r_i$
$C_{x[0]} = \mathsf{Com}(x[0], r_0)$
$\textbf{foreach } i \in [0, |x| - 1] \textbf{ do}$
    $(st_{x[i]}, a_{x[i]}) \leftarrow \mathcal{P}\{(C_{x[i]} \equiv C_0) \vee (C_{x[i]} \equiv C_1)\}.\mathit{Commit}(x[i], r_i)$
$\textbf{foreach } \rho \in [1, t] \textbf{ do}$
    Extract shares $x_1^{(\rho)}, x_2^{(\rho)}, x_3^{(\rho)}$ from $st_\zeta$
    $C_{x_1^{(\rho)}} = \mathsf{Com}(x_1^{(\rho)}, r_{x_1^{(\rho)}}), C_{x_2^{(\rho)}} = \mathsf{Com}(x_2^{(\rho)}, r_{x_2^{(\rho)}}), C_{x_3^{(\rho)}} = \mathsf{Com}(x_3^{(\rho)}, r_{x_3^{(\rho)}})$
$a = (a_\zeta, (C_{x[i]})_{|x|}, (a_{x[i]})_{|x|}, (C_{x_1^{(\rho)}})_t, (C_{x_2^{(\rho)}})_t, (C_{x_3^{(\rho)}})_t)$    // output of (Commit step)
    // ($\mathit{Challenge}$ step)
$e \leftarrow H(a)$
    // ($\mathit{Response}$ step)
$\mathbf{r}_\zeta \leftarrow \mathsf{ZKB}_F.\mathit{Response}(e, st_\zeta)$
$\textbf{foreach } i \in [0, |x| - 1] \textbf{ do}$
    $\mathbf{r}_{x[i]} \leftarrow \mathcal{P}\{(C_{x[i]} \equiv C_0) \vee (C_{x[i]} \equiv C_1)\}.\mathit{Response}(e, st_{x[i]})$
$\textbf{foreach } \rho \in [1, t] \textbf{ do}$
    $\alpha^{(\rho)} = x_e^{(\rho)} \oplus x_{e+1}^{(\rho)}$
    $C_z^{(\rho)} = \prod_{i \in [0, |x|-1]} [2^i] C_{x[i] \oplus \alpha^{(\rho)}[i]}$
    $r_z^{(\rho)} = r_{x_{e+2}^{(\rho)}} - \prod_{i \in [0, |x|-1]} (-1)^{\alpha[i]} r_{x[i]}$
$\textbf{return } (e, a, \mathbf{r}_\zeta, (\mathbf{r}_{x[i]})_{|x|}, (x_e^{(\rho)}, r_{x_e^{(\rho)}})_t, (x_{e+1}^{(\rho)}, r_{x_{e+1}^{(\rho)}})_t, (r_z^{(\rho)})_t)$
```

Figure 4.3: Description of Cross-ZKB++ Prove algorithm for function $F(x) = 1$ with a committed input $C_x = \mathsf{Com}(x, r)$, made non-interactive using the Fiat-Shamir transformation and with t rounds of ZKB++.

```
$\{Reject, Accept\} \leftarrow \mathsf{Verify}(C_x, p)$
 1:     // $Reconstruct$ step
 2: Parse $p$ as $(e, a, \mathbf{r}_\zeta, (\mathbf{r}_{x[i]})_{|x|}, (x_e^{(\rho)}, r_{x_e^{(\rho)}})_t, (x_{e+1}^{(\rho)}, r_{x_{e+1}^{(\rho)}})_t, (r_z^{(\rho)})_t)$
 3: Parse $a$ as $(a_\zeta, (C_{x[i]})_{|x|}, (a_{x[i]})_{|x|}, (C_{x_1^{(\rho)}})_t, (C_{x_2^{(\rho)}})_t, (C_{x_3^{(\rho)}})_t)$
 4: $Reject$ if $C_x \neq \prod_{i \in [0,|x|-1]} [2^i] C_{x[i]}$
 5: foreach $\rho \in [1, t]$ do
 6:     $Reject$ if $C_{x_e^{(\rho)}} \neq \mathsf{Com}(x_e^{(\rho)}, r_{x_e^{(\rho)}})$ or $C_{x_{e+1}^{(\rho)}} \neq \mathsf{Com}(x_{e+1}^{(\rho)}, r_{x_{e+1}^{(\rho)}})$
 7:     $\alpha^{(\rho)} = x_e^{(\rho)} \oplus x_{e+1}^{(\rho)}$
 8:     $C_z^{(\rho)} = \prod_{i \in [0,|x|-1]} [2^i] C_{x[i] \oplus \alpha^{(\rho)}[i]}$
 9:     $Reject$ if $C_z^{(\rho)} \cdot \mathsf{Com}(0, r_z^{(\rho)}) \neq C_{x_{e+2}^{(\rho)}}$
10: $(st'_\zeta, a'_\zeta) \leftarrow \mathsf{ZKB}_F.Reconstruct(e, \mathbf{r}_\zeta)$
11: $Reject$ if $(x_e^{(\rho)})_t, (x_{e+1}^{(\rho)})_t$ do not match respective values in $st'_\zeta$
12: foreach $i \in [0, |x|-1]$ do
13:     $a'_{x[i]} \leftarrow \mathcal{V}\{(C_{x[i]} \equiv C_0) \vee (C_{x[i]} \equiv C_1)\}.Reconstruct(e, \mathbf{r}_{x[i]})$
14: $a' = (a'_\zeta, (C_{x[i]})_{|x|}, (a'_{x[i]})_{|x|}, (C_{x_1^{(\rho)}})_t, (C_{x_2^{(\rho)}})_t, (C_{x_3^{(\rho)}})_t)$
15: $e' \leftarrow H(a')$
16: $Accept$ if $e' = e$, otherwise $Reject$.
```

Figure 4.4: Description of Cross-ZKB++ Verify algorithm for function $F(x) = 1$ with a committed input $C_x = \mathsf{Com}(x, r)$, made non-interactive using the Fiat-Shamir transformation and with t rounds of ZKB++.

- First it uses the 3-special soundness of the ZKB++ protocol to extract a value x_{ZKB} for which $F(x_{ZKB}) = 1$.
- It uses the 2-special soundness of the proof system for Eq. (4.2) to extract the bits $x[i]$, for all $i \in [0, |x| - 1]$, and the opening information $r_{x[i]}$.
- It computes r_{ZKB}, as described below, and returns $(x^*, r^*) = (x_{ZKB}, r_{ZKB})$ as a valid witness.

We will now show how Extr_{Cross} computes witness $w^* = (x^*, r^*)$ and that the returned values are valid. Let $w_2 = (\{x[i], r_{x[i]}\}_{i\in[0,|x|-1]})$ be the witness extracted in step 2 and $w_1 = (x_{ZKB})$ be the witness extracted in step 1. Moreover, let r_{x_e} and $r_{x_{e+1}}$ be the opening information to commitments $C_{x_{e_1}}$, $C_{x_{e_2}}$ and $C_{x_{e_3}}$, where we know that $C_{x_{e_1}} = \mathsf{Com}(x_{e_1}, r_{x_{e_1}})$, $C_{x_{e_2}} = \mathsf{Com}(x_{e_2}, r_{x_{e_2}})$ and $C_{x_{e_3}} = \mathsf{Com}(x_{e_3}, r_{x_{e_3}})$.

Again we observe that if the algorithm Extr_{Cross} encounters two different opening information to one commitment the equivocal trapdoor can be used to open the commitment C_x to the value $w_1 = x_{ZKB}$.

We now proceed with the proof and notice that since all the tuples are accepting, we conclude that the openings of the commitments $C_{x_1}, C_{x_2}, C_{x_3}$ are valid. This is the case because we have valid openings $(x_{e_1}, r_{x_{e_1}})$, $(x_{e_2}, r_{x_{e_2}})$, $(x_{e_3}, r_{x_{e_3}})$. It follows that the binary representations of x_1, x_2, x_3 correspond to the correct input of the ZKB++ protocol and we have $x_{ZKB} = x_1 \oplus x_2 \oplus x_3$. Note that this is only true because x_{ZKB} is shorter that the order of the used group. Moreover, we know that by construction:

$$C_x = \prod_{i\in[0,|x|-1]} [2^i]\mathsf{Com}(x[i], r_{x[i]}),$$

and that $x[i]$ are bits.

Let $e = e_1$, the Extr_{Cross} computes commitment $C_z = \prod_{i\in[0,|x|-1]}[2^i]C_{x[i]\oplus\alpha_i}$, where $\alpha = x_e \oplus x_{e+1}$. Since we know that for e_1 we receive an accepting state, we know that $C_z \cdot \mathsf{Com}(0, r_z) = C_{x_{e+2}} = \mathsf{Com}(x_{e+2}, r_{x_{e+2}})$. This basically means that Extr_{Cross} can open C_z to x_{e+2} using randomness $r_{x_{e+2}} - r_z$. We now distinguish two cases:

1. the openings of C_z and $C_{x_{e+2}}$ are different, i.e. this means that
$$\sum_{i\in[0,|x|-1]} 2^i(x[i] \oplus \alpha_i) \neq x_{e+2},$$
2. the openings of C_z and $C_{x_{e+2}}$ are the same.

In the first case we notice that Extr_{Cross} knows openings of the commitment C_z to two different values. Thus, it can use an extractor $\mathsf{Extr}_{\mathsf{ck}}$ to compute the equivocality trapdoor for the commitment scheme and compute r_{ZKB} as $\mathsf{Eval}(\tau, x_{ZKB}, (C_x, \sum_{i\in[0,|x|-1]} 2^i x[i], \sum_{i\in[0,|x|-1]} 2^i r_{x[i]}))$. In other words, the extraction algorithm Extr_{Cross} uses the trapdoor to open the commitment from the statement to the value x_{ZKB} for which $F(x_{ZKB}) = 1$. This means that the returned values are a valid witness for the proven statement. In the second case we know that:

$$\sum_{i\in[0,|x|-1]} 2^i(x[i] \oplus \alpha_i) = x_{e+2}.$$

This means that $r^* = \sum_{i\in[0,|x|-1]} 2^i r_{x[i]}$ is an opening of the commitment C_x to a value x' for which we know that $x' \oplus x_e \oplus x_{e+1} = x_{e+2}$. It follows that $x' = x_e \oplus x_{e+1} \oplus x_{e+2} = x_{ZKB}$. Thus, in this case Extr_{Cross} can also return $w^* = (x_{ZKB}, r^*)$, which ends the proof. □

Theorem 4.3.4. *Assuming the ZKB++ protocol is HVZK and the commitment scheme is perfectly hiding, then Construction 4.3.2 is a HVZK Σ-protocol for Statement 4.3.1.*

Proof. We will show how to construct a simulator $\mathcal{SIM}$ that on input of a statement as in Statement 4.3.1 with commitment C_x, outputs a transcript (a, e, z). The simulator works as follows:

- It runs the simulator for ZKB++ receiving a transcript (a', e, z'), where z' contains the shares x_e and x_{e+1}.
- $\mathcal{SIM}$ chooses randomness r_{x_e}, $r_{x_{e+1}}$ and computes commitments $C_{x_e} = \mathsf{Com}(x_e, r_{x_e})$, $C_{x[i]_{e+1}} = \mathsf{Com}(x_{e+1}, r_{x_{e+1}})$. Note that the openings for those commitments are part of the response z. Commitments to the bits of $x[i]$ and to x_{e+2} are not opened, so the simulator can compute the commitments $C_{x[i]}$ and $C_{x_{e+2}}$ as follows.
- For $i \in [1, |x|-1]$ it computes commitments $C_{x[i]}$ as commitments to 0. For $j = 0$ it uses the homomorphic properties of the commitment scheme to compute $C_{x[j]}$ such that $C_x = \prod_{i=0}^{|x|-1}[2^i]C_{x[i]}$.
- It then chooses a randomness r_z and computes

$$C_{x_{e+2}} = \prod_{i\in[0,|x|-1]} C_{2^i\cdot(x[i]\oplus x_e[i]\oplus x_{e+1}[i])} \cdot \mathsf{Com}(0, -(r_z)).$$

- $\mathcal{SIM}$ runs the simulator for the Σ-protocol for the algebraic statement receiving (a'', e'', z''). Note that since this simulator should work for all possible challenges, there is a non-negligible probability that $e'' = e$. Otherwise, $\mathcal{SIM}$ just restarts it.
- Finally, $\mathcal{SIM}$ sets $a = (a', a'', C_{x_1}, C_{x_2}, C_{x_3}, \{C_{x[i]}\}_{i\in[0,|x|-1]})$ and $z = (z', z'', r_{x_e}, r_{x_{e+1}}, r_z)$

□

Lemma 4.3.4. *The soundness error of the Σ-protocol presented in Construction 4.3.2 is* $2/3$*.*

Proof. It is implied directly from 3-special soundness of the protocol (Theorem 4.3.3) and the challenge space of cardinality 3. $\square$

4.3.4 Optimization for large input space

We now show how to reduce the number of public key operations to be proportional to the message space of the commitment scheme and independent of the input size of the function F, which is desirable when the input to the ZKB++ circuit is large and required if we want to use Construction 4.3.2 for such circuits. This optimization will utilize the properties of modular arithmetics.

Let $F(m) = 1$ be a function that has to be proven in the cross-domains, and let $m \geq q$ where $[0, q-1]$ is the message space of the commitment scheme. The prover proceeds as follows. Instead of committing to m, it commits to $C = \mathsf{Com}_q(m')$, where m' satisfies $m' < q$ and $m = k \cdot q + m'$ and proves the relation between m and m' as part of F. Let the original cross-domain statement be described as: $\mathcal{P}\{m : (F(m) = 1) \wedge (C_m = \mathsf{Com}_q(m, r))\}$. Then the optimized version is defined as:

$$\mathcal{P}\{m, m', k : (F_{opt}(m, m', k, q) = 1) \wedge C_m = \mathsf{Com}_q(m', r)\},$$

where

$$F_{opt}(m, m', k, q) = (F(m) = 1 \wedge (m = m' + k \cdot q)) .$$

It is easy to see that C_m can be opened either to m, or to m', as both values are equal modulo q. Furthermore, the prover indeed proves that m and m' are equal modulo q. Finally, the prover proves that m' is the value committed to in C_m.

This solution requires us to create an arithmetic circuit as part of the statement proven by ZKB++. Fortunately, this is a standard integer multiplication circuit of a number $k < |x|$ and $q = O(\lambda)$. We can view such a multiplication as the addition of q, k-bit numbers. Since adding two k-bit numbers can be done using $O(k)$ gates, it follows that this multiplication can be done using $O(k \cdot q)$ gates, which is also $O(|x| \cdot \lambda)$. In particular, we have that this can be done using $O(|F| \cdot \lambda)$ gates, because $|x| < |F|$. Thus, the asymptotic number of symmetric operations remains the same and we only introduce a slight overhead using this technique.

4.3.5 Efficiency

We will discuss the computation overhead and increase in the proof size of our techniques. We will compare both constructions for Statement 4.3.1 and focus only on public key operations, i.e. exponentiations and multiplications in the used group $\mathbb{G}$ of order q, where $\ell q = \log q$. Let us assume that we run both protocols ρ times for input x and that we use Pedersen commitments. Moreover, we will by

ℓ_{ZKB} denote the proof size of the ZKB++ protocol, by ℓ_Σ the proof size of the Σ-protocol for Eq. (4.1) and by $\ell_\mathbb{G}$ the size of group elements.

In such a case the proof size of Construction 4.3.1 is $\rho \cdot (\ell_{ZKB} + \ell_\Sigma + 4 \cdot |x| \cdot \ell_\mathbb{G} + 2 \cdot |x| \cdot \ell q)$, which asymptotically is $O(|x| \cdot \rho)$. Construction 4.3.2 was introduced to decrease this by depending less on Σ-protocols for algebraic statements and using the homomorphic properties of the commitment scheme. When executed in parallel, the proof size is $\rho \cdot (\ell_{ZKB} + (3 \cdot |x| + 3) \cdot \ell_\mathbb{G} + 2 \cdot |x| \cdot \ell q + 3 \cdot \ell q)$, which is better but still $O(|x| \cdot \rho)$. Fortunately, we have shown that certain parts of the computations can be reused throughout every round. Therefore, for an optimized version of Construction 4.3.2 we end up with a proof size of $\rho \cdot (\ell_{ZKB} + 3 \cdot \ell_\mathbb{G} + 3 \cdot \ell q) + |x| \cdot (3 \cdot \ell_\mathbb{G} + 2 \cdot \ell q)$, which is $O(|x| + \rho)$.

To compute the proof in Construction 4.3.1 we have to compute $4 \cdot \rho \cdot |x|$ commitments and compute the proof for statement Eq. (4.1), which strongly depends on the instantiation but it requires at least $O(\rho \cdot |x|)$ exponentiations. Computing commitments to bits costs one exponentiation and one multiplication. In the end, for this construction we require $O(\rho \cdot |x|)$ exponentiations. In case of Construction 4.3.2 we have to compute $|x| \cdot (3 \cdot \ell_\mathbb{G}) + \rho \cdot 3 \cdot \ell_\mathbb{G}$ commitments and $2 \cdot |x|$ exponentiations for the proof for statement Eq. (4.2). We also have to compute the commitment C_z, which requires us to compute $|x| \cdot \rho$ multiplications in $\mathbb{G}$. Given the fact, that we assumed that $|x|$ is of the size of $\log q$ it follows that the cost of those multiplications is comparable with ρ exponentiations in $\mathbb{G}$. It follows, that for this construction we require only $O(|x| + \rho)$ exponentiations.

4.4 NIZK OR-proofs in cross-domains

Proofs of partial knowledge [49], also known as OR-proofs, allow to efficiently prove only a part of a statement, without revealing, which part has been proven. Below we show how to prove the most simple OR-statement in cross-domains, which can be used as a basis for proving more complex statements.

Statement 4.4.1. *Prove knowledge of x_1 s.t. $F(x_1) = 1$ or knowledge of x_2 such that $y = g^{x_2}$, where F is an arithmetic circuit.*

We are going to use ZKB++ for proving the first part and the standard Schnorr proof for the second part. Since the both parts of the proof system are Σ-protocols, a challenge e will be "distributed" between these parts as $e = e_1 + e_2$. Assume $e_1 \in \mathbb{Z}_p$ and $e_2 \in \mathbb{Z}_q$, where $p > q$. The prover generates e_1 or e_2 and derives the remaining element based on e. Both e_1 and e_2 should have the same distribution regardless of the part that is being proved. Depending on which part is being proved, we proceed as follows. Given $e \in \mathbb{Z}_p$, to prove the first part the prover picks $e_2 \leftarrow_R \mathbb{Z}_q$ and computes $e_1 = e - k \cdot e_2$, where $k = \lfloor p/q \rfloor$. Given $e \in \mathbb{Z}_p$, to prove the second part the prover picks $e_1 \leftarrow_R \mathbb{Z}_p$ and computes $e'_2 = e - e_1 \in \mathbb{Z}_p$. To preserve the distribution of e_2, the prover performs rejection sampling: it further computes the largest p' that satisfies $p' = k \cdot q \leq p$ for integer k and rejects and regenerates e_1 if $e_1 > \mathbb{Z}_{p'}$, otherwise $e_2 \leftarrow e'_2 (\bmod q)$. It is easy to see that the probability of rejection is at most 1/2, and e_1 and e_2 are distributed identically regardless of which part has been proven.

