Zero-Knowledge Proofs: Succinct Verification, Distributed Proofs and Lookup Arguments

A THESIS

SUBMITTED FOR THE DEGREE OF

Doctor of Philosophy

IN THE

Faculty of Engineering

BY

Moumita Dutta

Computer Science and Automation
Indian Institute of Science
Bangalore – 560 012 (INDIA)

November, 2025

Declaration of Originality

I, **Moumita Dutta**, with SR No. **04-04-00-10-12-20-1-18335** hereby declare that the material presented in the thesis titled

Zero-Knowledge Proofs: Succinct Verification, Distributed Proofs and Lookup Arguments

represents original work carried out by me in the **Department of Computer Science and Automation** at **Indian Institute of Science** during the years **2020-2025**.
With my signature, I certify that:

- I have not manipulated any of the data or results.
- I have not committed any plagiarism of intellectual property. I have clearly indicated and referenced the contributions of others.
- I have explicitly acknowledged all collaborative research and discussions.
- I have understood that any false claim will result in severe disciplinary action.
- I have understood that the work may be screened for any form of academic misconduct.

Date: Monday, June 2nd, 2025 Student Signature

In my capacity as supervisor of the above-mentioned work, I certify that the above statements are true to the best of my knowledge, and I have carried out due diligence to ensure the originality of the report.

Advisor Name: Chaya Ganesh, Arpita Patra Advisor Signature

DEDICATED TO

Mom, Dad & Sis

Acknowledgements

Before I start, I would like to thank my mom, dad, and sister without whom I would not have been able to embark on this journey or complete it.

First and foremost, I would like to express my sincerest gratitude to my supervisors, Prof. Chaya Ganesh and Prof. Arpita Patra, for introducing me to the exciting world of cryptography and providing constant guidance, intellectual freedom, and unwavering support. I am grateful to Prof. Chaya Ganesh for her crucial role in shaping my academic direction through numerous insightful meetings and valuable feedback, and I am grateful to Prof. Arpita Patra for her invaluable support, understanding, and the trust she placed in me to pursue my research directions. Their extensive knowledge, deep passion for research, and continuous encouragement remain a constant source of inspiration, consistently motivating me to strive to become a better researcher.

I would like to extend my gratitude to my mentors, Sikhar Patranabis and Nitin Singh, for their patient guidance during my internships. I am especially grateful for their mentorship in academic writing and the time they generously spent listening to my evolving ideas and engaging in numerous insightful discussions. I also extend my thanks to my co-authors for their hard work and dedication to our collaborative projects.

Beyond my thesis research, I am also grateful to professors whose passion and guidance made my coursework a delightful learning experience. I am particularly thankful to Prof. L. Sunil Chandran for facilitating a highly interactive session that empowers us to arrive at our own solutions, as well as his constant encouragement and guidance. I am also grateful to Prof. Apoorva Khare for his enthusiastic outlook on problems and support. I am also grateful to Dr. Buddhadeb Sau, Prof. Shamik Ghosh and other professors at Jadavpur University for their enthusiastic encouragement during my bachelors and masters, and Raju Sir for believing in me since childhood, which was instrumental in my decision to pursue research.

In addition to the individual contributions enumerated above, I would like to thank all my amazing colleagues at CrIS and CDTG labs for making this journey enjoyable and memorable. I am deeply grateful to my seniors, Protik Paul and Shravani Patil, for their invaluable assis-

tance and willingness to discuss any challenges I encountered, both within and beyond research. Navigating research problems with Protik Paul, Banashri Karmakar and Sikhar Patranabis has always been very enjoyable and fulfilling. It has been exciting sharing my Ph.D. journey with Girisha B. Shankar, who was a wise and insightful presence throughout this process. During my research journey, I treasure all my interactions with my precious labmates, seniors and colleagues: Protik Paul, Shravani Patil, Banashri Karmakar, Girisha B. Shankar, Bhavish Raj Gopal, Soumya Kanti Saha, Riddhiman Dutta Roy, Shyam Murthy, Siddharth Agarwal, Siddharth Kapoor, Nishat Koti, Varsha Bhat Kukkala, Sruthi Sekar, Ajith Suresh, and Divya Ravi. I also cherish the fun times with Arunachaleshwaran, Anoushka, Aditya, Anantha, Vandana, Sandhya, Sankha, Shreyas, Anirban, Adarsh, Pabitra, Suvankar, Ravghavendra and Sindhura. I would also like to thank Kushael ma'am, Shubha ma'am, Padmavati ma'am, Akshaynath, Sudesh bhaiyya, and other staff at the CSA office for all of their help in resolving administrative queries.

I would like to take this opportunity to also express my gratitude to my incredible friends. A special thanks to Mayur and Sanyogita for always being there and listening to my research ramblings more often than not. I am grateful to them, as well as Prem bhaiyya and Kiran di, for illuminating my thoughts through random debates and thought-provoking discussions. I deeply thank Atreyee for the solace of our late-night discussions. I am thankful to have met amazing batchmates like Rishikesh, Atasi, Prashanthi, Swetha, Neha and Aditya, who also made the (online) coursework a lovely experience. I am also thankful to my co-interns, Animesh, Santosh, Ashita and others, for contributing to an enjoyable and productive internship experience. I would like to thank my friends at Jadavpur University and Kendriya Vidyalaya for sharing this journey with me towards research.

Finally, I would like to express my gratitude to my extended family for their love and constant support throughout. Thank you!

Abstract

Zero-Knowledge Proofs (ZKPs) are fundamental cryptographic tools enabling a prover to convince a verifier about the knowledge of a secret witness related to a public statement, without revealing any information beyond the validity of the claim. zk-SNARKs, acronym for zero-knowledge Succinct Non-interactive ARguments of Knowledge, offer efficient ZKPs with succinct communication, prover, and verifier complexities. Additionally, contrary to traditional ZKPs that tackles privacy concerns, there exists applications where privacy is not a requirement and efficiency is the primary concern - for instance to enable powerful (and potentially untrusted) server(s) to perform computationally expensive tasks and provide efficiently verifiable proofs of correct computation of the specified function. Furthermore, there are applications where proof generation being distributed enhances trust and security.

This thesis explores various efficiency dimensions in the context of zero-knowledge proofs. First, we study compressed sigma protocols, enhancing a core ZKP building block of sigma protocols, to achieve logarithmic verification complexity while maintaining logarithmic proof size. Second, we explore applications where privacy is a requirement in an event of distribution of proof generation. In particular, we elevate the building blocks for ZKPs for scenarios where the prover wishes to distribute the proof generation to a set of workers by secret-sharing its private witness; and explore its significance on the application of providing efficient framework for authentication of inputs used inside secure Multi-Party Computation (MPC). Finally, we investigate applications like scalable blockchain rollups, where the primary goal is obtaining efficiently verifiable proofs. We present a batching-efficient Random Access Memory (RAM) framework that proves the correctness of m updates on a RAM of size N (where m is significantly smaller than N), where we improve efficiency of proof generation while further improving proof size and preserving optimal verification complexity. This is achieved by realising efficient updatable lookup arguments, that helps us perform m lookups on a N-size table, even after the table has been updated at a few positions.

Publications based on this Thesis

The results of this thesis have led to the following publications. The author names in the publications are ordered lexicographically based on the surname.

1. Succinct Verification of Compressed Sigma Protocols in the Updatable SRS setting [53]
   Authors: **Moumita Dutta**, Chaya Ganesh, Neha Jawalkar
   Accepted at PKC 2024

2. Compute, but Verify: Efficient Multiparty Computation over Authenticated Inputs [55]
   Authors: **Moumita Dutta**, Chaya Ganesh, Sikhar Patranabis, Nitin Singh
   Accepted at ASIACRYPT 2024

3. Batching-Efficient RAM using Updatable Lookup Arguments [54]
   Authors: **Moumita Dutta**, Chaya Ganesh, Sikhar Patranabis, Shubh Prakash, Nitin Singh
   Accepted at ACM CCS 2024

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Zero-Knowledge Proofs (ZKP) enable a party, known as the *prover*, to convince another party, known as the *verifier*, of the validity of a statement without revealing anything else about the witness beyond the validity of the statement. Introduced by Goldwasser, Micali and Rackoff in [71], ZKPs are a powerful cryptographic tool that are used as fundamental building blocks in various real-world applications, e.g. facilitating decentralized anonymous payments like Zerocash by Ben Sasson et al. [20]. The efficiency of a ZKP is determined by various parameters, such as communication complexity, verification complexity, prover complexity, and round complexity.

1.1 Zero-Knowledge Proofs

Zero-knowledge proofs typically refer to an *interactive proof* that satisfies three fundamental properties: completeness, soundness and zero-knowledge, and we will expand on these properties next. An interactive proof is said to satisfy *completeness*, if an honest prover always succeeds in convincing an honest verifier of a valid claim. Similarly, interactive proof is said to satisfy *soundness*, if a cheating prover always fails to convince an honest verifier of an invalid claim with high probability. Interactive arguments are proofs that provide soundness guarantees only against cheating prover strategies that run in polynomial time. Note that we use the term arguments and proofs interchangeably throughout the thesis since we only work with algorithms that run in polynomial time. In an interactive proof, the prover does not wish to send the witness to the verifier, either due to privacy concerns or due to communication constraints. To facilitate privacy, an interactive proof is said to satisfy *zero-knowledge* property, if a malicious verifier learns nothing beyond the validity of the statement. Beyond traditional guarantee of ensuring that a cheating prover fails to convince the verifier of an invalid claim, it

is sometimes imperative for the prover to convince the verifier that it 'knows' the witness for a public statement – for instance the prover may want to prove that its attributes are certified by a credential issuer by proving it knows a signature (eg. PS signature [97]) on its attributes. For such scenarios, an interactive proof is said to satisfy *knowledge-soundness* if the prover can convince the verifier that it 'knows' the secret witness, and the interactive proof is known as *proof of knowledge*. Intuitively, this property is ensured by proving the existence of a PPT algorithm that can 'extract' the witness from the prover, in the event that the prover succeeds in convincing the verifier with high probability. We call an interactive proof *public-coin* if the verifier's randomness is uniformly sampled and sent to the prover during the proof.

NIZKs. So far we have discussed interactive proofs, but there are applications of ZKPs where we require *one-shot* proofs that can be sent across to the verifier – for instance, in blockchain where the proof of valid state update needs to be 'posted' in a public forum; or electronic voting where the proof of valid ballot submission is required per vote. The class of ZKPs that offers *non-interactivity* is known as *non-interactive zero-knowledge proofs* (NIZKs). From here on, we also refer to communication complexity as the *proof size*.

Succinct Arguments. ZKPs are often deployed in applications like blockchain – to ensure that state updates are consistent with the public ledger and perform verifiable off-chain computations, and in verifiable credentials – to enable selective disclosure of personal attributes. In all such real-world applications, the prover is computationally bounded. If the *soundness* guarantee holds against a PPT prover, i.e. if we can ensure that a PPT prover cannot convince a verifier of a false claim, then such protocols are known as *arguments*. Considering practical applications of ZKPs, for instance, in blockchain applications where a prover submits cryptographic evidence of a valid state transition, or in electronic voting systems where proof of a legitimate ballot submission is required, the generated proofs being *very short* is a critical factor for efficiency. To facilitate efficient communication for such use cases, an interactive proof is said to have a *succinct proof*, if the proof size is logarithmic in the size of the witness or constant. Similarly, to facilitate efficient verification, we define interactive proofs with *succinct verification complexity* as proofs that require logarithmic/constant verifier computation.

zkSNARKs. Building upon the fundamental concepts of ZKPs and the efficiency gains offered by succinct arguments, we now discuss zkSNARKs – a class of ZKPs that inherits and amplifies such attractive efficiency benefits. zkSNARKs stands for zero-knowledge Succinct Non-interactive ARguments of Knowledge, and we expand on each property below. Here *zk* in **zk**SNARKs stands for the classical *zero-knowledge* property that ensures that the verifier would not learn anything 'extra' from the proofs. *Succinct* in zk**S**NARKs stands for succinct

proofs with succinct verification, which ensures that the proof is significantly smaller in size than the computation being verified and that the verification is much faster than re-executing the computation. *Non-interactive* in zkS**N**ARKs stands for one-shot proofs that can be sent across to the verifier in a single round, which is facilitated by a one-time setup (preprocessing) phase. This is particularly required in applications like blockchain, where the proof of correct state updates needs to be posted in the public ledger for *public verifiability* (i.e. anyone interested should be able to verify), and cannot be reliant on interaction with *every* potential verifier. Recall that arguments are proofs with computational soundness. Hence, *ARguments of Knowledge* in zkSN**ARK**s refers to the proof of knowledge protocols with knowledge-soundness property against a computationally bounded prover. On a related note, if the proof only satisfies the traditional *soundness* as opposed to the stronger *knowledge-soundness* property, it is known as a *zkSNARG* that refers to zero-knowledge Succinct Non-interactive ARGument.

1.2 Summary of the contributions of this Thesis

As part of this thesis, we primarily explore several well-studied theoretical primitives that are often used as foundational building blocks. We then examine the applications of some of these primitives in practical real-world applications. The primary objective of this thesis is to analyze these theoretical primitives through the lens of various dimensions of efficiency known in the context of ZKP constructions – specifically proof size, verification complexity, and prover complexity.

1.2.1 Succinct Verification

Kilian provided the first construction of *succinct arguments* based on probabilistically checkable proofs (PCP) in [80], where the proof size is logarithmic in the size of the statement. Micali made this non-interactive in the random oracle model (ROM) in [90] – where we assume existence of a common random function RO. Traditionally, non-interactivity in the standard model is achieved by assuming a Common Reference String (CRS) generated during a one-time setup phase. Depending on how the CRS is generated, the setup is known as trusted (where the randomness used to generate the CRS is a 'toxic waste' and is required to be discarded for the security to hold), transparent (where the CRS consists of public randomness) or updatable (where the CRS can be updated by anyone, and the security guarantees only assume the existence of at least one honest update). If the CRS is structured, it is known as a structured reference string (SRS). There has been a series of works on constructing zkSNARKs [73, 84, 25, 69, 93, 85, 21, 74], which have very short proofs with efficient verification complexity.

Sigma protocols are well-understood class of *zero-knowledge proof of knowledge* (ZKPoK)

that is also *public-coin* (where the verifier's randomness is sent to the prover). A public-coin interactive proof can be made non-interactive in the Random Oracle Model using the Fiat-Shamir transformation [59]. Earlier work of Bulletproofs [36] uses a split-and-fold technique to achieve logarithmic communication complexity, which is extended by Attema and Cramer [8] to the sigma protocol framework for the discrete log relation (which proves that a party knows x for public group elements P, g such that $P = g^x$). This split-and-fold technique involves splitting the witness in two halves and taking a random linear combination to construct a new smaller witness for a similar relation, and it integrates well with scenarios where hiding the smaller witness is not a requirement. To use this technique to obtain a ZKPoK that hides the witness, we first need to run a 'pivot' ZKPoK protocol (in our case, a sigma protocol) as a starting point, which ensures zero-knowledge. The split-and-fold technique is then used to compress the final message of the pivot sigma protocol, which was initially being sent in clear (leading to linear communication complexity). Using this technique of applying the split-and-fold compression mechanism on a pivot sigma-protocol has led to a series of works on compressed sigma protocols, such as for lattices [10], bilinear group arithmetic circuit [11], and k-out-of-n proofs [9]. When we refer to compressed sigma protocol (CSP), we would consider CSP for discrete log relation from here on, unless specified otherwise. CSP for discrete log relation extends the well-understood and deployed framework for sigma protocols to have logarithmic proof size in the transparent setup.

Our Contributions. In Chapter 3, we lift Attema and Cramer's CSP [8] to have logarithmic verification in addition to logarithmic proof size, by increasing the degree of trust in the setup phase and using structured reference string (SRS). This is done by relying on an updatable setup as opposed to the earlier transparent setup. Note that translating to this setup is not prohibitive, since this setup only relies on existence of at least one honest update to the SRS during the setup phase.

On this front, we first construct an inner product argument with logarithmic proof size and logarithmic verification complexity by relying on an updatable SRS and discrete log assumption in bilinear groups. Specifically, this lets the prover convince the verifier that $\langle \boldsymbol{a}, \boldsymbol{b} \rangle = y$ for committed vectors $\boldsymbol{a}, \boldsymbol{b}$ and public value y.

Using our above construction, we provide a ZKPoK for arithmetic circuit satisfiability with logarithmic proof size and logarithmic verification complexity – that has concretely better proof size, prover or verifier complexity than the state-of-the-art constructions under similar assumptions and setup. Note that our aforementioned succinct inner product argument can help us tackle addition gates in the arithmetic circuit. Since arithmetic circuits also consist of multiplication gates, we need additional primitives with succinct verification. Both of these

constructions are public-coin and can be made non-interactive in the ROM using the Fiat-Shamir transformation [59].

Next, we use our techniques discussed above to provide construction for an inner-product-like relation in groups, against a designated verifier. Specifically, this allows the prover to 'open' a group homomorphism f on a particular vector – i.e. prover can convince the verifier that $f(\boldsymbol{x}) = y$ for committed homomorphism f, committed vector $\boldsymbol{x}$ of field elements, and public group element y.

Our technique for achieving logarithmic verification in the compression framework is of independent interest.

1.2.2 Distributed Proofs

We motivate our next question with the following scenario – a party wishes to know the industry average salary for its role, for which it uses a job portal (eg. glassdoor) that computes the same in a privacy-preserving manner. This privacy preserving computation can be done using the powerful and well-explored tool of secure multiparty computation (MPC). However, if a party participating in the computation decides to participate with a monthly salary of 100 crore rupees, then the computed average would not be close to the true reflection of the industry average. This kind of data-poisoning attacks can be avoided by ensuring that only *authentic* inputs are being used inside the MPC execution.

MPC allows n parties to compute a joint function on their private inputs, while ensuring that (1) the privacy of their inputs is maintained and (2) the correctness of the computed output is guaranteed. In such a framework, if each party wishes to prove to every other party that its input is *authentic* leveraging well-known ZKP tools, the traditional ZKP would require each pair of parties to run a *proof of authentication* among themselves. This would lead to a total of $O(n^2)$ proofs – which would incur significant computation and communication overhead, while also requiring additional checks to ensure the same input – that is *authenticated* with proof – is being used inside the MPC execution, rendering such a direct application impractical. With such applications in mind, we propose usage of distributed ZKPs that considers multiple provers.

Different notions of distributed ZKPs have appeared in recent works [95, 16, 102, 91, 51] that explores unconventional usage of ZKPs. Pedersen's distributed ZKP [95] aims to ensure that a signature can be verified when the signature is publicly available and the secret key is held distributively by multiple provers. Another extensive line of work [102, 91, 51] focuses on distributed proof generation, where multiple provers interact with each other to jointly generate a non-interactive publicly verifiable proof, when the witness is secret-shared across the provers.

Our Contributions. In Chapter 4, we put forward a notion of distributed proof of knowledge (DPoK). In DPoK, we separate the notion of classical prover that holds the complete witness, and call these parties *workers* – which holds the secret-shares of the witness and interacts with the verifier on behalf of the prover to convince the verifier of the claim. In particular, prover holds the witness $\boldsymbol{w}$ and secret-shares the witness by sending the i^{th} share to the i^{th} worker W_i, and all the workers then interact with the verifier over broadcast channel to convince the verifier of the claim. Our setting ensures that workers require no private communication and the verifier is public-coin and only needs to share its random challenges over broadcast, making the proof publicly verifiable.

Our DPoK ensures that if the prover and the workers are honest, the proof succeeds and if any of them is corrupt, the proof fails. This is analogous to the traditional *completeness* property in classical ZKPs. Additionally, we have *zero-knowledge*, ensuring that the verifier learns nothing beyond the validity of the claim, Finally, we also have *knowledge-soundness* where we establish that the prover *knows* a valid witness if the proof succeeds with high probability. We also provide a variant of DPoK that has *robust completeness*, that can ensure an honest prover succeeds even if some of the workers participate with dishonest shares. We call such variants *Robust Complete DPoKs* or *Robust DPoKs*. Next we provide constructions of these DPoKs for discrete log relation, using which we also provide DPoKs for algebraic signatures schemes, namely BBS+ [29, 41] and PS [97]. These DPoKs have logarithmic round complexity, and hence we also provide constructions of round-efficient versions of these DPoKs (with constant rounds) that are secure in the Random Oracle Model.

Application of DPoKs in input authentication in MPC. Using our DPoKs for BBS+ (or PS) signature schemes, we provide a compiler that transforms any honest majority secret-sharing based MPC to an honest majority secret-sharing based MPC that supports input authentication. The output MPC retains the security guarantees of the input MPC, eg. identifiable abort (id-abort) or guaranteed output delivery (if default input is allowed). The input authentication phase of our compiler can be instantiated with (i) our regular DPoK whose n-parallel executions can be efficient batched into a single instance – providing enhanced efficiency and security with abort; or (ii) our robust complete DPoK providing enhanced security against dishonest usage of share, but against lower corruption threshold. Our robust complete DPoK preserves id-abort guarantees of the underlying MPC, which was not achieved by any of the prior works irrespective of the corruption threshold. Our compiler avoids protocol-specific techniques, unlike prior works for input authentication in MPC [27, 5], and provides a generic compiler with improved computational overhead.

The prior works of [27, 5] focus on generating authenticated shares and therefore would

require additional checks to ensure the same authenticated shares are used in a later MPC execution. On the contrary, our work focuses on *authenticating* existing shares held by the parties (workers).

1.2.3 Lookup Arguments

To motivate the next part of the thesis, let us consider a simple real-world scenario. A cryptocurrency blockchain maintains a public digest (commitment) of the account ledger, and one of the customers wishes to provide a *proof of coin ownership* with respect to the public digest to avail an external service, by proving that it holds two accounts – the i^{th} and j^{th} accounts in the ledger.

Lookup arguments allow us to prove that a large vector $\mathbf{T} = (T_1, \ldots, T_N)$ contains the small vector $\mathbf{S} = (S_1, \ldots, S_m)$ in the indices specified by the index vector $\boldsymbol{a} = (a_1, \ldots, a_m) \subset [N]$, i.e. $S_i = T_{a_i}$ for all $i \in [m]$ where $m << N$. This version considers the indexed variant of sub-vector relation, and is a stronger statement than just proving $\mathbf{S} \subset \mathbf{T}$. It is important to note in this context that we often consider these vectors as tables, and use the terms interchangeably. The series of works in lookup arguments [98, 111, 56, 43, 110] already achieves constant proof size and constant verification complexity and the primary goal of these works is to lessen the prover complexity. These works are in the preprocessing paradigm, where the prover precomputes the potentially expensive terms in an offline phase, to ensure that less prover computation is required in the online phase. In particular, state-of-the-art lookup arguments require $O(N \log N)$ preprocessing (dependent on the large table) to ensure the prover effort in the online phase is independent of N.

Now looking at our earlier example from the perspective of real-world application, it is evident that the cryptocurrency's public commitment shall periodically change due to updates in the underlying account ledger from currency transfer transactions. However, all recent works in the field of lookup arguments consider a static table – which implies that we can do multiple lookups on a static table $\mathbf{T}$ efficiently, but if even one of the T_i is changed then the precomputed parameters are no longer useful and cannot be used in the online phase. All preprocessed parameters need to be recomputed with $O(N \log N)$ effort in the event of *any* updates to the large table $\mathbf{T}$.

Our Contributions. In Chapter 5, we provide *updatable lookup arguments* which supports efficient lookup arguments for tables that have undergone some update after the precomputation of preprocessed parameters is done. In essence, we remove the strict dependence of the lookup arguments on the table-specific preprocessing, and enable the prover to provide efficient proof for $\mathbf{S} \subset \mathbf{T}'$ when $\mathbf{T}'$ is within certain Hamming distance of the preprocessed table $\mathbf{T}$.

We also provide construction for *committed index lookup arguments* that – given the commitment to three vectors $\mathbf{T}, \mathbf{S}, \boldsymbol{a}$ – allows us to prove that $\mathbf{S}$ is equivalent to looking up $\mathbf{T}$ at indices specified by $\boldsymbol{a}$, i.e. $S_i = T_{a_i}$ for all i. We also provide a construction to lift any proof of sub-vector relation $\mathbf{S} \subset \mathbf{T}$ using homomorphic commitment scheme to support committed index lookup argument.

Next, using our *updatable lookup argument* as a building block, along with additional primitives, we construct a batching-efficient RAM (Random Access Memory), which can efficiently prove that a RAM of size N has undergone m updates when $m << N$. Here, RAM is modeled as a table, and RAM updates refer to READ/WRITE operations. Recent works of [92, 42] based on RSA accumulators also present efficient constructions for batching-efficient RAM with constant proof size and constant verification complexity; however, their prover complexity is linear in N. We use our updatable lookup argument to ensure our batching-efficient RAM incurs constant proof size and constant verification complexity, while having improved prover complexity that is sublinear in the size of the RAM. This has application in achieving verifiable outsourcing of state updates in blockchain, in particular for blockchain rollups that aims to offload expensive computation off the blockchain while ensuring correctness of the expensive off-chain computation.

In our above constructions of lookup arguments and batching-efficient RAM, we primarily focus on succinctness and do not necessarily require privacy, and we believe it to be achievable through slight modification to our constructions.

1.3 Roadmap of the Thesis

First, we present some preliminary tools required for this thesis in Chapter 2. Additional foundational concepts are integrated into later chapters as they become relevant to the discussion. In Chapter 3, we present constructions of zero-knowledge proofs of knowledge with succinct proof size and succinct verification complexity. Then, in Chapter 4, we discuss distributed proofs of knowledge and present constructions of the same for discrete log relation and some algebraic signature schemes, and thereafter we see how to use it to ensure input authentication in secure Multiparty Computation (MPC). Finally, in Chapter 5, we discuss how to perform efficient lookups on tables undergoing continuous updates and its application in providing efficient proofs of correct update on large RAMs (where the number of updates are small). The results in this thesis are based on the works in [53], [54], and [55], and some passages are presented verbatim.

Chapter 2

Preliminaries

In this section, we present some of the required background for Zero-Knowledge Proofs. Further concepts are deferred to the 'preliminaries' section of later chapters as per their relevance.

Notations. Let $\mathbb{N}$ be the set of all natural numbers. Let $(\mathbb{G}, \circ)$ denote a group $\mathbb{G}$ corresponding to the binary operation $\circ$. We drop the operation $\circ$ from the group description when it is evident from the context. Let $(\mathbb{F}, +, \cdot)$ denote a field with respect to the two binary operations $+$ (addition) and $\cdot$ (multiplication). For $x \in X$, let $x \leftarrow_R X$ denote uniformly random sampling of an element x from the set X. Let $[n]$ denote the set $\{1, \ldots, n\}$, such that $[n] \subset \mathbb{N}$.

Let $\mathbb{F}_q$ denote a finite field of order q, also denoted by $\mathbb{F}$ when the order is not specified or is clear from context. Let $\mathbb{G}$ be a group of prime order q. We refer to $\lambda \in \mathbb{N}$ as the security parameter, and denote by $\mathsf{poly}(\lambda)$ and $\mathsf{negl}(\lambda)$ any generic (unspecified) polynomial function and negligible function in λ, respectively. A function $f : \mathbb{N} \to \mathbb{N}$ is said to be negligible in λ if for every positive polynomial p, $f(\lambda) < 1/p(\lambda)$ when λ is sufficiently large. The output x of a deterministic algorithm $\mathcal{A}$ is denoted by $x = \mathcal{A}$ and the output x' of a randomized algorithm $\mathcal{A}'$ is denoted by $x' \leftarrow_R \mathcal{A}'$.

We denote vectors by boldface letters. For vectors $\boldsymbol{g} = (g_1, \ldots, g_n) \in \mathbb{G}^n$ and $\boldsymbol{x} = (x_1, \ldots, x_n) \in \mathbb{F}_q^n$, where $\mathbb{G}$ is a group of prime order q, the multi-exponentiation $\boldsymbol{g}^{\boldsymbol{x}}$ is defined by $\boldsymbol{g}^{\boldsymbol{x}} = g_1^{x_1} \cdots g_n^{x_n}$. Also, for $g \in \mathbb{G}$ and $\boldsymbol{x} = (x_1, \ldots, x_n) \in \mathbb{F}_q^n$, $g^{\boldsymbol{x}}$ is defined by $g^{\boldsymbol{x}} = (g^{x_1}, \cdots, g^{x_n})$.

Bilinear Group. We assume a bilinear group generator BG on input λ outputs parameters for the protocols, i.e. $\mathsf{BG}(1^\lambda)$ outputs $(q, \mathbb{F}, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2)$. Here, $\mathbb{F} = \mathbb{F}_q$ is a prime field of order q, $\mathbb{G}_1, \mathbb{G}_2$ and $\mathbb{G}_T$ are groups of order q, and e is an efficiently computable non-degenerate bilinear pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$. g_1, g_2 are uniformly chosen generators of $\mathbb{G}_1$ and $\mathbb{G}_2$ respectively, and $e(g_1, g_2)$ is the generator for $\mathbb{G}_T$.

2.1 Interactive Proofs

Let $\mathcal{R} = \{(x, w)\}$ be a relation and $\mathcal{L}$ be the corresponding NP language. We consider interactive proofs for relations, where a prover P convinces the verifier that it knows a witness w such that for a public statement $x, (x, w) \in \mathcal{R}$. For a pair of PPT interactive algorithms P, V, $\langle P(w), V\rangle(x)$ denotes the output of V on its interaction with P, where w is P's private input and x is a common input.

Proof of Knowledge. An interactive protocol is known as a proof of knowledge (PoK) for a relation $\mathcal{R}$ if it consists of a PPT algorithm $\mathsf{Setup}(1^\lambda)$ that takes a security parameter λ and outputs public parameters srs, and a pair of PPT interactive algorithms $\langle \mathcal{P}, \mathcal{V}\rangle$. The triple $(\mathsf{Setup}, \mathcal{P}, \mathcal{V})$ satisfy the following properties of completeness and knowledge-soundness.

Definition 1 (Completeness) *For all $\lambda \in \mathbb{N}$, $(x, w) \in \mathcal{R}$,*

$$\Pr\left(\langle \mathcal{P}(w), \mathcal{V}\rangle(\mathsf{srs}, x) = 1 \,:\, \mathsf{srs} \leftarrow \mathsf{Setup}(1^\lambda)\right) = 1.$$

Definition 2 (Knowledge Soundness) *An interactive protocol $(\mathcal{P}, \mathcal{V})$ for a relation $\mathcal{R}$ is knowledge sound with error κ if there exists an expected polynomial time extractor Ext such that for every efficient adversary $\tilde{\mathcal{P}}$, for every $x \in \{0,1\}^*$, whenever $\tilde{\mathcal{P}}$ makes $\mathcal{V}$ accept with probability $\epsilon > \kappa$, $\mathsf{Ext}^{\tilde{\mathcal{P}}}(x)$ outputs w^* such that $(x, w^*) \in \mathcal{R}$ with probability at least $\frac{\epsilon-\kappa}{q}$ for some polynomial q.*

Zero-Knowledge Proof of Knowledge. An interactive protocol is known as a Zero-Knowledge Proof of Knowledge (ZKPoK) if it is a Proof of Knowledge which satisfies the following zero-knowledge property.

Definition 3 (Zero-Knowledge) *An interactive protocol is said to satisfy zero-knowledge property if for every polynomial time verifier $\tilde{\mathcal{V}}$, there exists an efficient simulator algorithm $\mathsf{Sim}(s)$ such that for every $(x, w) \in \mathcal{R}$, the distribution $\mathsf{Sim}(x)$ is identical to $\mathsf{View}_{\langle \mathcal{P}(x,w), \tilde{\mathcal{V}}(x)\rangle}$, where $\mathsf{View}_{\langle \mathcal{P}(x,w), \tilde{\mathcal{V}}(x)\rangle}$ is the distribution of the view of the verifier in the protocol on common input x and prover's witness w.*

Honest Verifier Zero-Knowledge. An interactive protocol is said to be Honest Verifier Zero-Knowledge (HVZK) if there exists an efficient simulator Sim such that for every $(x, w) \in \mathcal{R}$, the distribution $\mathsf{Sim}(x)$ is identical to $\mathsf{View}_{\langle \mathcal{P}(x,w), \mathcal{V}(x)\rangle}$, where $\mathsf{View}_{\langle \mathcal{P}(x,w), \mathcal{V}(x)\rangle}$ is the distribution of the view of the verifier in the protocol on common input x and prover's witness w. Contrary to the earlier zero-knowledge property, this version assumes that the verifier follows the prescribed algorithm.

SNARKs. A proof of knowledge protocol is also known as an argument of knowledge (AoK) if the knowledge-soundness property holds against a PPT prover, i.e. if it has computational knowledge-soundness. An interactive protocol is said to have a succinct proof, if the communication complexity between the prover and the verifier is bounded by $\mathsf{poly}(\lambda)$, and is said to have a succinct verification complexity if the running time of the verifier is bounded by $\mathsf{poly}(\lambda + |x|)$. A succinct argument of knowledge which is also non-interactive is known as a SNARK, which is an acronym for Succinct Non-Interactive ARgument of Knowledge.

Updatable SRS model. In common reference string (CRS) model, we assume the presence of a trusted setup phase, where a random string is generated and made available to both the prover and the verifier. If the CRS is structured, then it is known as a structured reference string (SRS), and the model is known as an SRS model. An SRS is known as an updatable universal SRS if it enables parties to update the parameters, while retaining computational soundness against any probabilistic-polynomial time adversary, as long as at least one honest update is performed, and such a model is known as an updatable SRS model. [28] introduced non-interactive zero-knowledge proofs (NIZKs) in the CRS model, and later [75] introduced updatable SRS with its application to SNARKs.

Fiat-Shamir and Random Oracle Model. An interactive protocol is *public-coin* if the verifier's messages are uniformly random strings. Public-coin protocols can be transformed into non-interactive arguments in the Random Oracle Model (ROM) by using the Fiat-Shamir [59] heuristic to derive the verifier's messages as the output of a Random Oracle.

Note that all of our protocols are public-coin, hence we are only required to prove that our protocols satisfy knowledge-soundness (or special-soundness, which also implies knowledge-soundness) for our interactive protocols, and thereafter rely on the Fiat-Shamir transform to obtain the non-interactive version of our protocols.

Algebraic Group Model. An adversary $\mathcal{A}$ is called *algebraic* if every group element output by $\mathcal{A}$ is accompanied by a representation of that group element in terms of all the group elements that $\mathcal{A}$ has seen so far (input and output). Introduced in [61], the Algebraic Group Model (AGM) restricts the adversary $\mathcal{A}$ to be *algebraic*.

2.2 Sigma Protocols

Sigma protocols are three-round public-coin zero-knowledge proof of knowledge protocols. The structure of a sigma protocol for $(x, w) \in \mathcal{R}$ for a relation $\mathcal{R}$ is as follows, where x is the common input and w is the private witness:

– In the first round, the prover sends a message a.

– In the second round, the verifier responds with a randomly sampled challenge c.

– In the third round, the prover computes and sends a response z based on the witness w, its first message a, and the random challenge c.

– The verifier then performs local checks based on the public x, prover's first message a, its challenge c, and the prover's response z, and outputs if it accepts/rejects.

Here, (a, c, z) is known as the *transcript*, and it is known as an *accepting transcript* if the verifier accepts at the end of the protocol.

An interactive protocol is known as Sigma protocol if it is a three-round public-coin protocol that satisfies the following properties of completeness, special-soundness and special honest verifier zero-knowledge.

Definition 4 (Completeness) *If $(x, w) \in \mathcal{R}$, and $\mathcal{P}$ and $\mathcal{V}$ follow the protocol specification, then* $\Pr\left(\langle \mathcal{P}(w), \mathcal{V} \rangle(x) = 1\right) = 1$.

Definition 5 (Special Soundness) *There exists a PPT extractor* Ext *that takes as input two accepting transcripts (a, c, z) and (a, c', z') for a given x, such that $c \neq c'$ and the first message a is identical, and outputs a witness w^* such that $(x, w^*) \in \mathcal{R}$.*

Definition 6 (Special Honest Verifier Zero-Knowledge) *There exists a PPT simulator* Sim*, such that for every $(x, w) \in \mathcal{R}$ and any given c, the distribution* $\mathsf{Sim}(x, c)$ *is identical to* $\mathsf{View}_{\langle \mathcal{P}(x,w), \mathcal{V}(x,r) \rangle}$*, where* $\mathsf{Sim}(x, c)$ *is the output of the simulator with input x and c, and* $\mathsf{View}_{\langle \mathcal{P}(x,w), \mathcal{V}(x,r) \rangle}$ *is the distribution of the view of the verifier in the protocol on common input x and prover's witness w.*

Now, we briefly recall the sigma protocol for proving knowledge of discrete logarithm $\boldsymbol{x} \in \mathbb{F}_p^n$ of a vector of group elements $\boldsymbol{g} \in \mathbb{G}^n$, such that $\boldsymbol{g}^{\boldsymbol{x}} = z$. Here, a prover $\mathcal{P}$ with knowledge of the secret vector $\boldsymbol{x}$, samples a random vector of scalars $\boldsymbol{r} \leftarrow_R \mathbb{F}_p^n$, and sends $\alpha = \boldsymbol{g}^{\boldsymbol{r}}$ to the verifier $\mathcal{V}$. $\mathcal{V}$ then samples a challenge $c \leftarrow_R \mathbb{F}_p$ and sends it to $\mathcal{P}$ and in the next round $\mathcal{P}$ replies with $\boldsymbol{x} = c\boldsymbol{x} + \boldsymbol{r}$. $\mathcal{V}$ considers the transcript (α, c, z), and checks if $\boldsymbol{g}^{\boldsymbol{x}} = z^c \alpha$. Note that the last message of the prover is linear in the size of the witness, and hence the traditional sigma protocol has linear communication complexity. [8] presents compressed sigma protocol with logarithmic communication complexity, albeit with logarithmic round complexity, which can be compressed to obtain NIZK with logarithmic proof size by relying on the Fiat-Shamir heuristic [59] in the Random Oracle Model.

2.3 Polynomial Commitment Scheme

The notion of a polynomial commitment scheme (PCS) that allows the prover to open evaluations of the committed polynomial succinctly was introduced in [77] who gave a construction under the trusted setup assumption. A polynomial commitment scheme over $\mathbb{F}$ is a tuple $\mathsf{PC} = (\mathsf{Setup}, \mathsf{Commit}, \mathsf{open}, \mathsf{eval})$ where:

- $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, D)$. On input security parameter λ, and an upper bound $D \in \mathbb{N}$ on the degree, Setup generates public parameters pp.
- $(C, \tilde{\mathbf{c}}) \leftarrow \mathsf{Commit}(\mathsf{pp}, f(X), d)$. On input the public parameters pp, and a univariate polynomial $f(X) \in \mathbb{F}[X]$ with degree at most $d \leq D$, Commit outputs a commitment to the polynomial C, and additionally an opening hint $\tilde{\mathbf{c}}$.
- $b \leftarrow \mathsf{open}(\mathsf{pp}, f(X), d, C, \tilde{\mathbf{c}})$. On input the public parameters pp, the commitment C and the opening hint $\tilde{\mathbf{c}}$, a polynomial $f(X)$ of degree $d \leq D$, open outputs a bit indicating accept or reject.
- $b \leftarrow \mathsf{eval}(\mathsf{pp}, C, d, x, v; f(X))$. A public coin interactive protocol $\langle P_{\mathsf{eval}}(f(X)), V_{\mathsf{eval}} \rangle(\mathsf{pp}, C, d, z, v)$ between a PPT prover and a PPT verifier. The parties have as common input public parameters pp, commitment C, degree d, evaluation point x, and claimed evaluation v. The prover has, in addition, the opening $f(X)$ of C, with $\deg(f) \leq d$. At the end of the protocol, the verifier outputs 1 indicating accepting the proof that $f(x) = v$, or outputs 0 indicating rejecting the proof.

A polynomial commitment scheme must satisfy the following properties of completeness, binding and extractability.

Definition 2.1 (Completeness) *For all polynomials $f(X) \in \mathbb{F}[X]$ of degree $d \leq D$, for all $x \in \mathbb{F}$,*

$$\Pr\left[b = 1 \ : \ \begin{array}{c} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, D) \\ (C, \tilde{\mathbf{c}}) \leftarrow \mathsf{Commit}(\mathsf{pp}, f(X), d) \\ v \leftarrow f(x) \\ b \leftarrow \mathsf{eval}(\mathsf{pp}, C, d, x, v; f(X)) \end{array} \right] = 1.$$

Definition 2.2 (Binding) *A polynomial commitment scheme* PC *is binding if for all PPT* $\mathcal{A}$*, the following probability is negligible in* λ*:*

$$\Pr\left[\begin{array}{c}\mathsf{open}(\mathsf{pp}, f_0, d, C, \tilde{\mathbf{c}}_\mathbf{0}) = 1 \wedge \\ \mathsf{open}(\mathsf{pp}, f_1, d, C, \tilde{\mathbf{c}}_\mathbf{1}) = 1 \wedge \\ f_0 \neq f_1\end{array} : \begin{array}{c}\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, D) \\ (C, f_0, f_1, \tilde{\mathbf{c}}_\mathbf{0}, \tilde{\mathbf{c}}_\mathbf{1}, d) \leftarrow \mathcal{A}(\mathsf{pp})\end{array}\right].$$

Definition 2.3 (Knowledge Soundness) *For any PPT adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$*, there exists a PPT algorithm* Ext *such that the following probability is negligible in* λ*:*

$$\Pr\left[\begin{array}{c} b = 1 \wedge \\ \mathcal{R}_{\mathsf{eval}}(\mathsf{pp}, C, x, v; \tilde{f}, \tilde{\mathbf{c}}) = 0\end{array} : \begin{array}{c}\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, D) \\ (C, d, x, v, \mathsf{st}) \leftarrow \mathcal{A}_1(\mathsf{pp}) \\ (\tilde{f}, \tilde{\mathbf{c}}) \leftarrow \mathsf{Ext}^{\mathcal{A}_2}(\mathsf{pp}) \\ b \leftarrow \langle \mathcal{A}_2(\mathsf{st}), V_{\mathsf{eval}} \rangle(\mathsf{pp}, C, d, x, v)\end{array}\right].$$

where the relation $\mathcal{R}_{\mathsf{eval}}$ *is defined as follows:*

$$\mathcal{R}_{\mathsf{eval}} = \{((\mathsf{pp}, C \in \mathbb{G},\ x \in \mathbb{F},\ v \in \mathbb{F});\ (f(X), \tilde{\mathbf{c}})) : \\ (\mathsf{open}(\mathsf{pp}, f, d, C, \tilde{\mathbf{c}}) = 1) \wedge v = f(x)\}$$

We denote by $\mathsf{Prove}, \mathsf{Verify}$, the non-interactive prover and verifier algorithms obtained by applying FS to the eval public-coin interactive protocol, giving a non-interactive PCS scheme $(\mathsf{Setup}, \mathsf{Commit}, \mathsf{Prove}, \mathsf{Verify})$.

Definition 2.4 (Succinctness) *We require the commitments and the evaluation proofs to be of size independent of the degree of the polynomial, that is the scheme is* proof succinct *if* $|C|$ *is* $\mathsf{poly}(\lambda)$*,* $|\pi|$ *is* $\mathsf{poly}(\lambda)$ *where* π *is the transcript obtained by applying FS to* eval*. Additionally, the scheme is* verifier succinct *if* eval *runs in time* $\mathsf{poly}(\lambda) \cdot \mathsf{log}(d)$ *for the verifier.*

Chapter 3

Succinct Verification of Compressed Sigma Protocols using Updatable SRS

In this chapter[1], we explore a class of zero-knowledge proofs of knowledge that has succinct communication complexity to additionally support succinct verification complexity. In particular, we lift compressed sigma protocols, introduced by Attema and Cramer in [8], to achieve logarithmic verification complexity while retaining their logarithmic proof size, by using a structured reference string (SRS).

3.1 Introduction

NIZKs can be facilitated by having oracle access to a common function which in an idealised setting is modeled as a random function (known as Random Oracle or RO), which makes the protocol secure in the idealised setting known as the Random Oracle Model (ROM). The setup phase generally consists of having a Common Reference String (CRS) shared across the prover and the verifier. The setup is known as *transparent* if the CRS consists of a truly random string. Note that a structured CRS is often referred to as Structured Reference String (SRS). The construction with better concrete efficiency with respect to communication complexity are based on *trusted* SRS, where the randomness used to generate the SRS is akin to toxic waste (also known as the *trapdoor*) which is required to be discarded upon the generation of SRS, and the security guarantees hold only if the randomness is not known.

A line of work aims to reduce the degree of trust required in the generation of the CRS, by having an *updatable* SRS, giving rise to SNARKs in an updatable setup model [75, 88, 63, 45]. The *updatable* SRS can be updated by any of the involved parties, ensuring that the security

[1]This chapter is based on the joint work [53] with Chaya Ganesh and Neha Jawalkar, that appeared in PKC 2024.

guarantees hold as long as there has been at least one honest update to the SRS. Intuitively, an update to the SRS is done by having the party contribute additional randomness, and the trapdoor for a given 'updatable' SRS is hidden due to re-randomization of the trapdoor from each contribution of randomness.

Bulletproofs [36], building on the work of [32] introduced techniques that achieve logarithmic communication complexity in discrete logarithm (DL) based zero-knowledge proofs. Thereafter, Attema and Cramer [8] introduced *compressed sigma protocol* theory by using a blackbox compression technique, which also places Bulletproofs in the framework of Sigma protocol theory. These protocols relies on the compression mechanism of the split-and-fold technique, which uses a 'pivot' ZKPoK protocol as a starting point to ensure zero-knowledge, and thereafter compresses the linear-sized messages of the pivot sigma protocol. The split-and-fold technique involves splitting the message in half, and then folding it using a randomly sampled challenge, to attain a new (smaller) witness satisfying a similar relation. The idea of employing a compression mechanism on a pivot protocol has become a versatile tool, leading to compressed sigma protocol theory for lattices [10], and compressed sigma protocols for bilinear group arithmetic circuits [11].

Compressed Sigma Protocols (CSP) are attractive in applications due to their reliance on weaker assumptions (DL), conceptual simplicity, logarithmic proof size and transparent setup. One downside of this class of protocols is that they are only proof-succinct but not verifier-succinct – the verification is linear. In this chapter, we focus on studying succinct verification of compressed sigma protocols while retaining the succinct proof size.

3.1.1 Our Contributions

In this chapter, we present compressed sigma protocols that are both proof and verifier-succinct in the updatable SRS model. CSP compresses a pivot Σ-protocol for proving knowledge of a long vector $\boldsymbol{x}$ while revealing a public linear form $L(\boldsymbol{x})$, given a Pedersen commitment, resulting in a protocol for opening linear forms on committed vectors with proof size $O(\log n)$ and verification complexity $O(n)$ where n is the size of $\boldsymbol{x}$.

Protocol for opening a committed linear form. A core building block of our succinct verifier constructions is a protocol to open a committed linear form on a committed vector. That is, we can prove knowledge of a long vector $\boldsymbol{x} \in \mathbb{F}_q^n$ given a commitment to a linear form L and a public element $y \in \mathbb{F}_q$, such that $L(\boldsymbol{x}) = y$. Here, L can be equivalently represented by a vector of group elements $(a_1, \ldots, a_n)$ such that $L(\boldsymbol{x}) = a_1x_1 + \cdots + a_nx_n$, where $\boldsymbol{x} = (x_1, \ldots, x_n)$ is a vector of field elements. In essence, since both L and $\boldsymbol{x}$ are vectors of field elements, we are proving that an inner product of two elements has been computed correctly. It is an inner

product argument, but in the spirit of CSP, one of the vectors – the linear form, is public and not a witness. This vector is committed to only for the sake of succinctness, and looking ahead, this commitment encodes the structure of the circuit and is computed during preprocessing. We use a commitment scheme with a structured key introduced by [52]. This protocol with logarithmic proof size and logarithmic verification complexity is secure under the DL assumption (same as CSP), albeit in a bilinear group and at the cost of moving to an updatable SRS setting. We compare our inner product protocol with [52] and [83] in Table 3.1.

Succinct verifier protocol for circuit satisfiability. We construct a succinct argument of knowledge for circuit satisfiability in the universal updatable SRS model. The proof size and verifier are logarithmic in the size of the circuit. This is secure under DL and can be made non-interactive in the ROM using the Fiat-Shamir transform [59] since our protocols are public-coin. We compare the concrete costs of our protocol, with that of [52] in Table 3.2[1]. Since we use the same structure of SRS, the complexity of updating the SRS and verifying the updates remain the same as in [52]. Dory's [83] polynomial commitment can be used to obtain a protocol for circuit-satisfiability. The exact costs will depend on the underlying information-theoretic object (Polynomial interactive oracle proof) that is compiled using Dory. For univariate polynomials of degree n, and opening one evaluation, Dory's costs are: proof size of $(4\log n + 10)\mathbb{G} + (\log n + 8)\mathbb{F}_q$, prover's computation of $(n + \log n)E + n^{1/2}P$, verifier's computation of $4\log nE + O(1)P$. Note that Dory's prover requires pairing operations and the security relies on a decisional assumption.

Protocol	Setup	Assumption	Proof size	$\mathcal{P}$ complexity	$\mathcal{V}$ complexity
[52]	Updatable	DL	$8\log n$ $\mathbb{G}$ $2\log n$ $\mathbb{F}_q$	$(8n-4)$ E^*	$2\log n$ P $4\log n$ E
[83]	Transparent	SXDH	$12\log n$ $\mathbb{G}$ $\log n$ $\mathbb{F}_q$	$O(n^{1/2})$ P	$9\log n$ E 1 P
This work ($\Pi_{1\text{-}\mathcal{R}}$)	Updatable	DL	$4\log n$ $\mathbb{G}$ $3\log n$ $\mathbb{F}_q$	$(8n-4)$ E^* $2\log n$ E	$2\log n$ P $2\log n$ E
This work ($\Pi_{2\text{-}\mathcal{R}}$)	Updatable	DL	$2\log n$ $\mathbb{G}$ $\log n$ $\mathbb{F}_q$	$(4n-2)$ E^* $\log n$ E	$\log n$ P $\log n$ E

Table 3.1: Comparison of our Linear Form opening protocols (or equivalently inner product arguments) for vectors of length n. We compare in terms of most expensive operations, i.e. multi-exponentiations E, pairings P and exponentiations E^* (to provide aggregate values for non-constant multi-exponentiations).

[1]We note that other SNARKs in the universal, updatable setting that have better communication and/verifier complexity ($O_\lambda(1)$) rely on the Algebraic Group Model or Knowledge Type assumptions *in addition* to the Random Oracle Model. In this work, we are interested in constructions in the Random Oracle Model, and relying on standard assumptions.

Protocol	Proof size	$\mathcal{P}$ complexity	$\mathcal{V}$ complexity
[52]	$12 \log n\ \mathbb{G}$ $4 \log n\ \mathbb{F}_q$	$(22+10M)n\ E^*$	$12 \log n\ E^*$ $8 \log n\ P$
[83]	$(27/2) \log n\ \mathbb{G}$ $3 \log n\ \mathbb{F}_q$	$O(n^{1/2})\ P$	$((27/2) \log n$ $O(1))\ E$ $O(1)\ P$
This work (Π_{csat})	$2 \log n\ \mathbb{G}$ $\log n\ \mathbb{F}_q$	$(13+5M)n\ E^*$	$\log n\ E$ $\log n\ P$

Table 3.2: Circuit SAT protocol for a preprocessed circuit of size n (which is roughly $3m$ for m multiplication gates). Both protocols are updatable zkSNARKs that rely on the DL assumption. Similar to [52], we only compare in terms of the most expensive operations, exponentiations E and pairings P, and omit constant terms. M is a parameter that determines the processed circuit's fan-in and fan-out upper bound, and can be fine-tuned to balance the prover/verifier computations.

Protocol for opening a committed homomorphism. We then construct a protocol for opening a homomorphism, where both the vector and the homomorphism are committed to. That is, we can prove knowledge of a long vector $\boldsymbol{x} \in \mathbb{F}_q^n$ given a commitment to a homomorphism f and a public group element y, such that $f(\boldsymbol{x}) = y$. Throughout this thesis, we consider group homomorphisms of the form $f : \mathbb{F}_q^n \to \mathbb{G}$, and f can be equivalently represented by a vector of group elements $(g_1, \ldots, g_n)$ such that $f(\boldsymbol{x}) = g_1^{x_1} \cdots g_n^{x_n}$, where $\boldsymbol{x} = (x_1, \ldots, x_n)$ is a vector of field elements. We extend commitment schemes to group elements from [81, 11] to one that uses a structured key and show binding based on SXDH. Succinct verifier protocols for opening homomorphisms are useful in constructing proofs of partial knowledge with succinct verifier. We then extend our protocol to general homomorphisms (on commitments to $\mathbb{F}_q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ elements simultaneously) motivated by applications to bilinear group arithmetic circuit zero-knowledge protocols. Our constructions for opening homomorphisms are in the *designated-verifier* setting. In applications like verification of structure preserving signatures and attribute-based authentication, public verifiability might not be necessary since there is a designated credential verifier, and indeed the homomorphism itself is given by the statement to prove (signature verification algorithm) that can be committed to in a preprocessing phase. Therefore, our protocols can be used in similar applications as in [81], like proving knowledge of signature for complex access structures. While [81] has proof that scales logarithmically with the size of the statement, our protocol additionally yields logarithmic verification (albeit for designated verifier, which we believe is not a limitation for credential verification).

3.1.2 Related Work

Daza et al. [52] constructs inner product argument with logarithmic verifier by replacing the unstructured CRS or commitment key with a structured one. This also yields a protocol for circuit satisfiability with logarithmic verification in the updatable setting. Our construction and the protocols of [52] achieve the same result asymptotically. However, while we crucially rely on a structured commitment key to make the verifier logarithmic like in [52], our techniques are different. The work of [52] extend the protocols of [32, 36], while we take the approach of CSP. This has the advantage of applying *compression mechanism* on standard protocols for linear relations (or non-linear relations after linearization). The CSP approach also allows us to extend our techniques to other applications where compression applies in a black-box way. Second, our protocols are concretely better than [52] with smaller constants (See Table 3.2).

Dory [83] presents a transparent protocol for inner products between committed vectors with logarithmic proof size and logarithmic verification. Dory relies on a decisional assumption (SXDH) whereas our inner product protocol relies on DL. Additionally, our prover work is only group operations as opposed to $(O(n^{1/2}))$ pairing operations required by the prover in Dory, and our constants in the proof size are better.

Other SNARKs in the updatable setting [88, 63, 45, 86] rely on knowledge-type assumptions or Algebraic Group Model (AGM), and constructions in the transparent setting with similar and better asymptotics [37, 6] require unknown order groups with concretely expensive operations.

Lai et al. [81] show a generalization of Bulletproof's circuit zero-knowledge protocol to work for bilinear group arithmetic circuits directly, without requiring these circuits to be translated into arithmetic circuits. Attema et al. [11] generalize compressed sigma protocols for bilinear group arithmetic circuits. Both these constructions rely on a protocol for opening a group homomorphism where the verifier is linear. Using our protocol for opening a committed homomorphism will yield a succinct verifier at the expense of making it a designated verifier system. We provide the comparison in Table 3.4. Note that for application like Threshold Signature Schemes (following Algorithm 4 of [11]), we retain the logarithmic size of the signature similar to prior works, however we improve the verification complexity from linear to logarithmic.

Performace Comparison for MiMC and Poseidon Hash. We report the timing using a third party implementation calculator `https://zka.lc`, where we estimate using BLS12-381 curve implemented in arkworks-rs provided using Amazon Linux 2 8-core Intel(R) Xeon(R) Platinum 8259CL CPU @ 2.50GHz, 32GB. In Table 3.3, we use the reported number of gates for MiMC in [3], having 1293 multiplication gates and 646 addition gates. We achieve 1.77× improvement in prover time and more than 7× improvement in verification time as compared to Daza et al.

[52]. Similarly, we achieve 1.79× improvement in prover time and more than 7× improvement in verification time compared to Daza et al. for Poseidon (assuming number of R1CS constraints is equal to the number of multiplication gates, with 276 R1CS constraints [1]).

Similarly, we obtain 1.42× improvement in prover time as compared to Dory [83] for MiMC hash instantiation and 2.69× improvement in prover time for Poseidon hash instantiation, at a slight cost of verifier time. Note that for circuits of smaller sizes, we do fairly better than Dory in prover time complexity, while not losing much in verifier time complexity. Also, for both comparisons we assume that Dory has at least $6n^{1/2}$ pairings computation by prover, and 3 pairing checks performed by the verifier. For statements that show up in practice, like proving knowledge of opening of a Merkle tree leaf using MiMC/Poseidon Hash functions, the reported number for the prover increases by a factor of depth d of the tree.

Hash	Protocol	Prover Time	Verifier Time
MiMC	[52]	729,272	151,831
	[83]	586,381	7,054
	Us	411,809	19,319
Poseidon	[52]	181,058	123,610
	[83]	270, 912	6, 769
	Us	100,636	15,780

Table 3.3: Performance of MiMC and Poseidon Hash Instantiation using `https://zka.lc` and in μs.

3.1.3 Technical Overview

An inner product argument allows us to prove that an inner product of two vectors has been computed correctly, that is $\langle \boldsymbol{a}, \boldsymbol{b} \rangle = z$, for two vectors $\boldsymbol{a}$ and $\boldsymbol{b}$, and a public value z. The high-level idea behind the inner product argument of [32] and the compressed sigma protocol of [8] is to compress a vector using a Pedersen commitment, and then in each round reduce the instance and the commitment key to another one of half the size by using the verifier's challenge. At a high level, the source of the verifier's linear complexity is in having to compute the new keys at every step.

We use a structured commitment key proposed in [52] that consists of encodings of multilinear monomials of a secret vector of logarithmic length. That is, for vectors of length n, we first set $\ell = \log n$ and consider the vector $\dot{\boldsymbol{a}} = (\dot{a}_1, \ldots, \dot{a}_\ell)$ to construct $\overline{\boldsymbol{a}} = \left(\prod_{i=1}^{\ell} \dot{a}_i^{b_i}\right)_{b_i \in \{0,1\}}$ and finally, the commitment key is set as $g^{\overline{\boldsymbol{a}}}$ where $g^{\boldsymbol{x}} = (g^{x_1}, \ldots, g^{x_n})$ for a vector $\boldsymbol{x} = (x_1, \ldots, x_n)$.

Protocol	Proof size	Prover Complexity	Verifier Complexity
[81]	$O(n)$ $\mathbb{G}_T$ $O(\log n)$ $\mathbb{G}_1$ $O(\log n)$ $\mathbb{G}_2$ $O(\log n)$ $\mathbb{F}_q$	$O(n)$ E	$O(n)$ E
[11]	$O(\log n)$ $\mathbb{G}_T$ $O(\log n)$ $\mathbb{F}_q$	$O(n)$ E	$O(n)$ E
Us	$O(\log n)$ $\mathbb{G}_T$ $O(\log n)$ $\mathbb{G}_1$ $O(\log n)$ $\mathbb{G}_2$ $O(\log n)$ $\mathbb{F}_q$	$O(n)$ E	$O(\log n)$ E $O(\log n)$ P

Table 3.4: Comparison of protocols for opening homomorphism for vectors of length n. We compare in terms of most expensive operations, i.e. pairings P and exponentiations E and dominant communication cost with respect to elements of the field $\mathbb{F}_q$ and groups $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$. Note that our verifier complexity is $2 \log n$ E + $\log n$ P.

The commitment to a vector $\boldsymbol{x}$ under key $g^{\overline{\boldsymbol{a}}}$ is $g^{\langle \overline{\boldsymbol{a}}, \boldsymbol{x} \rangle}$. This key is updatable, a party can sample new ℓ secrets and update the encoding in a verifiable way. A compressed version of this key, $g^{\dot{\boldsymbol{a}}} \in \mathbb{G}_1^{\ell}$ allows the verifier to be logarithmic.

Linear Form Opening with Succinct Verifier. We build on the inner product arguments of [32] and [52]. The verifier's work in [32] involves computing an updated key in each round, and in [52], the verifier is only given a compressed key (of logarithmic size) and the prover convinces the verifier that the reduced statement in each round is with respect to a new key that is correctly updated. New commitment keys with size half of the original one are created by splitting them in half and then combining them based on the verifier's challenge. A logarithmic verifier can check that a structured key has been updated correctly using a pairing operation. The source of the verifier's linear complexity in the compressed sigma protocol of [8] was due to computation of the new keys at every step. On the contrary, our verifier remains logarithmic as the verifier is only required to parse a compressed key of logarithmic size, which is used for performing constant number of checks per round, and we only require logarithmic number of rounds which does not involve updating the shared key.

While our protocol uses the same structured key, we take a slightly different approach: we exploit the fact that we can go from 'a commitment to a vector with respect to the second half of the original basis' to 'a commitment to the same vector with respect to the first half of the original basis'. Now, if a prover produces commitment to both halves of a vector with respect to the first half of the basis, the verifier can perform one multiplication in the exponent to check the

consistency. Consider, $\dot{\boldsymbol{a}} = (\dot{a}_1, \ldots, \dot{a}_\ell)$ and $\overline{\boldsymbol{a}} = \left(\prod_{i=1}^{\ell} \dot{a}_i^{b_i}\right)_{b_i \in \{0,1\}}$ for $n = 2^\ell$, the commitment key is $g^{\overline{\boldsymbol{a}}} = \left(g^{\overline{\boldsymbol{a}}_L} \| g^{\dot{a}_\ell \overline{\boldsymbol{a}}_L}\right)$, the verification key is $H^{\dot{\boldsymbol{a}}}$, and $P = \mathrm{COM}_{\overline{\boldsymbol{a}}}(\boldsymbol{x})$. In each step, for $\boldsymbol{x} = (\boldsymbol{x}_L \| \boldsymbol{x}_R)$, if the prover produces $A_1 = \mathrm{COM}_{\overline{\boldsymbol{a}}_L}(\boldsymbol{x}_L), A_2 = \mathrm{COM}_{\overline{\boldsymbol{a}}_L}(\boldsymbol{x}_R)$, the verifier can perform the check $e(\frac{P}{A_1}, H) = e(A_2, H^{\dot{a}_\ell})$ to ensure consistency. Thus, the commitment key updates are done implicitly by simply dropping off the last element $\dot{a}_\ell$, and using the challenge only to fold the instance vectors $\boldsymbol{x}' = \boldsymbol{x}_L + c\boldsymbol{x}_R$. This observation allows us to shave off about 4 group elements in each round from the Daza et al.'s inner product argument. Our protocol also has the advantage of allowing efficient batching, i.e., for vectors of length n, the prover can prove, for distinct $L_1, \ldots, L_m$ and $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m$ that $L_1(\boldsymbol{x}_1) = y_1, \ldots, L_m(\boldsymbol{x}_m) = y_m$ while incurring a cost that is $\mathcal{O}(m + \log n)$ as opposed to $\mathcal{O}(m \log n)$.

Succinct ZK Argument for Circuit Satisfiability. We construct an improved protocol for arithmetic circuit satisfiability in the universal updatable SRS setting. The CSP approach to handle multiplication gates by linearizing them renders the verifier linear. We propose a protocol for computing a commitment to the linear form that captures the multiplication gates in the circuit in a verifiable way while keeping the verifier succinct. We use the ideas from [52] to preprocess a circuit, and obtain a commitment to the linear constraints. Now, all relations are linearized, we have commitments to all linear forms, and we show how to batch all linear form openings into one protocol for opening a committed linear form on a committed vector.

The following are two key new ideas to make a CSP-like proof have succinct verification. (i) The first relates to how we handle multiplication gates. For linearizing multiplication constraints, CSP uses a linear combination of polynomial evaluations at $1, \ldots, m$ to evaluate a polynomial at a new random value z rendering the verifier linear. We handle multiplication in the same way as CSP, but instead of computing the public linear form for multiplication, the verifier instead *succinctly* verifies a commitment to it. We construct a succinct-verifier protocol for obtaining a *commitment* to the linear form used for verifying multiplication constraints. In order to do this, we impose some structure; specifically, we use a linear combination of polynomial evaluations at $2, 2^2, \ldots, 2^m$. This choice allows us to compute the value of a polynomial at any point z while keeping the verifier succinct. This idea gives a protocol where the prover computes a commitment to the linear form that the verifier can efficiently check. The linearization is now done via a committed linear form. (ii) The second idea relates to how we handle linear gates. Here, we employ the ideas of Daza et al., by reducing the problem of verifying linear gates to checking that two committed vectors are permutations of each other; and a Hadamard product argument. We deviate from [52] by first reducing the permutation argument to the CSP pivot of opening linear forms on committed vectors. We then use our techniques from (i)

to obtain commitments to these linear forms. Finally, we take advantage of our ability to batch the openings of linear forms, which allows us to prove circuit constraints while paying the cost of essentially a single invocation of our linear form protocol.

Homomorphism Opening with Succinct Verifier. Our ideas for succinct verifier in linear form openings do not extend to opening homomorphism. First, we need to commit to a homomorphism, and we extend the commitment scheme for group elements used in [81, 11] to a commitment scheme with a structured key, in order to make the verifier logarithmic. We show that binding is implied by SXDH (same assumption as the scheme with uniform key). Since we rely on SXDH, we cannot encode the commitment key randomness in the second group as the verification key. Thus a pairing check to verify correct key updates is not possible anymore, making our constructions designated verifier. A second hurdle is that the commitment itself lives in the target group. This means that our idea to check correct updation of the key in each round of split-and-fold (which involved a pairing operation) does not work anymore. We tackle this by having the commitment key in both $\mathbb{G}_1$ and $\mathbb{G}_T$. Now, the prover updates the commitment key in $\mathbb{G}_1$ and proves that this has been done correctly. The verifier can check this using a pairing, move this updated commitment key in $\mathbb{G}_1$ to $\mathbb{G}_T$, and then finally at the end of the recursion verify the commitment with respect to the updated key in $\mathbb{G}_T$.

3.2 Preliminaries

In this section, we present the required notation [1] and relevant background for this chapter.

Notation. Let $\mathbb{F}_q$ denote a finite field of order q, also denoted by $\mathbb{F}$ when the order is not specified or is clear from context. Let $\mathbb{G}$ be a group of order q. λ denotes the security parameter, and negl denotes a negligible function, i.e. for any integer $c > 0$, there exists $n \in \mathbb{N}$, such that $\forall\ x > n$, $\mathsf{negl}(x) \leq 1/n^c$. We denote vectors by boldface letters, and the inner product between $\mathbf{a}$ and $\mathbf{b}$ by $\langle \mathbf{a}, \mathbf{b} \rangle$.

We define $\mathcal{L}(\mathbb{F}_q^n)$ as $\mathcal{L}(\mathbb{F}_q^n) = \{L : L \text{ is a linear map from } \mathbb{F}_q^n \text{ to } \mathbb{F}_q\}$. A linear map $L \in \mathcal{L}(\mathbb{F}_q^n)$ is equivalently represented by a vector, i.e. $L(x_1, \ldots, x_n) = a_1 x_1 + \cdots + a_n x_n$ (for $(x_1, \ldots, x_n) \in \mathbb{F}_q^n$) is equivalently represented by $L = (a_1, \ldots, a_n) \in \mathbb{F}_q^n$. For $x = (x_1, \ldots, x_n) \in \mathbb{F}_q^n$, $\mathsf{rev}(x)$ denotes its reversal and is defined as $\mathsf{rev}(x) = (x_n, \ldots, x_1)$.