Remark 4.4.1. *If $p >> q$, it suffices to stay in $\mathbb{Z}_p$ and convert an element from $\mathbb{Z}_p$ to $\mathbb{Z}_q$ by taking its residue.*

4.5 Conclusion

Zero-knowledge proofs are an essential component in various protocols, including payment, electronic voting, anonymous credential systems. Proofs based on algebraic groups and for arithmetic circuits represent two different domains. In this work, we presented an efficient Σ-protocol in cross-domains, which can be used to prove the possession of standard RSA/DSA signatures. Moreover, the protocol can be executed non-interactively using the Fiat-Shamir transformation. It follows, that our results can be applied to build round-optimal and concurrent-secure anonymous credentials based on standard signature schemes. Our techniques are especially beneficial when applied for large circuits and when the prover's running time is critical. As future work, it would be interesting to explore whether the approach by Ames et al. [4] can be used to achieve yet more efficient and compact NIZK proofs in cross-domains.

The prover knows x to a public function F, such that $y = F(x)$, where y is public. t denotes the number of (parallel) rounds.

$\underline{p \leftarrow Prove(x)}$

1. (Commit step) For each round $\rho \in [1, t]$: Sample random tapes $k_1^{(\rho)}$, $k_2^{(\rho)}$, $k_3^{(\rho)}$ and simulate the MPC protocol to get an output view $\mathsf{View}_j^{(\rho)}$ and output share $y_j^{(\rho)}$.

$$\begin{aligned}
&(x_1^{(\rho)}, x_2^{(\rho)}, x_3^{(\rho)}) \leftarrow \mathsf{Share}(x, k_1^{(\rho)}, k_2^{(\rho)}, k_3^{(\rho)}) \\
&= (G(k_1^{(\rho)}), G(k_2^{(\rho)}), x \oplus G(k_1^{(\rho)}) \oplus G(k_2^{(\rho)})) \\
&\mathsf{View}_j^{(\rho)} \leftarrow \mathsf{Upd}(...\mathsf{Upd}(x_j^{(\rho)}, x_{j+1}^{(\rho)}, k_j^{(\rho)}, k_{j+1}^{(\rho)})...) \\
&y_j^{(\rho)} \leftarrow \mathsf{Output}(\mathsf{View}_j^{(\rho)})
\end{aligned}$$

   Commit $D_j^{(\rho)} \leftarrow H'(k_j^{(\rho)}, \mathsf{View}_j^{(\rho)})$, let $a^{(\rho)} = (y_1^{(\rho)}, y_2^{(\rho)}, y_3^{(\rho)}, D_1^{(\rho)}, D_2^{(\rho)}, D_3^{(\rho)})$ and let $a = a^{(1)}, \dots, a^{(t)}$ be the output of this step.
2. Compute the challenge: $e \leftarrow H(a)$. Interpret e such that for $\rho \in [1, t]$, $e^{(\rho)} \in \{1, 2, 3\}$.
3. (Response step) For each round $\rho \in [1, t]$: let $b^{(\rho)} = (y_{e^{(\rho)}+2}^{(\rho)}, D_{e^{(\rho)}+2}^{(\rho)})$ and set $z^{(\rho)} \leftarrow (\mathsf{View}_{e^{(\rho)}+1}^{(\rho)}, k_{e^{(\rho)}}^{(\rho)}, k_{e^{(\rho)}+1}^{(\rho)})$. If $e^{(\rho)} \neq 1$, add $x_3^{(\rho)}$ to $z^{(\rho)}$. Let $\mathbf{r} \leftarrow [(b^{(1)}, z^{(1)}), \dots, (b^{(t)}, z^{(t)})]$ be the output of this step.
4. Output $p \leftarrow [e, \mathbf{r}]$.

$\underline{b \leftarrow Verify(y, p)}$:

1. (Reconstruct step) For each round $\rho \in [1, t]$: Run the MPC protocol to reconstruct the views. In particular: compute $x_{e^{(\rho)}}^{(\rho)}, x_{e^{(\rho)}+1}^{(\rho)}$ using $z^{(\rho)}$ as part of $\mathbf{r}$ of p in one of the following ways: $x_1^{(\rho)} \leftarrow G(k_1^{(\rho)})$, $x_2^{(\rho)} \leftarrow G(k_2^{(\rho)})$, or $x_3^{(\rho)}$ given as part of $z_{(\rho)}$.
   Obtain $\mathsf{View}_{e^{(\rho)}+1}^{(\rho)}$ from $z_{(\rho)}$ and compute
   $\mathsf{View}_e^{(\rho)} \leftarrow \mathsf{Upd}(...\mathsf{Upd}(x_j^{(\rho)}, x_{j+1}^{(\rho)}, k_j^{(\rho)}, k_{j+1}^{(\rho)})...)$, $y_{e^{(\rho)}}^{(\rho)} \leftarrow \mathsf{Output}(\mathsf{View}_{e^{(\rho)}}^{(\rho)})$, $y_{e^{(\rho)}+1}^{(\rho)} \leftarrow \mathsf{Output}(\mathsf{View}_{e^{(\rho)}+1}^{(i)})$, $y_{e^{(\rho)}+2}^{(i)} \leftarrow y \oplus y_{e^{(\rho)}}^{(i)} \oplus y_{e^{(\rho)}+1}^{(i)}$.
   Compute the commitments for views $\mathsf{View}_{e^{(\rho)}}^{(\rho)}$ and $\mathsf{View}_{e^{(\rho)}}^{(\rho+1)}$.
   For $j \in \{e^{(\rho)}, e^{(\rho)} + 1\}$: $D_j^{(\rho)} \leftarrow H'(k_j^{(\rho)}, \mathsf{View}_j^{(\rho)})$.
   Let $a'^{(\rho)} = (y_1^{(\rho)}, y_2^{(\rho)}, y_3^{(\rho)}, D_1^{(\rho)}, D_2^{(\rho)}, D_3^{(\rho)})$ and note that $y_{e^{(\rho)}+2}^{(\rho)}$ and $D_{e^{(\rho)}+2}^{(\rho)}$ are part of $z_{(\rho)}$. Let $a' = (a'^{(1)}, \dots, a'^{(t)})$ be the output of this step.
2. Compute the challenge: $e' \leftarrow H(a')$. If $e' = e$, output Accept, otherwise output Reject.

Figure 4.5: Non-interactive ZKB++ (37).

5 Single Secret Leader Election from MPC

5.1 Introduction

In 2008, Bitcoin [99] laid the foundation for the increasingly important areas of cryptocurrencies and distributed ledgers. One of the main advantages of such technology is that there is no single central authority that controls transaction flow (*censorship resistance*). Anyone can access a public ledger, which is a sequence of blocks that contains transactions. For example, in Bitcoin, participants called “miners” are randomly selected to produce and append a new block to the chain. This selection process relies on the “proof-of-work” concept (PoW). To be able to append a block to the chain, the participant has to find a value, such that a pseudo-random function (cryptographic hash function) is evaluated below some threshold. The threshold is adapted by the protocol at intervals to yield an average time span of 10 minutes between blocks. Although PoW allows to remove a central authority from the model and to find a consensus of what blocks constitute the correct chain, most of the computation the miners do is solving this hash puzzle. Recent reports estimate that the energy used to keep the Bitcoin network running exceeded the power supply of a small state [103].

To avoid extreme energy consumption induced by PoW protocols, an alternative approach, “proof-of-stake” (PoS), has been proposed. Here, the probability of being selected for appending the chain depends on the stake a party owns. It does not matter whether the party owns an account with some stake v, or several accounts whose accumulated stake amounts to v. The protocol consensus works as long as the majority of all stake is controlled by honest users.

In cryptocurrencies based on proof-of-stake [72, 91, 62, 65], a single party that produces a block is chosen randomly from a set of participants, called validators (which is the equivalent to miners in a PoW protocol). In a PoS cryptocurrency there could be potentially thousands or millions users, who may come and go. It is up to a PoS protocol to determine and fix a relatively small (typically tens or hundreds) set of validators [72] and a time window, in which a selected validator can append a blockchain. To create a consistent picture for all validators, this selection has to be deterministic, but pseudo-random – properties often achieved by relying on Verifiable Random Functions (VRF). However, if an adversary knows in advance which of the validators is selected, it can launch a targeted attack and cause a denial-of-service.

Previous approaches to solving this issue aim to run the selection process in private, with the selected participant publishing a proof alongside the block. Until recently, these approaches failed to guarantee only a single participant to be chosen [72]. After much interest in a solution that provides such a guarantee [118], Boneh et al. proposed a formal definition and several instantiations of a Single Secret Leader Election [18].

The primary motivation of having a single leader is a simple consensus design, as there are no forks in the blockchain. This property encourages the leader to solely perform heavy computations, which may even exceed the running time of SSLE and/or require multiple cores. For example, the leader's task may consist of prover-heavy computations, whereas verification is very fast (SNARKs). Many protocols assume uniqueness, and it is easy to update them with a SSLE solution. They may require a full redesign if the leader uniqueness assumption no longer holds.

5.1.1 Our contribution

1. In this work, we propose a framework for constructing an efficient Single Secret Leader Election (SSLE), which relies on secure multi-party computation (MPC). We formulate a simulation-based definition of the SSLE problem. We first develop, step by step, an efficient t-threshold SSLE scheme that is based on Shamir's secret sharing in the random oracle model. We prove that our construction is secure in the honest-but-curious and malicious adversary models. For the latter, we additionally assume DDH. For N parties, the leader election requires $O(\log N)$ communication rounds and $O(N)$ of basic operations on the underlying primitives. Furthermore, we instantiated our SSLE scheme using the MPC framework by Wang et al. [129], which is secure against any number of malicious parties and is more scalable, but requires all parties to be online.
2. Our SSLE scheme can handle arbitrary stake distributions very efficiently. For N parties and the overall sum of their stake units S, our construction achieves $O(N \log S)$ cost of the election. Compared with a standard multi-registration technique, in which a party registers multiple times for the election proportionally to her stake, this cost may go up to $O(S)$, which makes our solution exceptionally efficient if $N \ll S$.
3. We implemented and microbenchmarked our solution using two different MPC frameworks. The performance evaluation indicates that our DDH-based SSLE protocol can be used in practical scenarios up to 30-40 parties when instantiated with the textbook $O(N^2)$ techniques using the verifiable secret sharing scheme (VSS). Furthermore, we implemented our SSLE in the MPC framework based on garbled circuits [129]. The overall time to set up and complete the protocol for 128 parties in a practical scenario is less than 7 minutes.

5.1.2 Related work

The idea of proof-of-stake was first discussed on the Bitcoin forum[1] in 2011. Kiayaias et al. presented a provably-secure PoS protocol "Ouroboros" in CRYPTO 2017 [91], in which the participants that produce the blocks are elected publically.

[1]https://bitcointalk.org/index.php?topic=27787.0 (accessed 14.06.2021)

Such a leader election may be public as in Ouroboros or private as in Algorand [72]. In a private leader election, each node needs to check whether it will be the next leader using its private information but then can prove to others using only public information that it is indeed the next leader. Such a design makes it impossible for others to predict and carry out DoS attacks against the next leader until it is too late.

Algorand achieves this private leader election using Verifiable Random Functions, for which a participant has to prove the outcome to be below a certain threshold. This, however, can result in either no participant or multiple participants being elected. Another protocol employing a private leader election has been formalized by Ganesh et al., who presented a privacy-preserving protocol called Ouroboros Praos [65], which does not guarantee existence and uniqueness of the leader either.

To mitigate these shortcomings of previous private leader elections, a problem statement of a single secret leader election was first posed at a GitHub page [118] in the form of a research proposal in the context of the Filecoin cryptocurrency. The protocol's goal is to elect a *single* leader among a finite set of participants. Informally, such a protocol consisting of n participants has to meet the following requirements: *fairness* – the probability of a particular party being elected should be proportional to her power or stake, *secrecy* – only the elected leader should learn the result of the protocol, *(public) verifiability* – the elected leader should be able to prove the leadership to the other participants or observers by showing a proof-string, *unpredictability* – no set of participants smaller than a threshold m of n should be able to predict the outcome of the protocol with a probability greater than a negligible factor. To tolerate sporadic drop outs, the protocol should satisfy *liveness* – it should terminate as long as the honest majority is participating. Moreover, the protocol should be reasonably efficient, that is, on-chain $O(\log n)$ bits per block, $O(n)$ communication complexity (per active party).

Following this call, Boneh et al. [18] formalized the problem of Single Secret Leader Election (SSLE) and presented three constructions: 1) a feasibility result based on indistinguishability obfuscation, 2) a construction based on threshold fully homomorphic encryption (TFHE), and 3) a construction based on DDH that achieves a weaker notion of security. To the best of our knowledge, this is the only work on the SSLE problem in the literature, which proposes provably secure SSLE constructions, so we will directly compare our solution to the latter two constructions in [18].

5.1.3 Comparison with Boneh et al. [18]

SSLE is a protocol between N parties, who secretly and randomly elect a leader among them. SSLE protocols are sought to be used as a subroutine in PoS protocols. Intuitively, an adversary may corrupt up to a specific number of parties

Construction	*Assumptions*	*Security notion*	*Number of rounds*	*Computation / Communication*
Obfuscation-based [18]	iO	full, non-interactive	0 + beacon	feasibility result
TFHE-based [18]	TFHE, weak PRF	full, t-threshold	1 + beacon	depends on a particular instance
DDH-based [18]	ROM, DDH	weak	1 + beacon	$O(\sqrt{N})$ pub. op. / $O(\sqrt{N})$ group el.
Our 5.4.1	ROM, DDH	full, t-threshold	$O(\log N)$	$O(N)$ MPC op.
Our 5.5.1	ROM	full	$O(\log N)$	$O(N)$ MPC op.

Table 5.1: Comparison of SSLE protocols, assuming all N users participate in election, amortized per one election

and observe public messages during an election, but should not be able to predict a chosen leader up to a point when the leader reveals herself (and does something useful, e.g. appends a blockchain), nor should the adversary be able to influence the election by sending malformed messages, or pretending that it is the elected leader when it is not. In order to let anyone to verify transactions in a chain, a leader has to append new transactions along with a proof of leadership, and the verification algorithm should use only the data stored on the blockchain. Therefore, in the context of PoS systems, we distinguish on-chain and off-chain messages sent by the parties according to a SSLE scheme. A multi-round SSLE protocol is not required to post intermediate messages in a blockchain, as long as the parties agree on the final on-chain message that should be appended to the chain. We may refer to on-chain messages as to a public state. An SSLE scheme may or may not require all N parties to be online during the election phase. If in a PoS protocol the parties' stakes are public, an SSLE scheme can be naturally used out-of-box. If the stakes are private, a PoS protocol and a SSLE scheme have to agree in advance on the setup parameters and how the PoS protocol supplies the inputs to the SSLE scheme. An SSLE scheme may use a public randomness beacon R that becomes available to the parties before each election starts.

We begin by first comparing how arbitrary stake distributions are handled in [18] and our work. While a scenario with equal stakes is easier to analyze, in practice one has to also account for arbitrary stake distributions and how they affect the overall performance of the scheme. Boneh et al. [18] suggest a multi-registration technique (one registration corresponds to one unit of stake) to address arbitrary stake distributions, which makes the associated costs grow linearly with the user's stake. In contrast, our construction offers a more efficient tree-based solution to this setting with the associated costs grow logarithmically in the total stake S of participating parties. In the rest of our comparison, we will assume equal stakes.

The TFHE-based SSLE [18] uses TFHE [19] as a building block, which in turn is based on fully homomorphic encryption (FHE) [70]. Its security relies on learning with errors assumption (LWE) [111]. FHE [70] is parametrized by a circuit depth that the scheme can handle without bootstrapping. The TFHE-based SSLE is instantiated with a circuit multiplicative depth parameter $D := d + \log\log N + 1$, where d is a circuit depth of a low-depth block cipher f optimized for the FHE setting, in particular Boneh et al. [18] suggest that it can be instantiated as low as $d = 5$ with 127 bit blocks [56]. The evaluation of a circuit on encrypted data in FHE [70] has the complexity $\tilde{O}((nD)^{\omega})$ of field operations per gate, where $\omega < 2.3727$ is the matrix multiplication exponent, D is the depth of a circuit, and n the dimensional parameter, which depends on the security parameter λ. Depending on how the underlying building blocks are instantiated, the TFHE-based SSLE scheme offers various trade-offs in terms of assumptions and space/runtime. For more details, we refer the reader to [18, 19]. After a random beacon R is revealed, the parties engage in one round of communication to determine a leader.

The DDH-based construction [18] relies on more lightweight components than the TFHE-based one, but achieves only a weaker security notion of unpredictability. More specifically, for N parties, a potential leader is picked from a subset of data elements representing $O(\sqrt{N})$ parties, and an adversary is asked to predict a leader within this subset (excluding parties controlled by the adversary), whereas in the full notion of unpredictability an adversary has to guess a purported leader from the set of N parties. To register for an election, a party has to update and shuffle $O(\sqrt{N})$ group elements available and provide a NIZK proof of honest shuffling and re-randomization. These messages have to be considered on-chain, so that everyone could verify the outcome of an election. One can trade efficiency for security in this scheme by changing the number of elements that has to be reshuffled during registration. After a random beacon R is revealed, a leader can be determined locally, thus requiring no further communication.

Our SSLE construction runs $O(\log N)$ rounds of communication and does not use a randomness beacon R. A party is required to post as little as $O(1)$ of information on-chain during registration. From [18], we use game-based definitions as a starting point for our definitions, and the technique to prevent duplicate key attacks.

We compare our constructions with possible instantiations of Boneh et al. [18] in Table 5.1. For completeness, we included the obfuscation-based feasibility result in [18]. This construction is the only one among the discussed ones that does not require the leader to re-register in future elections and the only one non-interactive, that is the outcome of election is known right after the randomness beacon is revealed. By pub. op. we denote the number of public key operations such as exponentiation, by MPC op. we denote basic MPC operations such as multiplication.

In all discussed schemes, the leader has to re-register before next election, since she reveals a secret that was generated and used for the registration.

Randomness beacon A randomness beacon can be implemented in several ways, ranging from harnessing randomness from financial data [46, 21, 14] to cryptographic delay functions [95, 17], and specialized systems [75, 34, 122]. In blockchain based applications, however, the most frequent source of randomness is the output of *verifiable random functions* [98] (VRFs). Bootstrapped with one initial random value, the leader of each election can publish a new, verifiable random value based on the previous one without additional rounds of communication. One possible instantiation is based on BLS signatures [20], which assume random oracles and the intractability of the computational Diffie-Hellman problem in a gap Diffie-Hellman group and require a single exponentiation and a single call to the random oracle per signature and the verification additionally requires a single pairing operation.

5.1.4 On the practicality of our SSLE framework

The number of validators depends on the PoS protocol and can vary from dozens to a few hundred and in limited cases thousands. It does not necessarily correlate with the total number of users. Stake disbalances also vary, and therefore they need to be approximated in our framework by a tree of a sufficient height (Section 5.6.1). Our tree optimization technique has a better effect when applied to a smaller set of validators.

In our SSLE framework, we rely on existing MPC techniques. If one comes up with a more efficient MPC protocol than the ones used in our constructions, it will help to further improve the running time of the SSLE.