A vector $\boldsymbol{a} = (a_1, \ldots, a_n)$ naturally defines a $(n-1)$-degree polynomial by considering the vector $\boldsymbol{a}$ as the vector of coefficients, which gives us the polynomial $\boldsymbol{a}(X) = a_1 + a_2 X + a_3 X^2 + \cdots + a_n X^{n-1}$. Also, a commitment to a polynomial $\boldsymbol{a}(X) = a_1 + a_2 X + a_3 X^2 + \cdots + a_n X^{n-1}$ is provided by a commitment to the vector of coefficient $\boldsymbol{a} = (a_1, \ldots, a_n)$. We use the vector and

[1] Note that some of the notations are redefined here for ease of access.

polynomial notation interchangeably throughout the first chapter for ease of notation.

For vectors $\boldsymbol{g} = (g_1, \ldots, g_n) \in \mathbb{G}^n$ and $\boldsymbol{x} = (x_1, \ldots, x_n) \in \mathbb{F}_q^n$, the multi-exponentiation $\boldsymbol{g}^{\boldsymbol{x}}$ is defined by $\boldsymbol{g}^{\boldsymbol{x}} = g_1^{x_1} \cdots g_n^{x_n}$. Also, for $g \in \mathbb{G}$ and $\boldsymbol{x} = (x_1, \ldots, x_n) \in \mathbb{F}_q^n$, $g^{\boldsymbol{x}}$ is defined by $g^{\boldsymbol{x}} = (g^{x_1}, \cdots, g^{x_n})$. The inner product between elements of $\mathbb{F}_q^n$, $\boldsymbol{a} = (a_1, \ldots, a_n)$ and $\boldsymbol{b} = (b_1, \ldots, b_n)$, is denoted by $\langle \boldsymbol{a}, \boldsymbol{b} \rangle = a_1 b_1 + \cdots + a_n b_n$. For $a \in \mathbb{F}_q^m, b \in \mathbb{F}_q^n$, $a \| b \in \mathbb{F}_q^{m+n}$ denotes concatenation of a and b in the respective order, and the notation is used similarly for the vectors in a group $\mathbb{G}$ to denote concatenation of two vectors. For $\boldsymbol{a} = (a_1, \ldots, a_n) \in \mathbb{G}^n$ and $\boldsymbol{b} = (b_1, \ldots, b_n) \in \mathbb{G}^n$, the hadamard product $\boldsymbol{a} \circ \boldsymbol{b}$ is defined by $\boldsymbol{a} \circ \boldsymbol{b} = (a_1 b_1, \ldots, a_n b_n)$. For $v, n \in \mathbb{F}_q$, $\boldsymbol{v^n}$ denotes the vector $\boldsymbol{v^n} = (1, \ldots, v^n)$.

A bilinear group is denoted by the tuple $(q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, G, H) \leftarrow_R \mathcal{G}(1^\lambda)$, where $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ are groups of prime order q, G and H are generators of $\mathbb{G}_1$ and $\mathbb{G}_2$, and $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is an efficiently computable bilinear map.

We define $\mathcal{ML}_{2^\ell} \in \mathbb{F}_q^n$ to be the set of all n-length multilinear vectors of the form $(1, \dot{a}_1, \dot{a}_2, \ldots, \dot{a}_1 \cdots \dot{a}_\ell)$, determined by ℓ-mutually independent scalars $\dot{a}_1, \ldots, \dot{a}_\ell$. We denote the set of ℓ scalars by $\dot{\boldsymbol{a}} = (\dot{a}_1, \ldots, \dot{a}_\ell)$ and the n-length vector by $\overline{\boldsymbol{a}} = (1, \dot{a}_1, \dot{a}_2, \ldots, \dot{a}_1 \cdots \dot{a}_\ell)$. More formally, $\mathcal{ML}_n = \{\overline{\boldsymbol{a}} : \dot{\boldsymbol{a}} = (\dot{a}_1, \ldots, \dot{a}_\ell) \in \mathbb{F}_q^\ell,\ \overline{\boldsymbol{a}} = (\prod_{i=1}^{\ell} \dot{a}_i^{x_i})_{x_i \in \{0,1\}}\}$.

3.2.1 Interactive Arguments

We consider interactive arguments for relations, where a prover P convinces the verifier that it knows a witness w such that for a public statement x, $(x, w) \in \mathcal{R}$. For a pair of PPT interactive algorithms P, V, we denote by $\langle P(w), V \rangle(x)$, the output of V on its interaction with P, where w is P's private input and x is a common input. Let $\mathcal{R} = \{(x, w)\}$ be a relation and $\mathcal{L}$ be the corresponding NP language.

Definition 7 (Argument of Knowledge) *An interactive argument of knowledge (AoK) for a relation $\mathcal{R}$ consists of a PPT algorithm $\mathsf{Setup}(1^\lambda)$ that takes a security parameter λ and outputs public parameters srs, and a pair of PPT interactive algorithms $\langle \mathcal{P}, \mathcal{V} \rangle$. The triple $(\mathsf{Setup}, \mathcal{P}, \mathcal{V})$ satisfy the following properties.*

1. *Completeness. For all $\lambda \in \mathbb{N}$, $(x, w) \in \mathcal{R}$,*

$$\Pr\left(\langle \mathcal{P}(w), \mathcal{V} \rangle(\mathsf{srs}, x) = 1 \ : \ \mathsf{srs} \leftarrow \mathsf{Setup}(1^\lambda)\right) = 1.$$

2. *Knowledge Soundness. An argument system $(\mathcal{P}, \mathcal{V})$ for a relation $\mathcal{R}$ is knowledge sound with error κ if there exists an expected polynomial time extractor Ext such that for every efficient adversary $\tilde{\mathcal{P}}$, for every $x \in \{0,1\}^*$, whenever $\tilde{\mathcal{P}}$ makes $\mathcal{V}$ accept with probability*

$\epsilon > \kappa$, $\mathsf{Ext}^{\tilde{\mathcal{P}}}(x)$ *outputs* w^* *such that* $(x, w^*) \in \mathcal{R}$ *with probability at least* $\frac{\epsilon-\kappa}{q}$ *for some polynomial* q.

Definition 8 (Honest verifier zero-knowledge (HVZK)) *An argument system* $(\mathcal{P}, \mathcal{V})$ *for a relation* $\mathcal{R}$ *is HVZK if there exists an efficient simulator* Sim *such that for every* $(x, w) \in \mathcal{R}$, *the distribution* $\mathsf{Sim}(x)$ *is identical to* $\mathsf{View}_{\langle \mathcal{P}(x,w), \mathcal{V}(x) \rangle}$, *where* $\mathsf{View}_{\langle \mathcal{P}(x,w), \mathcal{V}(x) \rangle}$ *is the distribution of the view of the verifier in the protocol on common input* x *and prover's witness* w.

We now recall the special soundness property, which is typically simpler than knowledge soundness.

Definition 9 (Tree of transcripts) *A set of* $k = \prod_{i=1}^{\ell} k_i$ *accepting transcripts for an argument system* $(\mathcal{P}, \mathcal{V})$ *is a* $(k_1, \ldots, k_\ell)$*-tree of accepting transcripts if they are in the following tree structure: The nodes of the tree are formed by* $\mathcal{P}$*'s messages, and the edges correspond to* $\mathcal{V}$*'s messages. Each node at depth* i *has exactly* k_i *children corresponding to* k_i *distinct messages from the verifier. Each transcript is given by a path from a leaf node to the root.*

Definition 10 (Special Soundness) *A* $(2\ell + 1)$ *move protocol is said to be* $(k_1, \ldots, k_\ell)$ *special sound if there exists an extractor* Ext *that takes as input a* $(k_1, \ldots, k_\ell)$*-tree of accepting transcripts for an instance* x, *and outputs* w *such that* $(x, w) \in \mathcal{R}$.

Definition 11 (Succinct Argument of knowledge) *An argument system is* proof-succinct *if the communication complexity between prover and verifier is bounded by* $\mathsf{poly}(\lambda)$, *and* verifier-succinct *if the running time of* $\mathcal{V}$ *is bounded by* $\mathsf{poly}(\lambda + |x|)$ *and independent of the size of the circuit computing* $\mathcal{R}$.

Fiat Shamir and Non-Interactive AoK. An argument system is said to be *public-coin* if the verifier's messages are uniformly random strings. Public-coin interactive protocols can be heuristically compiled into non-interactive arguments by applying the Fiat-Shamir [59] transform (FS) in the Random Oracle Model (ROM). Building on the above fundamental ZKP concepts, SNARKs stands for the Succinct Non-interactive ARguments of Knowledge.

Note that all of our protocols are public-coin, hence we are only required to prove that our protocols satisfy special soundness for our interactive protocols, and thereafter rely on the FS transform to obtain the non-interactive version of our protocols.

SNARKs in the updatable SRS model. A common reference string (CRS) model assumes the presence of a trusted setup phase before the onset of the protocol execution, which generates a truly random string and makes it available to both the prover and the verifier at the beginning of the protocol. If the CRS has a structure that is leveraged in the protocol to obtain efficient ZKP, then the CRS is known as a structured reference string (SRS). Additionally, an SRS which enables parties to update the parameters during the setup phase, while retaining computational soundness against any probabilistic polynomial time adversary as long as at least one honest update is performed, is known as a universal updatable SRS. In the updatable SRS model, we assume that the setup phase has generated the universal updatable SRS [75] securely, and then the protocol leverages the structure of the SRS shared between the prover and the verifier to enable efficient ZKPs.

We follow the model used by Daza et al. [52], based on [75], where anyone can deterministically compute the circuit-specific preprocessing material given the (updated) universal SRS, which ensures that the circuit-specific preprocessing is performed publicly without any involvement of secrets.

3.2.2 Assumptions

Definition 12 (DLOG Assumption) *The discrete logarithm (DLOG) assumption for a group* $\mathbb{G}$ *states that, given a generator* g *of the group* $\mathbb{G}$*, for all PPT adversaries* $\mathcal{A}$ *we have*

$$\Pr\left(r = r' \mid r \leftarrow_R \mathbb{F}_q \wedge r' \leftarrow \mathcal{A}(g^r)\right) \leq \mathsf{negl}(\lambda)$$

For structured keys following multilinear distribution ($\mathcal{ML}_n$), the following Find-rep assumption holds in bilinear groups, which is known to follow from asymmetric DLOG Assumption [52] (a natural extension of DLOG to accommodate bilinear groups). In the asymmetric DLOG assumption, the adversary receives each element in both groups, that is, for generators $g \in \mathbb{G}_1, h \in \mathbb{G}_2$ and for all $\alpha \in \mathbb{F}_q$, any g^α as an input to the adversary will be accompanied by h^α, and vice versa.

Definition 13 (Find-rep Assumption) *Find-rep assumption holds with respect to a bilinear group generator* BG *for all PPT adversaries* $\mathcal{A}$ *we have*

$$\Pr\left(\begin{array}{c} g^{\langle \boldsymbol{a}, \boldsymbol{x} \rangle} = 1_{\mathbb{G}_1} \wedge \boldsymbol{x} \neq 0 \\ (q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g, h) \leftarrow_R \mathsf{BG}(1^\lambda) \\ \boldsymbol{a} \leftarrow_R \mathcal{ML}_n, \boldsymbol{x} \leftarrow \mathcal{A}(g^{\boldsymbol{a}}, h^{\boldsymbol{a}}) \end{array}\right) \leq \mathsf{negl}(\lambda)$$

Definition 14 (DDH Assumption) *For a group* $\mathbb{G}$*, the decisional Diffie-Hellman (DDH)*

problem is to determine, when given a tuple (g, g^a, g^b, g^c) *for some* $g \in \mathbb{G}$*, whether* $c = ab$*. Decisional Diffie-Hellman (DDH) assumption in a group* $\mathbb{G}$ *states that DDH problem is hard in that group.*

Definition 15 (SXDH Assumption) *For* $(q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g, h) \leftarrow_R \mathcal{G}(1^\lambda)$*, the Symmetric External Diffie-Hellman (SXDH) assumption states that the decisional Diffie-Hellman (DDH) assumption holds for both* $\mathbb{G}_1$ *and* $\mathbb{G}_2$*.*

3.3 Compressed Sigma Protocol for Committed Linear Forms

We construct a protocol to reveal the value $L(x)$ for a committed vector $\boldsymbol{x}$ and committed linear form L, i.e. a public value y satisfies the constraint $y = L(\boldsymbol{x})$. The key idea is to honestly generate a commitment to the (public) linear form in a preprocessing phase. Once generated, a commitment to a linear form L can be used to open L on any committed vector. Note that while using this as a subprotocol for arithmetic circuit SAT, we generate these commitments during a one-time circuit-specific setup phase.

Definition 3.1 (Commitment to $\mathbb{F}_q$-vectors [52]) *Let* $(q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g, H)$ *be a bilinear group and let* $n \geq 0$*. We define a commitment scheme for vectors in* $\mathbb{F}_q^n$ *with the following setup and commitment phase:*

- *Setup: Let* $\dot{\boldsymbol{a}} := (\dot{a}_1, \ldots, \dot{a}_\ell) \leftarrow \mathbb{F}_q^\ell$ *where* $\ell = \log(n+1)$*. Let* $\overline{\boldsymbol{a}} = (a_1, \ldots, a_n) \in \mathbb{F}_q^{n+1}$ *be defined as* $a_j = \prod_{i=1}^{\ell} \dot{a}_i^{b_{ji}}$ *,where* $(b_{j1}, \ldots, b_{j\ell})$ *is the binary representation of* j*. Output* $(g^{\overline{\boldsymbol{a}}}, H^{\dot{\boldsymbol{a}}})$*, where* $g^{\overline{\boldsymbol{a}}} \in \mathbb{G}_1^{n+1}$ *is the commitment key, and* $H^{\dot{\boldsymbol{a}}} \in \mathbb{G}_2^\ell$ *is the verification key.*
- *Commit:* $\mathrm{COM} : \mathbb{F}_q^{n+1} \to \mathbb{G}_1$*,* $\gamma \leftarrow_R \mathbb{F}_q$ *and define*

$$\mathrm{COM}_{\overline{\boldsymbol{a}}}(\boldsymbol{x}; \gamma) \to g^{\langle \overline{\boldsymbol{a}}, (\boldsymbol{x} \| \gamma) \rangle}$$

Lemma 3.1 *The above scheme is perfectly hiding and computationally binding under the DLOG assumption [52].*

Proof: The prover is given the (public) commitment key $g^{\overline{\boldsymbol{a}}}$ where $\overline{\boldsymbol{a}} = \{\overline{a}_1, \ldots, \overline{a}_{n+1}\}$, i.e. $g^{\overline{\boldsymbol{a}}} = (g^{\overline{a}_1}, \ldots, g^{\overline{a}_{n+1}})$.

Proof of Hiding. We note that, given a commitment P to any $\mathbb{F}_q$ vector, any vector of elements of $\mathbb{F}_q$, $\boldsymbol{x} = (x_1, \ldots, x_n) \in \mathbb{F}_q^n$, could have been the element that is chosen to compute P, if the uniformly randomly chosen randomness γ for obtaining the commitment is such that $g^{\langle \overline{\boldsymbol{a}}, (\boldsymbol{x} \| \gamma) \rangle} = P$ i.e. γ satisfies :

$$\left(g^{\overline{a}_{n+1}}\right)^{\gamma} = \frac{P}{\left(g^{\overline{a}_1}\right)^{x_1} \cdots \left(g^{\overline{a}_n}\right)^{x_n}}$$

Hence, the aforementioned scheme is perfectly hiding.

Proof of Binding. If the binding of this commitment scheme is broken, then we have $\boldsymbol{x} = (x_1, \ldots, x_n), \gamma$ and $\boldsymbol{y} = (y_1, \ldots, y_n), \delta$ such that $\boldsymbol{x} \neq \boldsymbol{y}$ and

$$\begin{aligned}
& & \mathrm{COM}_{\overline{\boldsymbol{a}}}\ (\boldsymbol{x}; \gamma) &= \mathrm{COM}_{\overline{\boldsymbol{a}}}\ (\boldsymbol{y}; \delta) \\
\implies & & g^{\langle \overline{\boldsymbol{a}}, (\boldsymbol{x} \| \gamma) \rangle} &= g^{\langle \overline{\boldsymbol{a}}, (\boldsymbol{y} \| \delta) \rangle} \\
\implies & & g^{\langle \overline{\boldsymbol{a}}, (\boldsymbol{x} - \boldsymbol{y} \| \gamma - \delta) \rangle} &= 1_{\mathbb{G}_1} \\
\implies & \left(g^{\overline{a}_1}\right)^{(x_1 - y_1)} \cdots \left(g^{\overline{a}_n}\right)^{(x_n - y_n)} \cdot \left(g^{\overline{a}_{n+1}}\right)^{(\gamma - \delta)} &&= 1_{\mathbb{G}_1}
\end{aligned}$$

which breaks the (extended) discrete logarithm assumption.

□

We start with a Σ-Protocol for opening a linear form which is similar to the initial protocol in [8] but using structured keys instead of uniformly random keys for the commitments. We consider the following relation

$$\mathcal{R} = \{(P \in \mathbb{G}, L \in \mathcal{L}(\mathbb{F}_q^n), y \in \mathbb{F}_q; \boldsymbol{x} \in \mathbb{F}_q^n, \gamma \in \mathbb{F}_q) : P = \mathrm{COM}_{\overline{\boldsymbol{a}}}\ (\boldsymbol{x}; \gamma)\ \wedge\ L(\boldsymbol{x}) = y\}$$

which corresponds to showing opening of a public commitment P and a public value y, obtained by operating a linear form L on a secret $\mathbb{F}_q^n$ vector $\boldsymbol{x}$. This is the same relation as in [8] but using the commitment COM with structured commitment key $(g^{\overline{\boldsymbol{a}}}, H^{\dot{\boldsymbol{a}}})$ (Definition 3.1). We rely on the SXDH assumption for providing the structured key $(g^{\overline{\boldsymbol{a}}}, H^{\dot{\boldsymbol{a}}})$ while maintaining security.

Theorem 3.1 *Π_0 is a 3-move protocol for relation $\mathcal{R}$. It is perfectly complete, special honest-verifier zero-knowledge and computationally special sound.*

Proof: Completeness. If protocol steps by the prover is executed correctly, then we have

Parameters

- Common parameters : $(P \in \mathbb{G}, L \in \mathcal{L}(\mathbb{F}_q^n), y \in \mathbb{F}_q),\ P = \mathrm{COM}_{\overline{\boldsymbol{a}}}\ (\boldsymbol{x}; \gamma), y = L(\boldsymbol{x})$
- $\mathcal{P}$'s input : $(\boldsymbol{x} \in \mathbb{F}_q^n, \gamma \in \mathbb{F}_q)$

Protocol

1. $\mathcal{P}$ samples $\boldsymbol{r} \leftarrow_R \mathbb{F}_q^n, \rho \leftarrow_R \mathbb{F}_q$, computes $A = \mathrm{COM}_{\overline{\boldsymbol{a}}}\ (\boldsymbol{r}; \rho),\ t = L(\boldsymbol{r})$ and sends A, t to $\mathcal{V}$
2. $\mathcal{V}$ samples $c \leftarrow_R \mathbb{F}_q$ and sends c to $\mathcal{P}$
3. $\mathcal{P}$ computes $\boldsymbol{z} = c\boldsymbol{x} + \boldsymbol{r}$ and $\phi = c\gamma + \rho$ and sends $\boldsymbol{z}, \phi$ to $\mathcal{V}$
4. $\mathcal{V}$ checks if $\mathrm{COM}_{\overline{\boldsymbol{a}}}\ (\boldsymbol{z}; \phi) = AP^c$ and $L(\boldsymbol{z}) = cy + t$, outputs 1 if it holds, outputs 0 otherwise.

Figure 3.1: Protocol Π_0 for relation $\mathcal{R}$

$\boldsymbol{z} = c\boldsymbol{x} + \boldsymbol{r}$, and it satisfies the final two checks by the verifier

$$\begin{aligned} \mathrm{COM}_{\overline{\boldsymbol{a}}}\ (\boldsymbol{z}; \phi) &= g^{\langle \overline{\boldsymbol{a}}, (\boldsymbol{z} \| \phi) \rangle}, \\ &= g^{\langle \overline{\boldsymbol{a}}, (c\boldsymbol{x} + \boldsymbol{r} \| c\gamma + \rho) \rangle} \\ &= g^{c\langle \overline{\boldsymbol{a}}, (\boldsymbol{x} \| \gamma) \rangle} g^{\langle \overline{\boldsymbol{a}}, (\boldsymbol{r} \| \rho) \rangle} \\ &= P^c A \end{aligned} \qquad \begin{aligned} L(\boldsymbol{z}) &= L(c\boldsymbol{x} + \boldsymbol{r}) \\ &= cL(\boldsymbol{x}) + L(\boldsymbol{r}) \\ &= cy + t \end{aligned}$$

Special Honest-Verifier Zero-Knowledge. We construct a simulator Sim, which produces a transcript indistinguishable from the transcript of the real execution of the protocol, provided a challenge $c \in \mathbb{F}_q$. (i) Sim samples $\boldsymbol{z}, \phi$ (ii) Sim computes $\mathrm{COM}_{\overline{\boldsymbol{a}}}\ (\boldsymbol{z}; \phi)$ and sets $A = \frac{\mathrm{COM}_{\overline{\boldsymbol{a}}}\ (\boldsymbol{z}; \phi)}{P^c}$ and $t = L(\boldsymbol{z}) - cy$.

The transcript produced by the simulator Sim is indistinguishable from the transcript of the real execution of the protocol due to the hiding property of the commitment scheme $\mathrm{COM}_{(.)}\ (.)$, which ensures that a commitment sampled uniformly at random from the set of all possible commitments is indistinguishable from a commitment computed from a message chosen uniformly at random.

Special Soundness. We consider 2 accepting transcripts $(A, t, c_1, \boldsymbol{z}_1, \phi_1)$ and $(A, t, c_2, \boldsymbol{z}_2, \phi_2)$, such that $c_1 \neq c_2$. Then we have,

$$\begin{aligned} &\mathrm{COM}_{\overline{\boldsymbol{a}}}\ (\boldsymbol{z}_1; \phi_1) = AP^{c_1}, L(\boldsymbol{z}_1) = c_1 y + t, \qquad \text{and} \\ &\mathrm{COM}_{\overline{\boldsymbol{a}}}\ (\boldsymbol{z}_2; \phi_2) = AP^{c_2}, L(\boldsymbol{z}_2) = c_2 y + t \end{aligned}$$

$$\implies g^{\langle \overline{\boldsymbol{a}}, (\boldsymbol{z}_1 \| \phi_1) \rangle} = AP^{c_1}, L(\boldsymbol{z}_1) = c_1 y + t, \qquad \text{and}$$
$$g^{\langle \overline{\boldsymbol{a}}, (\boldsymbol{z}_2 \| \phi_2) \rangle} = AP^{c_2}, L(\boldsymbol{z}_2) = c_2 y + t$$

Dividing the first two equations, and subtracting the second equations, we get

$$g^{\langle \overline{\boldsymbol{a}}, (\boldsymbol{z}_1 - \boldsymbol{z}_2 \| \phi_1 - \phi_2) \rangle} = P^{c_1 - c_2}, L(\boldsymbol{z}_1 - \boldsymbol{z}_2) = (c_1 - c_2) y$$

We define $\boldsymbol{x} = (\boldsymbol{z}_1 - \boldsymbol{z}_2)/(c_1 - c_2)$ and $\gamma = (\phi_1 - \phi_2)/(c_1 - c_2)$, and this gives us $g^{\langle \overline{\boldsymbol{a}}, (\boldsymbol{x}, \gamma) \rangle} = \mathrm{COM}_{\overline{\boldsymbol{a}}}(\boldsymbol{x}, \gamma) = P$, and $L(\boldsymbol{x}) = y$. □

3.3.1 Opening a Committed Linear Form

In Π_0, the communication complexity as well as the verifier complexity is linear due to the last message sent by the prover and the last check performed by the verifier. To improve both complexities, we replace the message sent in the last step of Π_0 with a proof of knowledge. We define a relation that captures this and reduce the verifier's work by committing to the linear form and compressing the check using split-and-fold technique used in [8]. The protocol Π_1 is in Fig 3.2. We compress recursively until the size of instance is constant and can be sent in the clear.

We now consider the new relation $\mathcal{R}_{\mathrm{CLF}}$ with respect to an updated linear form, where the new linear form L is defined as $L(\boldsymbol{z}, \phi) := L(\boldsymbol{z})$ and hence, the check performed by the verifier in step 4 of Π_0 (Fig 3.1) corresponds to the new relation, where the message sent by the prover $\mathcal{P}$ to the verifier $\mathcal{V}$ in step 3 corresponds to a witness in the new relation.

$$\mathcal{R}_{\mathrm{CLF}} = \{(P \in \mathbb{G}, Q \in \mathbb{G}, y \in \mathbb{F}_q; \boldsymbol{x} \in \mathbb{F}_q^n, L \in \mathcal{L}(\mathbb{F}_q^n)) : \\ P = \mathrm{COM}_{\overline{\boldsymbol{a}}}(\boldsymbol{x}) \ \wedge \ Q = \mathrm{COM}_{\overline{\boldsymbol{a}}}(L) \ \wedge \ L(\boldsymbol{x}) = y\}\}$$

This corresponds to showing opening of a public commitment P and a public value y, which is the output of a linear form L on a secret $\mathbb{F}_q^n$ vector $\boldsymbol{x}$, given a commitment to the linear form L. We present the Σ-Protocol Π_1 for $\mathcal{R}_{\mathrm{CLF}}$ in Fig 3.2, and use this protocol instead of step 4 of Π_0 to improve the communication and verifier complexity.

Finally, we define $\Pi_{1\text{-}\mathcal{R}}$ as $\Pi_{1\text{-}\mathcal{R}} = \Pi_1 \circ \Pi_0$ for relation $\mathcal{R}$, whose communication and computational complexity are dominated by that of Π_1. The concatenation of the protocols Π_1 and Π_0 proceeds by replacing the last message sent in clear by the prover in Π_0 with a proof of knowledge using Π_1.

Parameters

- **Common parameters** : $(P \in \mathbb{G}, Q \in \mathbb{G}, y \in \mathbb{F}_q, H^{\dot{\boldsymbol{a}}} \in \mathbb{G}_2^\ell)$,
  - $P = \mathrm{COM}_{\overline{\boldsymbol{a}}}(\boldsymbol{x}), Q = \mathrm{COM}_{\overline{\boldsymbol{a}}}(L), y = L(\boldsymbol{x})$,
  - $n = 2^\ell, \dot{\boldsymbol{a}} = (\dot{a}_1, \ldots, \dot{a}_\ell), \overline{\boldsymbol{a}} = \left(\prod_{i=1}^{\ell} \dot{a}_i^{b_i}\right)_{b_i \in \{0,1\}}$
  - A bilinear group description $(q, \mathbb{G}, \mathbb{G}_2, \mathbb{G}_T, e, g, H)$, where $e : \mathbb{G} \times \mathbb{G}_2 \mapsto \mathbb{G}_T$ is an efficient bilinear map and g, H and $e(g, H)$ are generators of groups $\mathbb{G}, \mathbb{G}_2$ and $\mathbb{G}_T$, respectively, each of order q.
- $\mathcal{P}$**'s input** : $(g^{\overline{\boldsymbol{a}}} \in \mathbb{G}^n, \boldsymbol{x} \in \mathbb{F}_q^n, L \in \mathcal{L}(\mathbb{F}_q^n))$

Protocol

1. $\mathcal{P}$ parses $\boldsymbol{x} = (\boldsymbol{x}_L \| \boldsymbol{x}_R), L = (L_L \| L_R)$ and $g^{\overline{\boldsymbol{a}}} = \left(g^{\overline{\boldsymbol{a}}_L} \| g^{\dot{a}_\ell \overline{\boldsymbol{a}}_L}\right)$, and computes and sends the following to $\mathcal{V}$:
   (a) $A_1 = \mathrm{COM}_{\overline{\boldsymbol{a}}_L}(\boldsymbol{x}_L), A_2 = \mathrm{COM}_{\overline{\boldsymbol{a}}_L}(\boldsymbol{x}_R)$
   (b) $B_1 = \mathrm{COM}_{\overline{\boldsymbol{a}}_L}(L_L), B_2 = \mathrm{COM}_{\overline{\boldsymbol{a}}_L}(L_R)$
   (c) $y_1 = \langle L_R, \boldsymbol{x}_L \rangle, y_2 = \langle L_L, \boldsymbol{x}_R \rangle$
2. $\mathcal{V}$ checks the following, proceeds to step 3 if it holds, and aborts otherwise
$$e\left(\frac{P}{A_1}, H\right) = e\left(A_2, H^{\dot{a}_\ell}\right) \ \wedge \ e\left(\frac{Q}{B_1}, H\right) = e\left(B_2, H^{\dot{a}_\ell}\right)$$
3. $\mathcal{V}$ samples $c \leftarrow_R \mathbb{F}_q$ and sends c to $\mathcal{P}$
4. $\mathcal{P}$ sets $\boldsymbol{x}' = \boldsymbol{x}_L + c\boldsymbol{x}_R$, $L' = cL_L + L_R$ and implicitly sets $\dot{\boldsymbol{a}}' = (\dot{a}_1, \ldots, \dot{a}_{\ell-1})$ and $\overline{\boldsymbol{a}}' = \overline{\boldsymbol{a}}_L$.
5. $\mathcal{P}$ and $\mathcal{V}$ both compute the following : $P' = A_1 A_2^c, Q' = B_1^c B_2, y' = y_1 + cy + c^2 y_2$
6. If $\boldsymbol{x}' \notin \mathbb{F}_q^2$: $\mathcal{P}$ runs PoK Π_1 to prove knowledge of $\boldsymbol{x}', L'$ such that $\mathrm{COM}_{\overline{\boldsymbol{a}}'}(\boldsymbol{x}') = P'$, $\mathrm{COM}_{\overline{\boldsymbol{a}}'}(L') = Q'$ and $\langle L', \boldsymbol{x}' \rangle = y'$.
   Hence, $\mathcal{P}$ and $\mathcal{V}$ run the protocol Π_1 with updated common parameters $(P', Q', y', g^{\dot{\boldsymbol{a}}'})$ and prover's input $(g^{\overline{\boldsymbol{a}}'}, \boldsymbol{x}', L')$, for $(P', Q', y'; \boldsymbol{x}') \in \mathcal{R}_{\mathrm{CLF}}$
7. If $\boldsymbol{x}' \in \mathbb{F}_q^2$:
   (a) $\mathcal{P}$ sends $\boldsymbol{x}', L'$ to $\mathcal{V}$
   (b) $\mathcal{V}$ outputs 1 if the following checks hold, and 0 otherwise:
$$\mathrm{COM}_{\overline{\boldsymbol{a}}'}(\boldsymbol{x}') = P' \ \wedge \ \mathrm{COM}_{\overline{\boldsymbol{a}}'}(L') = Q' \ \wedge \ \langle L', \boldsymbol{x}' \rangle = y'$$

Figure 3.2: Protocol Π_1 for relation $\mathcal{R}_{\mathrm{CLF}}$

Theorem 3.2 *Π_1 is a $(k_1, \ldots, k_\ell)$-move protocol for relation $\mathcal{R}_{CLF}$, where $k_i = 3,\ \forall i \in [\ell], \ell = \log n$. It is perfectly complete and computationally special sound. It incurs total communication of $4 \log n$ group elements and $4 + 3 \log n$ field elements.*

Proof Sketch. Here we present the proof sketch for the special soundness of Π_1. Given 3 accepting transcripts $(A_1, A_2, B_1, B_2, y_1, y_2, c_i, \boldsymbol{x}'_i, L'_i)$ for one iteration of Π_1 (where one iteration consists of steps 1-5, and step 6 follows by sending $\boldsymbol{x}', L'$ instead of providing a PoK) for three distinct challenges c_1, c_2 and c_3, extractor proceeds as follows. It computes a_1, a_2, a_3 as $(a_1, a_2, a_3)^T = V^{-1}(0, 1, 0)^T$, where V is the Vandermonde matrix defined by the the challenges, and sets $\boldsymbol{w} = \sum_i a_i(c_i \boldsymbol{x}'_i \| \boldsymbol{x}'_i)$ to be the extracted value. We show that $\mathrm{COM}_{\overline{\boldsymbol{a}}}(\boldsymbol{w}) = P$; and similarly we extract a valid opening $\boldsymbol{m}$ of the commitment Q.

We then show that the extracted $\boldsymbol{w}, \boldsymbol{m}$ satisfy $\boldsymbol{x}'_i = \boldsymbol{w}_L + c_i \boldsymbol{w}_R$ and $L'_i = \boldsymbol{m}_L + c_i \boldsymbol{m}_R$ for all $i = 1, 2, 3$, which when substituted in the verification equation $\langle L'_i, \boldsymbol{x}'_i \rangle = y'_i$ (Step 7(b)) gives us $\langle \boldsymbol{m}_R, \boldsymbol{w}_L \rangle + c_i \langle \boldsymbol{m}, \boldsymbol{w} \rangle + c_i^2 \langle \boldsymbol{m}_L, \boldsymbol{w}_R \rangle = y_1 + c_i y + c_i^2 y_2$, for the distinct challenges c_1, c_2 and c_3. Hence, $\langle \boldsymbol{m}, \boldsymbol{w} \rangle = y$ holds, which shows that $(\boldsymbol{w}, \boldsymbol{m})$ is a valid witness for $(P, Q, y) \in \mathcal{R}_{\mathrm{CLF}}$.