Chapter Outline The rest of the chapter is organized as follows. Definitions are in Section 5.2 and Section 5.8. In Section 5.3, we discuss how naive attempts fail to solve the problem of SSLE and present another attempt based on a new two-party primitive called *Oblivious Select*. In Section 5.4, we develop our attempt further and present our SSLE protocol from DDH based on MPC. In Section 5.5, we instantiate our construction using the MPC framework on boolean circuits by Wang et al. In Section 5.6, we discuss some practical considerations for SSLE. In Section 5.7, we evaluate the performance of our SSLE instantiated with two different MPC frameworks in a practical scenario. Security analysis of our SSLE framework is in Section 5.9. Finally, we conclude in Section 5.10.

5.2 Definitions

5.2.1 Preliminaries

DDH Assumption (54) Let g be a generator of a group G of a prime order q. For any probabilistic polynomial time (PPT) machine Adv and $(x, y, z) \leftarrow (\mathbb{Z}_q)^3$,

$$|Pr[\mathsf{Adv}(g, g^x, g^y, g^{xy}) = 1] - Pr[\mathsf{Adv}(g, g^x, g^y, g^z) = 1]| \leq negl(\lambda).$$

Secret Sharing Secret sharing schemes allow a dealer to share a secret s among parties such that later a qualified set of parties can jointly reconstruct s, whereas a non-qualified set of parties learns no information about it. We use Shamir's Secret Sharing [115], which is a t-threshold scheme. Let $\mathcal{P}_1, ..., \mathcal{P}_N$ be N parties and there is a threshold $t < N/2$. In Shamir's secret sharing scheme, a secret can be shared among N parties such that $t + 1$ parties can reconstruct it, whereas t parties learn no information about the secret. We denote $[x]$ a Shamir sharing of x in a prime field $\mathbb{Z}_q$, for which each party $\mathcal{P}_i$ gets a secret share $x_i \in \mathbb{Z}_q$. For this scheme to work, it is required that $N < q$. We denote Share a protocol to share a secret x as $[x]$, and Rec to reconstruct x from $[x]$. Whereas Shamir's Secret Sharing is only secure against passive adversaries, Verifiable Secret Share (VSS) schemes [107] can protect against active adversaries.

Communication and adversary models We assume secure point-to-point communication channels between parties. An adversary is allowed to corrupt up to t parties. We consider two models of adversaries: honest-but-curious and malicious. In the honest-but-curious model, adversaries follow the protocol honestly and try to learn as much as possible from observed communication by corrupted parties. In the malicious model, the parties controlled by an adversary can stop communicating or send arbitrary messages to other parties, not necessarily following the prescribed protocols.

Secure Multi-Party Computation (MPC) MPC allows a set of parties $\mathcal{P} = \{\mathcal{P}_1, ..., \mathcal{P}_N\}$ to jointly compute a function on their private inputs in a privacy-preserving manner [131]. Our SSLE scheme is based on MPC.

We borrow the standard definitions of $VIEW$ and t-Privacy from [3].

Definition 5.2.1 ($VIEW$)**.** *Let $\mathcal{P} = \{\mathcal{P}_1, ..., \mathcal{P}_N\}$ engage in a protocol Π that computes function $f(\mathsf{in}_1, ..., \mathsf{in}_N) = (\mathsf{out}_1, ..., \mathsf{out}_N)$. Let $VIEW_\Pi(\mathcal{P}_i)$ denote the view of participant $\mathcal{P}_i$ during the execution of protocol Π. More precisely, $\mathcal{P}_i$'s view is formed by its input and internal random coin tosses r_i, as well as messages $m_1, ..., m_l$ passed between the parties during protocol execution:*

$$VIEW_\Pi(\mathcal{P}_i) = (\mathsf{in}_i, r_i; m_1, ..., m_l).$$

We denote the combined view of a set of participants $\mathcal{I} \subseteq \mathcal{P}$ (i.e., the union of the views of the participants in I) by $VIEW_\Pi(\mathcal{I})$.

Definition 5.2.2 (t-Privacy)**.** *We say that protocol* Π *is* t*-private in the presense of honest-but-curious adversaries if for all* $\mathcal{I} \subset \mathcal{P}$ *with* $|\mathcal{I}| \leq t < N$ *there exist a PPT simulator* $\mathcal{S}_{\mathcal{I}}$ *such that*

$$\{\mathcal{S}_{\mathcal{I}}(\mathsf{in}_{\mathcal{I}}, f(\mathsf{in}_1, ..., \mathsf{in}_N))\} \equiv \{\mathit{VIEW}_{\Pi}(\mathcal{I}), \mathsf{out}_{\mathcal{I}}\},$$

where $\mathsf{in}_{\mathcal{I}} = \bigcup_{\mathcal{P}_i \in \mathcal{I}}\{\mathsf{in}_i\}$*,* $\mathsf{out}_{\mathcal{I}} = \bigcup_{\mathcal{P}_i \in \mathcal{I}}\{\mathsf{out}_i\}$*, and* $\equiv$ *denotes computational indistinguishability.*

In the malicious setting, the subset $\mathcal{I}$ of honest-but-curious parties in Def. 5.2.1 and 5.2.2 is replaced with an equal-sized subset of malicious PPT parties $\mathcal{I}^M$, and the protocol Π is replaced with Π^M, its maliciously-secure version.

We instantiate our SSLE scheme using the following underlying protocols:

1. the VSS-based MPC protocols [107, 110, 69, 67, 50, 36], in which secrets are shared between the parties using Shamir's secret sharing scheme:
   - protocols for adding shares, substracting, and multiplying by a scalar: $[x] + [y]$, $[x] - [y]$, $[\alpha \cdot x]$,
   - RndFld to generate a share of a random field element in $\mathbb{Z}_p$,
   - RndBit to generate a share of a random bit,
   - Mul to compute $[x \cdot y]$ given $[x]$ and $[y]$.
2. garbled circuit based MPC [129] on boolean circuits, where each party can privately input her input to a computing circuit.

5.2.2 Single Secret Leader Election

We consider the following problem. Given a set of N parties. The parties do some interactive pre-computation. Then, each party can run a local function that takes the transcript as input to determine whether it is the leader or not. The leader can show a proof that it is the leader.

Informally, we require the leader selection protocol to satisfy the following properties: a) there should be a unique leader (uniqueness), b) only the leader and no one else should be able to convince other parties that it is the leader (soundness), c) the parties should have equal chances of becoming the leader (fairness).

Game-based formulation of the SSLE problem We first present the game-based definition. Our syntax and security properties of SSLE are based on that of [18], with a slight difference that we do not have an external source of randomness (random beacon) and we allow multiple rounds of communication between the parties during the election, whereas the definition of SSLE in [18] allows a single round of communication. Due to page limits, we postpone the game-based definition to Section 5.8.

Simulation-based definition of the SSLE problem We now formulate the SSLE problem as an ideal functionality $\mathcal{F}_{\mathsf{SSLE}}^{N,\ell,c}$, which is presented in Figure 5.1. In the description of the ideal functionality, we denote election id as *eid*, and registration numbers as C_i. We then show that the simulation-based definition implies the game-based one.

Our modeling of the ideal functionality $\mathcal{F}_{\mathsf{SSLE}}^{N,\ell,c}$ for N parties with an adversary statically corrupting up to t of them is influenced by the game-based definition (Definition 5.8.1), which defines the registration and verification algorithms that surround the election itself. We follow the same approach and define messages in the ideal functionality for registration, election, and their verification.

In $\mathcal{F}_{\mathsf{SSLE}}^{N,\ell,c}$, the parties send messages to the ideal functionality that correspond to a specific stage of the election. First, the parties register for an election with id *eid* via sending register messages containing the registration number C. They receive notifications from the ideal functionality for every registered party. To verify registration, the parties send messages regVerify to the ideal functionality, which outputs 1 if all registered numbers are distinct, otherwise it outputs 0 and the execution of $\mathcal{F}_{\mathsf{SSLE}}^{N,\ell,c}$ stops. If regVerify returned 1, the parties participate in the election by sending messages elect to $\mathcal{F}_{\mathsf{SSLE}}^{N,\ell,c}$, which returns one of the registered numbers as the elected number. Finally, the parties can verify whether some party $\mathcal{P}_i$ is the elected leader by sending a message verify with the identifier for $\mathcal{P}_i$ and the elected number.

Next, we discuss some of the design choices that we made in $\mathcal{F}_{\mathsf{SSLE}}^{N,\ell,c}$:

1. With the explicit inputs associated to parties, the definition naturally captures the adversarial ability to register multiple parties using the same private material and thereby break the uniqueness property.
2. The result of election is returned to the parties as one of the numbers, used for the registration. In this way we model the information leakage, which suggests an efficient way of running multiple elections by the same parties. To run a subsequent election, the leader has to simply re-register, while other parties can keep their previously registered numbers.

Intuitively, the security properties from the game-based definitions are captured in the ideal functionality $\mathcal{F}_{\mathsf{SSLE}}^{N,\ell,c}$ as follows:

1. **Uniqueness** - provided by answering regVerify messages, which excludes the case that two parties register the same number, and elect messages are answered with exactly one number.
2. **Unpredictability** - provided by answering elect messages with one of n registered numbers, which are known only to the respective parties. In the beginning, party $\mathcal{P}_i$ sends her input C_i only to the ideal functionality and never discloses C_i to other parties until the election is finished. $\mathcal{P}_i$ discloses her registered number only when $\mathcal{P}_i$ is the elected leader.
3. **Fairness** - provided by answering elect messages by *uniformly at random* selecting one of n registered distinct numbers as the elected value.

$\mathcal{F}_{\mathsf{SSLE}}^{N,\ell,c}$ for a set of parties $\mathcal{P} = \{\mathcal{P}_1, \dots, \mathcal{P}_N\}$, c of which are corrupted by an adversary, consists of the following steps:

- Upon receiving a message $(eid, \mathsf{register}, C)$ from $\mathcal{P}_i$, check if $(eid, \mathcal{P}_i, \cdot)$ or $(eid, elected, \cdot, \cdot)$ is stored. If so, ignore the message. Otherwise, store $(eid, \mathcal{P}_i, C)$. When storing tuples, we write $\mathcal{P}_i$ to denote the party's unique identifier. Send $(eid, registered, \mathcal{P}_i)$ to all parties and the environment.
- Upon receiving a message $(eid, \mathsf{regVerify})$ from $\mathcal{P}_i$, reply 0 if there exist two stored tuples $(eid, \mathcal{P}_j, C_j)$ and $(eid, \mathcal{P}_k, C_k)$ such that $j \neq k$ and $C_j = C_k$. Otherwise, reply 1.
- Upon receiving a message (eid, elect) from $\mathcal{P}_i$, check if there are at least ℓ registered parties that have corresponding stored tuples $(eid, \cdot, \cdot)$. If not, ignore the message, otherwise proceed. Check if $(eid, elected, \mathcal{P}_u, C_u)$ is stored. If not, pick one of the stored tuples $(eid, \cdot, \cdot)$ uniformly at random as $(eid, \mathcal{P}_u, C_u)$, append it as $(eid, elected, \mathcal{P}_u, C_u)$, and send $(eid, elected, C_u)$ to the environment. Send $(eid, elected, C_u)$ to $\mathcal{P}_i$.
- Upon receiving a message $(eid, \mathsf{verify}, \mathcal{P}_j, C)$ from $\mathcal{P}_i$, check if $(eid, elected, \mathcal{P}_u, C_u)$ is stored. If such a tuple exists, reply 1 if $\mathcal{P}_u = \mathcal{P}_j$ and $C_u = C$. In all other cases, reply 0.

Figure 5.1: Ideal functionality $\mathcal{F}_{\mathsf{SSLE}}^{N,\ell,c}$.

We formally prove that the ideal functionality implies the game-based definitions by showing the non-existence of a simulator given any of the game-based attackers.

Proposition 5.2.1. *The ideal functionality $\mathcal{F}_{\mathsf{SSLE}}^{N,\ell,c}$ implies the game-based definitions for uniqueness (Definition 5.8.3), unpredictability (Definition 5.8.4), and fairness (Definition 5.8.5).*

We refer to Section 5.9 for the proof of Proposition 5.2.1 and proofs of subsequent theorems.

In this work, we only consider SSLE schemes with *expiring registration*. In such schemes, in a single SSLE instance elections are run sequentially and the eventual leader has to re-register for subsequent elections. In the remainder of the chapter we will only consider the modified ideal functionality that ensures sequentiality. To this end, the ideal functionality keeps track of the current election id eid^*. As soon as it receives a message with $eid' \neq eid^*$, it stops responding to any further messages with eid^* and updates the current election id to eid'. In contrast to the real world, in the ideal world non-leaders have to register for subsequent elections explicitly using the same registration number C.

5.3 (Non-secret) single leader election constructions

In this section, we start by discussing how naive solutions to the problem of SSLE fail in keeping the leader secret. We then introduce a protocol based on our primitive called *Two-Party Oblivious Select*. While this protocol still does not meet all of our desired properties, it will serve as the base for our SSLE scheme, which we present in Section 5.4.

5.3.1 Naive attempts

Designing a secure SSLE protocol is not a trivial task. Below, we briefly mention three naive leader selection protocols and discuss where they fail to meet our requirements.

Protocol 5.3.1. *Run any secure MPC protocol for N parties to generate a fresh random number (selection phase) and later reveal it and take it modulo N to determine which party is selected (reveal phase).*

Protocol 5.3.2. *Each party commits to a number and sends the commitment to all parties (selection phase). To determine the leader, each party opens the commitment, and the leader is computed as a sum of these N numbers modulo N (reveal phase).*

Protocol 5.3.3. *Let an array of distinct numbers represent the participants in the election. To determine a leader, we randomly permute this array (or sort it according to some unpredictable criteria) (selection phase) and pick the first number as the leader and discard the rest (reveal phase).*

Problem The three naive protocols defined above are *not* secret leader election protocols, as they follow a two-phase pattern: the *selection* phase and the *reveal* phase. After the selection phase is over, the parties have already committed to some leader, which is not yet known to anyone. After the reveal phase, everyone knows who the leader is. The missing intermediate point (the "check" phase) is the one that would allow the leader to learn the outcome exclusively.

Nevertheless, these protocols can serve as the basis for a secret leader election protocol. Our construction is inspired by the idea in Protocol 5.3.3. While there exist cryptographic protocols for multi-party sorting ([78, 104]) that all rely on a pairwise comparison subroutine, we take a more efficient approach: We observe that the order of the discarded numbers does not matter and take this into account when designing our solution. This observation allows us to eliminate the requirement for this comparison subroutine.

5.3.2 Two-party Oblivious Select

In this section, we start introducing the basis for our final SSLE protocol. Note that, while the constructions in this section do not yet meet our requirements and are considered non-secret, they will form the basis of the protocol presented in Section 5.4.

We begin by defining a *two-party Oblivious Select* protocol, whose goal is to secretly select one of two commitments. Once the commitment is selected, the parties can open the selected commitment. This sub-protocol essentially makes use of the observation from the previous section that we can discard any information except the chosen leader.

We define first the probabilistic swap algorithm, denoted as PSwap, and then the probabilistic select algorithm, denoted as PSelect.

Definition 5.3.1 (PSwap)**.** *On input of two commitments C_0 and C_1, draw randomness (r_0, r_1) and a random bit $b \in \{0,1\}$, compute $C'_0 = Com(C_0, r_0)$, $C'_1 = Com(C_1, r_1)$, and output (C'_b, C'_{1-b}).*

Definition 5.3.2 (PSelect)**.** *On input of two commitments C_0 and C_1, draw randomness (r) and a random bit $b \in \{0,1\}$, compute and output $C' = Com(C_b, r)$.*

It is easy to show for both algorithms that if the commitment scheme is hiding, then an adversary cannot find the value of b better than pure guessing (up to a negligible factor).

Lemma 5.3.1. *If the commitment scheme is hiding, then for any pair of commitments (C_0, C_1) and any PPT adversary* Adv*, and security parameter λ,*

$$Pr\begin{bmatrix} (C'_0, C'_1) \leftarrow \mathsf{PSwap}(C_0, C_1), \\ b' \leftarrow \mathsf{Adv}(C_0, C_1, C'_0, C'_1), b' = b \end{bmatrix} \leq 1/2 + negl(\lambda)$$

Lemma 5.3.2. *If the commitment scheme is hiding, then for any pair of commitments (C_0, C_1) and any PPT adversary* Adv*, and security parameter λ,*

$$\begin{aligned} &Pr[C' \leftarrow \mathsf{PSelect}(C_0, C_1), b' \leftarrow \mathsf{Adv}(C_0, C_1, C'), b' = b] \\ &\leq 1/2 + negl(\lambda) \end{aligned}$$

We now describe $\mathsf{OSelect}$, the two-party oblivious select protocol, which is based on the probabilistic swap (PSwap) and probabilistic select ($\mathsf{PSelect}$) algorithms. $\mathsf{OSelect}$ allows parties to select one of the users' commitments without revealing, which commitment is selected.

The protocol consists of the select and the opening phases. The select phase consists of the following steps: Two parties, which we will call Alice and Bob, make their initial commitments public so that both Alice and Bob know (C_A, C_B). In the next step, Alice performs PSwap on (C_A, C_B) and sends the resulting (C_0, C_1) to Bob. Since the commitments are hiding, Bob does not yet know how the two commitments are ordered. Bob proceeds analogously by running $\mathsf{PSelect}$ on (C_0, C_1) and sends the output commitment C' to Alice, which completes the select phase. Yet again, since the commitments are hiding, Alice does not yet know which commitment was selected by Bob. To open the selected commitment, Alice and Bob reveal the random tape used during their previous computation so that the parties can verify that the protocol was carried out correctly by the parties. More specifically, $C' = Com(Com(C_E))$ for $C_E \in \{C_A, C_B\}$ and some randomness used by Alice and Bob in Com. A schematic description of the protocol is given in Figure 5.2.

Let b_0 denote the bit used by Alice in a PSwap execution to determine the output order of (C_0, C_1). Let b_1 denote the bit used by Bob in a $\mathsf{PSelect}$ execution to determine the selected commitment. Let $\hat{b}$ denote a bit, which indicates the selected commitment, C_A or C_B, encoded in the output $\hat{C}$. We have that $\hat{b} = b_0 \oplus b_1$ with overwhelming probability.

Lemma 5.3.3. *A passive adversary, controlling either party in* $\mathsf{OSelect}$*, can output the correct value of $\hat{b}$ after the commit phase with the probability at most $1/2$ plus some negligible factor.*

5.3.3 Leader Election based on Two-party Oblivious Select

In this section, we define and analyze $\mathsf{LeaderElection}$, our intermediate non-secret construction, which essentially uses the oblivious select protocol ($\mathsf{OSelect}$) multiple times in a black-box manner.

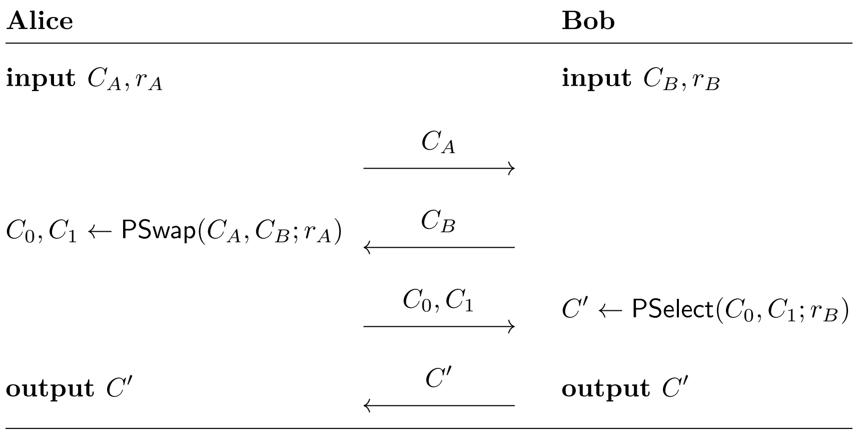

Figure 5.2: Two-party oblivious select OSelect protocol between Alice and Bob.

There are N users. In the selection phase, each user U_i initially holds a distinct number m_i and commits to it as $C_i = Com(m_i; r_i)$. Then, the users run a generalization of two-party OSelect to N parties, which we call $\mathsf{OSelect}_N$. $\mathsf{OSelect}_N$ essentially forms a binary tree of OSelect instances in a black-box manner. Whereas OSelect reduces two inputs to one output, $\mathsf{OSelect}_N$ reduces N inputs to one output. We will use this logical tree-like structure used in $\mathsf{OSelect}_N$ as the basis for our final SSLE construction.

5.3.3.1 Extending OSelect to N parties

The protocol consists of multiple rounds. One of N users is appointed as an *operational leader*. W.l.o.g., assume U_1 is the operational leader. In a round, users U_a and U_b holding C_a and C_b, respectively, perform $C' \leftarrow \mathsf{OSelect}(C_a, C_b)$, and U_a keeps C' and proceeds to the next round, whereas U_b terminates. Indices a and b are chosen according to the following procedure. In the first round, users U_{2i+1} and U_{2i+2} run OSelect on their original inputs, and the resulting value (commitment) is kept by U_{2i+1}, where $i \in [0, N/2-1]$. In the second round, users U_{2i+1} and U_{2i+3} run OSelect using the output from the previous round as input, where $i \in [0, N/4-1]$. There will be $log(N)$ such rounds. In the final round, U_1 and $\mathsf{U}_{N/2+1}$ run an instance of OSelect. The resulting output, $\bar{C}$, is stored by the operational leader U_1 (and broadcasted to other $N-1$ users).

Thanks to the properties of OSelect, the output of $\mathsf{OSelect}_N$ protocol, $\bar{C}$, is a commitment to one of the user's inputs. If $\bar{C}$ is a commitment to m_i, then U_i is the elected user. In the opening phase, all users broadcast their input message and randomness, so that anyone could verify that $\mathsf{OSelect}_N$ protocol was carried out correctly.

$$\begin{tikzcd}
\mathcal{P}_1 & \mathcal{P}_2 & \mathcal{P}_3 & \mathcal{P}_4 \\
C_1 \arrow[d] & C_2 \arrow[dl] & C_3 \arrow[d] & C_4 \arrow[dl] \\
C'_1 \leftarrow \mathsf{OSelect}(C_1, C_2) \arrow[d] & & C'_3 \leftarrow \mathsf{OSelect}(C_3, C_4) \arrow[dll] & \\
\bar{C} \leftarrow \mathsf{OSelect}(C'_1, C'_3) & & &
\end{tikzcd}$$

Figure 5.3: $\mathsf{OSelect}_N$: Extension of two-party OSelect to N users. Example for $N = 4$ and party $\mathcal{P}_1$ as the operational leader.

5.3.3.2 Amortizing the communication cost

Rotation of operational leaders allows to amortize the communication cost. The operational leader has to execute OSelect $log(N)$ times. After N executions of LeaderElection with different operational leaders, the total number of OSelect executions for any user amounts to $2 \cdot N$, which means on average each user has to perform two OSelect-s per single execution of LeaderElection, which makes it a considerably efficient protocol.

5.3.3.3 Problem

The resulting protocol LeaderElection is still a *non-secret* leader election, as there is no way for the leader to learn the outcome of the protocol exclusively. Moreover, the *unpredictability* property does not hold. Imagine the last OSelect execution is run by two parties, controlled by an adversary. The adversary then knows exactly which half of the users contains the selected leader, breaking the unpredictability property. Lastly, all parties are required to participate in the protocol in at least one instance of OSelect, which makes it impossible to tolerate a single faulty party. In the next section, we will address these problems and present our secure SSLE protocol.

5.4 Our SSLE from DDH

In this section, we present our secret single leader election SSLE protocol, which we gradually develop starting from Section 5.3. We first add an intermediate representation layer to the tree-based construction LeaderElection (Section 5.3.3) in order to let the secret leader actually check whether she is the elected leader. Then, we replace the oblivious select protocol used in the construction with its MPC version to achieve the desired security properties. More specifically, we will show that our SSLE protocol satisfies uniqueness, fairness, and unpredictability.

5.4.1 Upgrading to secret leader

We begin with the non-secret protocol LeaderElection (Section 5.3.3) and modify it to let the elected leader exclusively learn the outcome of the election. Here, we make use of a distributed key generation and threshold decryption. The resulting secret leader election protocol does not satisfy all our requirements to SSLE but serves as an intermediate point towards our final construction in Section 5.4.2.

5.4.1.1 Preliminaries

Distributed Key Generation (DKG) Distributed Key Generation [107] allows several parties to agree on a joint secret key. The corresponding public key is computed and published jointly by the honest majority of the parties. In t-out-of-N DKG protocol [68], the secret key is shared according to Shamir's secret sharing scheme. The protocol can be efficiently simulated against passive and active adversaries, which can corrupt up to t parties.

Threshold cryptography In threshold cryptography, parties jointly generate a group public key to encrypt messages and a qualified subset of parties can collaboratively decrypt ciphertexts encrypted using that key. We consider Shamir's t-out-of-N threshold ElGamal-based decryption schemes, for which any coalition of t parties cannot decrypt a given ciphertext or learn any information about the plaintext, whereas any coalition of $t+1$ parties can recover it, even if the remaining $N-t-1$ parties stop communicating.

5.4.1.2 Intuition

Instead of OSelect, we use a new subroutine OSelectD, which is a two-party verifiable oblivious select protocol in the discrete log setting. In contrast to OSelect, the users can publicly verify that a OSelectD instance was executed correctly without learning which input was selected. $\mathsf{OSelectD_N}$ is an extension of OSelectD to N parties, which follows the tree structure as shown in Figure 5.3. The input to OSelectD is an Elgamal encryption of two group elements $e_i := (g^r, g^{k_i \cdot r})$ under a group public key for some user's registration key k_i and randomness r. Algebraic properties of the underlying group allow us to construct OSelectD in a way that its output will be an encryption of one of the inputs in a randomized form, i.e., $(g^{r'}, g^{k_i \cdot r'})$ for some r'. Conversely, $\mathsf{OSelectD_N}$ will output an encryption of one of the user's inputs in a randomized form. Users jointly decrypt the output of $\mathsf{OSelectD_N}$ and obtain $\bar{C}$. There will be a unique pair $(e_i, \bar{e})$, which forms a valid DDH tuple, for which the elected leader knows an exponent; all other pairs $(e_j, \bar{e})$ where $j \neq i$ are random tuples. The leader presents the exponent as proof of leadership.

OSelectD is a two-party oblivious select protocol between Alice and Bob, which takes as input two messages of the following structure: $(g^{r'}, (y_{\mathcal{G}})^{r'} \cdot g^r, (y_{\mathcal{G}})^{r'} \cdot g^{k_i \cdot r}) \in$

$(G_q)^3$, for some i and random r, r'. We construct OSelectD in such a way that the output message preserves this structure, while changing only the randomness. Alice holds $C_A \in (G_q)^3$ as input, and Bob holds $C_B \in (G_q)^3$. The description of the commit phase is exactly the same as it was defined in Figure 5.2 for OSelect, except that OSelectD relies on the discrete log variants of PSwap and PSelect, which we call PSwapD and PSelectD. We will also use appropriate non-interactive zero-knowledge (NIZK) proofs that each OSelectD computation was carried out correctly. This will help a leader later exclusively learn the outcome of the election. These proofs are generalizations [49, 30] of Schnorr signature [113] and can be efficiently instantiated in the random oracle model using the Fiat-Shamir transform [60].

Definition 5.4.1 (PSwapD). *On an input of two valid messages $C_0, C_1 \in (G_q)^3$, draw randomness (r_0, r_1) and a random bit $b \in \{0,1\}$, compute $C_0' = (C_0)^{r_0}$, $C_1' = (C_1)^{r_1}$, and output (C_b', C_{1-b}') and appropriate NIZK proofs.*

Definition 5.4.2 (PSelectD). *On an input of two valid messages $C_0, C_1 \in (G_q)^3$, draw randomness (r) and a random bit $b \in \{0,1\}$, compute and output $C' = (C_b)^r$ and appropriate NIZK proofs.*

Definition 5.4.3 (Valid tuples). *We call a tuple $(G_q)^3$ valid w.r.t. $k_i \in \mathbb{Z}_p$ if it can be represented as $(g^{r'}, (pk_{\mathcal{G}})^{r'} \cdot g^r, (pk_{\mathcal{G}})^{r'} \cdot g^{k_i \cdot r})$ for some $r, r' \in \mathbb{Z}_p$.*

It is easy to see that if the inputs to OSelectD are valid tuples w.r.t. k_i and k_j, then the output of OSelectD is also a valid tuple w.r.t. $k \in \{k_i, k_j\}$. Conversely, the output to $\mathsf{OSelectD_N}$ is a valid tuple w.r.t. $k \in \{k_1, \ldots, k_N\}$.

Lemma 5.4.1. *Assuming DDH in groups G_q of a prime order q, for any pair of valid tuples $C_0, C_1 \in (G_q)^3$ and any PPT adversary* Adv*, and security parameter λ,*

$$Pr\begin{bmatrix} (C_0', C_1') \leftarrow \mathsf{PSwapD}(C_0, C_1), \\ b' \leftarrow \mathsf{Adv}(C_0, C_1, C_0', C_1'), b' = b \end{bmatrix} \leq 1/2 + negl(\lambda).$$

Lemma 5.4.2. *Assuming DDH in groups G_q of a prime order q, for any pair of valid tuples $C_0, C_1 \in (G_q)^3$ and any PPT adversary* Adv*, and security parameter λ,*

$$\begin{array}{c} Pr[C' \leftarrow \mathsf{PSelectD}(C_0, C_1), b' \leftarrow \mathsf{Adv}(C_0, C_1, C'), b' = b] \\ \leq 1/2 + negl(\lambda) \end{array}.$$

5.4.1.3 Protocol description

Let t denote the number of dishonest (malicious) users, where $2 \cdot t < N$. Let g be a generator of a group G of a prime order $\mathbb{Z}_p$. User U_i registers for the election by generating a registration key $k_i \in \mathbb{Z}_p$ and computing a registration token as $e_i \leftarrow (g^r, g^{k_i \cdot r})$ for some random r. The values k_i are e_i are kept private.

```
Elect(i, k_i)
1 : $r \leftarrow \mathbb{Z}_q$
2 : $e_i \leftarrow (g^r, g^{k_i \cdot r})$
3 : $(y_\mathcal{G}, (y_i, x_i)) \leftarrow \mathsf{DKG}(t, N)$
4 : $C_i \leftarrow \mathsf{Enc}_\mathcal{G}(e_i)$
5 : $\bar{C} \leftarrow \mathsf{OSelectD_N}(..., C_i, ...)$
6 : $(\bar{e}_1, \bar{e}_2) \leftarrow \mathsf{Dec}_\mathcal{G}(\bar{C})$
7 : if $(\bar{e}_1)^{k_i} \neq \bar{e}_2$
8 :     output $\perp$
9 : output $\pi := k_i$
```

Figure 5.4: The intermediate Secret Leader Election protocol.

The users generate a temporary shared public key using t-out-of-N distributed key generation (DKG), $pk_\mathcal{G} = g^{sk_\mathcal{G}}$. The corresponding group secret key, $sk_\mathcal{G}$, is shared between N parties, such that $t+1$ parties have to collaborate to decrypt a ciphertext C, which we denote by $\mathsf{Dec}_\mathcal{G}(C)$.

Party $\mathcal{P}_i$ prepares her input C_i to $\mathsf{OSelectD_N}$ by encrypting e_i under a group public key, $C_i \leftarrow \mathsf{Enc}_\mathcal{G}(e_i) := (g^{r_1}, (pk_\mathcal{G})^{r_1} \cdot e_i)$. The parties then proceed with $\mathsf{OSelectD_N}$ and collaboratively decrypt its output $\bar{C}$ to $\bar{e}$, while all the intermediate results of $\mathsf{OSelectD}$ instances in $\mathsf{OSelectD_N}$ are not decrypted. Thanks to algebraic properties of the underlying group, the decrypted value will be $(g^{r'}, g^{k_j \cdot r'})$ for some r' and $j \in [N]$. The election is concluded by running a DDH tuple check for the registration token and $\bar{e}$ using the registration key. We give a compact description of the protocol in Figure 5.4. To verify a proof π, one verifies all NIZK proofs used in $\mathsf{OSelectD_N}$ and checks whether $(\bar{e}[0])^\pi = \bar{e}[1]$. The leader will have to re-register and get a fresh k_i before participating in another election.

5.4.1.4 Problem

While, in this version of the leader election protocol, the leader can learn the outcome of the election in private, there remain several problems to address. First, an adversary can run a *duplicate key* attack [18], where she obtains multiple registration tokens that correspond to a single registration key, and thus break fairness. To see why the protocol suffers from this attack, we observe that for a fixed registration key k_i, two valid tokens $(g^{r_1}, g^{k_i \cdot r_1})$ and $(g^{r_2}, g^{k_i \cdot r_2})$ are indistinguishable from two valid tokens that correspond to different keys, because of DDH assumption. The mitigation measures proposed in [18] work in our setting, too. Second, a malicious adversary can use biased coins when computing $\mathsf{OSelectD}$. If both parties are under her control, she can break the obliviousness of $\mathsf{OSelectD}$ and, in turn, the unpredictability and fairness of the SSLE. Finally, even an *honest-but-curious* adversary, who controls both parties in $\mathsf{OSelectD}$ and

```
OSelectM([C_A], [C_B])
1 : $[b] \leftarrow \mathsf{RandBit}()$
2 :  do   //run in parallel
3 :      $[b \cdot C_A] \leftarrow \mathsf{Mul}([b], [C_A])$
4 :      $[(1-b) \cdot C_B] \leftarrow \mathsf{Mul}([1-b], [C_B])$
5 : $[C'] \leftarrow [b \cdot C_A] + [(1-b) \cdot C_B]$
6 : output $[C']$
```

Figure 5.5: OSelectM: Oblivious Select in the MPC setting.

follows the protocol, exactly knows which input has been selected, thus *breaking* unpredictability of SSLE. Since our goal is to satisfy *all* the three properties (uniqueness, unpredictability, and fairness), we will need one more modification to our construction, which we discuss in the following section.

5.4.2 Full construction

To obtain the full construction, we modify the secret leader election from Section 5.4.1 by replacing the two-party OSelect protocol with its MPC variant, OSelectM. Thereby, we ensure that no adversary in our model can learn the outcome of a OSelectM protocol instance. The extension of OSelectM to N inputs, which we call $\mathsf{OSelectM_N}$, retains the binary tree layout of inputs and outputs as it was shown in Figure 5.3. Each OSelectM instance is now executed by all parties simultaneously. This modification incurs additional communication costs compared to the previous (insecure) version of our SSLE construction. Fortunately, the number of communication rounds needed for a leader election remains $O(\log N)$, as OSelectM instances on the same level in the tree can run in parallel.

OSelectM is an oblivious select protocol in the MPC setting, which means that it can be completed as long as at least $t+1$ parties remain online and honestly execute the protocol. The protocol takes two secret shares $[C_A], [C_B]$ as input and outputs a new secret share $[C']$. It has a property, that the output secret C' is either C_A or C_B with equal probability. A description of OSelectM protocol is shown in Figure 5.5. OSelectM extension to N inputs, called $\mathsf{OSelectM_N}$, follows the same binary-tree structure of inputs and outputs to OSelectM, as in the previous extensions of oblivious select protocols. As there are no dedicated parties that execute the protocol, the notion of an operational leader introduced in the context of $\mathsf{OSelect}_N$ is no longer applicable. The binary-tree layout in $\mathsf{OSelectM_N}$ is shown in Figure 5.6.

To prevent duplicate key attacks, we incorporate into our SSLE scheme a technique used in [18]. The technique works as follow. The registration key k_i is now used to produce a secret part k_{iL} and a public fingerprint k_{iR} using a

$$\begin{tikzcd}[column sep=small]
{[C_1]} \arrow[dr] & & {[C_2]} \arrow[dl] & & {[C_3]} \arrow[dr] & & {[C_4]} \arrow[dl] \\
& {[C'_1] \leftarrow \mathsf{OSelectM}([C_1],[C_2])} \arrow[drr] & & & & {[C'_3] \leftarrow \mathsf{OSelectM}([C_3],[C_4])} \arrow[dll] & \\
& & & {[\bar{C}] \leftarrow \mathsf{OSelectM}([C'_1],[C'_3])} & & &
\end{tikzcd}$$

Figure 5.6: $\mathsf{OSelectM_N}$: Extension $\mathsf{OSelectM}$ to N inputs. Example for $N = 4$.

cryptographic hash function H, where $(k_{iL}, k_{iR}) \leftarrow H(k_i)$. Before the election starts, each user verifies that there are no duplicate fingerprints in the public state st. The security properties of the hash function ensure that chances for an adversary to succeed in a duplicate key attack are negligible.

The election proceed as follows. For each $i \in \{1, \ldots, N\}$, the parties jointly generate $[C_i]$, a MPC version of the secret part k_{iL} of the registration key k_i, which is $[k_{iL}]$). In the MPC setting we do not need to additionally hide the key using Elgamal encryption, since secret sharing already hides the results of the computation.

The parties then proceed with $\mathsf{OSelectM_N}$ and obtain $[\bar{C}]$, which is a secret share of one of the secret inputs to $\mathsf{OSelectM_N}$. The parties jointly reconstruct two group elements $(\bar{e}_1, \bar{e}_2)$ from $[\bar{C}]$, which turn out to be a randomization of the secret part of a party participated in the election, which we denote $\bar{k}_L$. If $\mathcal{P}_i$ is the elected leader, the following equation will hold $\bar{k}_L = k_{iL}$, i.e. each party learns the secret key k_{iL} of the leader, but does not know which one. The leader $\mathcal{P}_i$ sends the registration key k_i as a proof of leadership. To verify a proof π, one recomputes the secret part π_L of the registration key and its fingerprint π_R and checks that the computed fingerprint matches the one stored as st_i, and that the equation $\pi_L = \bar{k}_L$ holds.

In the malicious adversary model, we can use standard techniques [69, 48] to protect the underlying MPC primitives used in the scheme against active adversaries.

We now formally define our fully-fledged SSLE construction.