We now present the full proof.

Proof: Completeness. If the protocol is correctly executed by the prover $\mathcal{P}$, then we have

1. $A_1 = \mathrm{COM}_{\overline{\boldsymbol{a}}_L}(\boldsymbol{x}_L) = g^{\langle \overline{\boldsymbol{a}}_L, \boldsymbol{x}_L \rangle}, A_2 = \mathrm{COM}_{\overline{\boldsymbol{a}}_L}(\boldsymbol{x}_R) = g^{\langle \overline{\boldsymbol{a}}_L, \boldsymbol{x}_R \rangle}$
2. $B_1 = \mathrm{COM}_{\overline{\boldsymbol{a}}_L}(L_L) = g^{\langle \overline{\boldsymbol{a}}_L, L_L \rangle},\ B_2 = \mathrm{COM}_{\overline{\boldsymbol{a}}_L}(L_R) = g^{\langle \overline{\boldsymbol{a}}_L, L_R \rangle}$
3. $y_1 = \langle \boldsymbol{x}_L, L_R \rangle,\ y_2 = \langle \boldsymbol{x}_R, L_L \rangle$
4. $\boldsymbol{x}' = \boldsymbol{x}_L + c\boldsymbol{x}_R,\ L' = cL_L + L_R$

Hence, the following verifier checks are satisfied as shown below :

$$
\begin{aligned}
e\left(\frac{P}{A_1}, H\right) &= e\left(\frac{g^{\langle \overline{\boldsymbol{a}}, \boldsymbol{x} \rangle}}{g^{\langle \overline{\boldsymbol{a}}_L, \boldsymbol{x}_L \rangle}}, H\right) \\
&= e(g^{\langle \overline{\boldsymbol{a}}_R, \boldsymbol{x}_R \rangle}, H) \\
&= e(g^{\langle \dot{a}_\ell \overline{\boldsymbol{a}}_L, \boldsymbol{x}_R \rangle}, H) \\
&= e(g^{\langle \overline{\boldsymbol{a}}_L, \boldsymbol{x}_R \rangle}, H^{\dot{a}_\ell}) \\
&= e(A_2, , H^{\dot{a}_\ell})
\end{aligned}
\qquad
\begin{aligned}
e\left(\frac{Q}{B_1}, H\right) &= e\left(\frac{g^{\langle \overline{\boldsymbol{a}}, L \rangle}}{g^{\langle \overline{\boldsymbol{a}}_L, L_L \rangle}}, H\right) \\
&= e(g^{\langle \overline{\boldsymbol{a}}_R, L_R \rangle}, H) \\
&= e(g^{\langle \dot{a}_\ell \overline{\boldsymbol{a}}_L, L_R \rangle}, H) \\
&= e(g^{\langle \overline{\boldsymbol{a}}_L, L_R \rangle}, H^{\dot{a}_\ell}) \\
&= e(B_2, , H^{\dot{a}_\ell})
\end{aligned}
$$

For each iteration of PoK Π_1 (where one iteration consists of steps 1-5, and step 6 follows by sending $\boldsymbol{x}', L'$ instead of providing a PoK), we have

$$\begin{aligned}\mathrm{COM}_{\overline{\boldsymbol{a}}'}(\boldsymbol{x}') &= g^{\langle \overline{\boldsymbol{a}}', \boldsymbol{x}' \rangle} \\ &= g^{\langle \overline{\boldsymbol{a}}_L, \boldsymbol{x}_L + c\boldsymbol{x}_R \rangle} \\ &= g^{\langle \overline{\boldsymbol{a}}_L, \boldsymbol{x}_L \rangle} g^{c\langle \overline{\boldsymbol{a}}_L, \boldsymbol{x}_R \rangle} \\ &= A_1 A_2^c\end{aligned} \qquad \begin{aligned}\mathrm{COM}_{\overline{\boldsymbol{a}}'}(L') &= g^{\langle \overline{\boldsymbol{a}}', L' \rangle} \\ &= g^{\langle \overline{\boldsymbol{a}}_L, cL_L + L_R \rangle} \\ &= g^{c\langle \overline{\boldsymbol{a}}_L, L_L \rangle} g^{\langle \overline{\boldsymbol{a}}_L, L_R \rangle} \\ &= B_1^c B_2\end{aligned}$$

$$\begin{aligned}\langle L', \boldsymbol{x}' \rangle &= \langle cL_L + L_R, \boldsymbol{x}_L + c\boldsymbol{x}_R \rangle \\ &= c\langle L_L, \boldsymbol{x}_L \rangle + c^2 \langle L_L, \boldsymbol{x}_R \rangle + \langle L_R, \boldsymbol{x}_L \rangle + c\langle L_R, \boldsymbol{x}_R \rangle \\ &= \langle L_R, \boldsymbol{x}_L \rangle + c(\langle L_L, \boldsymbol{x}_L \rangle + \langle L_R, \boldsymbol{x}_R \rangle) + c^2 \langle L_L, \boldsymbol{x}_R \rangle \\ &= \langle L_R, \boldsymbol{x}_L \rangle + c\langle L, \boldsymbol{x} \rangle + c^2 \langle L_L, \boldsymbol{x}_R \rangle \\ &= y_1 + cy + c^2 y_2\end{aligned}$$

Special Soundness. We first illustrate the extraction of the witness, and thereafter proceed with the argument of the correctness of the extracted value. We begin with three accepting transcripts for one iteration of PoK Π_1 (where one iteration consists of steps 1-5, and step 6 follows by sending $\boldsymbol{x}', L'$ instead of providing a PoK) as follows, where c_1, c_2, c_3 are all distinct challenges:

$$\begin{aligned}&(A_1, A_2, B_1, B_2, y_1, y_2, c_1, \boldsymbol{x}_1', L_1') \\ &(A_1, A_2, B_1, B_2, y_1, y_2, c_2, \boldsymbol{x}_2', L_2') \\ &(A_1, A_2, B_1, B_2, y_1, y_2, c_3, \boldsymbol{x}_3', L_3')\end{aligned}$$

Extraction. The extraction proceeds as follows. We aim to find $\boldsymbol{w}, \boldsymbol{m}$ such that $\langle \boldsymbol{m}, \boldsymbol{w} \rangle = y$, and $\boldsymbol{w}, \boldsymbol{m}$ are openings of P and Q. We note that, as c_1, c_2 and c_3 are such that $c_i \neq c_j$ for all $(i \neq j)$ $i, j \in \{1, 2, 3\}$, the Vandermonde matrix V described below is invertible.

$$V = \begin{pmatrix} 1 & 1 & 1 \\ c_1 & c_2 & c_3 \\ c_1^2 & c_2^2 & c_3^2 \end{pmatrix}$$

Hence, we can compute a_1, a_2, a_3 as $(a_1, a_2, a_3)^T = V^{-1}(0, 1, 0)^T$. The computed a_1, a_2, a_3 satisfy $\sum_i a_i = 0, \sum_i a_i c_i = 1$ and $\sum_i a_i c_i^2 = 0$. Define $\boldsymbol{z}_i = (c_i \boldsymbol{x}_i' \| \boldsymbol{x}_i')$. Now let $\boldsymbol{w} = a_1\boldsymbol{z}_1 + a_2\boldsymbol{z}_2 + a_3\boldsymbol{z}_3$ be the extracted value. Following a similar procedure, we extract $\boldsymbol{m}$ and output $(\boldsymbol{w}, \boldsymbol{m})$ as the witness for $\mathcal{R}_{\mathrm{CLF}}$.

Proof of correctness of extracted value. We first prove that the extracted $\boldsymbol{w}$ and $\boldsymbol{m}$ are openings of commitments P and Q, respectively. Then we prove that $\boldsymbol{w}$ and $\boldsymbol{m}$ also satisfies the

constraint $\langle \boldsymbol{m}, \boldsymbol{w} \rangle = y$. This shows that the extracted $(\boldsymbol{w}, \boldsymbol{m})$ is a valid witness for $(P, Q, y) \in \mathcal{R}_{\text{CLF}}$.

We recall that $\boldsymbol{w} = a_1\boldsymbol{z}_1 + a_2\boldsymbol{z}_2 + a_3\boldsymbol{z}_3$ by definition, and $\text{COM}_{\overline{\boldsymbol{a}}'}(\boldsymbol{x}'_i) = A_1 A_2^{c_i}$ holds from verification equation in step 6.

$$
\begin{aligned}
\text{COM}_{\overline{\boldsymbol{a}}}(\boldsymbol{w}) &= g^{\langle \overline{\boldsymbol{a}}, \boldsymbol{w} \rangle} \\
&= g^{\langle \overline{\boldsymbol{a}}, a_1\boldsymbol{z}_1 + a_2\boldsymbol{z}_2 + a_3\boldsymbol{z}_3 \rangle} \\
&= g^{\langle \overline{\boldsymbol{a}}_L \| \overline{\boldsymbol{a}}_R, a_1(c_1\boldsymbol{x}'_1 \| \boldsymbol{x}'_1) + a_2(c_2\boldsymbol{x}'_2 \| \boldsymbol{x}'_2) + a_3(c_3\boldsymbol{x}'_3 \| \boldsymbol{x}'_3) \rangle} \\
&= g^{\langle \overline{\boldsymbol{a}}_L, a_1c_1\boldsymbol{x}'_1 + a_2c_2\boldsymbol{x}'_2 + a_3c_3\boldsymbol{x}'_3 \rangle} g^{\langle \overline{\boldsymbol{a}}_R, a_1\boldsymbol{x}'_1 + a_2\boldsymbol{x}'_2 + a_3\boldsymbol{x}'_3 \rangle} \\
&= g^{\langle \overline{\boldsymbol{a}}', a_1c_1\boldsymbol{x}'_1 + a_2c_2\boldsymbol{x}'_2 + a_3c_3\boldsymbol{x}'_3 \rangle} g^{\langle \dot{a}_\ell \overline{\boldsymbol{a}}', a_1\boldsymbol{x}'_1 + a_2\boldsymbol{x}'_2 + a_3\boldsymbol{x}'_3 \rangle} \\
&= (A_1 A_2^{c_1})^{a_1c_1} (A_1 A_2^{c_2})^{a_2c_2} (A_1 A_2^{c_3})^{a_3c_3} (A_1 A_2^{c_1})^{a_1\dot{a}_\ell} (A_1 A_2^{c_2})^{a_2\dot{a}_\ell} (A_1 A_2^{c_3})^{a_3\dot{a}_\ell} \text{ (from step 6)} \\
&= A_1^{(a_1c_1 + a_2c_2 + a_3c_3) + \dot{a}_\ell(a_1 + a_2 + a_3)} A_2^{(a_1c_1^2 + a_2c_2^2 + a_3c_3^2) + \dot{a}_\ell(a_1c_1 + a_2c_2 + a_3c_3)} \\
&= A_1 A_2^{\dot{a}_\ell} \\
&= P \quad \left(\text{since we have } e\left(\frac{P}{A_1}, H\right) = e\left(A_2, H^{\dot{a}_\ell}\right) \text{ which ensures } P = A_1 A_2^{\dot{a}_\ell}\right)
\end{aligned}
$$

Hence, the extracted $\boldsymbol{w}$ is an opening of the commitment P. Similarly, we can prove that $\boldsymbol{m}$ is an opening of the commitment Q. From the binding of the commitment scheme, we can ensure that $\boldsymbol{w} = \boldsymbol{x}$ and $\boldsymbol{m} = L$ except with negligible probability.

Let $\boldsymbol{b} \in \mathbb{F}_q^n$ be such that $\boldsymbol{x}'_i = \boldsymbol{b}_L + c_i\boldsymbol{b}_R$ holds for all $i = 1, 2, 3$. Then our defined $\boldsymbol{z}_i$ can be interpreted as $\boldsymbol{z}_i = (c_i\boldsymbol{x}'_i \| \boldsymbol{x}'_i) = (0 \| \boldsymbol{b}_L) + c_i(\boldsymbol{b}_L \| \boldsymbol{b}_R) + c_i^2(\boldsymbol{b}_R \| 0)$ for all $i = 1, 2, 3$. Now, given a_1, a_2, a_3 that satisfy $\sum_i a_i = 0, \sum_i a_i c_i = 1$ and $\sum_i a_i c_i^2 = 0$, we have $\boldsymbol{w} = \sum_i a_i\boldsymbol{z}_i = \sum_i a_i(0 \| \boldsymbol{b}_L) + \sum_i a_i c_i(\boldsymbol{b}_L \| \boldsymbol{b}_R) + \sum_i a_i c_i^2(\boldsymbol{b}_R \| 0) = \boldsymbol{b}$. Hence, the extracted $\boldsymbol{w}$ satisfies $\boldsymbol{x}'_i = \boldsymbol{w}_L + c_i\boldsymbol{w}_R$. Similarly, the extracted value $\boldsymbol{m}$ also satisfies $L'_i = \boldsymbol{m}_L + c_i\boldsymbol{m}_R$ for all $i = 1, 2, 3$.

Since the transcripts are accepting, step 7(b) of the verification equation, $\langle L'_i, \boldsymbol{x}'_i \rangle = y'_i$ holds; that is $\langle L'_i, \boldsymbol{x}'_i \rangle = y_1 + c_i y + c_i^2 y_2$, for all $i = 1, 2, 3$. Substituting the values of L'_i and $\boldsymbol{x}'_i$, we get that $\langle \boldsymbol{m}_R, \boldsymbol{w}_L \rangle + c_i\langle \boldsymbol{m}, \boldsymbol{w} \rangle + c_i^2\langle \boldsymbol{m}_L, \boldsymbol{w}_R \rangle = y_1 + c_i y + c_i^2 y_2$ holds for all $i = 1, 2, 3$. Hence, we obtain that $\langle \boldsymbol{m}, \boldsymbol{w} \rangle = y$. □

3.3.2 Improved Protocol for Opening a Committed Linear Form

We recall that for $x = (x_1, \ldots, x_n) \in \mathbb{F}_q^n$, $\mathsf{rev}(x)$ is defined as $\mathsf{rev}(x) = (x_n, \ldots, x_1)$. We present an alternative protocol that achieves better communication complexity at the cost of degrading soundness; it needs $2n$ transcripts to extract. Consider a modified version of the relation $\mathcal{R}_{\text{CLF}}$

defined earlier, where instead of committing to the linear form we now commit to the reverse of the linear form, and define the new relation $\mathcal{R}_{\text{CLF-rev}}$ as follows :

$$\mathcal{R}_{\text{CLF-rev}} = \{(P \in \mathbb{G}, Q \in \mathbb{G}, y \in \mathbb{F}_q; \boldsymbol{x} \in \mathbb{F}_q^n, L \in \mathcal{L}(\mathbb{F}_q^n)) : \\ P = \text{COM}_{\overline{\boldsymbol{a}}}(\boldsymbol{x}) \ \wedge \ Q = \text{COM}_{\overline{\boldsymbol{a}}}(\mathsf{rev}(L)) \ \wedge \ L(\boldsymbol{x}) = y\}\}$$

where the relation $\mathcal{R}_{\text{CLF-rev}}$ corresponds to showing opening of a public commitment P and a public value y, obtained by operating a linear form L on a secret $\mathbb{F}_q^n$ vector $\boldsymbol{x}$, where we also have a commitment to the reverse of linear form L which is represented as a vector. We note that the randomness used for the commitments is implicitly assumed from here onwards. We have the following protocol for the relation $\mathcal{R}_{\text{CLF-rev}}$ (Fig 3.3).

The protocol aims to reduce the verification of the statement $(P, Q, y; \boldsymbol{x}) \in \mathcal{R}_{\text{CLF-rev}}$ by prover $\mathcal{P}$ and verifier $\mathcal{V}$, to a polynomial check where we have the equation

$$\boldsymbol{x}(U) \cdot \mathsf{rev}(L)(U) = p_L(U) \cdot U^{-1} + y \cdot U^{n-1} + p_R(U) \cdot U^n$$

and we have commitment to each polynomial. The polynomials are then evaluated at the random challenge sent by the verifier $\mathcal{V}$, and the consistency of the evaluations with the equation satisfied by the polynomial is checked.

Theorem 3.3 *Π_2 is a protocol for relation $\mathcal{R}_{CLF\text{-}rev}$. It is perfectly complete and computationally special sound.*

Proof: **Completeness** follows directly.

Special Soundness. We consider 4 accepting transcripts for one iteration of PoK Π_2 with different challenges $t_i, i \in \{1, 2, 3\}$ as follows, where t_1, t_2, t_3 are all distinct challenges:

$$(A_1, A_2, c, z_1, z_2, z_3, z_4, t_1, \boldsymbol{w}_1)$$
$$(A_1, A_2, c, z_1, z_2, z_3, z_4, t_2, \boldsymbol{w}_2)$$
$$(A_1, A_2, c, z_1, z_2, z_3, z_4, t_3, \boldsymbol{w}_3)$$
$$(A_1, A_2, c, z_1, z_2, z_3, z_4, t_4, \boldsymbol{w}_4)$$

We note that, as t_1, t_2, t_3 and t_4 are such that $t_i \neq t_j$ for all $(i \neq j)$ $i, j \in \{1, 2, 3, 4\}$, the

Parameters

- **Common parameters** : $(P \in \mathbb{G}, Q \in \mathbb{G}, y \in \mathbb{F}_q, H^{\dot{\boldsymbol{a}}} \in \mathbb{G}^\ell)$,
  - $P = \mathrm{COM}_{\overline{\boldsymbol{a}}}(\boldsymbol{x}), Q = \mathrm{COM}_{\overline{\boldsymbol{a}}}(\mathsf{rev}(L)), y = L(\boldsymbol{x})$,
  - $n = 2^\ell, \dot{\boldsymbol{a}} = (\dot{a}_1, \ldots, \dot{a}_n), \overline{\boldsymbol{a}} = \left(\prod_{i=1}^{\ell} \dot{a}_i^{b_i}\right)_{b_i \in \{0,1\}}$
- $\mathcal{P}$**'s input** : $(g^{\overline{\boldsymbol{a}}} \in \mathbb{G}^n, \boldsymbol{x} \in \mathbb{F}_q^n, L \in \mathcal{L}(\mathbb{F}_q^n))$

Protocol

1. Let us define $\mathbf{B} \in \mathbb{F}_q^n$ as $\mathbf{B} = \mathsf{rev}(L)$. Let $\boldsymbol{x}(U)$ be a polynomial of degree $(n-1)$ defined with coefficient vector $\boldsymbol{x} = (x_1, \ldots, x_n)$, such that $\boldsymbol{x}(U) = \sum_{i=0}^{n-1} x_{i+1} U^i$. Similarly, we define the polynomial $\mathbf{B}(U)$ of degree $(n-1)$ for the vector $\mathbf{B}$.
2. $\mathcal{P}$ defines a $(2n-2)$ degree polynomial $\boldsymbol{p}$ by
$$\boldsymbol{p}(U) = \boldsymbol{x}(U) \cdot \mathbf{B}(U) = \sum_{i,j} x_{i+1} B_{j+1} U^{i+j},$$
and parses the computed polynomial as
$$\boldsymbol{p}(U) = \boldsymbol{p}_L(U) \cdot U^{-1} + y \cdot U^{n-1} + \boldsymbol{p}_R(U) \cdot U^n,$$
where p_L is a polynomial of degree $(n-1)$ and p_R is a polynomial of degree $(n-2)$ (which is trivially extended to a vector of length n by appending 0 appropriately).
3. $\mathcal{P}$ computes $A_1 = \mathrm{COM}_{\overline{\boldsymbol{a}}}(\boldsymbol{p}_L)$ and $A_2 = \mathrm{COM}_{\overline{\boldsymbol{a}}}(\boldsymbol{p}_R)$, and sends A_1, A_2 to $\mathcal{V}$
4. $\mathcal{V}$ samples $c \leftarrow_R \mathbb{F}_q$ and sends c to $\mathcal{P}$
5. $\mathcal{P}$ computes the evaluations of the polynomials on the random challenge c as follows, and then sends z_1, z_2, z_3 and z_4 to $\mathcal{V}$: $z_1 = \boldsymbol{x}(c)$, $z_2 = \mathbf{B}(c)$, $z_3 = p_L(c)$, $z_4 = p_R(c)$.
6. $\mathcal{V}$ checks if the following relation holds, aborts if the check fails, and continues to the next step otherwise.
$$z_3 \cdot c^{-1} + y \cdot c^{n-1} + z_4 \cdot c^n = z_1 \cdot z_2$$
7. $\mathcal{V}$ samples $t \leftarrow_R \mathbb{F}_q$ and sends t to $\mathcal{P}$

Figure 3.3: Protocol Π_2 for relation $\mathcal{R}_{\text{CLF-rev}}$

8. $\mathcal{P}$ sets $\boldsymbol{w} = \boldsymbol{x} + t \cdot \mathbf{B} + t^2 \cdot \boldsymbol{p}_L + t^3 \cdot \boldsymbol{p}_R$ and sends $\boldsymbol{w}$ to $\mathcal{V}$

9. $\mathcal{P}$ and $\mathcal{V}$ both compute the following :

$$R = P \cdot Q^t \cdot A_1^{t^2} \cdot A_2^{t^3} \quad \text{and} \quad z = z_1 + t \cdot z_2 + t^2 \cdot z_3 + t^3 \cdot z_4$$

10. $\mathcal{V}$ outputs 1 if for $\boldsymbol{c^{n-1}} = (1, \ldots, c^{n-1}) \in \mathcal{PW}_n$ the following relation holds, and outputs 0 otherwise:

$$\text{COM}_{\overline{\boldsymbol{a}}}(\boldsymbol{w}) = R \quad \wedge \quad \langle \boldsymbol{w}, \boldsymbol{c^{n-1}} \rangle = z$$

Figure 3.3: Protocol Π_2 for relation $\mathcal{R}_{\text{CLF-rev}}$

matrix V described below is invertible.

$$V = \begin{pmatrix} 1 & 1 & 1 & 1 \\ t_1 & t_2 & t_3 & t_4 \\ t_1^2 & t_2^2 & t_3^2 & t_4^2 \\ t_1^3 & t_2^3 & t_3^3 & t_4^3 \end{pmatrix}$$

Let us denote $e_i, i \in \{1,2,3,4\}$ where j^{th} entry of e_i is 1 for $j = i$, and 0 otherwise. Let us consider a vector $\boldsymbol{\rho}^j = (\rho_1^j, \rho_2^j, \rho_3^j, \rho_4^j)$ for $j = 1,2,3,4$. Hence, we can compute $(\boldsymbol{\rho}^j)^T = V^{-1} e_j^T$. The computed $\rho_1^1, \rho_2^1, \rho_3^1, \rho_4^1$ satisfy $\sum_i \rho_i^1 = 1, \sum_i \rho_i^1 t_i = 0, \sum_i \rho_i^1 t_i^2 = 0$ and $\sum_i \rho_i^1 t_i^3 = 0$. Similarly, it holds for $j = 2,3,4$.

We define $\boldsymbol{r}_x$ to be the extracted value of $\boldsymbol{x}$ and compute it as $\boldsymbol{r}_x = \rho_1^1 \boldsymbol{w}_1 + \rho_2^1 \boldsymbol{w}_2 + \rho_3^1 \boldsymbol{w}_3 + \rho_4^1 \boldsymbol{w}_4$, given that $\text{COM}_{\overline{\boldsymbol{a}}}(\boldsymbol{w}_i) = P \cdot Q^{t_i} \cdot A_1^{t_i^2} \cdot A_2^{t_i^3}$ then we consider

$$\begin{aligned} \text{COM}_{\overline{\boldsymbol{a}}}(\boldsymbol{r}_x) &= g^{\langle \overline{\boldsymbol{a}}, \boldsymbol{r}_x \rangle} \\ &= g^{\langle \overline{\boldsymbol{a}}, \rho_1^1 \boldsymbol{w}_1 + \rho_2^1 \boldsymbol{w}_2 + \rho_3^1 \boldsymbol{w}_3 + \rho_4^1 \boldsymbol{w}_4 \rangle} \\ &= (PQ^{t_1} A_1^{t_1^2} A_2^{t_1^3})^{\rho_1} (PQ^{t_2} A_1^{t_2^2} A_2^{t_2^3})^{\rho_2} (PQ^{t_3} A_1^{t_3^2} A_2^{t_3^3})^{\rho_3} (PQ^{t_4} A_1^{t_4^2} A_2^{t_4^3})^{\rho_4} \\ &= P^{\sum_i \rho_i^1} Q^{\sum_i \rho_i^1 t_i} A_1^{\sum_i \rho_i^1 t_i^2} A_2^{\sum_i \rho_i^1 t_i^3} \\ &= P \end{aligned}$$

Hence, the extracted $\boldsymbol{r}_x$ is an opening of the commitment P.

Similarly, we define $\boldsymbol{r}_B, \boldsymbol{r}_{p_L}$ and $\boldsymbol{r}_{p_R}$ to be the extracted value of $\mathbf{B}, \boldsymbol{p}_L$ and $\boldsymbol{p}_R$,and compute them as $\boldsymbol{r}_B = \rho_1^2 \boldsymbol{w}_1 + \rho_2^2 \boldsymbol{w}_2 + \rho_3^2 \boldsymbol{w}_3 + \rho_4^2 \boldsymbol{w}_4$, $\boldsymbol{r}_{p_L} = \rho_1^3 \boldsymbol{w}_1 + \rho_2^3 \boldsymbol{w}_2 + \rho_3^3 \boldsymbol{w}_3 + \rho_4^3 \boldsymbol{w}_4$ and $\boldsymbol{r}_{p_R} = \rho_1^4 \boldsymbol{w}_1 + \rho_2^4 \boldsymbol{w}_2 + \rho_3^4 \boldsymbol{w}_3 + \rho_4^4 \boldsymbol{w}_4$. We also know that $\boldsymbol{w}(c) = z_1 + t_i \cdot z_2 + t_i^2 \cdot z_3 + t_i^3 \cdot z_4$ holds for

all $i = 1, 2, 3, 4$, since the transcripts are accepting transcripts.

$$\begin{aligned}
\boldsymbol{r}_x(c) &= \rho_1^1 \boldsymbol{w}_1(c) + \rho_2^1 \boldsymbol{w}_2(c) + \rho_3^1 \boldsymbol{w}_3(c) + \rho_4^1 \boldsymbol{w}_4(c) \\
&= \sum_i \rho_i^1 z_1 + \sum_i \rho_i^1 t_i \cdot z_2 + \sum_i \rho_i^1 t_i^2 \cdot z_3 + \sum_i \rho_i^1 t_i^3 \cdot z_4 \quad = z_1 \\
\boldsymbol{r}_B(c) &= \rho_1^2 \boldsymbol{w}_1(c) + \rho_2^2 \boldsymbol{w}_2(c) + \rho_3^2 \boldsymbol{w}_3(c) + \rho_4^2 \boldsymbol{w}_4(c) \\
&= \sum_i \rho_i^2 z_1 + \sum_i \rho_i^2 t_i \cdot z_2 + \sum_i \rho_i^2 t_i^2 \cdot z_3 + \sum_i \rho_i^2 t_i^3 \cdot z_4 \quad = z_2 \\
\boldsymbol{r}_{p_L}(c) &= \rho_1^3 \boldsymbol{w}_1(c) + \rho_2^3 \boldsymbol{w}_2(c) + \rho_3^3 \boldsymbol{w}_3(c) + \rho_4^3 \boldsymbol{w}_4(c) \\
&= \sum_i \rho_i^3 z_1 + \sum_i \rho_i^3 t_i \cdot z_2 + \sum_i \rho_i^3 t_i^2 \cdot z_3 + \sum_i \rho_i^3 t_i^3 \cdot z_4 \quad = z_3 \\
\boldsymbol{r}_{p_R}(c) &= \rho_1^4 \boldsymbol{w}_1(c) + \rho_2^4 \boldsymbol{w}_2(c) + \rho_3^4 \boldsymbol{w}_3(c) + \rho_4^4 \boldsymbol{w}_4(c) \\
&= \sum_i \rho_i^4 z_1 + \sum_i \rho_i^4 t_i \cdot z_2 + \sum_i \rho_i^4 t_i^2 \cdot z_3 + \sum_i \rho_i^4 t_i^3 \cdot z_4 \quad = z_4
\end{aligned}$$

Hence, we have that the extracted polynomials $\boldsymbol{r}_x, \boldsymbol{r}_B, \boldsymbol{r}_{p_L}$ and $\boldsymbol{r}_{p_R}$ satisfies the following constraint :

$$\begin{aligned}
& z_3 \cdot c^{-1} + y \cdot c^{n-1} + z_4 \cdot c^n = z_1 \cdot z_2 \qquad \text{(from accepting transcripts)} \\
\implies & \boldsymbol{r}_{p_L}(c) \cdot c^{-1} + y \cdot c^{n-1} + \boldsymbol{r}_{p_R}(c) \cdot c^n = \boldsymbol{r}_x(c) \cdot \boldsymbol{r}_B(c)
\end{aligned}$$

Now, we consider $2n$ such transcripts with different verifier challenges $c_i, i \in \{1, \ldots, 2n\}$, each with 4 different challenges $t_{ij}, j \in \{1, \ldots, 4\}, i \in \{1, \ldots, 2n\}$. Hence, the above constraint is satisfied by $2n-1$ random challenges, i.e. the polynomial evaluations are consistent with the constraint at $2n$ evaluation points, where the highest degree of the polynomial is $2n-1$. Hence, polynomials identically satisfy the constraints at all points, which implies that $\langle \boldsymbol{r}_x, \mathsf{rev}(\boldsymbol{r}_B) \rangle = y$ holds for the aforementioned polynomials.

□

Now we note that the last message $\boldsymbol{w}$ sent by the prover to the verifier in Π_2 (Fig 3.3) is a witness for the relation $\mathcal{R}$, where $\mathcal{R} = \{(P \in \mathbb{G}, L \in \mathcal{L}(\mathbb{F}_q^n), y \in \mathbb{F}_q; \boldsymbol{x} \in \mathbb{F}_q^n) : P = \mathrm{COM}_{\overline{\boldsymbol{a}}}(\boldsymbol{x}) \wedge L(\boldsymbol{x}) = y\}\}$, and the check computed by the verifier in step 10 of Π_2 corresponds to ensuring that $(R, \boldsymbol{c^{n-1}}, z; \boldsymbol{w}) \in \mathcal{R}$.

We state the following protocol for relation $\mathcal{R}$ which is the compressed proof of knowledge protocol stated in [8] with the following key differences: the linear form evaluation is checked in clear, commitment uses structured commitment key and commitment to the left and right half of the witness sent in the protocol being used to establish consistency with the commitment

to the whole key which is only possible due to the usage of structure in commitment key with keys being hidden in the exponent.

We note that even with the same protocol technique as [8] which inherently incurs linear computational complexity for verifier, we manage to retain a logarithmic computational complexity. This is due to the usage of structured commitment key, which does not require the verifier to compute a challenge dependent commitment key for the next iteration, and having a nicely-structured linear form which ensures that verifier can compute the challenge dependent linear form required for the next iteration efficiently. This suffices for our cause as we aim to use this protocol for providing proof of knowledge of the last message sent in step 8 of Π_2 such that it satisfies the verifier check in the step 10, which provides us a witness of the relation $\mathcal{R}$.

We note that the last message vector (polynomial) sent by the prover to the verifier of Π_2 is aimed to convince the verifier that the vector is consistent with opening of a group element computed by the verifier and evaluation of the polynomial at a random field element is consistent with a public field element computed by the verifier. We provide protocol Π_2' for this.

We treat the evaluation of the polynomial $\boldsymbol{w}$ at a fixed point, denoted by $\boldsymbol{w}(c)$, as an inner-product relation with a univariate polynomial, denoted by $\langle \boldsymbol{w}, \boldsymbol{c^{n-1}} \rangle$, where $\boldsymbol{c^{n-1}} = (1, c, \ldots, c^{n-1})$. Now, provided that the evaluation point is fixed at $c \in \mathbb{F}_q$, this inner product relation can be thought of as a linear form evaluation, where the public linear form $\boldsymbol{c^{n-1}}$ evaluation at a secret vector $\boldsymbol{w}$ is equal to the public value $z \in \mathbb{F}_q$. Now, we note that, the claim in step 10 is equivalent to providing a Proof of Knowledge of witness $\boldsymbol{w}$ in the following relation :

$$\mathcal{R} = \{(P \in \mathbb{G}, L \in \mathcal{L}(\mathbb{F}_q^n), y \in \mathbb{F}_q; \boldsymbol{x} \in \mathbb{F}_q^n) : P = \mathrm{COM}_{\overline{\boldsymbol{a}}}(\boldsymbol{x}) \ \wedge \ L(\boldsymbol{x}) = y\}\}$$

where we have that $(R, \boldsymbol{c^{n-1}}, z; \boldsymbol{w}) \in \mathcal{R}$.

Theorem 3.4 *Π_2' is a $(k_1, \ldots, k_\ell)$-move protocol for relation $(R, \boldsymbol{c^{n-1}}, z; \boldsymbol{w}) \in \mathcal{R}$, where $k_i = 3, \ \forall i \in [\ell]$. It is perfectly complete and computationally special sound.*

Proof: **Completeness** follows directly.