Construction 5.4.1 (Single secret leader election (SSLE))**.** *Our SSLE scheme is a tuple of PPT algorithms SSLE = (Setup, $\mathsf{Register}$, $\mathsf{RegisterVerify}$, Elect, Verify) that use a group G of a prime order p. Let g be a generator of G, let H be a function that maps $\{0,1\}^\lambda$ to $\mathbb{Z}_p \times \{0,1\}^{r(\lambda)}$. The description of the algorithms is shown in Figure 5.7.*

Theorem 5.4.1. *Assuming the underlying MPC primitives are secure in the honest-but-curious adversary model, H is a random oracle, then Construction 5.4.1 implements functionality $\mathcal{F}_{\mathsf{SSLE}}$.*

```
Setup($1^\lambda, t, N$)
  1 : $p, g \leftarrow \mathsf{FindParam}(1^\lambda)$
  2 : for $i \in 1..N$
  3 :     $st_i \leftarrow \perp$
  4 : return $p, g, st_1, ..., st_N$
```

```
Register($i$)
  1 : $k_i \leftarrow \mathbb{Z}_p$
  2 : $k_{iL}, k_{iR} \leftarrow H(k_i)$
  3 : $st_i \leftarrow k_{iR}$
  4 : $[k_{iL}] \leftarrow \mathsf{Share}(k_{iL})$
  5 : return $k_i$
```

```
Verify($i, \pi$)
  1 : $\pi_L, \pi_R \leftarrow H(\pi)$
  2 : if $\pi_L = \bar{k}_L$ and $\pi_R = st_i$
  3 :     return 1
  4 : return 0
```

```
RegisterVerify($i, k_i$)
  1 : for $j_1 \in 1..(N-1)$
  2 :   for $j_2 \in (j_1+1)..N$
  3 :     if $st_{j_1} = st_{j_2} \neq \perp$
  4 :       return 0
  5 : $k_{iL}, k_{iR} \leftarrow H(k_i)$
  6 : if $k_{iR} \neq st_i$
  7 :   return 0
  8 : return 1
```

```
Elect($i, k_i$)
  1 : for $i \in 1..N$
  2 :     $[C_i] := [k_{iL}]$
  3 : $[\bar{C}] \leftarrow \mathsf{OSelectM_N}([C_1],$
  4 :   $..., [C_N])$
  5 : $\bar{k}_L \leftarrow \mathsf{Rec}([\bar{C}])$
  6 : if $\bar{k}_L \neq k_{iL}$
  7 :     return $\perp$
  8 : return $\pi := k_i$
```

Figure 5.7: Single Secret Leader Election construction SSLE instantiated with $\mathsf{OSelectM_N}$.

5.5 Our SSLE based on garbled circuits

In this section, we present our SSLE protocol, instantiated in the MPC framework by Wang et al. [129].

5.5.1 Preliminaries

Wang et al. proposed an efficient secure constant-round MPC on boolean circuits by extending a two-party protocol [128] to the multi-party setting [129]. The protocol uses an ideal functionality $\mathcal{F}_{\mathsf{Pre}}$ as a preprocessing step to set up correlated randomness between the parties. At a high level, $\mathcal{F}_{\mathsf{Pre}}$ generates authenticated shares on random bits x, y, z such that $z = x \wedge y$, using information-theoretic MACs [102]. Those authenticated shares are then used to distributively construct a single, "authenticated" garbled circuit, which is evaluated by one of the parties. Wang et al. show the security of their protocol against a malicious adversary that compromises $N-1$ parties in the $\mathcal{F}_{\mathsf{Pre}}$-hybrid model and assuming a random oracle (ROM). We refer the reader to [129] for the description of $\mathcal{F}_{\mathsf{Pre}}$ and further details.

5.5.2 Construction

We use the MPC protocol [129] to instantiate our SSLE in a black-box manner. The SSLE construction shown in Figure 5.7 needs to be updated to account for technical details specific to the MPC part in the Elect algorithm, which we discuss below.

To implement the Oblivious Select, we use a part of the input as selection bits. The modified version of the Elect algorithm and a pseudocode of OSelectM instantiated in the framework [129] are shown in Fig. 5.8. Each party $\mathcal{P}_i$ provides her input of $(rBits + aBits)$ bits, which is stored in arrays of instances of MPC's class Integer, R and A. We require that $rBits \geq \log(aBits)$. Array R is used to compute selection bits $R[0]$. Then $\log(N)$ rounds follow, in which depending on a selection bit *bit*, $A[i]$ is assigned to either $A[2i]$ or $A[2i+1]$. We use an existing function *select* of MPC's class Integer for this specific computation. Finally, $A[0]$ is revealed.

Construction 5.5.1 (Single secret leader election (SSLE))**.** *Our SSLE scheme is a tuple of PPT algorithms* $\mathsf{SSLE} = (\mathsf{Setup}, \mathsf{Register}, \mathsf{RegisterVerify}, \mathsf{Elect}, \mathsf{Verify})$*. Let H be a function that maps $\{0,1\}^{\lambda}$ to $\{0,1\}^{l(\lambda)} \times \{0,1\}^{r(\lambda)}$. The description of the algorithms is shown in Fig. 5.7 expect for* Elect*, which is shown in Fig. 5.8.*

Theorem 5.5.1. *Assuming the underlying MPC primitives are secure in the malicious adversary model, H is a random oracle, then Construction 5.5.1 implements functionality $\mathcal{F}_{\mathsf{SSLE}}$.*

```
oselect_N
for $i \in 1..N$
    $R[i-1] = Integer(bits : rBits, party : i)$
    $A[i-1] = Integer(bits : aBits, party : i)$
for $i \in 1..N-1$
    $R[0] = R[0] \oplus R[i]$
$rem = N$
$round = 0$
while $rem > 1$
    $bit = R[0][round]$
    for $i \in 0..rem/2-1$
        $A[i] = A[2i].select(bit, A[2i+1])$
    $round = round + 1$
    $rem = rem/2$
$A[0].reveal()$
```

```
Elect$(i, k_i)$
$R_i, A_i \leftarrow k_{iL}$
$\bar{A} \leftarrow$ oselect_N$(party\ i : R_i, A_i)$
if $\bar{A} \neq A_i$
    return $\perp$
return $\pi := k_i$
```

Figure 5.8: Elect algorithm and Oblivious Select instantiated in the MPC framework by Wang et al. (129)

$$\begin{tikzcd}
{[C_1]} \arrow[dr] & & {[C_2]} \arrow[dl] & & {[C_2]} \arrow[dr] & & {[C_2]} \arrow[dl] & \rightarrow & {[C_1]} \arrow[dr] & & {[C_2]} \arrow[dl] & \\
& {[...]} \arrow[drr] & & & & {[...]} \arrow[dll] & & & & {[...]} \arrow[dr] & & {[C_2]} \arrow[dl] \\
& & & {[\bar{C}]} & & & & & & & {[\bar{C}]} &
\end{tikzcd}$$

Figure 5.9: Tree optimization technique. Example $\mathsf{OSelectM_N}$ for $\mathcal{P}_1$ and $\mathcal{P}_2$ with stakes $(1, 3)$.

5.6 Practical considerations

There are several constraints in Constructions 5.4.1 and 5.5.1 that affect its practicality. First, the definition of SSLE says that the probability for a party being elected should be equal among all participants. In practice, the stakeholders may have different stakes, and the probability for a party to be elected should be proportional to her stake. A straightforward solution to this constraint would be to adapt our SSLE construction to work with stake units and let each party control several units. If implemented naively, this approach results in a linear blow-up in computation and required storage (in the number of stake units). In the following, we will show an efficient technique to extend Construction 5.4.1 to support arbitrary (non-uniform) probability distributions in the election.

Second, we assumed the number of parties to be a power of two, in order to construct a complete binary tree in Oblivious Select. However, if the number of parties is arbitrary, the tree structure will likely unbalance the tree leaves, as some inputs will not be matched on the first level with other inputs. Therefore, such inputs would proceed to the next round without competition, i.e., with the probability of 1, whereas input C_i in a binary tree will proceed with the probability of $1/2$. We will show that the technique from the previous point addresses this concern, too.

Moreover, we will show how to reduce the number of communication rounds in Construction 5.4.1 by using appropriate MPC primitives.

5.6.1 Non-uniform distributions

We observe that it is possible to unbalance almost-for-free the probability of being selected (among two parties) if the sum of the weights is a *power of two*. To illustrate this idea, assume that the weights are $(1, 3)$, i.e., the probabilities for two parties being selected are determined by the ratio 1:3. We can construct a tree-structure with the probabilities $1/4$ and $3/4$, as shown in Figure 5.9.

Basically, we introduce a special case for $\mathsf{OSelectM}$ when handling shares of the same secret for free, $\mathsf{OSelectM}(\mathsf{Share}C, [C]) \rightarrow [C]$. The resulting tree can be optimized significantly by dropping the nodes with the same inputs.

Using this technique, we can handle weights of the form $(w, 2^L - w)$ with a logarithmic overhead, for some $L \geq 1$ and $1 \leq w < 2^L$. However, we cannot naturally handle arbitrary weight ratios. For example, weights such as $(1, 2)$ are problematic. Nevertheless, we can approximate the probabilities in the election according to any weights (a, b) by having a tree of sufficient depth.

Arbitrary N and stakes Let N be the number of parties participating in the election with their stakes $(s_1, ..., s_N)$, and let $S = \sum_{i=1}^{N} s_i$ be the sum of parties' stakes. The multi-registration solution may lead to $O(S)$ complexity of the election algorithm. We extend our technique for two parties to an arbitrary number of users.

We start with a similar idea: each party has a sequence of stake units on the first level in a $\mathsf{OSelectM_N}$ tree. If $N \ll S$, there will be many pairs of inputs that represent the same party. We observe that in this case, there is no need to run $\mathsf{OSelectM}$ on such inputs. Instead, we can pick any input and advance it to the next level in the tree. The worst case complexity (the number of $\mathsf{OSelectM}$ instances) of this technique is $O(N \log S)$, since each party $\mathcal{P}_i$'s inputs will be matched in a tree of depth $O(\log S)$ at most two times, against $\mathcal{P}_{i-1}$ and $\mathcal{P}_{i+1}$. With a tree of depth L we can get the absolute precision up to $2^{-L} \cdot S$.

5.6.2 Extensions to DDH-based MPC

It is possible to generate random bits in a fully non-interactive manner [47]. This will require an initial (trusted) setup for distributing randomness between the parties. For the base case where stakes are equal and N is a power of two, this improvement will not be significant, as the parties have to generate only $O(\log N)$ random bits, while requiring $O(N)$ interactive multiplications of shares. In a general case, for arbitrary stakes, it will show a better reduction of the cost.

5.7 Evaluation

5.7.1 Experimental setup

We evaluate our SSLE framework, we implemented Constructions 5.4.1 and 5.5.1 and ran two kind of tests: in a local setting (LAN) and in a global setting (WAN). In the LAN setting, we used machines located in the same Amazon EC2 region. In the WAN setting, we used machines located in four different regions (Europe, North America, South America, and Asia). If not specified otherwise, each machine is a t2.large instance with 2 cores Intel Xeon E5-2686v4 2.3 GHz, 8Gb of RAM, and installed Ubuntu 20.04. In some regions t2.large instances are not available; instead we used t3.large instances with 2 cores Xeon Platinum 8175 2.5 GHz, 8Gb of RAM.

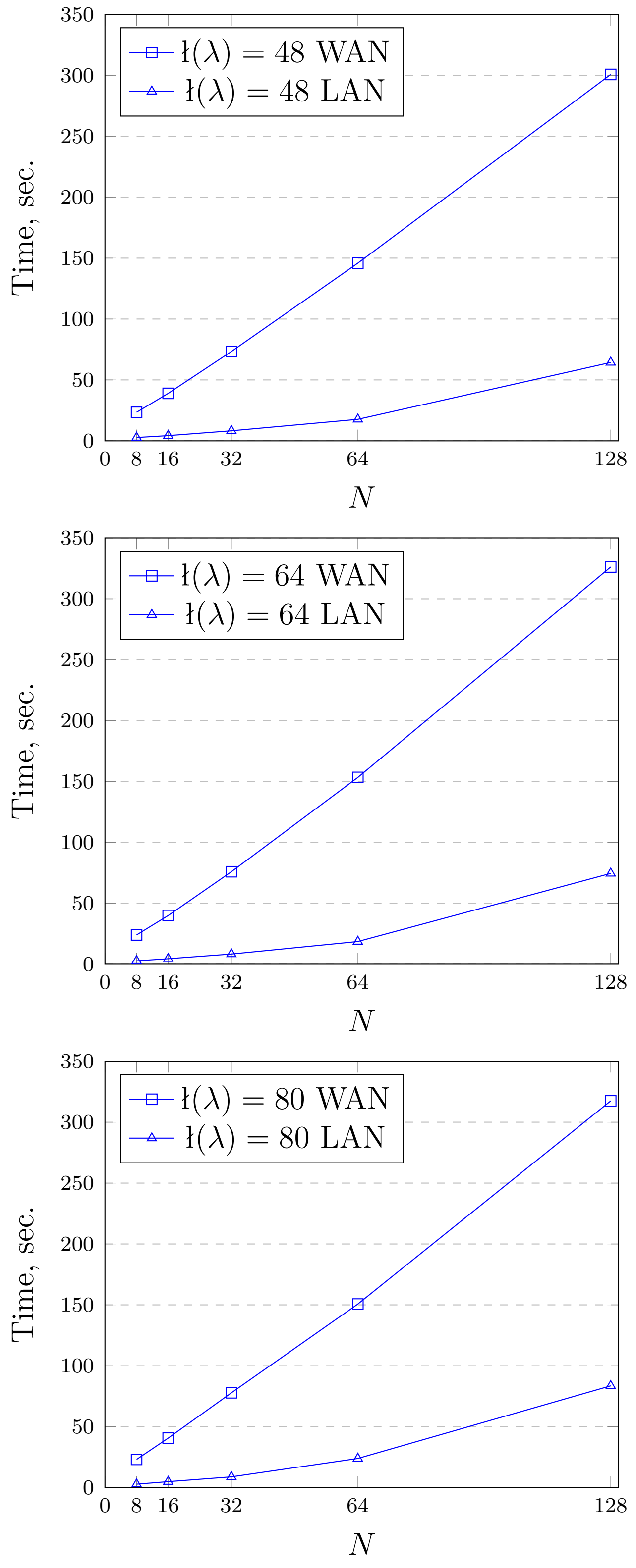

Figure 5.10: Comparison of timings for Oblivious Select in Construction 5.5.1 in the LAN and WAN settings.

In our experiments, we evaluate a complete OSelect tree in our SSLE framework, that is the number of users being a power of two, starting from 8 parties, and each party holding one unit of stake. For each experiment we take average of 10 runs, except that for lengthy experiments with a running time more than 1 minute we perform a single run. Next, we present implementation details and the evaluation results individually for each construction.

5.7.2 Construction 5.4.1 (Section 5.4.2)

Implementation details We implemented our Construction 5.4.1 in C++ in the honest-but-curious and malicious adversary models. We implemented the underlying MPC primitives for secret sharing, adding shares, substracting, multiplying by a scalar: $[x] + [y]$, $[x] - [y]$, $[\alpha \cdot x]$, protocols RndFld, RndBit, Mul [107, 110, 69, 67, 50, 36]. In the malicious adversary model, these primitives are accompanied with verifiable secret sharing (VSS). We set the threshold $t = N/2 - 1$ in all experiments. Our implementation uses the Relic toolkit [5] for operations on elliptic curves in groups of a prime order of 256 bits, the Boost and OpenSSL libraries for secure communication.

Experimental results We performed LAN tests for up to 128 parties in the honest-but-curious adversary model, and up to 32 parties in the malicious model. Timings are shown in Table 5.2.

Analysis The experimental results show that up to 128 parties can complete Elect protocol in under a minute. The running time grows rapidly as the number of parties increases. This is due to expensive public key operations for generating and reconstructing Shamir's secret shares. The explosion of running time is more visible in the malicious adversary model. In order to protect against such adversaries, we have to use verifiable secret sharing, which requires $O(N^2)$ public key operations in the textbook implementation. While Elect is the most heavy algorithm, the rest of the SSLE protocol is essentially for free. We conclude that Construction 5.4.1 offers a practical t-robust solution to the SSLE problem for a small number of parties (up to 32, according to our evaluation).

5.7.3 Construction 5.5.1 (Section 5.5.2)

Implementation details We implemented and evaluated Oblivious Select part of the Elect algorithm, as it is the most heavy part of the SSLE protocol (see experimental results for Construction 5.4.1 in the honest-but-curious adversary model in Section 5.7.2). Our implementation fully relies on the implementation of the MPC framework by Wang el at. [129], which is available as [127]. We can

N	t	Algorithm	HbC time, sec.	Mal. time, sec.
8	3	Register	<0.01	0.11
		RegisterVerify	<0.01	<0.01
		Elect	0.1	3.56
		Verify	<0.01	<0.01
16	7	Register	0.01	0.56
		RegisterVerify	<0.01	<0.01
		Elect	0.34	28.1
		Verify	<0.01	<0.01
32	15	Register	0.02	3.83
		RegisterVerify	<0.01	<0.01
		Elect	1.45	356.6
		Verify	<0.01	<0.01
64	31	Register	0.08	n.a.
		RegisterVerify	<0.01	
		Elect	7.63	
		Verify	<0.01	
128	63	Register	0.21	n.a.
		RegisterVerify	<0.01	
		Elect	54.4	
		Verify	<0.01	

Table 5.2: Experimenal results for Construction 5.4.1 in the honest-but-curious and malicious adversary models.

trade-off security for efficiency by controlling how many bits each party inputs to Oblivious Select (see Lemma 5.9.8).

Experimental results In the MPC framework, the evaluator of the garbled global circuit requires more RAM than any other party. Therefore, we set up one machine as a m5a.4xlarge instance with 16 cores and 64G of RAM, while the rest of machines remain t2.large or t3.large instances. We run experiments for each N up to 128. For the trade-off, we choose the length of user inputs to Oblivious Select, $l(\lambda)$, as 48, 64, and 80 bits. Additionally, each party provides 8 bits of selection bits, which satisfies the constraint that it should be at least as big as $\log(N)$ in all test cases. Timings the LAN and WAN settings are shown in Table 5.3 and in Fig. 5.10.

Analysis The experimental results show that the running time of Oblivious Select algorithm (and in turn, Elect) grows almost linearly as the number of parties gets increased. As we ran only 1 iteration for long test cases, we can see some unexpected fluctuations in the running time, which we think are caused by

N	$l(\lambda)$	LAN time, sec.	WAN time, sec.
8	48	2.73	23.42
	64	2.76	24.03
	80	2.80	24.27
16	48	4.28	38.95
	64	4.50	39.92
	80	4.86	40.61
32	48	8.25	73.34
	64	8.35	75.93
	80	8.80	77.81
64	48	17.64	145.87
	64	18.62	153.34
	80	23.90	150.67
128	48	64.33	300.77
	64	74.54	326.09
	80	83.54	317.46

Table 5.3: Experimental results for Construction 5.5.1 in the malicious adversary model.

fluctuations in the network and normally should be eliminated after averaging multiple iterations.

The LAN and WAN settings have identical computational and communication cost, as they only differ in the location of machines. We suspect that higher latency between machines in the WAN settings accounts for the increased running time. In the LAN setting, 128 parties can compute a leader in under 1.5 minutes, where as in the WAN setting, this number approaches 7 minutes.