Special Soundness. We consider 3 accepting transcripts for one iteration of PoK Π_2' (where one iteration consists of steps 1-5, and step 6 follows by sending $\boldsymbol{x}', L'$ instead of providing a

Parameters

- **Common parameters** : $(R \in \mathbb{G}, L_c \in \mathbb{F}_q^n, z \in \mathbb{F}_q, H^{\dot{\boldsymbol{a}}} \in \mathbb{G}^\ell)$
  - $R = \mathrm{COM}_{\overline{\boldsymbol{a}}}(\boldsymbol{w}), L_c = \boldsymbol{c^{n-1}} = (1, c, \ldots, c^{n-1}), z = L_c(\boldsymbol{w})$,
  - $n = 2^\ell, \dot{\boldsymbol{a}} = (\dot{a}_1, \ldots, \dot{a}_n), \overline{\boldsymbol{a}} = \left(\prod_{i=1}^{\ell} \dot{a}_i^{b_i}\right)_{b_i \in \{0,1\}}$
- $\mathcal{P}$**'s input** : $(g^{\overline{\boldsymbol{a}}} \in \mathbb{G}^n, \boldsymbol{w} \in \mathbb{F}_q^n, L_c = \boldsymbol{c^{n-1}} \in \mathbb{F}_q^n)$

Protocol

1. $\mathcal{P}$ computes and sends A_1, A_2, z' to $\mathcal{V}$
   (a) $A_1 = \mathrm{COM}_{\overline{\boldsymbol{a}}_L}(\boldsymbol{w}_L)$
   (b) $A_2 = \mathrm{COM}_{\overline{\boldsymbol{a}}_L}(\boldsymbol{w}_R)$
   (c) $z' = \langle \boldsymbol{w}_L, (L_c)_L \rangle = \langle \boldsymbol{w}_L, \boldsymbol{c}^{\frac{\boldsymbol{n-1}}{\boldsymbol{2}}} \rangle$
2. $\mathcal{V}$ checks if
$$e\left(\frac{R}{A_1}, g\right) = e\left(A_2, g^{\dot{a}_\ell}\right)$$
   If the check fails, $\mathcal{V}$ aborts, otherwise $\mathcal{V}$ continues.
3. $\mathcal{V}$ samples $s \leftarrow_R \mathbb{F}_q$ and sends s to $\mathcal{P}$
4. $\mathcal{P}$ sets $\boldsymbol{w}' = \boldsymbol{w}_L + s \cdot \boldsymbol{w}_R, L_c' = s(L_c)_L + (L_c)_R = (s + c^{n/2})\boldsymbol{c^{n/2-1}}$ and implicitly sets $\dot{\boldsymbol{a}}' = (\dot{a}_1, \ldots, \dot{a}_{\ell-1})$ and $\overline{\boldsymbol{a}}' = \overline{\boldsymbol{a}}_L$
5. $\mathcal{P}$ and $\mathcal{V}$ both compute the following :
$$R' = A_1 A_2^s \text{ and } d = c^{n/2} \cdot z' + s \cdot z + s^2 \cdot c^{-n/2} \cdot (z - z')$$
6. If $\boldsymbol{w}' \notin \mathbb{F}_q^2$: $\mathcal{P}$ runs PoK Π_2' to prove knowledge of $\boldsymbol{w}', L_c'$ such that $\mathrm{COM}_{\overline{\boldsymbol{a}}'}(\boldsymbol{w}') = P'$ and $\langle L_c', \boldsymbol{x}' \rangle = d$.
   Hence, $\mathcal{P}$ and $\mathcal{V}$ run the protocol Π_2' with updated common parameters $(P', L_c', d, g^{\dot{\boldsymbol{a}}'})$ and prover's input $(g^{\overline{\boldsymbol{a}}'}, \boldsymbol{w}')$, for $(P', L_c', d; \boldsymbol{w}') \in \mathcal{R}$
7. If $\boldsymbol{w}' \in \mathbb{F}_q^2$:
   (a) $\mathcal{P}$ sends $\boldsymbol{w}', L_c'$ to $\mathcal{V}$
   (b) $\mathcal{V}$ outputs 1 if the following holds, and outputs 0 otherwise:
$$\mathrm{COM}_{\overline{\boldsymbol{a}}'}(\boldsymbol{w}') = R' \ \wedge \ \langle L_c', \boldsymbol{w}' \rangle = d$$

Figure 3.4: Protocol Π_2' for $(R, L_c, z; \boldsymbol{w}) \in \mathcal{R}$

PoK) as follows, where s_1, s_2, s_3 are all distinct challenges. :

$$(A_1, A_2, z', s_1, \boldsymbol{w}'_1)$$
$$(A_1, A_2, z', s_2, \boldsymbol{w}'_2)$$
$$(A_1, A_2, z', s_3, \boldsymbol{w}'_3)$$

Let us consider $\boldsymbol{z}_i = (s_i \boldsymbol{w}'_i \| \boldsymbol{w}'_i)$. We note that, as s_1, s_2 and s_3 are such that $s_i \neq s_j$ for all $(i \neq j)$ $i, j \in \{1, 2, 3\}$, the matrix V described below is invertible.

$$V = \begin{pmatrix} 1 & 1 & 1 \\ s_1 & s_2 & s_3 \\ s_1^2 & s_2^2 & s_3^2 \end{pmatrix}$$

Hence, we can compute $(a_1, a_2, a_3)^T = V^{-1}(0, 1, 0)^T$. The computed a_1, a_2, a_3 satisfy $\sum_i a_i = 0, \sum_i a_i c_i = 1$ and $\sum_i a_i c_i^2 = 0$.

Let us consider $\boldsymbol{x} = a_1 \boldsymbol{z}_1 + a_2 \boldsymbol{z}_2 + a_3 \boldsymbol{z}_3$, given that $\mathrm{COM}_{\overline{\boldsymbol{a}}'}\ (\boldsymbol{w}'_i) = A_1 A_2^{s_i}$ then we consider

$$\begin{aligned}
\mathrm{COM}_{\overline{\boldsymbol{a}}}\ (\boldsymbol{x}) &= g^{\langle \overline{\boldsymbol{a}}, \boldsymbol{x} \rangle} \\
&= g^{\langle \overline{\boldsymbol{a}}, a_1 \boldsymbol{z}_1 + a_2 \boldsymbol{z}_2 + a_3 \boldsymbol{z}_3 \rangle} \\
&= g^{\langle \overline{\boldsymbol{a}}_L \| \overline{\boldsymbol{a}}_R, a_1(s_1 \boldsymbol{w}'_1 \| \boldsymbol{w}'_1) + a_2(s_2 \boldsymbol{w}'_2 \| \boldsymbol{w}'_2) + a_3(s_3 \boldsymbol{w}'_3 \| \boldsymbol{w}'_3) \rangle} \\
&= g^{\langle \overline{\boldsymbol{a}}_L, a_1 s_1 \boldsymbol{w}'_1 + a_2 s_2 \boldsymbol{w}'_2 + a_s s_3 \boldsymbol{w}'_3 \rangle} g^{\langle \overline{\boldsymbol{a}}_R, a_1 \boldsymbol{w}'_1 + a_2 \boldsymbol{w}'_2 + a_3 \boldsymbol{w}'_3 \rangle} \\
&= g^{\langle \overline{\boldsymbol{a}}', a_1 s_1 \boldsymbol{w}'_1 + a_2 s_2 \boldsymbol{w}'_2 + a_3 s_3 \boldsymbol{w}'_3 \rangle} g^{\langle \dot{a}_\ell \overline{\boldsymbol{a}}', a_1 \boldsymbol{w}'_1 + a_2 \boldsymbol{w}'_2 + a_3 \boldsymbol{w}'_3 \rangle} \\
&= (A_1 A_2^{s_1})^{a_1 s_1} (A_1 A_2^{s_2})^{a_2 s_2} (A_1 A_2^{s_3})^{a_3 s_3} (A_1 A_2^{s_1})^{a_1 \dot{a}_\ell} (A_1 A_2^{s_2})^{a_2 \dot{a}_\ell} (A_1 A_2^{s_3})^{a_3 \dot{a}_\ell} \\
&= A_1^{(a_1 s_1 + a_2 s_2 + a_3 s_3) + \dot{a}_\ell (a_1 + a_2 + a_3)} A_2^{(a_1 s_1^2 + a_2 s_2^2 + a_3 s_3^2) + \dot{a}_\ell (a_1 s_1 + a_2 s_2 + a_3 s_3)} \\
&= A_1 A_2^{\dot{a}_\ell} \\
&= R \quad \text{(since we have } e\left(\frac{R}{A_1}, H\right) = e\left(A_2, H^{\dot{a}_\ell}\right) \text{ which ensures } R = A_1 A_2^{\dot{a}_\ell})
\end{aligned}$$

Hence, the extracted $\boldsymbol{x}$ is an opening of the commitment R. From the binding of the commitment scheme, we can ensure that $\boldsymbol{x} = \boldsymbol{w}$ except with negligible probability.

From the accepting transcripts, we have that $\langle (L'_c)_i, \boldsymbol{x}'_i \rangle = y_1 + s_i y + s_i^2 y_2$ for $i = 1, 2, 3$. Now, we consider the following :

$$\begin{aligned}
\langle (L'_c)_i, \boldsymbol{w}'_i \rangle &= \langle s_i (L_c)_L + (L_c)_R, \boldsymbol{w}_L + s_i \boldsymbol{w}_R \rangle && \forall i \in \{1, 2, 3\} \\
\implies y_1 + s_i y + s_i^2 y_2 &= \langle (L_c)_R, \boldsymbol{w}_L \rangle + s_i \langle (L_c), \boldsymbol{w} \rangle + s_i^2 \langle (L_c)_L, \boldsymbol{w}_R \rangle && \forall i \in \{1, 2, 3\}
\end{aligned}$$

$$\implies y = \langle L_c, \boldsymbol{w} \rangle$$

□

Theorem 3.5 $(\Pi_2)_c = \Pi_2' \circ \Pi_2$ *is a* $(2n, 4, k_1, \ldots, k_\ell)$*-move protocol for relation* $\mathcal{R}_{CLF\text{-}rev}$*, where* $k_i = 3,\ \forall i \in [\ell]$*. It is perfectly complete and computationally special sound. It incurs total communication of* $(2 + 2\log n)$ *group elements and* $6 + 2\log n$ *field elements.*

We note that $(\Pi_2)_c$ performs better than Π_1 for the relation $\mathcal{R}_{\text{CLF-rev}}$, however the preprocessing step needs a commitment to reverse of the linear forms. This is fine in our application to construct proofs for circuit satisfiability, since the commitments to the reverse of these linear forms is computed in the preprocessing phase. In case we only a commitment to the linear form, we can still use our protocol by having the prover send the commitment to the reversed linear form, together with a proof that it is indeed correct. This can be achieved by the observation that for $L \in \mathcal{L}(\mathbb{F}_q^n)$ considered as a polynomial, being evaluated at c has equal value as that of its reverse being evaluated at c^{-1} and the result being multiplied with c^{n-1}.

$$L(c) = c^{n-1} \cdot (\mathsf{rev}(L))\left(c^{-1}\right) \quad \iff \quad \langle L, \boldsymbol{c^{n-1}} \rangle = c^{n-1} \cdot \langle \mathsf{rev}(L), \boldsymbol{(c^{-1})^{n-1}} \rangle$$

Hence, if $P = \text{COM}_{\overline{\boldsymbol{a}}}(L)$ is computed in the preprocessing phase, then the prover can compute $Q = \text{COM}_{\overline{\boldsymbol{a}}}(\mathsf{rev}(L))$ and send Q along with the proof that opening of P evaluated at a random challenge c is c^{n-1} times Q evaluated at c^{-1}, at the onset of the protocol and proceed with $(\Pi_2)_c$. This gives us an overhead of 1 group element and 2 field elements. Finally, we define $\Pi_{2\text{-}\mathcal{R}} = (\Pi_2)_c \circ \Pi_0$ for relation $\mathcal{R}$, whose communication and computational complexity are dominated by that of $(\Pi_2)_c$.

3.4 Updatable SRS zkSNARK for Circuit Satisfiability

In this section, we construct a zkSNARK with updatable SRS for circuit satisfiability by reducing a statement about a circuit with respect to a public input to opening a linear form.

We take the approach of Attema et al. [8] to handle multiplication gates by linearizing them, but we need to employ some new ideas to keep the verifier succinct. We recall the technique for handling multiplication gates in the work of Attema et al. [8], where we have the left input wire values $\boldsymbol{w}_a$, the right input wire values $\boldsymbol{w}_b$ and the output wire values $\boldsymbol{w}_o$, secret shared via packed secret sharing, where the randomness is embedded in the constant term. Let f, g and h be the polynomials with the packed secret sharing of $\boldsymbol{w}_a$, $\boldsymbol{w}_b$ and $\boldsymbol{w}_o$ respectively,such that $f(X) \cdot g(X) = h(X)$. Attema et al. [8] handles it by sending a commitment to the wire

values in a long vector and opening them at a random point c, and then using Schwartz Zippel lemma to argue that the polynomials are identical if $f(c) \cdot g(c) = h(c)$ holds. However, the protocol to check $f(c) \cdot g(c) = h(c)$ renders the verifier linear, since the linear form for opening the polynomials at the random value is linear in the size of the witness. We circumvent this drawback of linear verification complexity from having to read the linear form by obtaining *commitments* to the linear form. The goal is to commit to linear forms required for openings of f, g, and h, and then invoke our succinct-verifier linear form protocol. We then proceed to prove that, given A, B and C as commitment to some secret vectors $\boldsymbol{a}, \boldsymbol{b}$ and $\boldsymbol{c}$ respectively from $\mathbb{F}_q^n$, the committed vectors satisfies the hadamard relation $\boldsymbol{a} \circ \boldsymbol{b} = \boldsymbol{c}$, i.e. $a_i b_i = c_i$ for all $i \in [n]$.

Following that, we show how to prove that given commitments A, B to two vectors $\boldsymbol{s}, \boldsymbol{r} \in \mathbb{F}_q^n$, they are some committed permutation of each other. Concretely, $\boldsymbol{s}, \boldsymbol{r} \in \mathbb{F}_q^n$ are such that $\boldsymbol{s} = \sigma(\boldsymbol{r})$ for some known permutation σ. Finally, we show how to put together these building blocks to construct a protocol for circuit satisfiability with logarithmic proof size and verification complexity.

3.4.1 Committing to a Linear Form for Multiplication Gates

Let $\rho_c = V^{-1}(1\ c\ c^2\ \cdots\ c^{n-1})^T$ for a random challenge c chosen by the verifier, where V is the Vandermonde matrix of the public evaluation points $\alpha_i, i \in [n]$. This enables us to compute $f(c) = f \cdot V \cdot \rho_c = (f(\alpha_1) \ldots f(\alpha_n)) \cdot \rho_c$ for a polynomial $f \in \mathbb{F}_{q_{\leq n}}[X]$. Now, instead of having the verifier compute a commitment to ρ_c (which would render it linear), we instead offload the computation of ρ_c to the prover and have the verifier *check* this computation in logarithmic time.

To check if a group element is indeed a commitment to ρ_c in logarithmic time, our key idea is to instantiate V as follows

$$V = \begin{pmatrix} 1 & \cdots & 1 & \cdots & 1 \\ 2 & \cdots & 2^i & \cdots & 2^n \\ 2^2 & \cdots & 2^{2i} & \cdots & 2^{2n} \\ \vdots & & \vdots & & \vdots \\ 2^{(n-1)} & \ldots & 2^{(n-1)i} & \ldots & 2^{(n-1)n} \end{pmatrix} \tag{3.1}$$

This enables us to reduce the verification of ρ_c to a series of n linear form checks, where the linear forms correspond to the rows of V. We then use the structure of V to express a random linear combination of the rows of V in a way that is easily checkable.

Parameters

- Parameters from preprocessing:
  - $X := \mathrm{COM}_{\overline{\boldsymbol{a}}}(\boldsymbol{x})$ where $\boldsymbol{x} := (2, \ldots, 2^n) \in \mathbb{F}_q^n$,
  - $\mathbf{1}_{\mathsf{Com}} := \mathrm{COM}_{\overline{\boldsymbol{a}}}(\mathbf{1})$ where $\mathbf{1} = (1, \ldots, 1) \in \mathbb{F}_q^n$
- Common input:
  - V is the Vandermonde matrix defined in equation 3.1.

Protocol

1. $\mathcal{V}$ samples $c \leftarrow_R \mathbb{F}_q$ and sends it to $\mathcal{P}$.
2. $\mathcal{P}$ sets ρ_c, where $\rho_c = (\rho_{c1}, \cdots, \rho_{cn}) = V^{-1}(1\ c\ c^2 \cdots c^{n-1})^T$ and ${\rho_c}' = (0, \rho_{c1}, \cdots, \rho_{cn})$, and computes $A = \mathrm{COM}_{\overline{\boldsymbol{a}}}(\mathsf{rev}(\rho_c)), A' = \mathrm{COM}_{\overline{\boldsymbol{a}}}(\mathsf{rev}({\rho_c}'))$.
3. $\mathcal{P}$ sends A, A' to $\mathcal{V}$.
4. $\mathcal{V}$ samples $t \leftarrow_R \mathbb{F}_q \setminus \{2^{-1}, \cdots, 2^{-i}, \cdots, 2^{-n}\}$ and sends t to $\mathcal{P}$.
5. $\mathcal{P}$ sets the j^{th} row of V as V_j, i.e. $V_j := (2^{j-1},\ 2^{2(j-1)}, \cdots, 2^{i(j-1)}, \cdots, 2^{n(j-1)})\ \forall j \in \{1, \cdots, n\}$, and computes $B := \mathrm{COM}_{\overline{\boldsymbol{a}}}(V(t))$, where $V(t) := (\boldsymbol{t^{n-1}})^T V = \sum_{j=0}^{n} t^{n-1} V_j$.
6. $\mathcal{P}$ sends B to $\mathcal{V}$.
7. The verifier samples $y \leftarrow_R \mathbb{F}_q \setminus \{1, 2^{-1}\}, d \leftarrow_R \mathbb{F}_q$ and sends y, d to $\mathcal{P}$.
8. $\mathcal{P}$ sets $\boldsymbol{z} = (2^i t - 1)_{i \in [n]}$, $\gamma = \langle \rho_c, \boldsymbol{d^{n-1}} \rangle$ and sends γ to $\mathcal{V}$.
9. $\mathcal{P}$ and $\mathcal{V}$ independently computes $Z = \mathrm{COM}_{\overline{\boldsymbol{a}}}(\boldsymbol{y^{n-1}} \circ \boldsymbol{z}) = X^t \cdot (\mathbf{1}_{\mathsf{Com}})^{-1}$, $\alpha = 2^n t^n \frac{(2^n y)^n - 1}{2^n y - 1} - \frac{y^n - 1}{y - 1}$ and $\beta = \frac{(ct)^n - 1}{ct - 1}$.
10. $\mathcal{P}$ and $\mathcal{V}$ interact to prove the following relation:
    (a) run $(\Pi_2)_c$ for $(B, A, \beta; (V(t))^T, \rho_c) \in \mathcal{R}_{\text{CLF-rev}}$
    (b) run $\Pi_{2\text{-}\mathcal{R}}$ for $(A, \boldsymbol{d^{n-1}}, \gamma; \rho_c), (A', \boldsymbol{d^{n-1}}, d\gamma; {\rho_c}') \in \mathcal{R}$
    (c) run $\Pi_{2\text{-}\mathcal{R}}$ for $(B, \boldsymbol{y^{n-1}} \circ \boldsymbol{z}, \alpha; V(t)) \in \mathcal{R}$

Figure 3.5: Protocol $\Pi_{\mathsf{com\text{-}mult}}$ for obtaining commitment to linear form for multiplication gates.

Let us define the relation $\mathcal{R}_{\mathsf{com\text{-}mult}}$ as follows:

$$\mathcal{R}_{\mathsf{com\text{-}mult}} = \{ \quad (A_1 \in \mathbb{G}, A_2 \in \mathbb{G}, V \in \mathbb{F}_q^{n\times n}, \boldsymbol{c^{n-1}} \in \mathbb{F}_q^n; \rho_c, \rho_c') : \\ \boldsymbol{c^{n-1}} = (1 \ c \ \ldots \ c^{n-1}), \rho_c = V^{-1}\boldsymbol{c^{n-1}}, \rho_c' = 0 \| \rho_c, \\ A_1 = \mathrm{COM}_{\overline{a}}(\rho_c), A_2 = \mathrm{COM}_{\overline{a}}(\rho_c')\} \tag{3.2}$$

This relation captures obtaining commitment to a linear form consisting of public linear combination coefficients to obtain the evaluation of an n-degree polynomial at a randomly chosen point c by the verifier. We note that ρ_c' here denotes the vector (linear form) ρ_c shifted to the right by one, which is used in the protocols in subsequent sections to open polynomials defined by evaluations at the vector $(1, c_1, \ldots, c_n)$ as $(1, c_1, \ldots, c_{n-1})$ and $(c_1, \ldots, c_n)$ with the same vector description. That is, given a vector $(1, c_1, \ldots, c_n)$, we can use our relation to capture linear forms to evaluate polynomials defined by both $(1, c_1, \ldots, c_{n-1})$ and $(c_1, \ldots, c_n)$ simultaneously. Figure 3.5 presents the protocol $\Pi_{\mathsf{com\text{-}mult}}$ for the relation $\mathcal{R}_{\mathsf{com\text{-}mult}}$.

Note that it is easy to add zero checks to the protocol in Figure 3.5 to get a commitment to $\rho_{\mathsf{n}} \| \mathbf{0} \in \mathbb{F}_q^{n'}$. Let $n' > n$ be the length of the commitment key. The verifier samples a challenge $t \leftarrow_R \mathbb{F}_q$ and checks that the commitment P_n claimed to be to $\rho_{\mathsf{n}} \in \mathbb{F}_q^n$ satisfies $(P_n, \mathbf{0}^n \| \boldsymbol{t}^{n'-n}, 0; \rho_{\mathsf{n}} \| \mathbf{0}) \in \mathcal{R}$. Moreover, it is also easy to get a commitment to the reverse of ρ_{n}. For this, the verifier samples a challenge u and asks the prover to make a claim of the form $\langle \rho_{\mathsf{n}}, \boldsymbol{u^n} \rangle = v$. It then checks if the commitments P_n, Q_n claimed to be to be to ρ_{n} and its reverse satisfy $(P_n, \boldsymbol{u^n}, v), (Q_n, \mathsf{rev}(\boldsymbol{u^n}), v) \in \mathcal{R}_{\text{CLF-rev}}$.

Theorem 3.6 *$\Pi_{\mathsf{com\text{-}mult}}$ is a $(7, 4, k_1, \ldots, k_\ell)$-move protocol for relation $\mathcal{R}_{\mathsf{com\text{-}mult}}$ (equation 3.2). It is perfectly complete and computationally special sound.*

Proof: Completeness. The prover $\mathcal{P}$ computes $V(t) = \boldsymbol{t^{n-1}}V = \sum_{j=1}^{n} t^{j-1}V_j$ and $B = \mathrm{COM}_{\overline{a}}(V(t))$, where V_j is the j^{th} row of V (equation 3.1) and $t \neq 1, \ldots, 2^{-i}, \ldots, 2^{-(n-1)}$. Then,

$$V(t) = \left(\tfrac{(2t)^n - 1}{2t-1}, \ \ldots \ , \tfrac{(2^i t)^n - 1}{2^i t - 1}, \ \ldots \ , \tfrac{(2^n t)^n - 1}{2^n t - 1} \right)$$

Let us define $\boldsymbol{z} = (2t - 1, \ldots, \ 2^i t - 1, \ldots, 2^n t - 1)$, then we have $V(t) \circ \boldsymbol{z} = ((2t)^n - 1, \ldots, (2^i t)^n - 1, \ldots, (2^n t)^n - 1)$, which ensures that, for any $y \in \mathbb{F}_q$, we have $\langle \boldsymbol{y^{n-1}} \circ \boldsymbol{z}, V(t) \rangle = \langle \boldsymbol{y^{n-1}}, V(t) \circ \boldsymbol{z} \rangle = 2^n t^n \frac{(2^n y)^n - 1}{2^n y - 1} - \frac{y^n - 1}{y - 1} = \alpha$. This ensures that $(B, \boldsymbol{y^{n-1}} \circ \boldsymbol{z}, \alpha; V(t)) \in \mathcal{R}$. Also, $\langle (V(t))^T, \rho_c \rangle = \langle V^T \boldsymbol{t^{n-1}}, V^{-1} \boldsymbol{c^{n-1}} \rangle = \langle \boldsymbol{t^{n-1}}, \boldsymbol{c^{n-1}} \rangle = \frac{(ct)^n - 1}{ct - 1} = \beta$, ensures that $(B, A, \beta; (V(t))^T, \rho_c) \in \mathcal{R}_{\text{CLF-rev}}$.

Since $\langle \rho_c', \boldsymbol{d^{n-1}} \rangle = d\langle \rho_c, \boldsymbol{d^{n-1}} \rangle$, we have that $(A, \boldsymbol{d^{n-1}}, \gamma; \rho_c) \in \mathcal{R}$, and $(A', \boldsymbol{d^{n-1}}, d\gamma; \rho_c') \in \mathcal{R}$ for $\gamma = \langle \rho_c, \boldsymbol{d^{n-1}} \rangle \in \mathbb{F}_q$.

Special Soundness. Our extractor uses the extractor for $(\Pi_2)_c$ and $\Pi_{2\text{-}\mathcal{R}}$, invoked in step 10 of $\Pi_{\mathsf{com\text{-}mult}}$, as a subroutine. Given $(2n, 4, 3, \ldots, 3)$ tree of accepting transcripts for $(\Pi_2)_c$ invoked for relation $(B, A, \beta; V(t), \rho_c) \in \mathcal{R}_{\text{CLF-rev}}$, we run the extractor for $(\Pi_2)_c$ to obtain openings of B, A and the binding of the commitments and soundness of the protocol ensures that the extracted openings are $V(t)$ and ρ_c such that $\langle \rho_c, (V(t))^T \rangle = \beta$. Similarly, given $(2, 2n, 4, 3, \ldots, 3)$ tree of accepting transcripts for $\Pi_{2\text{-}\mathcal{R}}$ invoked for relations $(A, \boldsymbol{d^{n-1}}, \gamma; \rho_c)$, $(A', \boldsymbol{d^{n-1}}, d\gamma; \rho_c')$, and $(B, \boldsymbol{y^{n-1}} \circ \boldsymbol{z}, \alpha; V(t)) \in \mathcal{R}$, we run the extractor for $(\Pi_2)_c$ to obtain openings of A, A', B and the binding of the commitments and soundness of the protocol ensures that the extracted openings are ρ_c, ${\rho_c}'$ and $V(t)$ such that $\langle \boldsymbol{d^{n-1}}, \rho_c \rangle = \gamma$, $\langle \boldsymbol{d^{n-1}}, \rho_c' \rangle = d\gamma$ and $\langle \boldsymbol{y^{n-1}} \circ z, V(t) \rangle = \alpha$. Hence, we get that the following relations hold:

1. $\langle \boldsymbol{y^{n-1}}, V(t) \circ \boldsymbol{z} \rangle = \langle \boldsymbol{y^{n-1}} \circ \boldsymbol{z}, V(t) \rangle = \alpha = 2^n t^n \frac{(2^n y)^n - 1}{2^n y - 1} - \frac{y^n - 1}{y - 1}$
   $\implies \langle \boldsymbol{y^{n-1}}, V(t) \circ \boldsymbol{z} \rangle = 2^n t^n \frac{(2^n y)^n - 1}{2^n y - 1} - \frac{y^n - 1}{y - 1}$.

2. $\langle V\rho_c, \boldsymbol{t^{n-1}} \rangle = (V\rho_c)^T \boldsymbol{t^{n-1}} = {\rho_c}^T (V^T \boldsymbol{t^{n-1}}) = {\rho_c}^T V(t)^T = \langle \rho_c, V(t)^T \rangle = \frac{(ct)^n - 1}{ct - 1} \implies V\rho_c = \boldsymbol{c^{n-1}}$

3. $\langle \rho_c', \boldsymbol{d^{n-1}} \rangle = d\gamma = d\langle \rho_c, \boldsymbol{d^{n-1}} \rangle$
   $\implies \langle \rho_c', \boldsymbol{d^{n-1}} \rangle = d\langle \rho_c, \boldsymbol{d^{n-1}} \rangle$

The last relation provides us, that given two polynomials ρ_c and ${\rho_c}'$ defined by their vector of coefficients, we have $\rho_c(d) = d \cdot {\rho_c}'(d)$ for some $d \in \mathbb{F}_q$ sampled completely at random. Hence, given accepting transcripts with n-different challenges y, d and t each, following relations hold from Schwartz Zippel Lemma:

1. $V(t) \circ \boldsymbol{z} = ((2t)^n - 1, \ldots, (2^i t)^n - 1, \ldots, (2^n t)^n - 1)$, where $\boldsymbol{z} = (2^i t - 1)_{i \in [n]}$
   $\implies V(t) = \sum_{j=0}^{n} t^{n-1} V_j$, where $V_j = (2^{j-1},\ 2^{2(j-1)}, \cdots, 2^{i(j-1)}, \cdots, 2^{n(j-1)})$

2. $V\rho_c = \boldsymbol{c^{n-1}}$

3. ${\rho_c}' = 0 \| \rho_c = (0, \rho_{c1}, \ldots, \rho_{cn})$ when $\rho_c = (\rho_{c1}, \ldots, \rho_{cn})$

which ensures that our extracted vectors are such that ρ_c is a linear form whose commitment is provided, and ρ_c contains the coefficient linear combinations for obtaining evaluation at c. Also, ${\rho_c}'$ is a linear form which is ρ_c shifted by one place to the right.

□

3.4.2 Hadamard Product Argument

Let $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{F}_q^n$, recall that the hadamard product is defined as $\boldsymbol{a} \circ \boldsymbol{b} = (a_1 b_1, \ldots, a_n b_n) \in \mathbb{F}_q^n$. Our goal is to prove knowledge of three vectors that satisfy the hadamard product relation, given succinct commitment to the vectors.

Concretely, given three vectors $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c} \in \mathbb{F}_q^n$ such that $\boldsymbol{a} \circ \boldsymbol{b} = \boldsymbol{c}$, we define $p_{\boldsymbol{a}}(X), p_{\boldsymbol{b}}(X), p_{\boldsymbol{c}}(X) \in \mathbb{F}_q^n[X]$ such that $p_{\boldsymbol{a}}(2^i) = a_i, p_{\boldsymbol{b}}(2^i) = b_i$ for all $i \in [n]$ and $p_{\boldsymbol{c}}(X) := p_{\boldsymbol{a}}(X) \cdot p_{\boldsymbol{b}}(X)$. We define $h_{(\boldsymbol{c})} = (p_{\boldsymbol{c}}(2^{n+1}), \ldots, p_{\boldsymbol{c}}(2^{2n-1}))$. The protocol proceeds as follows. The prover computes commitments A, B, C to the vectors $\boldsymbol{a}, \boldsymbol{b}$ and $\boldsymbol{c}' := \boldsymbol{c} \| h_{(\boldsymbol{c})}$ respectively. The verifier then samples a challenge z, and the prover responds with commitments P_n, P_{2n} to the reverse of ρ_{n} and ρ_{2n}, where ρ_{n} and ρ_{2n} are defined as $\rho_{\mathsf{n}} = V^{-1}(1\ z\ z^2\ \cdots\ z^{n-1})^T$, $\rho_{\mathsf{2n}} = V^{-1}(1\ z\ z^2\ \cdots\ z^{2n-2})^T$. Then the prover opens the polynomial evaluations of $p_{\boldsymbol{a}}(X), p_{\boldsymbol{b}}(X), p_{\boldsymbol{c}}(X)$ at a random point chosen by the verifier, using the commitments to the vectors and the linear forms.

The hadamard relation $\mathcal{R}_{\mathsf{had}}$ with suitable modification to incorporate the commitments to the vectors is defined below, and the protocol Π_{had} presents the protocol for relation $\mathcal{R}_{\mathsf{had}}$. Note that to ensure zero-knowledge property of the protocol Π_{had}, to prove $\boldsymbol{a} \circ \boldsymbol{b} = \boldsymbol{c}$, we invoke the protocol for $(A, B, C; \boldsymbol{a} \| d, \boldsymbol{b} \| e, \boldsymbol{c} \| de) \in \mathcal{R}_{\mathsf{had}}$ where $d, e \leftarrow_R \mathbb{F}_q$.

$$\begin{aligned}\mathcal{R}_{\mathsf{had}} = \{(A \in \mathbb{G}, B \in \mathbb{G}, C \in \mathbb{G}; \boldsymbol{a} \in \mathbb{F}_q^n, \boldsymbol{b} \in \mathbb{F}_q^n, \boldsymbol{c} \in \mathbb{F}_q^n) :\\ A = \mathrm{COM}_{\overline{\boldsymbol{a}}}(\boldsymbol{a}), B = \mathrm{COM}_{\overline{\boldsymbol{a}}}(\boldsymbol{b}), \boldsymbol{c}' = \boldsymbol{c} \| h_{(\boldsymbol{c})}, C = \mathrm{COM}_{\overline{\boldsymbol{a}}}(\boldsymbol{c}')\}\end{aligned}$$

Theorem 3.7 *Π_{had} is a protocol for $\mathcal{R}_{\mathsf{had}}$. It is perfectly complete, special honest-verifier zero-knowledge and computationally special sound.*

Proof: The proof of completeness is straightforward to argue.

Special Soundness. Let $\boldsymbol{y}, \boldsymbol{y}'$ be defined as $\boldsymbol{y} = \boldsymbol{u} + r\boldsymbol{a} + r^2\boldsymbol{b}$, $\boldsymbol{y}' = \boldsymbol{u}' + r\boldsymbol{c}'$, $q = v_1 + rw_1 + r^2 w_2$ and $q' = v_2 + rw_1 w_2$. Now we note that the we invoke $(\Pi_2)_c$ for $(UA^r B^{r^2}, P_n, v_1 + rw_1 + r^2 w_2; \boldsymbol{u} + r\boldsymbol{a} + r^2\boldsymbol{b}, \rho_{\mathsf{n}})$ and $(U'C^r, P_{2n}, v_2 + rw_1 w_2; \boldsymbol{u}' + r\boldsymbol{c}', \rho_{\mathsf{2n}}) \in \mathcal{R}_{\text{CLF-rev}}$. Our extractor invokes the extractor for $(\Pi_2)_c$ to extract $\boldsymbol{y}, \boldsymbol{y}', \rho_{\mathsf{n}}$ and ρ_{2n} such that $\langle \rho_{\mathsf{n}}, \boldsymbol{y} \rangle = q, \langle \rho_{\mathsf{2n}}, \boldsymbol{y}' \rangle = q'$. Extracting $\boldsymbol{y}_1, \boldsymbol{y}'_1, \boldsymbol{y}_2, \boldsymbol{y}'_2, \boldsymbol{y}_3, \boldsymbol{y}'_3$ for three distinct challenges e_1, e_2, e_3, our extractor additionally computes $\boldsymbol{u}, \boldsymbol{u}', \boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}'$ such that $\langle \rho_{\mathsf{n}}, \boldsymbol{u} \rangle = v_1, \langle \rho_{\mathsf{2n}}, \boldsymbol{u}' \rangle = v_2, \langle \rho_{\mathsf{n}}, \boldsymbol{a} \rangle = w_1, \langle \rho_{\mathsf{n}}, \boldsymbol{b} \rangle = w_2, \langle \rho_{\mathsf{2n}}, \boldsymbol{c}' \rangle = w_1 w_2$. Note that the binding of the commitment ensures that the correct $\boldsymbol{u}, \boldsymbol{u}', \boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}', \rho_{\mathsf{n}}$ and ρ_{2n} have been extracted.

Our extractor again invokes the knowledge extractor of $\Pi_{\mathsf{com\text{-}mult}}$ to extract $\rho_{\mathsf{n}}, \rho_{\mathsf{2n}}$ such that $\rho_{\mathsf{n}} = V^{-1}(1\ z\ z^2\ \cdots\ z^n)^T$ and $\rho_{\mathsf{2n}} = V^{-1}(1\ z\ z^2\ \cdots\ z^{2n+1})^T$, and the binding of the commitment

Parameters

- Common input:
  - V is the Vandermonde matrix defined in equation 3.1.
  - $A = \mathrm{COM}_{\overline{\boldsymbol{a}}}(\boldsymbol{a}), B = \mathrm{COM}_{\overline{\boldsymbol{a}}}(\boldsymbol{b}), C = \mathrm{COM}_{\overline{\boldsymbol{a}}}(\boldsymbol{c}')$, such that $\boldsymbol{c}' = \boldsymbol{c} \| h_{(\boldsymbol{a} \circ \boldsymbol{b})}$
- $\mathcal{P}$'s input: $\boldsymbol{a} = \boldsymbol{a}^* \| d, \boldsymbol{b} = \boldsymbol{b}^* \| e, \boldsymbol{c} = \boldsymbol{c}^* \| de$ such that $\boldsymbol{a}^* \circ \boldsymbol{b}^* = \boldsymbol{c}^*$ and $d, e \leftarrow_R \mathbb{F}_q$

Protocol

1. $\mathcal{P}$ computes the polynomials $p_{\boldsymbol{a}}, p_{\boldsymbol{b}} \in \mathbb{F}_q^n[X]$ as $p_{\boldsymbol{a}}(2^i) := a_i, p_{\boldsymbol{b}}(2^i) := b_i \; \forall i \in [n]$. It defines $p_{\boldsymbol{c}}(X) := p_{\boldsymbol{a}}(X) \cdot p_{\boldsymbol{b}}(X)$.
2. $\mathcal{P}$ samples $\boldsymbol{u} \leftarrow_R \mathbb{F}_q^n, \boldsymbol{u}' \leftarrow_R \mathbb{F}_q^{2n-1}$ and defines $p_{\boldsymbol{u}} \in \mathbb{F}_q^n[X], p_{\boldsymbol{u}'}\mathbb{F}_q^{2n}[X]$ as $p_{\boldsymbol{u}}(2^i) := u_i \; \forall i \in [n]$, $p_{\boldsymbol{u}'}(2^i) := u'_i \; \forall i \in [2n]$. $\mathcal{P}$ computes $U = \mathrm{COM}_{\overline{\boldsymbol{a}}}(\boldsymbol{u}), U' = \mathrm{COM}_{\overline{\boldsymbol{a}}}(\boldsymbol{u}')$ and sends U, U' to $\mathcal{V}$.
3. $\mathcal{V}$ samples $z \leftarrow_R \mathbb{F}_q$ and sends z to $\mathcal{P}$.
4. Define $\rho_{\mathsf{n}} = V^{-1}(1 \; z \; z^2 \; \cdots \; z^n)^T$ and $\rho_{\mathsf{2n}} = V^{-1}(1 \; z \; z^2 \; \cdots \; z^{2n-2})^T$. $\mathcal{P}$ and $\mathcal{V}$ run $\Pi_{\mathsf{com\text{-}mult}}$ to obtain commitments P_n, P_{2n} to the reverse of $\rho_{\mathsf{n}}, \rho_{\mathsf{2n}}$.
5. $\mathcal{P}$ sets w_1, w_2 as $w_1 = p_{\boldsymbol{a}}(z)$, and $w_2 = p_{\boldsymbol{b}}(z)$. $\mathcal{P}$ also sets v_1, v_2 as $v_1 = p_{\boldsymbol{u}}(z), v_2 = p_{\boldsymbol{u}'}(z)$.
6. $\mathcal{P}$ sends w_1, w_2, v_1, v_2 to $\mathcal{V}$.
7. $\mathcal{V}$ samples $r \leftarrow_R \mathbb{F}_q$ and sends r to $\mathcal{P}$.
8. $\mathcal{P}$ and $\mathcal{V}$ independently computes $Y = UA^rB^{r^2}$, $Y' = U'C^r$, $\boldsymbol{y} = \boldsymbol{u} + r\boldsymbol{a} + r^2\boldsymbol{b}$, $\boldsymbol{y}' = \boldsymbol{u}' + r\boldsymbol{c}$, $q = v_1 + rw_1 + r^2w_2$ and $q' = v_2 + rw_1w_2$.
9. $\mathcal{P}$ and $\mathcal{V}$ run $(\Pi_2)_c$ for
   (a) $(Y, P_n, q;\ \boldsymbol{y}, \rho_{\mathsf{n}}) \in \mathcal{R}_{\text{CLF-rev}}$.
   (b) $(Y', P_{2n}, q';\ \boldsymbol{y}', \rho_{\mathsf{2n}}) \in \mathcal{R}_{\text{CLF-rev}}$.

Figure 3.6: Protocol Π_{had} for $\mathcal{R}_{\mathsf{had}}$

ensures the consistency of the extracted openings. Hence, it follows that the extracted witnesses satisfies $p_{\boldsymbol{u}}(z) = v_1, p_{\boldsymbol{u}'}(z) = v_2, p_{\boldsymbol{a}}(z) = w_1, p_{\boldsymbol{b}}(z) = w_2, p_{\boldsymbol{c}}(z) = w_1 w_2$.

We can extract $\boldsymbol{a}_i, \boldsymbol{b}_i, \boldsymbol{c}'_i$ for $i \in [2n+2]$ distinct challenges z_i such that $p_{\boldsymbol{c}'_i}(z_i) = p_{\boldsymbol{a}_i}(z_i) p_{\boldsymbol{b}_i}(z_i)$. If any of the extracted $\boldsymbol{a}_i, \boldsymbol{b}_i, \boldsymbol{c}'_i$ differ, we will have broken binding and so we have that except with negligible probability, all the extracted $\boldsymbol{a}_i, \boldsymbol{b}_i, \boldsymbol{c}'_i$ are identical. Thus, we have extracted $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}'$ that satisfy $p_{\boldsymbol{c}'}(z) = p_{\boldsymbol{a}}(z) p_{\boldsymbol{b}}(z)$ for $2n+2$ distinct z. This allows us to conclude (from the Schwartz-Zippel Lemma) that $p_{\boldsymbol{c}'}(X) = p_{\boldsymbol{a}}(X) p_{\boldsymbol{b}}(X)$.

Zero-Knowledge. Given access to verifier's randomness z, r, the simulator $\mathcal{S}_{\mathsf{had}}$ proceeds as follows:

1. $\mathcal{S}_{\mathsf{had}}$ samples $w_1, w_2 \leftarrow_R \mathbb{F}_q$, $\boldsymbol{y} \leftarrow_R \mathbb{F}_q^{n+1}, \boldsymbol{y}' \leftarrow_R \mathbb{F}_q^{2n+1}$, and sends $w_1, w_2, U = \frac{\mathrm{COM}_{\overline{\boldsymbol{a}}}(\boldsymbol{y})}{A^r B^{r^2}}, U' = \frac{\mathrm{COM}_{\overline{\boldsymbol{a}}}(\boldsymbol{y}')}{C^r}, v_1 = \langle \rho_{\mathsf{n}}, \boldsymbol{y} \rangle - r w_1 - r^2 w_2, v_2 = \langle \rho_{2\mathsf{n}}, \boldsymbol{y}' \rangle - r w_1 w_2$ to $\mathcal{V}$.

2. $\mathcal{S}_{\mathsf{had}}$ sets $Y = \mathrm{COM}_{\overline{\boldsymbol{a}}}(\boldsymbol{y}), Y' = \mathrm{COM}_{\overline{\boldsymbol{a}}}(\boldsymbol{y}'), q = \langle \rho_{\mathsf{n}}, \boldsymbol{y} \rangle, q' = \langle \rho_{2\mathsf{n}}, \boldsymbol{y}' \rangle$.

3. $\mathcal{S}_{\mathsf{had}}$ then honestly executes $(\Pi_2)_c$ to show that $(Y, P_n, q; \boldsymbol{y}, \rho_{\mathsf{n}})$, $(Y', P_{2n}, q'; \boldsymbol{y}', \rho_{2\mathsf{n}}) \in \mathcal{R}_{\text{CLF-rev}}$ in Step 9.

We now argue that the distribution of the simulated transcript is indistinguishable from the transcript obtained from real protocol execution. Since the underlying vector $\boldsymbol{y}, \boldsymbol{y}', w_1$ and w_2 are sampled uniformly at random, the computed U, U', v_1, v_2 subject to the constraints $\mathrm{COM}_{\overline{\boldsymbol{a}}}(\boldsymbol{y}) = U A^r B^{r^2}$, $\mathrm{COM}_{\overline{\boldsymbol{a}}}(\boldsymbol{y}') = U' C^r$, $q = v_1 + r w_1 + r^2 w_2$, and $q' = v_2 + r w_1 w_2$ outlined in Step 9, where $r \leftarrow_R \mathbb{F}_q$, are distributed uniformly at random in the transcript. The remaining computations are performed honestly and are thus indistinguishable from an actual protocol execution.

□

3.4.3 Permutation Argument

Our starting point is the Bayer-Groth protocol [17] for the permutation argument. Let $\mathsf{PERM}_n = \{f : f : [n] \to [n]$ such that f is a permutation$\}$ and $\sigma \in \mathsf{PERM}_n$. For two vectors $\boldsymbol{r} = (r_1, \ldots, r_n) \in \mathbb{F}_q^n$ and $\boldsymbol{s} = (s_1, \ldots, s_n) \in \mathbb{F}_q^n$, we aim to prove that $\sigma(\boldsymbol{r}) = \boldsymbol{s}$ for some publicly known σ. To prove the same, we leverage the technique introduced by Bayer and Groth of proving $\prod_{i=1}^n (r_i + i\beta + \gamma) = \prod_{i=1}^n (s_i + \sigma(i)\beta + \gamma)$ for verifier's choice of $\beta, \gamma \in \mathbb{F}_q$ sampled uniformly at random.

The proof is instantiated by having the verifier choose two challenges $\beta, \gamma \in \mathbb{F}_q$ and the prover constructing two vectors $\boldsymbol{a} = (a_1, \ldots, a_n)$ and $\boldsymbol{b} = (b_1, \ldots, b_n)$ defined as $a_i = r_i + i\beta + \gamma$ and $b_i = s_i + \sigma(i)\beta + \gamma$ for all $i = 1, \ldots, n$, and providing a proof that $\prod_{i=1}^n a_i = \prod_{i=1}^n b_i$ holds.

The proof for $\prod_{i=1}^{n} a_i = \prod_{i=1}^{n} b_i$ proceeds by constructing two vectors $\boldsymbol{c}', \boldsymbol{d}' \in \mathbb{F}_q^{n+1}$ such that $c'_0 = 1, d'_0 = 1$ and $c'_j := \prod_{i=1}^{j}(r_i + i\beta + \gamma), d'_j := \prod_{i=1}^{j}(s_i + \sigma(i)\beta + \gamma)$, for all $j \in [n]$. Now we consider two circuits consisting of n multiplication gates, first circuit with vector of left inputs $\boldsymbol{a} = (a_1, \ldots, a_n)$, vector of right inputs $\boldsymbol{e} = (e_1, \ldots, e_n) = (1, c_1, \ldots, c_{n-1})$ and vector of outputs $\boldsymbol{c} = (c_1, \ldots, c_n)$, and second circuit with left input $\boldsymbol{b} = (b_1, \ldots, b_n)$, vector of right inputs $\boldsymbol{f} = (f_1, \ldots, f_n) = (1, d_1, \ldots, d_{n-1})$ and vector of outputs $\boldsymbol{d} = (d_1, \ldots, d_n)$. Our idea now is to check the hadamard product relations $\boldsymbol{a} \circ \boldsymbol{e} = \boldsymbol{c}$ and $\boldsymbol{b} \circ \boldsymbol{f} = \boldsymbol{d}$ by leveraging the shifted structure of the vectors in the hadamard products; and using protocol $\Pi_{\mathsf{com\text{-}mult}}$ yielding a succinct verifier permutation argument.

We consider the following relation $\mathcal{R}_{\mathsf{perm}}$ for the permutation argument.

$$\begin{aligned}\mathcal{R}_{\mathsf{perm}} = \{\ (R, S, P; \boldsymbol{r}, \boldsymbol{s}, \sigma) : R = \mathrm{COM}_{\overline{\boldsymbol{a}}}\ (\boldsymbol{r}), S = \mathrm{COM}_{\overline{\boldsymbol{a}}}\ (\boldsymbol{s}), P = \mathrm{COM}_{\overline{\boldsymbol{a}}}\ (\sigma(I)),\\ I = (1, \ldots, n), \boldsymbol{s} = \sigma(\boldsymbol{r})\}\end{aligned}$$

We present the protocol Π_{perm} for the same in Fig 3.7. We define ρ_{n}, ρ_{2n}, δ_{n} and δ_{2n} as $\rho_{\mathsf{n}} = V^{-1}(1\ z\ z^2 \ldots z^{n-1})$, $\rho_{\mathsf{2n}} = V^{-1}(1\ z\ z^2 \ldots z^{2n-1})$, $\delta_{\mathsf{n}} = V^{-1}(1\ w\ w^2 \ldots w^{n-1})$ and $\delta_{\mathsf{2n}} = V^{-1}(1\ w\ w^2 \ldots w^{2n-1})$ where V is a Vandermonde matrix defined by the public evaluation points. We recall that the linear forms $\rho_{\mathsf{n}}, \rho_{\mathsf{2n}}$ are for computing evaluation at a random point z, and the linear forms $\delta_{\mathsf{n}}, \delta_{\mathsf{2n}}$ are for computing evaluation at a random point w. We note that we can batch the invocations of $(\Pi_2)_c$ for $\mathcal{R}_{\text{CLF-rev}}$ in each of the steps (a),(b) and (c) in Step 11 of Π_{perm} using the techniques of Attema et al. [8].

Theorem 3.8 *Π_{perm} is a protocol for $\mathcal{R}_{\mathsf{perm}}$. It is perfectly complete, special honest-verifier zero-knowledge and computationally special sound.*

Proof: The proof of completeness is straightforward to argue.

Special Soundness. We rely on the sub-routine $\Pi_{2\text{-}\mathcal{R}}$ invoked in step 10 to obtain openings of A, B, C and D provided appropriate $(2, 2n, 4, 3, \ldots, 3)$ tree of accepting transcripts. We denote $\boldsymbol{f}_a$, $\boldsymbol{f}_b, \boldsymbol{f}_c$ and $\boldsymbol{f}_d$ to denote the openings of A, B, C and D respectively which satisfies the following constraints, $\langle \rho_{\mathsf{n}}, \boldsymbol{f}_a \rangle \cdot \langle \rho_{\mathsf{n}}, \boldsymbol{f}_c \rangle = \langle \rho'_{\mathsf{2n}}, \boldsymbol{f}_c \rangle$ and $\langle \delta_{\mathsf{n}}, \boldsymbol{f}_b \rangle \cdot \langle \delta_{\mathsf{n}}, \boldsymbol{f}_d \rangle = \langle \delta'_{\mathsf{2n}}, \boldsymbol{f}_d \rangle$.

The binding of the commitment scheme ensures that $\boldsymbol{f}_a = \boldsymbol{a}$, $\boldsymbol{f}_b = \boldsymbol{b}$, $\boldsymbol{f}_c = \boldsymbol{c}''$ and $\boldsymbol{f}_d = \boldsymbol{d}''$. The constraints of the linear forms from $(A, P_n, z_1; \boldsymbol{a}, \rho_{\mathsf{n}})$,$(C, P_n, z_2; \boldsymbol{c}'', \rho_{\mathsf{n}})$, $(B, Q_n, w_1; \boldsymbol{b}, \delta_{\mathsf{n}})$, $(D, Q_n, w_2; \boldsymbol{d}'', \delta_{\mathsf{n}})$, $(C, P_{2n}, z_1 z_2; \boldsymbol{c}'', \rho'_{\mathsf{2n}})$, $(D, Q_{2n}, w_1 w_2; \boldsymbol{d}'', \delta'_{\mathsf{2n}}) \in \mathcal{R}_{\text{CLF}}$ and the verifier check in step 9 further ensure that the polynomials satisfy $p_{\boldsymbol{c}} = p_{\boldsymbol{a}} \cdot p_{\boldsymbol{e}}$ at a random point, i.e. the polynomials are identical with high probability via Schwartz-Zippel Lemma. Hence, the evaluations at each point satisfy the relation with high probability, which gives us $c_i = a_i \cdot e_i =$

Parameters

- Parameters from preprocessing:
  - $P = \mathrm{COM}_{\overline{\boldsymbol{a}}}(\sigma(I))$, $P' = \mathrm{COM}_{\overline{\boldsymbol{a}}}(I)$ for $I = (1, \ldots, n)$
  - $\mathsf{left} = \mathrm{COM}_{\overline{\boldsymbol{a}}}(\mathsf{rev}(1\|\mathbf{0}\|\mathbf{0}))$ and $\mathsf{right} = \mathrm{COM}_{\overline{\boldsymbol{a}}}(\mathsf{rev}(\mathbf{0}\|1\|\mathbf{0}))$ for linear forms $(1\|\mathbf{0}\|\mathbf{0})$ and $(\mathbf{0}\|1\|\mathbf{0})$, where $\mathbf{0} = (0, \ldots, 0) \in \mathbb{F}_q^n$
  - $T = \mathrm{COM}_{\overline{\boldsymbol{a}}}(\mathbf{1})$, $\mathbf{1} = (1, \ldots, 1) \in \mathbb{F}_q^n$
- Common Input: $R = \mathrm{COM}_{\overline{\boldsymbol{a}}}(\boldsymbol{r}), S = \mathrm{COM}_{\overline{\boldsymbol{a}}}(\boldsymbol{s})$
- $\mathcal{P}$'s input : $(\boldsymbol{r}, \boldsymbol{s}, \sigma, g^{\overline{\boldsymbol{a}}})$

Protocol

1. $\mathcal{V}$ samples $\beta, \gamma \leftarrow_R \mathbb{F}_q$ and sends β, γ to $\mathcal{P}$.
2. $\mathcal{P}$ computes $x := \prod_{i=1}^{n}(r_i + i\beta + \gamma)$ and sends x to $\mathcal{V}$.
3. $\mathcal{P}$ computes the vectors $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{F}_q^n$ such that $a_i = r_i + i\beta + \gamma$ and $b_i = s_i + \sigma(i)\beta + \gamma$ for all $i \in [n]$. $\mathcal{P}$ additionally computes $\boldsymbol{c'}, \boldsymbol{d'} \in \mathbb{F}_q^{n+1}$ such that $c'_1 = 1, d'_1 = 1$ and $c'_j := \prod_{i=1}^{j-1} a_i, d'_j := \prod_{i=1}^{j-1} b_i$, for all $j \in [n+1] \setminus \{1\}$, and defines $\boldsymbol{c}, \boldsymbol{d}, \boldsymbol{e}, \boldsymbol{f} \in \mathbb{F}_q^n$ such that $c_i = c'_{i+1}, d_i = d'_{i+1}, e_i = c'_i, f_i = d'_i$ for all $i \in [n]$, i.e. $c_j := \prod_{i=1}^{j} a_i, d_j := \prod_{i=1}^{j} b_i$, for all $j \in [n]$, and $e_1 = 1, f_1 = 1$ and $e_j := \prod_{i=1}^{j-1} a_i, d_j := \prod_{i=1}^{j-1} b_i$, for all $j \in [n]$.
4. $\mathcal{P}$ computes the polynomials $p_{\boldsymbol{a}}, p_{\boldsymbol{e}}$ and $p_{\boldsymbol{c}}$ as $p_{\boldsymbol{a}}(2^i) := a_i, p_{\boldsymbol{e}}(2^i) := e_i$ and $p_{\boldsymbol{c}} := p_{\boldsymbol{a}} \cdot p_{\boldsymbol{e}}$, and similarly computes $p_{\boldsymbol{b}}, p_{\boldsymbol{f}}$ and $p_{\boldsymbol{d}}$ as $p_{\boldsymbol{b}}(2^i) := b_i, p_{\boldsymbol{f}}(2^i) := f_i$ and $p_{\boldsymbol{d}} := p_{\boldsymbol{b}} \cdot p_{\boldsymbol{f}}$.
5. $\mathcal{P}$ and $\mathcal{V}$ independently computes $A = R(P')^{\beta}T^{\gamma}$ and $B = SP^{\beta}T^{\gamma}$.
6. $\mathcal{P}$ computes $\boldsymbol{c''} = \boldsymbol{c'}\|(p_{\boldsymbol{c}}(2^{n+1}), \ldots, p_{\boldsymbol{c}}(2^{2n}))$ and $\boldsymbol{d''} = \boldsymbol{d'}\|(p_{\boldsymbol{d}}(2^{n+1}), \ldots, p_{\boldsymbol{d}}(2^{2n}))$, $C = \mathrm{COM}_{\overline{\boldsymbol{a}}}(\boldsymbol{c''})$ and $D = \mathrm{COM}_{\overline{\boldsymbol{a}}}(\boldsymbol{d''})$ and sends C, D to $\mathcal{V}$.
7. $\mathcal{V}$ samples $z, w \leftarrow_R \mathbb{F}_q$ and sends z, w to $\mathcal{P}$.
8. $\mathcal{P}$ computes $\rho_{\mathsf{n}} = V^{-1}(1 \;\; z \;\; z^2 \ldots z^{n-1})$, $\rho_{\mathsf{2n}} = V^{-1}(1 \;\; z \;\; z^2 \ldots z^{2n-2})$, $\delta_{\mathsf{n}} = V^{-1}(1 \; w \; w^2 \ldots w^{n-1})$ and $\delta_{\mathsf{2n}} = V^{-1}(1 \; w \; w^2 \ldots w^{2n-2})$.
9. $\mathcal{P}$ and $\mathcal{V}$ run $\Pi_{\mathsf{com\text{-}mult}}$ to obtain commitments P_n, P_{2n}, Q_n and Q_{2n} to the reverse of ρ_{n},ρ'_{2n},δ_{n} and δ'_{2n} where $\rho'_{\mathsf{2n}} = 0\|\rho_{\mathsf{2n}}$ and $\delta'_{\mathsf{2n}} = 0\|\delta_{\mathsf{2n}}$.

Figure 3.7: Protocol Π_{perm} for Permutation Argument

10. $\mathcal{P}$ sets z_1, z_2, w_1, and w_2 as $z_1 = p_{\boldsymbol{a}}(z)$, $z_2 = p_{\boldsymbol{e}}(z)$, $w_1 = p_{\boldsymbol{b}}(w)$, and $w_2 = p_{\boldsymbol{f}}(w)$.

11. $\mathcal{P}$ and $\mathcal{V}$ run $(\Pi_2)_c$ to prove the following:

    (a) $(C, \mathsf{left}, 1; \boldsymbol{c}'', (1\|\boldsymbol{0}\|\boldsymbol{0}))$, $(C, \mathsf{right}, x; \boldsymbol{c}'', (\boldsymbol{0}\|1\|\boldsymbol{0}))$, $(D, \mathsf{left}, 1; \boldsymbol{d}'', (1\|\boldsymbol{0}\|\boldsymbol{0}))$, $(D, \mathsf{right}, x; \boldsymbol{d}'', (\boldsymbol{0}\|1\|\boldsymbol{0})) \in \mathcal{R}_{\text{CLF-rev}}$

    (b) $(A, P_n, z_1; \boldsymbol{a}, \rho_{\mathsf{n}})$,$(C, P_n, z_2; \boldsymbol{c}'', \rho_{\mathsf{n}})$, $(B, Q_n, w_1; \boldsymbol{b}, \delta_{\mathsf{n}})$, $(D, Q_n, w_2; \boldsymbol{d}'', \delta_{\mathsf{n}}) \in \mathcal{R}_{\text{CLF-rev}}$

    (c) $(C, P_{2n}, z_1 z_2; \boldsymbol{c}'', \rho'_{2\mathsf{n}})$, $(D, Q_{2n}, w_1 w_2; \boldsymbol{d}'', \delta'_{2\mathsf{n}}) \in \mathcal{R}_{\text{CLF-rev}}$.

Figure 3.7: Protocol Π_{perm} for Permutation Argument

$a_i \cdot c_{i-1}$ (considering $c_0 = 1$). We get $\boldsymbol{a}(z) \circ \boldsymbol{c}_{\mathsf{left}}(z) = \boldsymbol{c}_{\mathsf{right}}(z)$ and $\boldsymbol{b}(w) \circ \boldsymbol{d}_{\mathsf{left}}(w) = \boldsymbol{d}_{\mathsf{right}}(w)$ where $\boldsymbol{c}_{\mathsf{left}} = (1, c, \ldots, c^{n-1}) = \boldsymbol{e}$, $\boldsymbol{c}_{\mathsf{right}} = (c, \ldots, c^n) = \boldsymbol{c}$, $\boldsymbol{d}_{\mathsf{left}} = (1, d, \ldots, d^{n-1})$, and $\boldsymbol{d}_{\mathsf{right}} = (d, \ldots, d^n)$, that is $a_i \cdot c''_{i-1} = c''_i$ and $b_i \cdot d''_{i-1} = d''_i$ holds for all $i \in [n]$. Additionally, the constraints of the linear forms from $(C, \mathsf{left}, 1; \boldsymbol{c}'', (1\|\boldsymbol{0}\|\boldsymbol{0}))$, $(C, \mathsf{right}, x; \boldsymbol{c}'', (\boldsymbol{0}\|1\|\boldsymbol{0}))$, $(D, \mathsf{left}, 1; \boldsymbol{d}'', (1\|\boldsymbol{0}\|\boldsymbol{0}))$, $(D, \mathsf{right}, x; \boldsymbol{d}'', (\boldsymbol{0}\|1\|\boldsymbol{0})) \in \mathcal{R}_{\text{CLF}}$ ensures that the $c''_1 = 1$ and $d''_1 = 1$. The constraints of the linear forms from $(C, \mathsf{right}, x; \boldsymbol{c}'')$, $(D, \mathsf{right}, x; \boldsymbol{d}'') \in \mathcal{R}$ ensures that the $c''_n = d''_n = x$ which provides us $\prod_{i=1}^n a_i = x = \prod_{i=1}^n b_i$.

We define $\boldsymbol{r}, \boldsymbol{s} \in \mathbb{F}_q^n$ as $r_i = a_i - i\beta - \gamma, s_i = b_i - \sigma(i)\beta - \gamma$ for all $i \in [n]$, for the public permutation σ. We rely on the [17] to ensure that given $\prod_{i=1}^n a_i = x = \prod_{i=1}^n b_i$ holds which implies $\prod_{i=1}^n (r_i + i\beta + \gamma) = \prod_{i=1}^n (s_i + \sigma(i)\beta + \gamma)$ holds, we can ensure that the computed (extracted) vectors $\boldsymbol{r}, \boldsymbol{s}$ are such that $\sigma(\boldsymbol{r}) = \boldsymbol{s}$. The argument follows provided we have $O(n)$ accepting transcripts of Π_{perm}. □

3.4.4 zkSNARK for Circuit SAT

Given an upper bound on the circuit size n, the universal updatable SRS is generated by running COM .*Setup* to commit to $2n+2$-length vectors to obtain the commitment key $(g^{\overline{\boldsymbol{a}}}, H^{\dot{\boldsymbol{a}}})$. Here $g^{\overline{\boldsymbol{a}}}$ is the proving key and $H^{\dot{\boldsymbol{a}}}$ is the verification key. Since the SRS is universal, we need a circuit-dependent setup phase so the verifier will read the circuit only once. We omit the description of algorithms for updating and verifying the SRS since this corresponds to updating and verifying the commitment key, and is the same as in Daza et al. [52]. We note that the circuit-specific preprocessing material can be deterministically computed from the universal SRS and the circuit description, without any secrets.

We describe the protocol as an interactive public-coin argument. The final zkSNARK construction is in the Random Oracle model using the Fiat-Shamir heuristic.

Preprocessing. We use the preprocessing phase used by Daza et al. [52] to obtain a commit-

ment to the linear gates. They establish the existence of a circuit preprocessing methodology that effectively imposes constraints on the fan-in and fan-out of each gate in the circuit to a maximum value of M, which only incurs a linear expansion in the size of the circuit.

Let $\chi_1, \ldots, \chi_\nu$ be the public inputs of the circuit. Let m be the number of multiplication gates in the circuit. We then require a commitment key of size $n = 2m+2$. Let x_i^L, x_i^R, x_i^O denote the left input, right input and output of the i^{th} multiplication gate. Let $\boldsymbol{x}^L = (x_i^L)_{i\in[m]}, \boldsymbol{x}^R = (x_i^R)_{i\in[m]}, \boldsymbol{x}^O = (x_i^O)_{i\in[m]}$. Then $\boldsymbol{x}^L \circ \boldsymbol{x}^R = \boldsymbol{x}^O$. Additionally, there exist vectors $\boldsymbol{w}_i^L, \boldsymbol{w}_i^R \in \mathbb{F}_q^m$ with at most M non-zero entries such that $\langle \boldsymbol{w}_i^L, \boldsymbol{x}^O\rangle + x_i^L = \chi_i, \forall i \in [\nu]$, $\langle \boldsymbol{w}_i^L, \boldsymbol{x}^O\rangle = x_i^L, \forall i \in \{\nu+1, \ldots, m\}$ and $\langle \boldsymbol{w}_i^R, \boldsymbol{x}^O\rangle = x_i^R \ \forall i \in [m]$. Let $W^L, W^R \in \mathbb{F}_q^{m\times m}$ be matrices with their i^{th} rows equal to $\boldsymbol{w}_i^L$ and $\boldsymbol{w}_i^R$ respectively. Then W^L and W^R have $\leq M$ entries in each row and each column. The following applies to W^k for $k \in \{L, R\}$. W^k can be written as the sum of M permutation matrices, i.e. $W^k = \sum_{i=1}^M W_i^k$, where each W_i^k is a permutation matrix.

In addition to the preprocessing of [52], additional preprocessing material is generated that is required by our sub-protocols, $\Pi_{\mathsf{com\text{-}mult}}$ Π_{had}, and Π_{perm}. The verifier obtains commitments to W_i^k, I and $\sigma(I)$, where $I = (1, \ldots, n)$, W_i^k and $\sigma_i^k : [n] \to [n]$ are as defined above. The verifier also obtains commitments to $\mathbf{1}$, $\mathbf{2}_{[n]}$, $(\mathbf{0}\|1\|\mathbf{0})$, $(\mathbf{0}\|\mathbf{0}\|1)$ where $\mathbf{0} = (0, \ldots, 0) \in \mathbb{F}_q^n$, $\mathbf{1} = (1, \ldots, 1) \in \mathbb{F}_q^n$, $\mathbf{2}_{[n]} = (2, \ldots, 2^n) \in \mathbb{F}_q^n$.

Protocol Overview. Post circuit preprocessing, our circuit is now fully defined by $\tilde{\boldsymbol{w}}_i^k, \sigma_i^k$, where $\tilde{\boldsymbol{w}}_i^k$ is the vector containing the non-zero entry (if there is one) in each column of W_i^k and $\sigma_i^k : [n] \to [n]$ is the permutation that takes as input a column number j and outputs the row to which the j^{th} entry of $\boldsymbol{w}_i$ belongs. Our goal is to get a commitment to a random linear combination of the rows of W^k, i.e. a commitment to $W^k(c) = \sum_{i=1}^M \tilde{\boldsymbol{w}}_i^k \circ \ \sigma_i^k(\boldsymbol{c^m})$. To do this, we first demand commitments to $\sigma_i^k(\boldsymbol{c^m})$ from the prover, for a random challenge c chosen by the verifier. We can check that these commitments are honestly generated using Π_{perm}. We additionally ask the prover to provide us with commitments to $\tilde{\boldsymbol{w}}_i^k \circ \ \sigma_i^k(\boldsymbol{c^m})$ and a proof $h_{(\tilde{\boldsymbol{w}}_i^k \circ \ \sigma_i(\boldsymbol{c^m}))}$ that attests to the correct computation of a Hadamard product. To check this Hadamard product, we deploy our Π_{had} protocol. Since $\tilde{\boldsymbol{w}}_i^k, \sigma_i^k$ are public, Π_{had} can be invoked without requiring zero-knowledge.