5.8 Postponed definitions

Definition 5.8.1 (Single secret leader election (SSLE)). *A single secret leader election scheme is a tuple of PPT algorithms* $\mathit{SSLE} = (\mathit{SSLE}.\mathsf{Setup}, \mathit{SSLE}.\mathsf{Register}, \mathit{SSLE}.\mathsf{RegisterVerify}, \mathit{SSLE}.\mathsf{Elect}_1, \mathit{SSLE}.\mathsf{Elect}_2,$
$\mathit{SSLE}.\mathsf{Verify})$ *with the following behavior:*

- $\mathit{SSLE}.\mathsf{Setup}(1^\lambda, \ell, N) \to (\mathsf{pub}, sk_1, \dots, sk_N, st_0)$*: The setup process generates public parameters* pub*, a number of secrets, and an initial state* st_0*.* N *is an upper bound on the number of participants supported by the scheme, and* ℓ *is a lower bound on the number of required users per election.* $\mathit{SSLE}.\mathsf{Setup}$ *is a one-time setup process, followed by a series of elections.*
- $\mathit{SSLE}.\mathsf{Register}(i, \mathsf{pub}, st) \to (rk_i, rt_i, st')$*: Each user registers with a unique*

public identity $i \in [N]$, the public parameters pub*, and the current state st. Registration outputs a registration key rk_i, gives a user a registration token rt_i, and modifies the state to st'.* SSLE.Register *must be run to participate in a series of elections. The eventual leader may be required to re-register for subsequent elections.*

- SSLE.RegisterVerify$(i, rk_i, rt_i, \mathsf{pub}, st) \rightarrow \{0, 1\}$*:* SSLE.RegisterVerify *is run by previously registered users after a new user registers to verify that the registration was carried out correctly. Verification can use the user's registration key rk_i, registration token rt_i, the public parameters* pub*, and current state st. The user's registration token rt_i can be modified as the result of* SSLE.RegisterVerify*.*
- SSLE.Elect$_1(\mathsf{pub}, st, i, sk_i) \rightarrow (p_i, l_i)$*: Leader election begins by taking public parameters* pub*, current state st, a user U_i's secret key sk_i, and outputting intermediate values p_i and l_i.*
- SSLE.Elect$_2(\mathsf{pub}, st, i, p_i, l_1, ..., l_m, sk_i, rk_i, rt_i) \rightarrow (p'_i, l'_i) \cup \pi \cup \{\perp\}$*: Leader election proceeds by taking public parameters* pub*, current state st, intermediate values p_i and $l_1, ..., l_m$, user U_i's secrets sk_i, rk_i, rt_i, and outputting either 1) new intermediate values p'_i and l'_i, in which case the algorithm will be executed one more time on appropriate inputs, or eventually 2) a proof of leadership π in case user U_i has been chosen as the leader, or a distinguishable symbol $\perp$ otherwise.*
- SSLE.Verify$(\mathsf{pub}, i, st, \pi_i; p_i) \rightarrow \{0, 1\}$*: Given index i, the state st, and optionally an intermediate value p_i from the election, the verification algorithm accepts or rejects the proof that user U_i has been elected leader.* SSLE.Verify *is used to check the authenticity of a participant who claims to be the leader when it is time for the leader to reveal herself.*

If the eventual leader is required to re-register for subsequent elections, we say that an SSLE scheme with such a property has expiring registration*. Otherwise, if the parties may re-use their registration for multiple elections regardless of the outcome, an SSLE scheme has* non-expiring registration*. Note that re-registration will change the public state st.*

We could also formally include a revoke algorithm, to indicate that a user no longer wishes to participate. We refrain from such a formalism since it does not significantly impact the security properties, but our scheme can be modified to account for it.

Definition 5.8.1 specifies algorithms for setting up an SSLE instance, registering participants for the elections, verifying that the registration is performed correctly, electing the leader among registered parties, and verifying a proof of leadership. The setup algorithm generates private keys for the parties and introduces some initial state st_0, and the state can be updated by the registration algorithm. The state is public and accessible by all parties during the entire execution of an SSLE instance. The registration algorithm provides a party $\mathcal{P}_i$ with a registration key

rk_i and a registration token rt_i, which are kept private and used during elections. The difference between the key and the token is that the token depends on other parties participating in elections and therefore can be altered by the party holding it as the result of $\mathsf{SSLE.RegisterVerify}$. The election proceeds in several rounds. In each round, party $\mathcal{P}_i$ computes private p_i and public l_i intermediate values, and public output values from all participating parties are used as input for subsequent rounds ($\mathsf{SSLE.Elect}_2$). After the final round, each party holds either a proof of leadership π or $\perp$. The public state and, optionally, private value p_i from the last communication round are used to verify a proof of leadership.

Next, we define an experiment for N parties between the challenger and an adversary, where the adversary is controlling c parties. This experiment will serve as the initial step in the security games.

Definition 5.8.2 (Experiment). *We define an experiment* $\mathsf{EXPR}[\mathsf{Adv}, \lambda, \ell, N, c]$ *with security parameter λ, which is played between an adversary* Adv *and a challenger $\mathcal{C}$ as follows:*

- ***Setup phase.*** Adv *picks*[2] *a set of indices $M \subset [N]$, $|M| = c$, of users to corrupt. $\mathcal{C}$ runs* $(\mathsf{pub}, sk_1, ..., sk_N, st_0) \leftarrow \mathit{SSLE}.\mathsf{Setup}(1^\lambda, \ell, N)$ *and gives* Adv *the parameters* pub, *state st_0, and secrets sk_i for $i \in M$.*
- ***Elections phase.*** Adv *chooses a set of users to register for elections and for any polynomial number of elections to occur, where* Adv *plays the role of users* U_i *for $i \in M$ and $\mathcal{C}$ plays the role of the rest of the users.*
  To register an uncorrupted user, Adv *sends the index i of the user to $\mathcal{C}$, and $\mathcal{C}$ runs* $(rk_i, rt_i, st') \leftarrow \mathit{SSLE}.\mathsf{Register}(i, \mathsf{pub}, st)$. *To register a corrupted user,* Adv *sends the index i of the user to $\mathcal{C}$ along with an updated state st'. In either case, $\mathcal{C}$ then runs* $\mathit{SSLE}.\mathsf{RegisterVerify}(j, rk_j, rt_j, \mathsf{pub}, st)$ *for any previously registered user* U_j, *where $j \in [N] \setminus M$. If any call to* $\mathit{SSLE}.\mathsf{RegisterVerify}$ *returns 0, the game immediately ends with output 0. Otherwise, the state is updated to st'.*
  Each election begins with $\mathcal{C}$ generating $(p_i, l_i) \leftarrow \mathit{SSLE}.\mathsf{Elect}_1\,(\mathsf{pub}, st, i, sk_i)$ *on behalf of each uncorrupted registered user and* Adv *sending values l_i for any subset of corrupted registered users. Let $l_1, ..., l_t$ be the set of intermediate values l_i generated in this step. Then, elections may proceed with $\mathcal{C}$ generating* $(p'_i, l'_i) \leftarrow \mathit{SSLE}.\mathsf{Elect}_2(\mathsf{pub}, st, i, p_i, l_1, ..., l_t, sk_i, rk_i, rt_i)$ *on behalf of each uncorrupted registered user and* Adv *sending values l'_i for any subset of corrupted registered users. The intermediate values $l'_1, ..., l'_t$ are used as input for subsequent calls to* $\mathit{SSLE}.\mathsf{Elect}_2$. *Let $l''_1, ..., l''_t$ be the last intermediate values generated by* $\mathit{SSLE}.\mathsf{Elect}_2$, *and p''_j be the last private intermediate value of an uncorrupted registered user* U_j.
  Then, for all uncorrupted users, $\mathcal{C}$ sets $\pi_j \leftarrow \mathit{SSLE}.\mathsf{Elect}_2(\mathsf{pub}, st, j, p''_j, l''_1, ..., l''_t, sk_j, rk_j, rt_j)$ *if user* U_j *has registered for that election or $\perp$ otherwise. $\mathcal{C}$ sends π_j for each uncorrupted user to* Adv.

[2]We define the experiment is a way that c is given. Should Adv pick this number, we will denote such an experiment as $\mathsf{EXPR}[\mathsf{Adv}, \lambda, \ell, N]$.

If the SSLE scheme has expiring registration*, the leader of an election should repeat the registration procedure after the election is over, and uncorrupted parties should verify registration. During the elections phase, multiple elections can take place.*

The experiment in Definition 5.8.2 allows the adversary to corrupt parties statically during the setup. Then, the elections phase takes place, where an adversary chooses which parties will participate in an election. Those parties then register for the election. Each time a new party registers, all registered parties have to re-run SSLE.RegisterVerify to ensure that the registration data stored in the public state *st* is not malformed. During the election, the challenger plays the role of honest users, whereas a (malicious) adversary can send arbitrary messages on behave of the corrupted parties. As the outcome of an election, all non-registered (but participating in the election) parties receive $\perp$ symbol, whereas registered parties may receive either $\perp$ or a proof π.

Definition 5.8.3 (Uniqueness)**.** *The uniqueness experiment* $\mathsf{UNIQUE}[\mathsf{Adv}, \lambda, \ell, N]$ *between an adversary* Adv *and a challenger* $\mathcal{C}$ *with security parameter* λ *extends* $\mathsf{EXPR}[\mathsf{Adv}, \lambda, \ell, N]$ *as follows:*

- ***Output phase.*** *For each election in the elections phase,* Adv *outputs values* π_i *for each* $i \in M$. *The experiment outputs* 0 *if for each election with state st, there is at most one user* U_{i^*} *who wins that election. Otherwise the experiment outputs 1. We say user* U_{i^*} *wins an election if it outputs* $\pi_{i^*} \neq \perp$ *such that* $\mathit{SSLE}.\mathsf{Verify}(\mathsf{pub}, i^*, st, \pi_{i^*}) = 1$.

We say an SSLE scheme is unique *if no PPT adversary* Adv *can win the uniqueness game expect with negligible probability. That is, for all PPT* Adv *and for any* $\ell < N$, $Pr[\mathsf{UNIQUE}[\mathsf{Adv}, \lambda, \ell, N] = 1] \leq negl(\lambda)$. *If uniqueness only holds so long as there are at least t* uncorrupted *users participating in each election, we say that the scheme is t-threshold unique.*

Informally, an adversary wins the uniqueness experiment if in at least one election in a series of consecutive elections there is more than one verifiable leader.

Definition 5.8.4 (Unpredictability)**.** *The unpredictability experiment* $\mathsf{UNPRED}[\mathsf{Adv}, \lambda, \ell, N, n, c]$ *between an adversary* Adv *and a challenger* $\mathcal{C}$ *with security parameter* λ *extends* $\mathsf{EXPR}[\mathsf{Adv}, \lambda, \ell, N, c]$ *as follows:*

- ***Challenge phase.*** *At some point after the elections phase,* Adv *indicates that it wishes to receive a challenge, and one more election occurs. In this election,* $\mathcal{C}$ *does not send* (π_j) *for each uncorrupted user to* Adv*. Let* U_i *be the winner of this election. The game ends with* Adv *outputting an index* $i' \in [N]$*. If, for* U_i *elected in the challenge phase,* $i \in M$*, then the output of* $\mathsf{UNPRED}[\mathsf{Adv}, \lambda, \ell, N, n, c]$ *is set to 0. Otherwise,* $\mathsf{UNPRED}[\mathsf{Adv}, \lambda, \ell, N, n, c]$ *outputs 1 iff* $i = i'$.

We say that an SSLE scheme Π *is* unpredictable *if no PPT adversary* Adv *can win the unpredictability game with greater than negligible advantage. That is,* *for all PPT* Adv*, for any* $c \leq n-2$*,* $n \leq N$*, and for any* $\ell < N$*,*

$$Pr[\mathsf{UNPRED}[\mathsf{Adv}, \lambda, \ell, N, n, c] \mid i \in [N] \setminus M] \leq \frac{1}{n-c} + nelg(\lambda).$$

If Adv *wins with advantage* $\alpha + negl(\lambda)$ *for* $\alpha > \frac{1}{n-c}$*, with* α *potentially depending on* c*,* n*, or* N*, we say that* Π *is* α*-unpredictable. If the value of* α *depends on* N*, then we require that* $n = N$*. If unpredictability only holds for* $c < t$ *for some* $t > 0$*, we say that* Π *is* t*-threshold unpredictable.*

Informally, in the unpredictability experiment the adversary asks for a challenge election after a series of elections. In this special election, the challenger does not send to the adversary the outcome of elections. The adversary has to guess, who is the leader in this challenge election. If some corrupted party turns out to be the leader, the experiment is trivial and the result of the experiment will be always 0. Otherwise, if some honest party is the leader, the adversarial chances to correctly guess the leader should not be significantly greater than pure guessing.

Definition 5.8.5 (Fairness)**.** *The fairness experiment* $\mathsf{FAIR}[\mathsf{Adv}, \lambda, \ell, N, n, c]$ *between an adversary* Adv *and a challenger* $\mathcal{C}$ *with security parameter* λ *extends* $\mathsf{EXPR}[\mathsf{Adv}, \lambda, \ell, N, c]$ *as follows:*

– ***Challenge phase.*** *At some point after the elections phase,* Adv *indicates that it wishes to receive a challenge, and one more election occurs.* $\mathsf{FAIR}[\mathsf{Adv}, \lambda, \ell, N, n, c]$ *outputs 1 if there is no* $i \in [n] \setminus M$ *for which* $\mathit{SSLE}.\mathsf{Verify}(\mathsf{pub}, i, st, \pi_i) = 1$ *in the challenge election.*

We say that an SSLE scheme Π *is* fair *if no PPT adversary* Adv *can win the fairness game with greater than negligible advantage. That is, for all PPT* Adv*,* $n \leq N$*,* $c < n$*, and for any* $\ell < N$*,* $|Pr[\mathsf{FAIR}[\mathsf{Adv}, \lambda, \ell, N, n, c] = 1] - c/n| \leq negl(\lambda)$*.*

If fairness only holds for $c < t$ *for some* $t > 0$*, we say* Π *is* t*-threshold fair.*

Weaker selectively secure definitions of unpredictability and fairness are not considered in this work. A viable SSLE scheme must satisfy all the definitions above.

Informally, in the fairness experiment, the adversary asks for a challenge election after a series of elections. The probability of winning this challenge election by one of the corrupted parties should not be significantly greater than c/n, where c is the number of corrupted parties, and n is the number of parties registered for the challenge election.

5.9 Security analysis

Proof of Proposition 5.2.1. The goal is to demonstrate that any scheme that UC-realizes $\mathcal{F}_{\mathsf{SSLE}}$ fulfills all three game-based notions of (a) uniqueness, (b) unpredictability and (c) fairness. One needs to show the non-existence of a simulator given any of the game-based attackers: For some protocol π, assume an adversary against (a) and construct from it an environment distinguishing π and $\mathcal{F}_{\mathsf{SSLE}}$ (and same for (b) and (c)).

In the following experiments, assume to the contrary that some protocol π UC-realizes $\mathcal{F}_{\mathsf{SSLE}}$ and there is an ideal-world simulator $\mathcal{S}$ of the real-world adversary. Before an experiment starts, a random bit b is drawn; $\mathcal{Z}^*$ interacts with π in case $b = 0$, otherwise it interacts with $\mathcal{S}$ in case $b = 1$. Let $\mathcal{B}$ be an adversary successfully attacking some game-based property of π with non-negligible advantage $Adv_{\mathcal{B}}$. We construct an environment $\mathcal{Z}^*$ as follows: it uses $\mathcal{B}$ to set up malicious parties and communicate with the honest parties in the real world and with $\mathcal{S}$ in the ideal world. Let $Exec$ denote the output bit of $\mathcal{Z}^*$ in the experiment.

Uniqueness. An adversary wins the uniqueness experiment if there are at least two users $\mathsf{U}_i, \mathsf{U}_j$ for which the verification algorithm outputs 1 in the challenge election. At the end of the experiment, $\mathcal{Z}^*$ runs the verification algorithm for each user on the final output of the election (running SSLE.Verify in the real world or interacting with $\mathcal{S}$ that simulates the corresponding (verify) queries). If there is more than one user for which the verification algorithm (or $\mathcal{S}$) returns true, $\mathcal{Z}^*$ outputs 0 ($\mathcal{Z}^*$ believes it interacts with π). Otherwise, $\mathcal{Z}^*$ outputs 1 ($\mathcal{Z}^*$ believes it interacts with $\mathcal{S}$). On the one hand, we have that $Pr[Exec[\mathcal{F}_{\mathsf{SSLE}}, \mathcal{S}, \mathcal{Z}^*] = 0] = 0$ due to the definition of $\mathcal{F}_{\mathsf{SSLE}}$, which records only one leader for a single election and never changes it afterwards. On the other hand, $Pr[Exec[\pi, \mathsf{Adv}, \mathcal{Z}^*] = 0] = Adv_{\mathcal{B}}$, which is non-negligible. Hence, $\mathcal{Z}^*$ can distinguish between the real and ideal worlds with non-negligible probability, which contradicts the assumption we made about π.

Unpredictability. An adversary breaks unpredictability if it can predict the leader (among honest parties) significantly better than pure guessing. At the end of the experiment, $\mathcal{Z}^*$ computes the prediction as whatever $\mathcal{B}$ does. If the prediction was correct, $\mathcal{Z}^*$ outputs 0 ($\mathcal{Z}^*$ believes it interacts with π), otherwise it outputs 1 ($\mathcal{Z}^*$ believes it interacts with $\mathcal{S}$). On the one hand, we have that $Pr[Exec[\mathcal{F}_{\mathsf{SSLE}}, \mathcal{S}, \mathcal{Z}^*] = 0] = \frac{1}{n-c}$ due to the definition of $\mathcal{F}_{\mathsf{SSLE}}$, which uniformly at random chooses the leader among n registered parties, where c of them are controlled by the adversary. On the other hand, $Pr[Exec[\pi, \mathsf{Adv}, \mathcal{Z}^*] = 0] \geq \frac{1}{n-c} + Adv_{\mathcal{B}}$, where $Adv_{\mathcal{B}}$ is non-negligible. We have that $\mathcal{Z}^*$ can distinguish between the real and ideal worlds with non-negligible probability, which contradicts the initial assumption about π.

Fairness. An adversary breaks fairness if it can significantly increase the chances for the corrupted parties to be elected. At the end of the fairness experiment, $\mathcal{Z}^*$ learns whether one of the c controlled users (among n registered) is the leader. If this is the case, $\mathcal{Z}^*$ outputs 0 ($\mathcal{Z}^*$ believes it interacts with π), otherwise it outputs 1 ($\mathcal{Z}^*$ believes it interacts with $\mathcal{S}$). On the one hand, we have that $Pr[Exec[\mathcal{F}_{\mathsf{SSLE}}, \mathcal{S}, \mathcal{Z}^*] = 0] = \frac{c}{n}$ due to the definition of $\mathcal{F}_{\mathsf{SSLE}}$, which uniformly at random chooses the leader among n registered parties, where c of them are controlled by the adversary. On the other hand, $Pr[Exec[\pi, \mathsf{Adv}, \mathcal{Z}^*] = 0] \geq \frac{c}{n} + Adv_{\mathcal{B}}$, where $Adv_{\mathcal{B}}$ is non-negligible. We have that $\mathcal{Z}^*$ can distinguish between the real and ideal worlds with non-negligible probability. □

Lemma 5.9.1. *Let $[C_A]$ and $[C_B]$ be the inputs to* OSelectM *protocol, and let $[C']$ be the output. Then, assuming the underlying secret sharing scheme is linearly homomorphic and the primitives for multiplication secret shares and generating a random shared bit are secure, it holds that $C' \in \{C_A, C_B\}$.*

Proof. The underlying RandBit primitive produces shares of a random bit $[b]$. By homomorphic properties of the secret sharing scheme and security of the multiplication primitive, it follows that, if $b = 0$, C' evaluates to C_A, otherwise, if $b = 1$, C' evaluates to C_B. □

Lemma 5.9.2. *Let $[C_1], ..., [C_N]$ be the inputs to* $\mathsf{OSelectM_N}$ *protocol, and let $[\bar{C}]$ be the output. Then, it holds that $\bar{C} \in \{C_1, ..., C_N\}$.*

Proof. It follows from Lemma 5.9.1 and the binary tree structure of OSelectM instances in $\mathsf{OSelectM_N}$. □

Lemma 5.9.3. *Assuming secret sharing is secure, algorithm* Register *in Construction 5.4.1 called by some party, securely implements sending a* (register) *message in the ideal model.*

Proof. The party uses the output value from Register as input to the (register) message in the ideal model. The proof follows from simulatability of the secret sharing scheme. □

Lemma 5.9.4. *Assuming H is a random oracle, algorithm* RegisterVerify *in Construction 5.4.1 securely implements sending a* (regVerify) *message in the ideal model.*

Proof. By the properties of the random oracle, we have that the probability that $C_i \neq C_j$ in the ideal model and $k_{iR} = k_{jR}$ is $1/2^{\lambda}$, which is negligible in λ. □

Lemma 5.9.5. *Algorithm* Elect *in Construction 5.4.1 securely implements sending a* (elect) *message in the ideal model.*

Proof. We construct a simulator $\mathcal{S}$ for an ideal adversary Adv. $\mathcal{S}$ recovers the adversarial input from party $\mathcal{P}_i$ by reconstructing it from the shares available to the simulator ($\mathcal{S}$ controls enough honest parties to reconstruct any shared secret).