The above protocol allows us to get commitments to $W^L(c)$ and $W^R(c)$, but to show that the constraints of the circuit are satisfied, we need to prove that $\langle W^L(c) + uW^R(c), \boldsymbol{x}^O\rangle = \langle \boldsymbol{c^m}, \boldsymbol{x}^L\rangle - \sum_{i=1}^\nu c^{i-1}\chi_i + u\langle \boldsymbol{c^m}, \boldsymbol{x}^R\rangle = \langle \boldsymbol{c^m}, \boldsymbol{x}^L + u\boldsymbol{x}^R\rangle - \sum_{i=1}^\nu c^{i-1}\chi_i$ for $u \leftarrow_R \mathbb{F}_q$. We cannot test for equality directly since that would require the prover to send out linear combinations of $\boldsymbol{x}^L, \boldsymbol{x}^R$ and $\boldsymbol{x}^O$, violating zero-knowledge. Set $K = \sum_{i=1}^\nu c^{i-1}\chi_i, L_1 = W^L(c) + uW^R(c), L_2 = \boldsymbol{c^m}, \boldsymbol{y}_1 = \boldsymbol{x}^O$ and $\boldsymbol{y}_2 = \boldsymbol{x}^L + u\boldsymbol{x}^R$. Let $\tilde{L}_2$ be the $m-1$ vector comprising of the first $m-1$ elements of L_2. Let $(L_2)_m$ be the last element of L_2. The above constraint can then be

written as $\langle L_1, \boldsymbol{y}_1 \rangle = \langle L_2, \boldsymbol{y}_2 \rangle - K$. To prove this in zero-knowledge, we have the prover sample $\boldsymbol{r}_1 \leftarrow \mathbb{F}_q^m, \tilde{\boldsymbol{r}}_2 \leftarrow \mathbb{F}_q^{m-1}$ and set $\boldsymbol{r}_2 = \tilde{\boldsymbol{r}}_2 || (\langle L_1, \boldsymbol{r}_1 \rangle - \langle \tilde{L}_2, \tilde{\boldsymbol{r}}_2 \rangle)(L_2)_m^{-1}$. This ensures that $\langle L_1, \boldsymbol{r}_1 \rangle = \langle L_2, \boldsymbol{r}_2 \rangle$. The protocol now proceeds as follows: the verifier samples a challenge z and the prover proves that $\langle L_1, z\boldsymbol{y}_1 + \boldsymbol{r}_1 \rangle = \langle L_2, z\boldsymbol{y}_2 + \boldsymbol{r}_2 \rangle - zK$. We can directly test for equality here since the prover now needs to reveal $\langle L_1, z\boldsymbol{y}_1 + \boldsymbol{r}_1 \rangle$, which is a random value that reveals nothing about the input.

This allows us to conclude that the commitments to $\boldsymbol{x}^L, \boldsymbol{x}^R$ and $\boldsymbol{x}^O$ satisfy the linear combination constraints imposed by the circuit. Testing for multiplication, i.e. checking if $\boldsymbol{x}^L \circ \boldsymbol{x}^R = \boldsymbol{x}^O$ can be done by invoking our protocol Π_{had} by adding randomness to the input vectors in order to preserve zero-knowledge.

Since we reduce circuit satisfiability to opening a series of committed linear forms on committed vectors, we can optimize by batching the opening of several linear forms together. Consider two instances (P_1, Q_1, y_1) and (P_2, Q_2, y_2) claimed by the prover to belong to $\mathcal{R}_{\text{CLF-rev}}$. To prove this, we modify the protocol in Figure 3 as follows: let $\boldsymbol{x}_1, \boldsymbol{x}_2$ be the vectors to which P_1 and P_2 are commitments. Let B_1, B_2 be the linear forms to which Q_1 and Q_2 are commitments. We first demand that the prover send us commitments to $p_{L,1}, p_{R,1}, p_{L,2}$ and $p_{R,2}$ as it would in the original protocol. We then ask the prover to make claims about $\boldsymbol{x}_1(c), \boldsymbol{B}_1(c), \boldsymbol{p}_{L,1}(c), \boldsymbol{p}_{R,1}(c)$ and $\boldsymbol{x}_2(c), \boldsymbol{B}_2(c), \boldsymbol{p}_{L,2}(c), \boldsymbol{p}_{R,2}(c)$ with respect to the *same* challenge c. This allows us to combine the prover's claims to open $\boldsymbol{c^{n-1}}$ on a *single* vector given by $\boldsymbol{x}_1 + t\boldsymbol{B}_1 + t^2\boldsymbol{p}_{L,1} + t^3\boldsymbol{p}_{R,1} + t^4\boldsymbol{x}_2 + t^5\boldsymbol{B}_2 + t^6\boldsymbol{p}_{L,2} + t^7\boldsymbol{p}_{R,2}$ for a random challenge t. Thus, we can open $\mathcal{O}(M)$ linear forms while incurring the communication overhead of opening a single linear form. We present the complete protocol Π_{csat} in Fig. 3.8.

Theorem 3.9 Π_{csat} *is a public-coin, Honest Verifier Zero-Knowledge Argument of Knowledge for CSAT with* $\mathcal{O}(\log m)$ *round complexity,* $\mathcal{O}_\lambda(m)$ *prover complexity, and* $\mathcal{O}_\lambda(\log m)$ *communication and verification complexity, where* m *is the number of multiplication gates in the preprocessed circuit.*

Proof: **Completeness** follows directly.

Special soundness. We invoke the extractor of $(\Pi_2)_c$ to extract witnesses for $2m+2$ distinct ts in Step 10(a). Given these witnesses, we can either conclude that Σ_0 is a commitment to $\boldsymbol{c^m}||\boldsymbol{0}$ or break binding of the commitment scheme. Further, we can invoke the extractor of Π_{perm} to extract $\forall k \in \{L, R\}\ \forall i \in [M]$ vectors $\boldsymbol{u}_i^k$ such that $\boldsymbol{u}_i^k = \sigma_i^k(\boldsymbol{c^m})$. We invoke the extractor of Π_{had} to extract $\forall k \in \{L, R\}\ \forall i \in [M]$ vectors $\boldsymbol{t}_i^k = \sigma_i^k(\boldsymbol{c^m}) \circ \boldsymbol{w}_i^k || h_{(\sigma_i^k(\boldsymbol{c^m}) \circ \boldsymbol{w}_i^k)}$. In steps 10(d) and 10(e), given witnesses for $2m + 2$ distinct challenges, we either extract vectors L_1', L_1'' such that $L_1' = \sum_{i=1}^M \boldsymbol{t}_i^L + u\boldsymbol{t}_i^R$ and $L_1'' = (\boldsymbol{0} || ((L_1')_{m-i+1})_{i \in [M]}) = \mathsf{rev}(L_1)$ or break binding.

Universal updatable SRS: $(g^{\overline{\boldsymbol{a}}}, H^{\dot{\boldsymbol{a}}})$
Preprocessing Compute commitments to the following circuit-dependent vectors:

- $S_i^k = \mathrm{COM}_{\overline{\boldsymbol{a}}}\ (\tilde{\boldsymbol{w}}_i^k \| \mathbf{0})\ \forall i \in [M]\ k \in \{L, R\}$
- $P_i^k = \mathrm{COM}_{\overline{\boldsymbol{a}}}\ (\sigma_i^k \| \mathbf{0})\ \forall i \in [M]\ k \in \{L, R\}$
- $\mathtt{ones} = \mathrm{COM}_{\overline{\boldsymbol{a}}}\ (\mathbf{1}^{\boldsymbol{m}} \| \mathbf{0})$, $P_0 = \mathrm{COM}_{\overline{\boldsymbol{a}}}\ (\boldsymbol{m} \| \mathbf{0})$, where $\boldsymbol{m} = (1, 2, \cdots, m)$.

Input

- Public input $\chi_1, \ldots, \chi_n$
- $\mathcal{P}$'s input is the satisfying assignment $\boldsymbol{x}^L, \boldsymbol{x}^R, \boldsymbol{x}^O \in \mathbb{F}_q^m$. $\mathcal{P}$ samples $d, e \leftarrow_R \mathbb{F}_q$ and defines $\tilde{\boldsymbol{x}}^L = \boldsymbol{x}^L \| d$, $\tilde{\boldsymbol{x}}^R = \boldsymbol{x}^R \| e$, $\tilde{\boldsymbol{x}}^O = \boldsymbol{x}^O \| de \| h_{(\tilde{\boldsymbol{x}}^L \circ \tilde{\boldsymbol{x}}^R)} \in \mathbb{F}_q^{2m+1}$
- V's inputs are the commitments $X^k = \mathrm{COM}_{\overline{\boldsymbol{a}}}\ (\tilde{\boldsymbol{x}}^k; r^k)$ for $k \in \{L, R\}, X^O = \mathrm{COM}_{\overline{\boldsymbol{a}}}\ (\tilde{\boldsymbol{x}}^O; r^O)$ with $r^L, r^R, r^O \leftarrow_R \mathbb{F}_q$

Protocol

1. $\mathcal{V}$ sends $c \leftarrow_R \mathbb{F}_q$ to $\mathcal{P}$.
2. $\mathcal{P}$ computes for $k \in \{L, R\}$:
   (a) $\Sigma_0 = \mathrm{COM}_{\overline{\boldsymbol{a}}}\ (\boldsymbol{c}^{\boldsymbol{m}} \| \mathbf{0})$
   (b) $\Sigma_i^k = \mathrm{COM}_{\overline{\boldsymbol{a}}}\ (\sigma_i^k(\boldsymbol{c}^{\boldsymbol{m}}) \| \mathbf{0})\ \forall i \in [M]$
   (c) $W_i^k = \mathrm{COM}_{\overline{\boldsymbol{a}}}\ (\boldsymbol{w}_i^k \circ \sigma_i^k(\boldsymbol{c}^{\boldsymbol{m}}) \| h_{(\boldsymbol{w}_i^k \circ \sigma_i^k(\boldsymbol{c}^{\boldsymbol{m}}))})\ \forall i \in [M]$

   $\mathcal{P}$ sends all the computed commitments to $\mathcal{V}$.
3. $\mathcal{V}$ sends $u \leftarrow_R \mathbb{F}_q$ to $\mathcal{P}$.
4. $\mathcal{P}$ samples $\boldsymbol{r}, \tilde{\boldsymbol{r}} \leftarrow_R \mathbb{F}_q^m, s, \tilde{s} \leftarrow_R \mathbb{F}_q$ such that $\langle W^L(c) + uW^R(c), \boldsymbol{r} \rangle = \langle \boldsymbol{c}^{\boldsymbol{m}}, \tilde{\boldsymbol{r}} \rangle$ and sends $R = \mathrm{COM}_{\overline{\boldsymbol{a}}}\ (\boldsymbol{r}; s)$ and $\tilde{R} = \mathrm{COM}_{\overline{\boldsymbol{a}}}\ (\tilde{\boldsymbol{r}}; \tilde{s})$ to $\mathcal{V}$.
5. $\mathcal{V}$ sends $z \leftarrow_R \mathbb{F}_q$ to $\mathcal{P}$.
6. $\mathcal{P}$ sets $L_1 = W^L(c) + uW^R(c), L_2 = \boldsymbol{c}^{\boldsymbol{m}}, K = \sum_{i=1}^{\nu} c^{i-1} \chi_i$ and sends $v_1 = \langle L_1, z\boldsymbol{x}^O + \boldsymbol{r} \rangle = \langle L_2, z\boldsymbol{x}^L + zu\boldsymbol{x}^R + \tilde{\boldsymbol{r}} \rangle - zK$ and $\mathcal{L}_1^{rev} = \mathrm{COM}_{\overline{\boldsymbol{a}}}\ (\mathsf{rev}(L_1))$ to $\mathcal{V}$.
7. $\mathcal{V}$ sends $t \neq c^{-1}, t', t_h, t_{perm} \leftarrow_R \mathbb{F}_q$ to $\mathcal{P}$.

Figure 3.8: Protocol Π_{csat} for Circuit Satisfiability

8. $\mathcal{P}$ and $\mathcal{V}$ invoke $\Pi_{\mathsf{com\text{-}mult}}$ to obtain commitments to the reverse of ρ_{m+1}, ρ_{2m+2} with respect to the challenge t_h and to ρ_m, ρ_{2m} with respect to the challenge t_{perm}.

9. $\mathcal{P}$ sends $w = \langle L_1, \boldsymbol{t'^m} \rangle$ to $\mathcal{V}$, where $\boldsymbol{t'^m} = (1\ t'\ t'^2\ \ldots\ t'^{m-1} \| \mathbf{0})$.

10. $\mathcal{V}$ sets $\mathcal{L}'_1 = (\prod_{i=1}^{M} W_i^L)(\prod_{i=1}^{M} W_i^R)^u$. Eventually, we need a commitment to the reverse of the first m elements of the vector underlying $\mathcal{L}'_1$. This is accomplished in steps 11(d) and 11(e).

11. Set $V = (X^O)^z R, \tilde{V} = (X^L)^z (X^R)^{zu} \tilde{R}$. $\mathcal{V}$ checks if

    (a) $(\Sigma_0, \boldsymbol{t^{2m+2}}, \frac{(ct)^m - 1}{ct-1}) \in \mathcal{R}_{\text{CLF-rev}}$

    (b) $(\Sigma_0, \Sigma_i^k, S_i^k) \in \mathcal{R}_{\mathsf{perm}}\ \forall i \in [M]\ \forall k \in \{L, R\}$

    (c) $(S_i^k, \Sigma_i^k, W_i^k) \in \mathcal{R}_{\mathsf{had}} \forall i \in [M]\ \forall k \in \{L, R\}$

    (d) $(\mathcal{L}'_1, \boldsymbol{t'^m} \| \mathbf{0}, w) \in \mathcal{R}_{\text{CLF-rev}}$

    (e) $(\mathcal{L}_1^{rev}, \mathsf{rev}(\boldsymbol{t'^{2m+2}}), w) \in \mathcal{R}_{\text{CLF-rev}}$

    (f) $(V, \mathcal{L}_1^{rev}, v_1) \in \mathcal{R}$

    (g) $(\tilde{V}, \boldsymbol{c^m}, v_1 + zK) \in \mathcal{R}$

    (h) $(X^L, X^R, X^O) \in \mathcal{R}_{\mathsf{had}}$

    The checks in steps (c) and (h) use the commitments to ρ_{m+1}, ρ_{2m+2} obtained in Step 8, while the checks in step (b) use the commitments to ρ_m, ρ_{2m}. We don't need commitments to $\boldsymbol{t^{2m+2}}, \boldsymbol{t'^m}, \boldsymbol{c^m}$ in steps (a), (e) and (g) because the verifier can compute a random linear combination of these vectors without help from the prover. Moreover, all the claims about the openings of linear forms made by the prover in Steps 8 and 11 can be aggregated using our protocol for batched linear form openings.

Figure 3.8: Protocol Π_{csat} for Circuit Satisfiability

In steps 10(f) and 10(g), given witnesses for distinct z_1, z_2 that satisfy the given constraints, we can extract vectors $\boldsymbol{y}, \tilde{\boldsymbol{x}}^O, \boldsymbol{r}, \tilde{\boldsymbol{r}}$ such that $\forall i \in [2]$, $\langle L_1, z_i\tilde{\boldsymbol{x}}^O + \boldsymbol{r}\rangle = \langle \boldsymbol{c}^{\boldsymbol{m}}, z_i\boldsymbol{y} + \tilde{\boldsymbol{r}}\rangle - z_i K$. This implies that $\langle L_1, \tilde{\boldsymbol{x}}^O\rangle = \langle \boldsymbol{c}^{\boldsymbol{m}}, \boldsymbol{y}\rangle - K$. Given accepting $\boldsymbol{y}$ for distinct challenges u_1, u_2, we can extract $W^L(c), W^R(c), \tilde{\boldsymbol{x}}^L, \tilde{\boldsymbol{x}}^R$ such that $\langle W^L(c), \tilde{\boldsymbol{x}}^O\rangle = \langle \boldsymbol{c}^{\boldsymbol{m}}, \tilde{\boldsymbol{x}}^L\rangle - K$ and $\langle W^R(c), \tilde{\boldsymbol{x}}^O\rangle = \langle \boldsymbol{c}^{\boldsymbol{m}}, \tilde{\boldsymbol{x}}^R\rangle$. If the $\tilde{\boldsymbol{x}}^L, \tilde{\boldsymbol{x}}^R, \tilde{\boldsymbol{x}}^O$ obtained in this way are different from the ones extracted by the extractor of Π_{had} in Step (h), we will have broken binding of the commitment scheme.

Given $\tilde{\boldsymbol{x}}^L, \tilde{\boldsymbol{x}}^R, \tilde{\boldsymbol{x}}^O$ for $2m+1$ distinct c, we either break binding or conclude that $\tilde{\boldsymbol{x}}^L, \tilde{\boldsymbol{x}}^R, \tilde{\boldsymbol{x}}^O$ satisfy the linear constraints of the circuit as well as the multiplicative constraints, and so $\boldsymbol{x}^O = (\tilde{\boldsymbol{x}}^O)_{i\in[m]}$ must be a satisfying assignment.

Special HVZK. We describe a simulator that, given commitments to a satisfying assignment and the randomness of the verifier, computes a transcript which is perfectly indistinguishable from the transcript of a real execution. The simulator $\mathcal{S}$ acts as follows:

- It computes all commitments honestly in Step 2.
- In Step 4, it samples $\boldsymbol{r}, \tilde{\boldsymbol{r}} \leftarrow_R \mathbb{F}_q^m, s, \tilde{s} \leftarrow_R \mathbb{F}_q$ such that $\langle W^L(c) + uW^R(c), \boldsymbol{r}\rangle = \langle \boldsymbol{c}^{\boldsymbol{m}}, \tilde{\boldsymbol{r}}\rangle - K$. It sets $R = \frac{\mathrm{COM}_{\overline{\boldsymbol{a}}}(\boldsymbol{r}, s)}{(X^O)^z}$, $\tilde{R} = \frac{\mathrm{COM}_{\overline{\boldsymbol{a}}}(\tilde{\boldsymbol{r}}, \tilde{s})}{(X^L)^z (X^R)^{zu}}$.
- In Step 6, it sets $v_1 = \langle L_1, \boldsymbol{r}\rangle$.
- It honestly executes Step 8.
- It honestly executes Steps 10(a)-10(e), and invokes the simulator $\mathcal{S}$, $\mathcal{S}_{\mathsf{had}}$ of Π_0 with the corresponding verifier randomness in Steps 10(f)-(h).

We analyze the distribution of the transcript. In both executions, the elements $R, \tilde{R}, v_1$ are distributed uniformly at random. The indistinguishability of steps 10(f)-(h) follows from the simulators of Π_0 and $\Pi_{hadamard}$. All other computations are honestly executed.

□

3.5 Compressed Sigma Protocol for opening of Committed Homomorphism

A bilinear group arithmetic circuit is a circuit in which the wire values are from $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ or $\mathbb{F}_q$, and the gates are group operations, $\mathbb{F}_q$-scalar multiplication, or bilinear pairings. Bilinear circuits are of interest since they directly capture relations arising in identity-based and attribute-based encryption [99, 72], structure-preserving signatures [2] etc. Handling bilinear circuits directly in a ZK system avoids expensive NP reductions or arithmetizations to represent group operations as an arithmetic circuit. The work of Attema et al. [11], building on the

work of Lai et al. [81], gives a succinct argument system for bilinear group arithmetic circuits, by generalizing the compressed sigma protocol framework. A key building block is a protocol for opening a homomorphism on a committed vector. However, as in the case of arithmetic circuits, the verifier remains linear.

We construct a *designated-verifier* succinct argument for opening a committed homomorphism on a committed vector, where the verifier is *logarithmic*.

3.5.1 Commitment Scheme

In this section, we use additive notation for groups in line with prior works for bilinear circuits. We begin by generalizing the homomorphic commitment scheme of [81], to work with logarithmic amount of randomness. We note that $\dot{\boldsymbol{a}}$ denotes $\dot{\boldsymbol{a}} = (a_1, \ldots, a_\ell)$ and for $g \in \mathbb{G}$ and $\boldsymbol{x} = (x_1, \ldots, x_n) \in \mathbb{F}_q^n$, $g\boldsymbol{x}$ denotes $g\boldsymbol{x} = (gx_1, \ldots, gx_n)$ for $\boldsymbol{g} = (g_1, \ldots, g_n) \in \mathbb{G}^n$ and $\boldsymbol{x} = (x_1, \ldots, x_n) \in \mathbb{F}_q^n$, inner product with scalar $\langle \boldsymbol{g}, \boldsymbol{x} \rangle$ denotes $\langle \boldsymbol{g}, \boldsymbol{x} \rangle = g_1 x_1 + \ldots g_n x_n$; for $\boldsymbol{g} = (g_1, \ldots, g_n) \in \mathbb{G}_1^n$ and $\boldsymbol{h} = (h_1, \ldots, h_n) \in \mathbb{G}_2^n$, inner product $e(\boldsymbol{g}, \boldsymbol{h})$ denotes $e(\boldsymbol{g}, \boldsymbol{h}) = e(g_1, h_1) + e(g_2, h_2) + \ldots + e(g_n, h_n)$. Recall the key distribution $\mathcal{ML}_n$, for $n = 2^\ell$,

$$\mathcal{ML}_n = \{\overline{\boldsymbol{a}} : \dot{\boldsymbol{a}} = (\dot{a}_1, \ldots, \dot{a}_\ell) \leftarrow_R \mathbb{F}_q^\ell, \overline{\boldsymbol{a}} = (\prod_{i=1}^{\ell} \dot{a}_i^{x_i})_{x_i \in \{0,1\}}\}$$

We now consider a similar distribution over group elements,

$$\mathcal{ML}_n(\mathbb{G}) = \mathcal{ML}_{2^\ell}(\mathbb{G}) := \{g\overline{\boldsymbol{a}} : g \leftarrow_R \mathbb{G}, \dot{\boldsymbol{a}} = (\dot{a}_1, \ldots, \dot{a}_\ell) \leftarrow_R \mathbb{F}_q^\ell, \overline{\boldsymbol{a}} = (\prod_{i=1}^{\ell} \dot{a}_i^{x_i})_{x_i \in \{0,1\}}\}$$

We define a new commitment scheme which differs from the one proposed in [81] (and subsequently used in [11]) only in that we sample the commitment key from $\mathcal{ML}_n(\mathbb{G})$.

Definition 3.2 (Commitment to $(\mathbb{F}_q, \mathbb{G}_1, \mathbb{G}_2)$-vectors) *Let $(q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, G, H)$ be a bilinear group and $n_0, n_1, n_2 \geq 0$. We define a commitment scheme $\mathrm{COM}^{\mathbb{G}}$ for vectors in $\mathbb{F}_q^{n_0} \times \mathbb{G}_1^{n_1} \times \mathbb{G}_2^{n_2}$, given by the following setup and commitment phase:*

- *Setup :* $(\boldsymbol{h}, \boldsymbol{g}) \leftarrow_R \mathcal{ML}_{n_0+1}^2(\mathbb{G}_T), \mathbf{H} \leftarrow_R \mathcal{ML}_{n_1}^2(\mathbb{G}_2), \mathbf{G} \leftarrow_R \mathcal{ML}_{n_2}^2(\mathbb{G}_1)$
  Here, $(\boldsymbol{h}, \boldsymbol{g}, \mathbf{H}, \mathbf{G}) = (\overline{\boldsymbol{a}}h, \overline{\boldsymbol{b}}g, \overline{\boldsymbol{c}}H, \overline{\boldsymbol{d}}G)$ *for some structured $\mathbb{F}_q$ vectors $\overline{\boldsymbol{a}}, \overline{\boldsymbol{b}}, \overline{\boldsymbol{c}}, \overline{\boldsymbol{d}}$, where h, g is sampled to be $h = e(h_1, H), g = e(g_1, H)$ for some $h_1, g_1 \leftarrow_R \mathbb{G}_1$.*[1] *Then,* ($\mathsf{ck}_0 =$

[1] We note that the distribution remains the same even when $\overline{\boldsymbol{a}}g_1$ is sampled from $\mathcal{ML}_{n_0}(\mathbb{G}_1)$ and g is then set to $g = e(g_1, H)$, making the final commitment key for $\mathbb{F}_q$-vector to be $\boldsymbol{g} = \overline{\boldsymbol{a}}g$, as opposed to when $\boldsymbol{g}$ is directly sampled from $\mathcal{ML}_{n_0}(\mathbb{G}_T)$.

$((\overline{\boldsymbol{a}}h_1, \overline{\boldsymbol{a}}h), (\overline{\boldsymbol{b}}g_1, \overline{\boldsymbol{b}}g)), \mathsf{ck}_1 = \overline{\boldsymbol{c}}H, \mathsf{ck}_2 = \overline{\boldsymbol{d}}G)$ *are the commitment keys and* $(\dot{\mathsf{ck}}_0 = (\overline{\boldsymbol{a}}H, \overline{\boldsymbol{b}}H), \dot{\mathsf{ck}}_1 = \overline{\boldsymbol{c}}G, \dot{\mathsf{ck}}_2 = \overline{\boldsymbol{d}}H)$ *is the verification key.*

– *Commit :* $\mathrm{COM}^{\mathbb{G}} : \mathbb{F}_q^{n_0} \times \mathbb{G}_1^{n_1} \times \mathbb{G}_2^{n_2} \times \mathbb{F}_q \to \mathbb{G}_T^2$,

$$(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}; \gamma) \to \boldsymbol{h}\gamma + \langle \boldsymbol{g}, \boldsymbol{x} \rangle + e(\boldsymbol{y}, \mathbf{H}) + e(\mathbf{G}, \boldsymbol{z}),$$

$$\text{where } h\gamma + \langle \boldsymbol{g}, \boldsymbol{x} \rangle + e(\boldsymbol{y}, \mathbf{H}) + e(\mathbf{G}, \boldsymbol{z}) = \begin{pmatrix} h_1\gamma + \langle \boldsymbol{g}_1, \boldsymbol{x} \rangle + e(\boldsymbol{y}, \mathbf{H}_1) + e(\mathbf{G}_1, \boldsymbol{z}) \\ h_2\gamma + \langle \boldsymbol{g}_2, \boldsymbol{x} \rangle + e(\boldsymbol{y}, \mathbf{H}_2) + e(\mathbf{G}_2, \boldsymbol{z}) \end{pmatrix}$$

The verification key is used to check that the commitment key has been updated by the prover, by having the prover send the first element of the commitment key ck to the verifier, and the verifier using the pairing check to ensure that the split-and-fold technique has been used correctly to update the commitment key and check that the updated commitment (sent by the prover) with respect to the updated commitment key is consistent.

We define an assumption called eGDLR assumption 3.4 along the lines of GDLR assumption in [81] (restated in 3.3), show that it is implied by SXDH (Lemma 3.5) and prove binding of $\mathrm{COM}^{\mathbb{G}}$ under eGDLR.

Lemma 3.2 $\mathrm{COM}^{\mathbb{G}}$ *is computationally hiding under DDH in* $\mathbb{G}_T$*, and computationally binding under SXDH.*

We now formally present our hardness assumption eGDLR 3.4 and then present the proof of Lemma 3.2 in Lemma 3.3 (binding) and Lemma 3.4 (hiding).

3.5.2 Hardness Assumptions

Here, we first recall the GDLR assumption presented in [81], and then we introduce our eGDLR assumption that extends the guarantees of GDLR to structured strings.

Definition 3.3 (Generalized Discrete Logarithm Representation Assumption [81]) *Let* $m \geq 1$ *and* $n_1, n_2, n_T \geq 0$ *(not all zero).* (m, n_T, n_1, n_2)*-GDLR assumption holds in* $(q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, G, H)$*, if for any PPT adversary* $\mathcal{A}$*, we have*

$$\Pr\left[\begin{array}{c} e(\boldsymbol{a}_1, \mathbf{B}_2) + e(\mathbf{B}_1, \boldsymbol{a}_2) + \langle \mathbf{B}_T, \boldsymbol{a}_T \rangle = \mathbf{0}_T \wedge (\boldsymbol{a}_1, \boldsymbol{a}_2, \boldsymbol{a}_T) \neq (\mathbf{0}, \mathbf{0}, \mathbf{0}) \\ \overline{\boldsymbol{b}}_1 \leftarrow_R \mathbb{F}_q^{m \times n_2}, \overline{\boldsymbol{b}}_2 \leftarrow_R \mathbb{F}_q^{m \times n_1}, \overline{\boldsymbol{b}}_T \leftarrow_R \mathbb{F}_q^{m \times n_T} \\ \mathbf{B}_1 = G\overline{\boldsymbol{b}}_1, \mathbf{B}_2 = H\overline{\boldsymbol{b}}_2, \mathbf{B}_T = K\overline{\boldsymbol{b}}_T \\ (\boldsymbol{a}_1, \boldsymbol{a}_2, \boldsymbol{a}_T) \leftarrow \mathcal{A}(q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, G, H, \mathbf{B}_1, \mathbf{B}_2, \mathbf{B}_T) \end{array}\right] \leq \mathsf{negl}(\lambda)$$

where $\boldsymbol{a}_1 \in \mathbb{G}_1^{n_2}, \boldsymbol{a}_2 \in \mathbb{G}_2^{n_1}, \boldsymbol{a}_T \in \mathbb{F}_q^{n_T}$.

We now introduce the eGDLR assumption.

Definition 3.4 (extended Generalized Discrete Logarithm Representation Assumption) *Let $m \geq 1$ and $n_1, n_2, n_T \geq 0$ (not all zero). $(m, n_T, n_1, n_2)-$eGDLR assumption holds in $(q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, G, H)$, if for any PPT adversary $\mathcal{A}$, we have*

$$\Pr\left[\begin{array}{c} e(\boldsymbol{a}_1, \mathbf{B}_2) + e(\mathbf{B}_1, \boldsymbol{a}_2) + \langle \mathbf{B}_T, \boldsymbol{a}_T \rangle = \mathbf{0}_T \wedge (\boldsymbol{a}_1, \boldsymbol{a}_2, \boldsymbol{a}_T) \neq (\mathbf{0}, \mathbf{0}, \mathbf{0}) \\ \overline{\boldsymbol{b}}_1 \leftarrow_R \mathcal{ML}^m_{n_1}, \overline{\boldsymbol{b}}_2 \leftarrow_R \mathcal{ML}^m_{n_2}, \dot{\boldsymbol{b}}_T \leftarrow_R \mathcal{ML}^m_{n_0} \\ \mathbf{B}_1 = G\overline{\boldsymbol{b}}_1, \mathbf{B}_2 = H\overline{\boldsymbol{b}}_2, \mathbf{B}_T = K\overline{\boldsymbol{b}}_T \\ (\boldsymbol{a}_1, \boldsymbol{a}_2, \boldsymbol{a}_T) \leftarrow \mathcal{A}(q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, G, H, \mathbf{B}_1, \mathbf{B}_2, \mathbf{B}_T) \end{array}\right] \leq \mathsf{negl}(\lambda)$$

where $\boldsymbol{a}_1 \in \mathbb{G}_1^{n_2}, \boldsymbol{a}_2 \in \mathbb{G}_2^{n_1}, \boldsymbol{a}_T \in \mathbb{F}_q^{n_T}$.

Lemma 3.3 $\mathrm{COM}^{\mathbb{G}}$ *(Definition 3.2) is computationally binding under eGDLR assumption.*

Proof: If binding of the aforementioned commitment scheme is broken, then we get $\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}, \gamma$ and $\boldsymbol{x}', \boldsymbol{y}', \boldsymbol{z}', \gamma'$ where $\boldsymbol{x} \neq \boldsymbol{x}'$ or $\boldsymbol{y} \neq \boldsymbol{y}'$ or $\boldsymbol{z} \neq \boldsymbol{z}'$ or $\gamma \neq \gamma'$, such that $\mathrm{COM}^{\mathbb{G}}(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}; \gamma) = \mathrm{COM}^{\mathbb{G}}(\boldsymbol{x}', \boldsymbol{y}', \boldsymbol{z}'; \gamma')$

$$\implies \begin{pmatrix} h_1\gamma + \langle \boldsymbol{g}_1, \boldsymbol{x} \rangle + e(\boldsymbol{y}, \mathbf{H}_1) + e(\mathbf{G}_1, \boldsymbol{z}) \\ h_2\gamma + \langle \boldsymbol{g}_2, \boldsymbol{x} \rangle + e(\boldsymbol{y}, \mathbf{H}_2) + e(\mathbf{G}_2, \boldsymbol{z}) \end{pmatrix} = \begin{pmatrix} h_1\gamma' + \langle \boldsymbol{g}_1, \boldsymbol{x}' \rangle + e(\boldsymbol{y}', \mathbf{H}_1) + e(\mathbf{G}_1, \boldsymbol{z}') \\ h_2\gamma' + \langle \boldsymbol{g}_2, \boldsymbol{x}' \rangle + e(\boldsymbol{y}', \mathbf{H}_2) + e(\mathbf{G}_2, \boldsymbol{z}') \end{pmatrix}$$

$$\implies \begin{pmatrix} h_1(\gamma - \gamma') + \langle \boldsymbol{g}_1, (\boldsymbol{x} - \boldsymbol{x}') \rangle + e((\boldsymbol{y} - \boldsymbol{y}'), \mathbf{H}_1) + e(\mathbf{G}_1, (\boldsymbol{z} - \boldsymbol{z}')) \\ h_2(\gamma - \gamma') + \langle \boldsymbol{g}_2, (\boldsymbol{x} - \boldsymbol{x}') \rangle + e((\boldsymbol{y} - \boldsymbol{y}'), \mathbf{H}_2) + e(\mathbf{G}_2, (\boldsymbol{z} - \boldsymbol{z}')) \end{pmatrix} = \begin{pmatrix} \mathbf{0}_T \\ \mathbf{0}_T \end{pmatrix}$$

which breaks the $(2, n_0 + 1, n_1, n_2)$-eGDLR assumption. □

Lemma 3.4 $\mathrm{COM}^{\mathbb{G}}$ *(Definition 3.2) is computationally hiding under DDH assumption in $\mathbb{G}_T$.*

Proof: Hiding of $\mathrm{COM}^{\mathbb{G}}$ follows from the fact that $(h_1, h_2, h_1^\gamma, h_2^\gamma)$, where $h_1, h_2 \in \mathbb{G}_T$ and $\gamma \leftarrow_R \mathbb{F}_q$, is computationally indistinguishable from $(h_1, h_2, h_1^\gamma, r)$, where $h_1, h_2 \in \mathbb{G}_T$ and $\gamma \leftarrow_R \mathbb{F}_q$, $r \leftarrow_R \mathbb{G}_T$, when DDH holds in $\mathbb{G}_T$.