$\mathcal{S}$ sends all inputs from honest parties and the recovered adversarial inputs and receives $C_{U_1^N}$ from the ideal functionality as the result of the election. It is the same for all parties, including those controlled by the adversaries, so the simulator forwards this value to Adv. In order to let the adversary believe it interacts with the real protocol, the simulator has to produce a transcript of the $\mathsf{OSelectM_N}$ protocol that will result in a specific value U_N to be chosen and output. To do that, the simulator fixes the shares of the honest parties for random bits $[b]$ in OSelectM instances so that the reconstruction would output the specific fixed b, that will result $\mathsf{OSelectM_N}$ to select precisely the U_N-th element of the sequence $(st_1, \ldots, st_N)$. The underlying secret sharing scheme allows to simulate the transcripts for the honest parties that share a simulator-chosen secret. □

In the real protocol, it is possible that for some $i \neq j$, $k_{iL} = k_{jL}$, while $k_{iR} \neq k_{jR}$ and so the parties would pass the registration. However, this only happens with a low probability that we can control.

Lemma 5.9.6. *Assuming H is a random oracle, Construction 5.4.1 produces a unique leader with the probability at least* $1 - e^{-\frac{N(N-1)}{2p}}$.

Proof. The probability that there exist two parties $\mathcal{P}_i$ and $\mathcal{P}_j$ such that $k_{iL} = k_{jL}$ and $k_{iR} \neq k_{jR}$ can be estimated by the birthday paradox. Specifically, this probabilty is bounded by $e^{-\frac{N(N-1)}{2p}}$. □

Lemma 5.9.7. *Algorithm* Verify *in Construction 5.4.1 securely implements sending a* (verify) *message in the ideal model.*

Proof. It follows by a similar argument as in the proof of Lemma 5.9.4. □

Proof of Theorem 5.4.1. Since we only consider sequential execution, we need to show that adversarial views in the real and the ideal worlds are indistinguishable for one instance of the protocol, and the security of the whole protocol will follow by Canetti's composition theorem [31]. To this end, we construct a simulator for a real-world adversary as follows.

- For the registration, the real-world adversary and honest users use a call to the random oracle H for some (random) input and then share a string. Sharing algorithm can be simulated by a secure secret sharing scheme. Moreover, a t-private secret sharing scheme for $t < N$ allows $\mathcal{S}$ to reconstruct the input used by the corrupted user. Hence, all the numbers shared by the corrupted and honest users during registration are known to $\mathcal{S}$.
- To simulate the verification of registration, $\mathcal{S}$ verifies that there is no duplicate numbers recorded during registration. If this is the case, it outputs 1, otherwise 0.
- To simulate the election, $\mathcal{S}$ first consults the ideal functionality to elect the leader and then we use Lemma 5.9.5 to simulate the corresponding transcript.
- To simulate the verify algorithm, $\mathcal{S}$ compares the elected number with the number registered by the user (honest or malicious) and outputs 1 if the numbers are equal, otherwise it outputs 0.

We argue that any PPT environment $\mathcal{Z}$ cannot distinguish between the ideal world and the real world significantly better than negligible probability via a series or games.

G_0 - the real-world experiment.

G_1 - same as G_0, except that the election always returns a single leader. $\mathsf{G}_1 \approx \mathsf{G}_0$, since the probability of a collisions during registration is negligible by Lemma 5.9.6.

G_2 - same as G_1, except that MPC protocols (secret sharing, secret multiplication, and secret addition) are simulated by Lemma 5.9.5. We have that $\mathsf{G}_2 \approx \mathsf{G}_1$ due to the properties of the secure secret sharing scheme, which allows for efficient simulation.

G_3 - same as G_2, except that $\mathcal{S}$ consults the ideal functionality to determine the leader on the submitted numbers during registration. Since we have at least one honest user in the election who generates a random number for the registration and that the transcript can be simulated by the MPC functionality, we have that $\mathsf{G}_3 = \mathsf{G}_2$.

G_4 - same as G_3, except that during the registration, parties submit random numbers (instead of calls to H), and for the verification they send theses numbers to $\mathcal{S}$. We have that $\mathsf{G}_4 \approx \mathsf{G}_3$ by the properties of the random oracle.

G_5 - the ideal-world experiment. We have that $\mathsf{G}_5 = \mathsf{G}_4$ by construction (parties use random number for the registration and are consulted by the ideal functionality to determine the leader). □

Theorem 5.9.1. *Assuming the underlying primitives are secure in the malicious adversary model, the statement of Theorem 5.4.1 holds in the malicious adversary model.*

Proof. The theorem follows from Canetti's composition theorem [31] and Theorem 5.4.1. □

Lemma 5.9.8. *Assuming H is a random oracle, Construction 5.5.1 produces a unique leader with the probability at least* $1 - e^{-\frac{N(N-1)}{2^{l(\lambda)+1}}}$.

Proof. Analogously to Lemma 5.9.6 with the difference that k_{iL} has $2^{l(\lambda)}$ possible choices. □

Proof of Theorem 5.5.1. Since both our constructions Construction 5.4.1 and Construction 5.5.1 rely on secure MPC primitives and differ only in specifics of the used MPC frameworks, we simply follow the steps in the proof of Theorem 5.4.1 to prove the theorem. □

5.10 Conclusion

In current proof-of-stake cryptocurrencies, the a-priori knowledge of who will append the blockchain may be easily exploited by an adversary and lead to a denial-of-service of the system. The importance of hiding the identity of the block appender was recently recognized, and first solutions to the problem have been proposed by Boneh et al. [18]. In this work, we took a step further and presented an efficient SSLE protocol, whose security relies on MPC. We implemented our solution and microbenchmarked it in a real-world scenario. Our security analysis and performance evaluation indicate that our solution is practical and can be deployed into existing proof-of-stake cryptocurrencies.

Nevertheless, there are open questions that we leave for future work. First, our construction elects a unique leader. If the leader does not show up within some timeout, the parties have to restart the protocol from scratch. It would be interesting to come up with a more efficient solution for this scenario. The secret committee election of size > 1 can be seen as a generalization of the SSLE problem and could solve the mentioned problem. Such a primitive could be used, for example, to privately electing a committee for making treasury decisions as in [136]. However, the generalization of our techniques for this setting is not straightforward. If we naively skip the last OSelect subroutine and declare both candidates as leaders, we know that one candidate is from the left subtree, another from the right, which would contradict the committee's uniform selection. We leave this interesting problem as future work.

And second, the precision of our construction w.r.t. the stake distribution directly affects the communication and computational costs. It would be interesting to design an SSLE scheme that is cost-stable with regard to stake distributions.

6 Conclusion

Information systems that we use in our everyday lives provide various security guarantees and often consist of smaller building blocks. Anonymity properties are essential in many scenarios, therefore cryptographic primitives that capture anonymity by design are important tools when building more complex systems. In this thesis, we have studied four cryptographic primitives with anonymity properties in mind.

We have defined two new primitives, Anonymous RAM and Randomize-or-Change encryption, which capture anonymity properties by definition. Anonymous RAM can be seen as a natural generalization of Oblivious RAM to the setting of multiple mistrusting users, which share the same storage. We have shown two AnonRAM constructions, which differ in asymptotics and assumptions. The first construction is based on a secure ORAM and re-randomizable encryption scheme: users have their own ORAM instances where they store data, and they have to re-randomize the same locations in ORAM instances of all other users, thereby achieving anonymity. The second construction requires two non-colluding servers and achieves polylogarithmic complexity in the number of users and cells per access.

Randomize-or-Change (RoC) encryption allows users to chain ciphertexts in a way, that an external adversary cannot distinguish between re-randomizations or new encryptions, yet malicious parties cannot break integrity of plaintexts. In an extension for RoC called aggregatable RoC, multiple pairs of ciphertexts that have the same input ciphertext can be efficiently and publicly aggregated. Using aggregatable RoC, one can construct a parallel anonymous RAM, which has the benefit that the users can simultaneously access their storage.

Furthermore, we have presented efficient constructions for two important primitives. The first primitive is non-interactive zero-knowledge (NIZK) proofs. In this thesis, we have proposed an efficient way of constructing NIZK proofs for statements that consist of algebraic and arithmetic parts simultaneously. Each part is taken care of efficiently by the respective technique known in the literature, and we glue these parts together in a non-blackbox way, almost at no additional cost. NIZK proofs can be naturally used in anonymous scenarios, since the prover does not have to reveal her identity to the verifier(s).

Finally, we have proposed a framework for efficiently constructing Single Secret Leader Election protocols. The problem was recently recognized and formalized by Boneh et al. [18]. Our framework relies on secure multi-party computation (MPC) and does not require any sorting or shuffling of the underlying data.

Future work In this thesis, we have presented two new primitives and improved efficiency in another two primitives, where each of them has anonymity properties in mind. Clearly, we haven't covered all research problems related to anonymity that need to be addressed. Below, we list several directions for future work.

- Our single server AnonRAM construction has a linear cost per access in the number of users. It would be interesting to construct a single server AnonRAM with a polylogarithmic cost.

- It would be interesting to see how the ideas from randomize-or-change encryption can be extended to constructing *verifiable* proxy-reencryption with possibility of universal re-randomization.
- It would be interesting to explore whether the approach by Ames et al. [4] can be used to achieve yet more efficient and compact NIZK proofs in cross-domains.
- Currently, our experiments show that our SSLE protocol instance can run in under 7 minutes in a real-world scenario for a hundred of users. It would be an interesting challenge to bring this number to the order of tens of seconds for the same setting.

Future work does not stop at this point and may touch cryptographic building blocks which are not mentioned in this thesis or are not yet defined, e.g., the SSLE problem was formalized very recently. Another direction for future research is to address post-quantum security in the existing primitives, many of which will no longer be secure in the presence of a quantum attacker. The more efficient cryptographic building blocks exist at our disposal, the more complex systems we could efficiently build and achieve the desired security and anonymity properties.

Bibliography

Author's Papers for this Thesis

[P1] Backes, M., Herzberg, A., Kate, A., and Pryvalov, I. Anonymous ram. In: *European Symposium on Research in Computer Security–ESORICS 2016*. Springer. 2016, 344–362.

[P2] Backes, M., Herzberg, A., Kate, A., and Pryvalov, I. Randomize-or-Change Encryption. In: *Under submission*.

[P3] Backes, M., Herzberg, A., Kate, A., and Pryvalov, I. Touch-or-change: multi-user privacy and integrity in universally re-randomizable encryption. In: *Grande Region Security and Reliability Day (GRSRD)*. 2017.

[P4] Backes, M., Hanzlik, L., Herzberg, A., Kate, A., and Pryvalov, I. Efficient non-interactive zero-knowledge proofs in cross-domains without trusted setup. In: *IACR International Workshop on Public Key Cryptography*. Springer. 2019, 286–313.

[P5] Backes, M., Berrang, P., Hanzlik, L., and Pryvalov, I. A framework for constructing Single Secret Leader Election from MPC. In: *European Symposium on Research in Computer Security–ESORICS 2022*. 2022, to appear.

Other Papers of the Author

[S1] Eigner, F., Kate, A., Maffei, M., Pampaloni, F., and Pryvalov, I. Differentially private data aggregation with optimal utility. In: *Proceedings of the 30th Annual Computer Security Applications Conference*. 2014, 316–325.

[S2] Eigner, F., Kate, A., Maffei, M., Pampaloni, F., and Pryvalov, I. Privacy-preserving data aggregation with optimal utility using arithmetic smc. In: *Workshop on Usable and Efficient Secure Multiparty Computation (UaESMC)*. 2014.

[S3] Eigner, F., Kate, A., Maffei, M., Pampaloni, F., and Pryvalov, I. Privada: a generic framework for privacy-preserving data aggregation. In: *Grande Region Security and Reliability Day (GRSRD)*. 2014.

Other references

[1] Adida, B. Helios: web-based open-audit voting. In: *USENIX security symposium*. Vol. 17. 2008, 335–348.

[2] Agrawal, S., Ganesh, C., and Mohassel, P. Non-interactive zero-knowledge proofs for composite statements. In: *Advances in Cryptology–CRYPTO 2018 (3)*. Springer. 2018, 643–673.

[3] Aliasgari, M., Blanton, M., Zhang, Y., and Steele, A. Secure computation on floating point numbers. In: *NDSS*. 2013.

[4] Ames, S., Hazay, C., Ishai, Y., and Venkitasubramaniam, M. Ligero: lightweight sublinear arguments without a trusted setup. In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM. 2017, 2087–2104.

[5] Aranha, D. F., Gouvêa, C. P. L., Markmann, T., Wahby, R. S., and Liao, K. *RELIC is an Efficient LIbrary for Cryptography*. `https://github.com/relic-toolkit/relic`.

[6] Ateniese, G., Camenisch, J., and Medeiros, B. de. Untraceable RFID tags via insubvertible encryption. In: *Proceedings of the 12th ACM conference on Computer and communications security*. ACM. 2005, 92–101.

[7] Ateniese, G., Fu, K., Green, M., and Hohenberger, S. Improved proxy re-encryption schemes with applications to secure distributed storage. *ACM Transactions on Information and System Security (TISSEC)* 9, 1 (2006), 1–30.

[8] Baldimtsi, F. and Lysyanskaya, A. Anonymous credentials light. In: *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*. ACM. 2013, 1087–1098.

[9] Bangerter, E., Camenisch, J., and Maurer, U. Efficient proofs of knowledge of discrete logarithms and representations in groups with hidden order. In: *Public Key Cryptography–PKC 2005*. Springer. 2005, 154–171.

[10] Bayer, S. and Groth, J. Efficient zero-knowledge argument for correctness of a shuffle. In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 2012, 263–280.

[11] Belenkiy, M., Chase, M., Kohlweiss, M., and Lysyanskaya, A. P-signatures and noninteractive anonymous credentials. In: *Theory of Cryptography Conference*. Springer. 2008, 356–374.

[12] Bellare, M. and Namprempre, C. Authenticated encryption: relations among notions and analysis of the generic composition paradigm. In: *Advances in Cryptology–ASIACRYPT 2000*. Springer. 2000, 531–545.

[13] Ben-Sasson, E., Chiesa, A., Genkin, D., Tromer, E., and Virza, M. SNARKs for C: Verifying program executions succinctly and in zero knowledge. In: *Advances in Cryptology–CRYPTO 2013*. Springer, 2013, 90–108.

[14] Bentov, I., Gabizon, A., and Zuckerman, D. Bitcoin beacon. *arXiv preprint arXiv:1605.04559* (2016).

[15] Bindschaedler, V., Naveed, M., Pan, X., Wang, X., and Huang, Y. Practicing oblivious access on cloud storage: the gap, the fallacy, and the new way forward. In: *CCS*. 2015, 837–849.

[16] Boneh, D. The decision diffie-hellman problem. In: *Algorithmic number theory*. Springer, 1998, 48–63.

[17] Boneh, D., Bonneau, J., Bünz, B., and Fisch, B. Verifiable delay functions. In: *Annual international cryptology conference*. Springer. 2018, 757–788.

[18] Boneh, D., Eskandarian, S., Hanzlik, L., and Greco, N. Single secret leader election. In: *Proceedings of the 2nd ACM Conference on Advances in Financial Technologies*. ACM, 2020, 12–24.

[19] Boneh, D., Gennaro, R., Goldfeder, S., Jain, A., Kim, S., Rasmussen, P. M., and Sahai, A. Threshold cryptosystems from threshold fully homomorphic encryption. In: *Annual International Cryptology Conference*. Springer. 2018, 565–596.

[20] Boneh, D., Lynn, B., and Shacham, H. Short signatures from the weil pairing. *Journal of cryptology* 17, 4 (2004), 297–319.

[21] Bonneau, J., Clark, J., and Goldfeder, S. On bitcoin as a public randomness source. *IACR Cryptol. ePrint Arch.* 2015 (2015), 1015.

[22] Boyen, X. Multipurpose identity-based signcryption. In: *Advances in Cryptology–CRYPTO 2003*. Springer, 2003, 383–399.

[23] Boyle, E., Chung, K.-M., and Pass, R. Oblivious parallel RAM and applications. In: *TCC*. 2016, 175–204.

[24] Brands, S. A. *Rethinking public key infrastructures and digital certificates: building in privacy*. Mit Press, 2000.

[25] Bünz, B., Bootle, J., Boneh, D., Poelstra, A., Wuille, P., and Maxwell, G. Bulletproofs: short proofs for confidential transactions and more. In: *IEEE Symposium on Security and Privacy (SP)*. IEEE. 2018, 319–338.

[26] Camenisch, J. and Lysyanskaya, A. An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In: *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 2001, 93–118.

[27] Camenisch, J. and Lysyanskaya, A. Signature schemes and anonymous credentials from bilinear maps. In: *Advances in Cryptology–CRYPTO 2004*. Springer. 2004, 56–72.

[28] Camenisch, J. and Shoup, V. Practical verifiable encryption and decryption of discrete logarithms. In: *Advances in Cryptology-CRYPTO*. 2003, 126–144.

[29] Camenisch, J. and Stadler, M. Efficient group signature schemes for large groups. In: *Advances in Cryptology–CRYPTO '97*. Springer. 1997, 410–424.

[30] Camenisch, J. and Stadler, M. *Proof systems for general statements about discrete logarithms*. Tech. rep. TR260, Dept. of Computer Science, ETH Zürich. 1997.

[31] Canetti, R. Security and composition of multiparty cryptographic protocols. *Journal of CRYPTOLOGY* 13, 1 (2000), 143–202.

[32] Canetti, R. and Hohenberger, S. Chosen-ciphertext secure proxy re-encryption. In: *Proceedings of the 14th ACM conference on Computer and communications security*. ACM. 2007, 185–194.

[33] Canetti, R., Krawczyk, H., and Nielsen, J. B. Relaxing chosen-ciphertext security. In: *Advances in Cryptology–CRYPTO 2003*. Springer, 2003, 565–582.

[34] Cascudo, I. and David, B. Scrape: scalable randomness attested by public entities. In: *International Conference on Applied Cryptography and Network Security*. Springer. 2017, 537–556.

[35] Catalano, D. and Gennaro, R. New efficient and secure protocols for verifiable signature sharing and other applications. In: *CRYPTO*. 1998, 105–120.

[36] Catrina, O. and Saxena, A. Secure computation with fixed-point numbers. In: *International Conference on Financial Cryptography and Data Security*. Springer. 2010, 35–50.

[37] Chase, M., Derler, D., Goldfeder, S., Orlandi, C., Ramacher, S., Rechberger, C., Slamanig, D., and Zaverucha, G. Post-quantum zero-knowledge and signatures from symmetric-key primitives. In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM. 2017, 1825–1842.

[38] Chase, M., Ganesh, C., and Mohassel, P. Efficient zero-knowledge proof of algebraic and non-algebraic statements with applications to privacy preserving credentials. In: *Advances in Cryptology–CRYPTO 2016*. Springer. 2016, 499–530.

[39] Chase, M., Kohlweiss, M., Lysyanskaya, A., and Meiklejohn, S. Malleable proof systems and applications. In: *Advances in Cryptology–EUROCRYPT 2012*. Springer. 2012, 281–300.

[40] Chaum, D. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM* 4, 2 (1981), 84–88.

[41] Chaum, D. The dining cryptographers problem: unconditional sender and recipient untraceability. *J. Cryptology* 1, 1 (1988), 65–75.

[42] Chaum, D., Evertse, J.-H., Graaf, J. van de, and Peralta, R. Demonstrating possession of a discrete logarithm without revealing it. In: *Advances in Cryptology–CRYPTO '86*. Springer. 1986, 200–212.

[43] Chaum, D. and Pedersen, T. P. Wallet databases with observers. In: *CRYPTO*. 1992, 89–105.

[44] Chen, B., Lin, H., and Tessaro, S. Oblivious parallel RAM: Improved efficiency and generic constructions. In: *TCC*. 2016, 205–234.

[45] Chung, K.-M., Liu, Z., and Pass, R. Statistically-secure oram with $\tilde{O}(\log^2 n)$ overhead. In: *International Conference on the Theory and Application of Cryptology and Information Security*. Springer. 2014, 62–81.

[46] Clark, J. and Hengartner, U. On the use of financial data as a random beacon. *EVT/WOTE* 89 (2010).

[47] Cramer, R., Damgård, I., and Ishai, Y. Share conversion, pseudorandom secret-sharing and applications to secure computation. In: *Theory of Cryptography Conference*. Springer. 2005, 342–362.

[48] Cramer, R., Damgård, I., and Maurer, U. General secure multi-party computation from any linear secret-sharing scheme. In: *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 2000, 316–334.

[49] Cramer, R., Damgård, I., and Schoenmakers, B. Proofs of partial knowledge and simplified design of witness hiding protocols. In: *CRYPTO*. 1994, 174–187.

[50] Damgård, I., Fitzi, M., Kiltz, E., Nielsen, J. B., and Toft, T. Unconditionally secure constant-rounds multi-party computation for equality, comparison, bits and exponentiation. In: *Theory of Cryptography Conference*. Springer. 2006, 285–304.

[51] Danezis, G., Dingledine, R., and Mathewson, N. Mixminion: Design of a Type III anonymous remailer protocol. In: *Security and Privacy (S&P)*. 2003, 2–15.

[52] Danezis, G. Breaking four mix-related schemes based on universal re-encryption. In: *Information Security*. Springer, 2006, 46–59.

[53] De Santis, A., Di Crescenzo, G., Persiano, G., and Yung, M. On monotone formula closure of SZK. In: *FOCS*. 1994, 454–465.

[54] Diffie, W. and Hellman, M. New directions in cryptography. *IEEE transactions on Information Theory* 22, 6 (1976), 644–654.

[55] Dingledine, R., Mathewson, N., and Syverson, P. Tor: The second-generation onion router. In: *Usenix Security*. 2004, 303–320.

[56] Dobraunig, C., Eichlseder, M., Grassi, L., Lallemand, V., Leander, G., List, E., Mendel, F., and Rechberger, C. Rasta: a cipher with low anddepth and few ands per bit. In: *Annual International Cryptology Conference*. Springer. 2018, 662–692.

[57] Dodis, Y. and Yampolskiy, A. A verifiable random function with short proofs and keys. In: *Public Key Cryptography-PKC*. 2005, 416–431.

[58] Escala, A. and Groth, J. Fine-tuning Groth-Sahai proofs. In: *Public-Key Cryptography–PKC 2014*. Springer. 2014, 630–649.

[59] Fairbrother, P. An improved construction for universal re-encryption. In: *International Workshop on Privacy Enhancing Technologies*. Springer. 2004, 79–87.

[60] Fiat, A. and Shamir, A. How to prove yourself: practical solutions to identification and signature problems. In: *Conference on the theory and application of cryptographic techniques*. Springer. 1986, 186–194.

[61] Fiat, A. and Shamir, A. How to prove yourself: practical solutions to identification and signature problems. In: *Advances in Cryptology–CRYPTO '86*. Springer. 1987, 186–194.

[62] França, B., Wissfeld, M., Berrang, P., Styp-Rekowsky, P. von, and Trinkler, R. Albatross: an optimistic consensus algorithm. *arXiv preprint arXiv:1903.01589* (2019).

[63] Franz, M., Williams, P., Carbunar, B., Katzenbeisser, S., Peter, A., Sion, R., and Sotakova, M. Oblivious outsourced storage with delegation. In: *FC*. 2012, 127–140.

[64] Freedman, M. J., Ishai, Y., Pinkas, B., and Reingold, O. Keyword search and oblivious pseudorandom functions. In: *TCC*. 2005, 303–324.

[65] Ganesh, C., Orlandi, C., and Tschudi, D. Proof-of-Stake Protocols for Privacy-Aware Blockchains. In: *EUROCRYPT 2019*. Springer.

[66] Gennaro, R., Gentry, C., Parno, B., and Raykova, M. Quadratic span programs and succinct NIZKs without PCPs. In: *Advances in Cryptology–EUROCRYPT 2013*. Springer. 2013, 626–645.

[67] Gennaro, R., Jarecki, S., Krawczyk, H., and Rabin, T. Secure distributed key generation for discrete-log based cryptosystems. *Journal of Cryptology* 20, 1 (2007), 51–83.

[68] Gennaro, R., Jarecki, S., Krawczyk, H., and Rabin, T. Secure distributed key generation for discrete-log based cryptosystems. In: *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 1999, 295–310.

[69] Gennaro, R., Rabin, M. O., and Rabin, T. Simplified vss and fast-track multiparty computations with applications to threshold cryptography. In: *Proceedings of the seventeenth annual ACM symposium on Principles of distributed computing*. 1998, 101–111.

[70] Gentry, C., Sahai, A., and Waters, B. Homomorphic encryption from learning with errors: conceptually-simpler, asymptotically-faster, attribute-based. In: *Annual Cryptology Conference*. Springer. 2013, 75–92.

[71] Giacomelli, I., Madsen, J., and Orlandi, C. Zkboo: faster zero-knowledge for boolean circuits. In: *USENIX Security Symposium*. 2016, 1069–1083.

[72] Gilad, Y., Hemo, R., Micali, S., Vlachos, G., and Zeldovich, N. Algorand: scaling byzantine agreements for cryptocurrencies. In: *Proceedings of the 26th Symposium on Operating Systems Principles*. ACM. 2017, 51–68.

[73] Goldreich, O., Micali, S., and Wigderson, A. How to prove all NP statements in zero-knowledge and a methodology of cryptographic protocol design. In: *Advances in Cryptology–CRYPTO '86*. Springer. 1986, 171–185.

[74] Goldreich, O. and Ostrovsky, R. Software protection and simulation on oblivious RAMs. *Journal of the ACM (JACM)* 43, 3 (1996), 431–473.

[75] Goldschlag, D. M. and Stubblebine, S. G. Publicly verifiable lotteries: applications of delaying functions. In: *International Conference on Financial Cryptography*. Springer. 1998, 214–226.

[76] Goldwasser, S., Micali, S., and Rackoff, C. The knowledge complexity of interactive proof systems. *SIAM Journal on computing* 18, 1 (1989), 186–208.

[77] Golle, P., Jakobsson, M., Juels, A., and Syverson, P. Universal re-encryption for mixnets. In: *Topics in Cryptology–CT-RSA*. 2004, 163–178.

[78] Goodrich, M. T. Randomized shellsort: a simple oblivious sorting algorithm. In: *Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete Algorithms*. Society for Industrial and Applied Mathematics. 2010, 1262–1277.

[79] Goodrich, M. T. Randomized shellsort: a simple data-oblivious sorting algorithm. *Journal of the ACM (JACM)* 58, 6 (2011), 27.

[80] Goodrich, M. T., Mitzenmacher, M., Ohrimenko, O., and Tamassia, R. Oblivious RAM simulation with efficient worst-case access overhead. In: *ACM CCSW*. 2011, 95–100.

[81] Goodrich, M. T., Mitzenmacher, M., Ohrimenko, O., and Tamassia, R. Privacy-preserving group data access via stateless oblivious RAM simulation. In: *SODA*. 2012, 157–167.

[82] Groth, J. Rerandomizable and replayable adaptive chosen ciphertext attack secure cryptosystems. In: *Theory of Cryptography (TCC 2004)*. Springer. 2004, 152–170.

[83] Groth, J. Non-interactive zero-knowledge arguments for voting. In: *International Conference on Applied Cryptography and Network Security*. Springer. 2005, 467–482.

[84] Groth, J. Short pairing-based non-interactive zero-knowledge arguments. In: *Advances in Cryptology–ASIACRYPT 2010*. Springer. 2010, 321–340.

[85] Groth, J. and Sahai, A. Efficient non-interactive proof systems for bilinear groups. In: *Advances in Cryptology–EUROCRYPT 2008*. Springer, 2008, 415–432.

[86] Ishai, Y., Kushilevitz, E., Ostrovsky, R., and Sahai, A. Zero-knowledge from secure multiparty computation. In: *Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*. ACM. 2007, 21–30.

[87] Jarecki, S. and Liu, X. Efficient Oblivious Pseudorandom Function with applications to adaptive OT and secure computation of set intersection. In: *TCC*. 2009, 577–594.

[88] Jawurek, M., Kerschbaum, F., and Orlandi, C. Zero-knowledge using garbled circuits: how to prove non-algebraic statements efficiently. In: *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*. ACM. 2013, 955–966.

[89] Jinsheng, Z., Wensheng, Z., and Qiao, D. *A Multi-user Oblivious RAM for Outsourced Data*. `http://lib.dr.iastate.edu/cs_techreports/262/`. 2014.

[90] Katz, J. and Lindell, Y. *Introduction to modern cryptography*. CRC Press, 2007.

[91] Kiayias, A., Russell, A., David, B., and Oliynykov, R. Ouroboros: a provably secure proof-of-stake blockchain protocol. In: *Annual International Cryptology Conference*. Springer. 2017, 357–388.

[92] Klonowski, M., Kutylowski, M., Lauks, A., and Zagorski, F. Universal re-encryption of signatures and controlling anonymous information flow. *WARTACRYPT* 4 (2004), 1–3.

[93] Kolesnikov, V. and Schneider, T. Improved garbled circuit: free xor gates and applications. In: *International Colloquium on Automata, Languages, and Programming*. Springer. 2008, 486–498.

[94] Kushilevitz, E., Lu, S., and Ostrovsky, R. On the (in) security of hash-based oblivious RAM and a new balancing scheme. In: *SODA*. 2012, 143–156.

[95] Lenstra, A. K. and Wesolowski, B. A random zoo: sloth, unicorn, and trx. *IACR Cryptol. ePrint Arch.* 2015 (2015), 366.

[96] Lu, S. and Ostrovsky, R. Distributed oblivious RAM for secure two-party computation. In: *TCC*. 2013, 377–396.

[97] Maffei, M., Malavolta, G., Reinert, M., and Schröder, D. Privacy and access control for outsourced personal records. In: *Security and Privacy (S&P)*. 2015, 341–358.

[98] Micali, S., Rabin, M., and Vadhan, S. Verifiable random functions. In: *40th annual symposium on foundations of computer science (cat. No. 99CB37039)*. IEEE. 1999, 120–130.

[99] Nakamoto, S. *Bitcoin: A peer-to-peer electroniccash system.* `https://bitcoin.org/bitcoin.pdf`. 2008.

[100] Naor, M., Pinkas, B., and Sumner, R. Privacy preserving auctions and mechanism design. In: *Proceedings of the 1st ACM conference on Electronic commerce*. ACM. 1999, 129–139.

[101] Neff, C. A. A verifiable secret shuffle and its application to e-voting. In: *Proceedings of the 8th ACM conference on Computer and Communications Security*. ACM. 2001, 116–125.

[102] Nielsen, J. B., Nordholt, P. S., Orlandi, C., and Burra, S. S. A new approach to practical active-secure two-party computation. In: *Annual Cryptology Conference*. Springer. 2012, 681–700.

[103] O'Dwyer, K. J. and Malone, D. Bitcoin mining and its energy footprint (2014), 280–285.

[104] Ohrimenko, O., Goodrich, M. T., Tamassia, R., and Upfal, E. The melbourne shuffle: improving oblivious storage in the cloud. In: *International Colloquium on Automata, Languages, and Programming*. Springer. 2014, 556–567.

[105] Ostrovsky, R. Efficient computation on oblivious RAMs. In: *STOC*. 1990, 514–523.

[106] Park, C., Itoh, K., and Kurosawa, K. Efficient anonymous channel and all/nothing election scheme. In: *Workshop on the Theory and Application of of Cryptographic Techniques*. Springer. 1993, 248–259.

[107] Pedersen, T. P. Non-interactive and information-theoretic secure verifiable secret sharing. In: *Advances in Cryptology–CRYPTO '91*. Springer. 1991, 129–140.

[108] Pinkas, B. and Reinman, T. Oblivious RAM revisited. In: *CRYPTO*. 2010, 502–519.

[109] Prabhakaran, M. and Rosulek, M. Rerandomizable RCCA encryption. In: *CRYPTO*. 2007, 517–534.

[110] Rabin, T. and 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*. 1989, 73–85.

[111] Regev, O. On lattices, learning with errors, random linear codes, and cryptography. In: *Proceedings of the thirty-seventh annual ACM symposium on Theory of computing*. 2005, 84–93.

[112] Sahin, C., Zakhary, V., El Abbadi, A., Lin, H. R., and Tessaro, S. Taostore: overcoming asynchronicity in oblivious data storage. In: *Security and Privacy (S&P)*. 2016.

[113] Schnorr, C.-P. Efficient identification and signatures for smart cards. In: *Conference on the Theory and Application of Cryptology*. Springer. 1989, 239–252.

[114] Schnorr, C.-P. Efficient signature generation by smart cards. *J. Cryptology* 4, 3 (1991), 161–174.

[115] Shamir, A. How to share a secret. *Communications of the ACM* 22, 11 (1979), 612–613.

[116] Shi, E., Chan, T.-H. H., Stefanov, E., and Li, M. Oblivious RAM with $O((\log N)^3)$ worst-case cost. In: *International Conference on The Theory and Application of Cryptology and Information Security–ASIACRYPT*. 2011, 197–214.

[117] Shor, P. W. Algorithms for quantum computation: discrete logarithms and factoring. In: *Proceedings of the 35th Annual Symposium on Foundations of Computer Science*. IEEE. 1994, 124–134.

[118] *Single-Leader Election (SSLE)*. `https://github.com/protocol/research-RFPs/blob/master/RFPs/rfp-6-SSLE.md`. 2019.

[119] Songhori, E. M., Hussain, S. U., Sadeghi, A.-R., Schneider, T., and Koushanfar, F. Tinygarble: highly compressed and scalable sequential garbled circuits. In: *2015 IEEE Symposium on Security and Privacy (SP)*. IEEE. 2015, 411–428.

[120] Stefanov, E. and Shi, E. ObliviStore: High performance oblivious cloud storage. In: *Security and Privacy (S&P)*. 2013, 253–267.

[121] Stefanov, E., Van Dijk, M., Shi Elaine and Fletcher, C., Ren, L., Yu, X., and Devadas, S. Path ORAM: An extremely simple oblivious RAM protocol. In: *CCS*. 2013, 299–310.

[122] Syta, E., Jovanovic, P., Kogias, E. K., Gailly, N., Gasser, L., Khoffi, I., Fischer, M. J., and Ford, B. Scalable bias-resistant distributed randomness. In: *2017 IEEE Symposium on Security and Privacy (SP)*. Ieee. 2017, 444–460.

[123] *Technical background of version 1 bitcoin addresses*. `https://en.bitcoin.it/wiki/Technical_background_of_version_1_Bitcoin_addresses`. Accessed: 2018-10-09.

[124] *The Tor Project*. `https://www.torproject.org/`. Accessed Feb 2016. 2003.

[125] Tsoukalas, G., Papadimitriou, K., Louridas, P., and Tsanakas, P. From helios to zeus. In: *EVT/WOTE*. 2013.

[126] Wang, X., Chan, T.-H. H., and Shi, E. Circuit oram: on tightness of the goldreich-ostrovsky lower bound. In: *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. 2015, 850–861.

[127] Wang, X., Malozemoff, A. J., and Katz, J. *EMP-toolkit: Efficient MultiParty computation toolkit*. `https://github.com/emp-toolkit`. 2016.

[128] Wang, X., Ranellucci, S., and Katz, J. Authenticated garbling and efficient maliciously secure two-party computation. In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. 2017, 21–37.

[129] Wang, X., Ranellucci, S., and Katz, J. Global-scale secure multiparty computation. In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. 2017, 39–56.

[130] Williams, P., Sion, R., and Tomescu, A. PrivateFS: a parallel oblivious file system. In: *CCS*. 2012, 977–988.

[131] Yao, A. C. Protocols for secure computations. In: *23rd annual symposium on foundations of computer science (sfcs 1982)*. IEEE. 1982, 160–164.

[132] Yao, A. C.-C. How to generate and exchange secrets. In: *Proceedings of the 27th Annual Symposium on Foundations of Computer Science*. IEEE. 1986, 162–167.

[133] Young, A. L. and Yung, M. Semantically secure anonymity: foundations of re-encryption. In: *International Conference on Security and Cryptography for Networks*. Springer. 2018, 255–273.

[134] Zahur, S., Rosulek, M., and Evans, D. Two halves make a whole. In: *Advances in Cryptology–EUROCRYPT 2015*. Springer. 2015, 220–250.

[135] *Zcash Parameter Generation.* `https://z.cash/technology/paramgen.html`. Accessed: 2018-10-08.

[136] Zhang, B., Oliynykov, R., and Balogun, H. A treasury system for cryptocurrencies: enabling better collaborative intelligence. In: *The Network and Distributed System Security Symposium 2019.* 2019.

[137] Zheng, Y. Digital signcryption or how to achieve cost (signature & encryption) $\ll$ cost (signature)+ cost (encryption). In: *Advances in Cryptology–CRYPTO'97.* Springer, 1997, 165–179.