We construct a DDH adversary $\mathcal{A}$ for $\mathbb{G}_T$ given that we have a distinguisher $\mathcal{B}$ which distinguishes $(h_1, h_2, h_1^\gamma, h_2^\gamma)$ from $(h_1, h_2, h_1^\gamma, r)$, where $h_1, h_2 \in \mathbb{G}_T$ and $\gamma \leftarrow_R \mathbb{F}_q$, $r \leftarrow_R \mathbb{G}_T$.

1. $\mathcal{A}$ receives a DDH challenge $ch = (g, g^a, g^b, g^c)$
2. $\mathcal{A}$ sends the challenge vector ch to $\mathcal{B}$
3. If $\mathcal{B}$ outputs that ch is of the form $(h_1, h_2, h_1^\gamma, h_2^\gamma)$, then $\mathcal{A}$ outputs $c = ab$; otherwise $\mathcal{A}$ outputs $c \neq ab$.

$\mathcal{A}$ succeeds with overwhelming probability, if $\mathcal{B}$ does, which follows from the following observation :

- if $c = ab$, then the challenge vector $ch = (g, g^a, g^b, g^c) = (h_1, h_2, h_1^\gamma, h_2^\gamma)$ where $h_1 = g, h_2 = g^a, \gamma = b$, and
- if $c \neq ab$ and $c \in_R \mathbb{F}_q$, then $ch = (g, g^a, g^b, g^c) = (h_1, h_2, h_1^\gamma, r)$, where $h_1 = g, h_2 = g^a, \gamma = b$ and $r = g^c$.

Hence, we have that $(h_1, h_2, h_1^\gamma, h_2^\gamma)$ and $(h_1, h_2, h_1^\gamma, r)$ are computationally indistinguishable under DDH in $\mathbb{G}_T$, where $h_1, h_2 \in \mathbb{G}_T$ and $\gamma \leftarrow_R \mathbb{F}_q$, $r \leftarrow_R \mathbb{G}_T$. From the above property, we note that use of h_1^γ, h_2^γ to re-randomize the two components of $\text{COM}^{\mathbb{G}}$ is indistinguishable from using completely random elements to re-randomize the same, and hence $\text{COM}^{\mathbb{G}}$ is hiding under DDH in $\mathbb{G}_T$.

□

We now show that eGDLR is implied by SXDH.

Lemma 3.5 *Let q be such that $1/q = \mathsf{negl}$. Let $m = 2$; $n_i \geq 0, i = 0, 1, 2$ are not all zero. Then, the (m, n_0, n_1, n_2)-eGDLR assumption holds if the SXDH assumption holds.*

Proof: We know that, for q where $1/q = \mathsf{negl}$ and $m \geq 2$; $n_i \geq 0, i = 0, 1, 2$ are not all zero, (m, n_0, n_1, n_2)-GDLR assumption holds, if the SXDH assumption holds [81]. From the previous statement, we can infer that SXDH assumption implies that $(2, 1, 1, 1)$-GDLR assumption holds. Now, we wish to prove that, for $m = 2$, (m, n_0, n_1, n_2)-eGDLR assumption holds if $(2, 1, 1, 1)$-GDLR assumption holds. We additionally note that the distributions $\{(q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, G, H, (g_1\boldsymbol{r}_1, h_1\boldsymbol{r}_2)) : \boldsymbol{r} \leftarrow_R \mathbb{F}_q^n\}$ and $\{(q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, G, H, (g_1\boldsymbol{r}, h_1\boldsymbol{r})) : \boldsymbol{r}_1, \boldsymbol{r}_2 \leftarrow_R \mathbb{F}_q^n\}$ are identical when SXDH assumption holds.

We construct an adversary $\mathcal{A}$ for $(2, 1, 1, 1)$-GDLR assumption, given an adversary $\mathcal{B}$ for $(2, n_0, n_1, n_2)$-eGDLR assumption, as follows :

1. $\mathcal{A}$ receives a challenge for $(2, 1, 1, 1)$-GDLR assumption, $(q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, G, H, (g_1, h_1), (g_2, h_2), (g_T, h_T))$ such that $g_i, h_i \in \mathbb{G}_i, i \in \{1, 2, T\}$
2. $\mathcal{A}$ samples the keys for obtaining challenges for $\mathcal{B}$: $\overline{\boldsymbol{r}} \leftarrow_R \mathcal{ML}_{n_0}$, $\overline{\boldsymbol{s}} \leftarrow_R \mathcal{ML}_{n_1}$, and $\overline{\boldsymbol{t}} \leftarrow_R \mathcal{ML}_{n_2}$
3. $\mathcal{A}$ sends $\mathcal{B}$ the challenge $(q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, G, H, (g_1\overline{\boldsymbol{r}}, h_1\overline{\boldsymbol{r}}), (g_2\overline{\boldsymbol{s}}, h_2\overline{\boldsymbol{s}}), (g_T\overline{\boldsymbol{t}}, h_T\overline{\boldsymbol{t}}))$

4. $\mathcal{B}(q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, G, H, (g_1\overline{\boldsymbol{r}}, h_1\overline{\boldsymbol{r}}), (g_2\overline{\boldsymbol{s}}, h_2\overline{\boldsymbol{s}}), (g_T\overline{\boldsymbol{t}}, h_T\overline{\boldsymbol{t}})) \to (\boldsymbol{a}_1, \boldsymbol{a}_2, \boldsymbol{a}_T)$

5. $\mathcal{A}$ computes $x_1 = \langle \boldsymbol{a}_1, \overline{\boldsymbol{r}} \rangle, x_2 = \langle \boldsymbol{a}_2, \overline{\boldsymbol{s}} \rangle$ and $x_T = \langle \overline{\boldsymbol{t}}, \boldsymbol{a}_T \rangle$ and outputs (x_1, x_2, x_T), where the first two operations are inner product with scalar for groups $\mathbb{G}_2$ and $\mathbb{G}_1$, and the third operation is inner product of elements of $\mathbb{F}_q$.

We claim that $\mathcal{A}$ succeeds with overwhelming probability, if $\mathcal{B}$ does.

We note that, if $\mathcal{B}$ succeeds, then its output $(\boldsymbol{a}_1, \boldsymbol{a}_2, \boldsymbol{a}_T)$ is such that $(\boldsymbol{a}_1, \boldsymbol{a}_2, \boldsymbol{a}_T) \neq (0, 0, 0)$ and

$$g_T\langle \overline{\boldsymbol{t}}, \boldsymbol{a}_T \rangle + e(g_1\overline{\boldsymbol{r}}, \boldsymbol{a}_1) + e(\boldsymbol{a}_2, g_2\overline{\boldsymbol{s}}) = 0 \text{ , and } h_T\langle \overline{\boldsymbol{t}}, \boldsymbol{a}_T \rangle + e(h_1\overline{\boldsymbol{r}}, \boldsymbol{a}_1) + e(\boldsymbol{a}_2, h_2\overline{\boldsymbol{s}}) = 0$$

hence we have, for $x_1 = \langle \boldsymbol{a}_1, \overline{\boldsymbol{r}} \rangle, x_2 = \langle \boldsymbol{a}_2, \overline{\boldsymbol{r}} \rangle$ and $x_T = \langle \overline{\boldsymbol{t}}, \boldsymbol{a}_T \rangle$:

$$g_T x_T + e(g_1, x_1) + e(x_2, g_2) = 0 \text{ , and } h_T x_T + e(h_1, x_1) + e(x_2, h_2) = 0$$

We analyse that, $\mathcal{A}$'s breaks the assumption if $\mathcal{B}$ does, by showing that along with the satisfied equation, $(\boldsymbol{a}_1, \boldsymbol{a}_2, \boldsymbol{a}_T) \neq (0, 0, 0)$ ensures $(x_1, x_2, x_T) \neq (0, 0, 0)$, except with negligible probability.

If $\boldsymbol{a}_T \neq 0$, then we have $x_T = \langle \overline{\boldsymbol{t}}, \boldsymbol{a}_T \rangle \neq 0$ w.h.p. as otherwise we have $g_T\langle \overline{\boldsymbol{t}}, \boldsymbol{a}_T \rangle = 0$ which breaks dlog in $\mathbb{G}_T$, hence it ensures $(x_1, x_2, x_T) \neq (0, 0, 0)$.

If $\boldsymbol{a}_1 \neq 0$, and if $x_1 = \langle \boldsymbol{a}_1, \overline{\boldsymbol{r}} \rangle = e_2$, then we have $e(\langle g_1, \overline{\boldsymbol{r}} \rangle, \boldsymbol{a}_1) = e(g_1, \langle \boldsymbol{a}_1, \overline{\boldsymbol{r}} \rangle) = e(g_1, e_2) = e(g_1, qe_2) = e(qg_1, e_2) = e(e_1, e_2) = e_T$, which breaks the $(e)n$-BP assumption that holds when SXDH holds, as DDH is hard in $\mathbb{G}_1$.

Similarly, we can argue for x_2 being non-zero, when $\boldsymbol{a}_2$ is non-zero.

□

Definition 3.5 (n-BP Assumption) *For all non-uniform PPT Adversary $\mathcal{A}$,*

$$\Pr\begin{bmatrix} e(\boldsymbol{X}, \boldsymbol{Y}) = e_T, \boldsymbol{X} \in \mathbb{G}_1^n, \boldsymbol{Y} \in \mathbb{G}_2^n \\ \boldsymbol{x} \leftarrow_R \mathbb{F}_q^n, \boldsymbol{X} = G\boldsymbol{x}, \boldsymbol{Y} \neq e_2 \\ \boldsymbol{Y} \leftarrow \mathcal{A}(q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, G, H, \boldsymbol{X}) \end{bmatrix} = \mathsf{negl}(\lambda)$$

Definition 3.6 ($(e)n$-BP Assumption) *For all non-uniform PPT Adversary $\mathcal{A}$,*

$$\Pr\begin{bmatrix} e(\boldsymbol{X}, \boldsymbol{Y}) = e_T, \boldsymbol{X} \in \mathbb{G}_1^n, \boldsymbol{Y} \in \mathbb{G}_2^n \\ \boldsymbol{x} \leftarrow_R \mathcal{ML}_n, \boldsymbol{X} = G\boldsymbol{x}, \boldsymbol{Y} \neq e_2 \\ \boldsymbol{Y} \leftarrow \mathcal{A}(q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, G, H, \boldsymbol{X}) \end{bmatrix} = \mathsf{negl}(\lambda)$$

Lemma 3.6 *Let q be such that $1/q = \mathsf{negl}$. n-BP Assumption, for $n \in \mathbb{N}$, holds when DDH is hard in $\mathbb{G}_1$.*

Proof: We construct a DDH adversary $\mathcal{A}$ of $\mathbb{G}_1$, given a n-BP adversary $\mathcal{B}$, as follows:

1. $\mathcal{A}$ receives a DDH-challenge $(g, g \cdot a, g \cdot b, g \cdot c)$ of $\mathbb{G}_1$
2. $\mathcal{A}$ samples $j \leftarrow_R \{0, \ldots, n-1\}$ and $\boldsymbol{r} = (r_1, \ldots, r_{n-1}) \leftarrow_R \mathbb{F}_q^{n-1}$, and sets $\boldsymbol{x} = (x_0, \ldots, x_{n-1})$, where $x_j = c$, $x_i = ar_{i+1}$ for $i = 0, \ldots, j-1$ and $x_i = ar_i$ for $i = j+1, \ldots, n-1$
3. $\mathcal{A}$ computes $\boldsymbol{X} = G\boldsymbol{x}$ and sends $(q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, G, H, \boldsymbol{X})$ to $\mathcal{B}$
4. $\mathcal{A}$ receives $\boldsymbol{Y}$ from $\mathcal{B}$
5. $\mathcal{A}$ sets $\boldsymbol{z} = (z_0, \ldots, z_{n-1})$, where $z_0 = b$ and $z_i = r_i, i = 1, \ldots, n-1$, and sets $\boldsymbol{Z} = G\boldsymbol{z}$
6. If $e(\boldsymbol{X}, \boldsymbol{Y}) = e_T$ and $e(\boldsymbol{Z}, \boldsymbol{Y}) = e_T$, then $\mathcal{A}$ outputs 1, otherwise it outputs 0.

We claim that $\mathcal{A}$ succeeds with overwhelming probability, if $\mathcal{B}$ does.

We note that if $\mathcal{B}$ succeeds then we have $e(\boldsymbol{X}, \boldsymbol{Y}) = e_T$, which implies, assuming $\boldsymbol{Y} = H\boldsymbol{y}$, where $\boldsymbol{y} = (y_0, y_1, \ldots, y_{n-1}) \in \mathbb{F}_q^n$

$$e(G\boldsymbol{x}, H\boldsymbol{y}) = e_T \implies e(G{\cdot}x_1, H{\cdot}y_1) \cdots e(G{\cdot}x_{n-1}, H{\cdot}y_{n-1}) = e_T \implies e(G, H)\langle \boldsymbol{x}, \boldsymbol{y} \rangle = e_T$$

which gives us $\langle x, y \rangle = 0$.

If $c = ab$, then we have that $ar_1y_0 + \cdots + ar_jy_{j-1} + cy_j + ar_{j+1}y_{j+1} + \cdots + ar_{n-1}y_{n-1} = 0 \implies ar_1y_0 + \cdots + ar_jy_{j-1} + aby_j + ar_{j+1}y_{j+1} + \cdots + ar_{n-1}y_{n-1} = 0 \implies r_1y_0 + \cdots + r_jy_{j-1} + by_j + r_{j+1}y_{j+1} + \cdots + r_{n-1}y_{n-1} = 0 \implies e(\boldsymbol{Z}, \boldsymbol{Y}) = e_T$, then the adversary outputs 1.

If $c \neq ab$, then the adversary outputs 0 with high probability, as $c \neq ab$ and output 1 by adversary $\implies$ $ar_1y_0 + \cdots + ar_jy_{j-1} + cy_j + ar_{j+1}y_{j+1} + \cdots + ar_{n-1}y_{n-1} = 0$ and $r_1y_0 + \cdots + r_jy_{j-1} + by_j + r_{j+1}y_{j+1} + \cdots + r_{n-1}y_{n-1} = 0$, both equations are independently satisfied by $\boldsymbol{y}$, which happens with probability $O(1/q)$.

Additionally, since the position of embedding of the challenge is sampled at random, the probability that the adversary guesses the position j of embedding of the challenge and sets its own response such that $y_j = 0$ to remove the dependency of the solution from the challenge is $1/n$.

□

Definition 3.7 (**(P,Q)-DDH Assumption** [35]) *Let q be a prime number. Let $\mathbb{G}$ be a group of order q, g a generator of $\mathbb{G}$, and $(P,Q) \subseteq \mathbb{F}_q[X_1,\ldots,X_n]$ two sets of polynomials. We define the oracles $\mathsf{Real}_{(P,Q)}$ and $\mathsf{Fake}_{(P,Q)}$ as follows. Both oracles first select uniformly at random $x_i \leftarrow_R \mathbb{F}_q$, for $i \in [n]$. Then they answer two types of queries. In input (info, i) for $1 \leq i \leq |P|$, both $\mathsf{Real}_{(P,Q)}$ and $\mathsf{Fake}_{(P,Q)}$ answer with $g^{p_i(x_1,x_2,\ldots,x_n)}$ for $p_i \in P$. On each new input (chal, j) for some $1 \leq j \leq |Q|$, oracle $\mathsf{Real}_{(P,Q)}$ answers with $g^{q_j(x_1,x_2,\ldots,x_n)}$ whereas oracle $\mathsf{Fake}_{(P,Q)}$ selects $r_j \leftarrow_R \mathbb{F}_q$ and answers with g^{r_j}. The adversary can intertwine the info and chal queries. The goal of the adversary is to distinguish between these two oracles.*

[35] proves that when the challenge (P,Q) is non-trivial, i.e. if $\mathsf{span}(P) \cap \mathsf{span}(Q) = \{0\}$ and the polynomials in Q are linearly independent, that satisfies two conditions specified in Definition 3 of [35], then the (P,Q)-DDH Assumption hold whenever DDH holds. Additionally, we note that if we consider $P = \{X_1,\ldots,X_\ell\}$ and $Q = \{\prod_{i=1}^{\ell} X_i^{b_i}\}_{b_i \in \{0,1\}}$, then it satisfies the above criteria and hence, this particular variant of (P,Q)-DDH holds whenever DDH holds. Now, since DDH holds, we prove that our $(e)n$-BP assumption holds whenever (P,Q)-DDH assumption holds, given that n-BP assumption holds as well.

Lemma 3.7 *Let q be such that $1/q = \mathsf{negl}$. $(e)n$-BP Assumption holds when (P,Q)-DDH Assumption and n-BP Assumption holds, for all $n \in \mathbb{N}$.*

Proof: Let $n = 2^\ell$. Let us consider that, if possible, there exists an $(e)n$-BP adversary that breaks the $(e)n$-BP assumption. We construct an adversary $\mathcal{A}$ for (P,Q)-DDH Assumption in $\mathbb{G}_1$, given an adversary $\mathcal{B}$ for the $(e)n$-BP Assumption as follows, where we consider $P = \{X_1,\ldots,X_\ell\}$ and $Q = \{\prod_{i=1}^{\ell} X_i^{b_i}\}_{b_i \in \{0,1\}}$, such that $|P| = \ell$ and $|Q| = n - \ell$.

- $\mathcal{A}$ queries the (P,Q)-DDH challenger with (info, i) for all $i \in \{1,\ldots,\ell\}$ and receives response $w_1,\ldots,w_\ell$
- $\mathcal{A}$ then queries the (P,Q)-DDH challenger with (chal, i) for all $i \in \{1,\ldots,n-\ell\}$ and receives response $z_1,\ldots,z_{n-\ell}$
- $\mathcal{A}$ defines $\boldsymbol{X} = (x_1,\ldots,x_n)$ as $x_i = w_i$ for all $i \in [\ell]$ and $x_i = z_{i-\ell}$ for all $i \in \{\ell+1,\ldots,n\}$, and sends $\boldsymbol{X}$ along with the bilinear group $(q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, G, H)$ to $\mathcal{B}$
- $\mathcal{B}$ returns $\boldsymbol{Y}$ to $\mathcal{A}$
- $\mathcal{A}$ checks if $\boldsymbol{Y} \neq e_2$ and $e(\boldsymbol{X}, \boldsymbol{Y}) = e_T$

– If the above are satisfied, $\mathcal{A}$ concludes that $\boldsymbol{X}$ is a $\mathsf{Real}_{(P,Q)}$ challenge and outputs 1, and outputs 0 otherwise.

We claim that $\mathcal{A}$ succeeds with non-negligible advantage if $\mathcal{B}$ succeeds with non-negligible probability. Now, let us consider the (P,Q)-DDH challenges when $P = \{X_1, \ldots, X_\ell\}$ and $Q = \{\prod_{i=1}^{\ell} X_i^{b_i}\}_{b_i \in \{0,1\}}$. Note that a $\mathsf{Real}_{(P,Q)}$ challenge inherits the $\mathcal{ML}_n$ distribution in the exponent, which is the required distribution for a structured $(e)n$-BP challenge, whereas a $\mathsf{Fake}_{(P,Q)}$ challenge inherits the random distribution. Hence, we can denote a $\mathsf{Real}_{(P,Q)}$ challenge vector $\boldsymbol{X}$ as $\boldsymbol{X} \leftarrow_R \mathcal{ML}_n(\mathbb{G})$ and we can denote a $\mathsf{Fake}_{(P,Q)}$ challenge vector $\boldsymbol{X}$ as $\boldsymbol{X} \leftarrow_R \mathbb{G}^n$.

Let us assume that $\mathcal{B}$ succeeds with probability ϵ_1 for a structured $(e)n$-BP challenge, i.e. $\Pr[\mathcal{B} \text{ succeeds} \mid \boldsymbol{X} \leftarrow_R \mathcal{ML}_n(\mathbb{G})] = \epsilon_1$. Note that $\mathcal{B}$ succeeds for a $(e)n$-BP challenge $\boldsymbol{X}$ if it outputs $\boldsymbol{Y}$ such that $\boldsymbol{Y} \neq e_2$ and $e(\boldsymbol{X}, \boldsymbol{Y}) = e_T$. Let Additionally, we note that if n-BP holds and $\Pr[\mathcal{B} \text{ succeeds} \mid \boldsymbol{X} \leftarrow_R \mathbb{G}^n] = \epsilon_2$, since the challenge $\boldsymbol{X} \leftarrow_R \mathbb{G}^n$ follows the required distribution for a n-BP challenge ϵ_2 must be negligible. Then we compute the probability of success of $\mathcal{A}$.

$$
\begin{aligned}
&\Pr[\mathcal{A} \text{ guesses correctly}] \\
&= \Pr[\mathcal{A} \text{ outputs } 1 \mid \mathsf{Real}_{(P,Q)}] \Pr[\mathsf{Real}_{(P,Q)}] + \Pr[\mathcal{A} \text{ outputs } 0 \mid \mathsf{Fake}_{(P,Q)}] \Pr[\mathsf{Fake}_{(P,Q)}] \\
&= \Pr[\mathcal{A} \text{ outputs } 1 \mid \mathsf{Real}_{(P,Q)}] \times 1/2 + \Pr[\mathcal{A} \text{ outputs } 0 \mid \mathsf{Fake}_{(P,Q)}] \times 1/2 \\
&= \Pr[\mathcal{A} \text{ outputs } 1 \mid \boldsymbol{X} \leftarrow_R \mathcal{ML}_n(\mathbb{G})] \times 1/2 + \Pr[\mathcal{A} \text{ outputs } 0 \mid \boldsymbol{X} \leftarrow_R \mathbb{G}^n] \times 1/2 \\
&= \Pr[\mathcal{B} \text{ succeeds} \mid \boldsymbol{X} \leftarrow_R \mathcal{ML}_n(\mathbb{G})] \times 1/2 + \Pr[\mathcal{B} \text{ fails} \mid \boldsymbol{X} \leftarrow_R \mathbb{G}^n] \times 1/2 \\
&= \epsilon_1/2 + \Pr[\mathcal{B} \text{ fails} \mid \boldsymbol{X} \leftarrow_R \mathbb{G}^n] \times 1/2 \\
&= \epsilon_1/2 + (1 - \Pr[\mathcal{B} \text{ succeeds} \mid \boldsymbol{X} \leftarrow_R \mathbb{G}^n]) \times 1/2 \\
&= \epsilon_1/2 + (1 - \epsilon_2)/2 = 1/2 + (\epsilon_1 - \epsilon_2)/2
\end{aligned}
$$

Now, since $\epsilon_1 = \Pr[\mathcal{B} \text{ succeeds} \mid \boldsymbol{X} \leftarrow_R \mathcal{ML}_n(\mathbb{G})]$ is non-negligible and $\epsilon_2 = \Pr[\mathcal{B} \text{ succeeds} \mid \boldsymbol{X} \leftarrow_R \mathbb{G}^n]$ is negligible, $\epsilon_1 - \epsilon_2$ is non-negligible. Hence, $\mathcal{A}$ succeeds with a non-negligible advantage, which is a contradiction. □

3.5.3 Succinct Verifier Σ-Protocol for Opening Committed Homomorphism

Notation. Let $(q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, G, H)$ be a bilinear group. Let $g^{\overline{\boldsymbol{a}}} \in \mathbb{G}_1$ be the commitment key used to commit to a vector of $\mathbb{F}_q$ elements in $\mathrm{COM}_{\overline{\boldsymbol{a}}}(\boldsymbol{x}) = \langle \overline{\boldsymbol{a}}, \boldsymbol{x} \rangle g \in \mathbb{G}_1$, where $\boldsymbol{x} \in \mathbb{F}_q^n, \overline{\boldsymbol{a}} \in \mathbb{F}_q^n$. We consider the group homomorphism $f : \mathbb{F}_q^n \to \mathbb{G}_2$, and define $\mathrm{HOM}(\mathbb{F}_q^n, \mathbb{G}_2) = \{f : f \text{ is a homomorphism from } \mathbb{F}_q^n \text{ to } \mathbb{G}_2\}$. We use $\mathrm{COM}^{\mathbb{G}}$ given in definition 3.2 and use a

modified version to commit to element of only one source group of bilinear pairing as follows : $\text{COM}^{\mathbb{G}} : \mathbb{G}_2^n \to \mathbb{G}_T$, where $\text{COM}^{\mathbb{G}}(\boldsymbol{x}) = e(\mathbf{G}, \boldsymbol{x})$, for $n = 2^\ell, h \leftarrow_R \mathbb{G}_T, \dot{\boldsymbol{a}} = (\dot{a}_1, \ldots, \dot{a}_\ell) \leftarrow_R \mathbb{F}_q^\ell, \overline{\boldsymbol{a}} = \left(\prod_{i=1}^{\ell} \dot{a}_i^{b_i}\right)_{b_i \in \{0,1\}}, \mathbf{G} = \overline{\boldsymbol{a}}G$, and we use the notation to $\text{COM}_{\overline{\boldsymbol{a}}}^{\mathbb{G}}$ to explicitly specify the commitment key for ease of exposition, and define it as $\text{COM}_{\overline{\boldsymbol{a}}}^{\mathbb{G}}(\boldsymbol{x}) = \text{COM}^{\mathbb{G}}(\boldsymbol{x}) = e(\mathbf{G}, \boldsymbol{x})$, where $\mathbf{G} = \overline{\boldsymbol{a}}G$.

Opening group homomorphism. We aim to prove that a committed vector $\boldsymbol{x} \in \mathbb{F}_q^n$ is opening of an element $y \in \mathbb{G}$ with respect to group homomorphism defined by $f : \mathbb{F}_q^n \to \mathbb{G}_2$, i.e. the opening of a given commitment $\text{COM}_{\overline{\boldsymbol{a}}}(\boldsymbol{x})$, $\boldsymbol{x} \in \mathbb{F}_q^n$ is such that $f(\boldsymbol{x}) = y$ for some $y \in \mathbb{G}_2$. We note that the homomorphism $f : \mathbb{F}_q^n \to \mathbb{G}$ can be defined as $f \in \mathbb{G}_2^n$, and we extend the techniques discussed in Section 3.3. We use the commitment scheme from Definition 3.2 $\text{COM}^{\mathbb{G}} : \mathbb{F}_q \times \mathbb{G}_1^{n_1} \times \mathbb{G}_2^{n_2} \to \mathbb{G}_T^2$ to succinctly commit to $f = (f_1, \ldots, f_n) \in \mathbb{G}_2^n$ and $\mathsf{rev}(f) = (f_n, \ldots, f_1) \in \mathbb{G}_2^n$ using the structured commitment key $g^{\overline{\boldsymbol{a}}} \in \mathbb{G}_1$ used to commit to the vector.

We note that while techniques of Section 3.3.2 for committed linear forms can extend to a committed homomorphism, there are some differences that we need to handle. First, the representation of a group homomorphism is given by group elements as opposed to field elements in linear forms, and this requires a commitment to group elements. Since the commitment scheme relies on SXDH, we cannot encode the commitment randomness in the second group anymore. This is, however, crucial to verify that the commitment key is updated correctly in each step of split-and-fold. This makes our protocol designated-verifier since the encoding of the randomness is available only to the verifier and binding still holds under SXDH. We define the relation $\mathcal{R}$ for opening a group homomorphism f below, and then present the protocol $\Pi_{\mathsf{0\text{-}hom}}$ for relation $\mathcal{R}$.

$$\mathcal{R} = \{(P \in \mathbb{G}_1, f \in \text{HOM}(\mathbb{F}_q^n, \mathbb{G}_2), y \in \mathbb{G}_2; \boldsymbol{x} \in \mathbb{F}_q^n, \gamma \in \mathbb{F}_q) : P = \text{COM}_{\overline{\boldsymbol{a}}}(\boldsymbol{x}; \gamma) \ \wedge \ f(\boldsymbol{x}) = y\}$$

Theorem 3.10 $\Pi_{\mathsf{0\text{-}hom}}$ *(Fig 3.9) is a 3-move protocol for relation $\mathcal{R}$. It is perfectly complete, special honest-verifier zero-knowledge and computationally special sound.*

Proof Sketch. Note that this theorem follows from the fact that this protocol is identical to the one introduced in [11], and the properties of the protocol relies on the hiding and binding of the commitment scheme which are satisfied by our commitment scheme 3.2 used here.

Now, we note that the last message sent in step 4 of $\Pi_{\mathsf{0\text{-}hom}}$ (Fig 3.9) along with the check computed by the verifier can be captured by the relations defined below, and we provide a

Parameters

- Common parameters : $(P \in \mathbb{G}_1, f \in \text{HOM}(\mathbb{F}_q^n, \mathbb{G}_2), y \in \mathbb{G}_2),\ P = \text{COM}_{\overline{\boldsymbol{a}}}(\boldsymbol{x}; \gamma), y = f(\boldsymbol{x})$
- $\mathcal{P}$'s input : $(\boldsymbol{x} \in \mathbb{F}_q^n, \gamma \in \mathbb{F}_q)$

Protocol

1. $\mathcal{P}$ samples $\boldsymbol{r} \leftarrow_R \mathbb{G}, \rho \leftarrow_R \mathbb{F}_q$, computes $A = \text{COM}_{\overline{\boldsymbol{a}}}(\boldsymbol{r}; \rho),\ t = f(\boldsymbol{r})$ and sends A, t to $\mathcal{V}$.
2. $\mathcal{V}$ samples $c \leftarrow_R \mathbb{F}_q$ and sends c to $\mathcal{P}$
3. $\mathcal{P}$ computes $\boldsymbol{z} = c\boldsymbol{x} + \boldsymbol{r}$ and $\phi = c\gamma + \rho$ and sends $\boldsymbol{z}, \phi$ to $\mathcal{V}$
4. $\mathcal{V}$ checks if $\text{COM}_{\overline{\boldsymbol{a}}}(\boldsymbol{z}; \phi) = A + cP$ and $f(\boldsymbol{z}) = cy + t$, outputs 1 if it holds, outputs 0 otherwise.

Figure 3.9: Protocol $\Pi_{\text{0-hom}}$ for relation $\mathcal{R}$

Proof of Knowledge of the last message instead with the protocols for the following relation.

$$\mathcal{R}_{\text{CH}} = \{(P \in \mathbb{G}_1, Q \in \mathbb{G}_T, y \in \mathbb{G}_2, g \in \mathbb{G}_1; f, \boldsymbol{x} \in \mathbb{F}_q^n) : \\ P = \text{COM}_{\overline{\boldsymbol{a}}}(\boldsymbol{x}) \ \wedge \ Q = \text{COM}^{\mathbb{G}}_{\overline{\boldsymbol{a}}}(f) \ \wedge \ f(\boldsymbol{x}) = y\}$$

Note that in the above, $P = \langle \overline{\boldsymbol{a}}, \boldsymbol{x} \rangle g \ \wedge \ Q = e(g\overline{\boldsymbol{a}}, f)$

We provide the protocol $\Pi_{\text{1-hom}}$ for handling $\mathcal{R}_{\text{CH}}$ in Fig 3.10. Note that the protocol starts with having value of the common parameter intended as the first element of the commitment key as equal to the generator of the group, i.e. $g = G$, and it is later updated accordingly to encompass the commitment key updates in the protocol.

Proof: Special Soundness. We consider 3 accepting transcripts for one iteration of PoK $\Pi_{\text{1-hom}}$ (where one iteration consists of steps 1-5, and step 6 follows by sending $\boldsymbol{x}', L'$ instead of providing a PoK) as follows, where c_1, c_2, c_3 are all distinct challenges. :

$$(A_1, A_2, B_1, B_2, y_1, y_2, c_1, g_1', \boldsymbol{x}_1', f_1')$$
$$(A_1, A_2, B_1, B_2, y_1, y_2, c_2, g_2', \boldsymbol{x}_2', f_2')$$
$$(A_1, A_2, B_1, B_2, y_1, y_2, c_3, g_3', \boldsymbol{x}_3', f_3')$$

Let us consider $\boldsymbol{z}_i = (c_i \boldsymbol{x}_i' \| \boldsymbol{x}_i')$. We note that, as c_1, c_2 and c_3 are such that $c_i \neq c_j$ for all

$(i \neq j)$ $i, j \in \{1, 2, 3\}$, the matrix V described below is invertible.

$$V = \begin{pmatrix} 1 & 1 & 1 \\ c_1 & c_2 & c_3 \\ c_1^2 & c_2^2 & c_3^2 \end{pmatrix}$$

Hence, we can compute $(a_1, a_2, a_3)^T = V^{-1}(0, 1, 0)^T$. The computed a_1, a_2, a_3 satisfy $\sum_i a_i = 0, \sum_i a_i c_i = 1$ and $\sum_i a_i c_i^2 = 0$.

We define $\boldsymbol{w}$ to be the extracted value of $\boldsymbol{x}$ and compute it as $\boldsymbol{w} = a_1\boldsymbol{z}_1 + a_2\boldsymbol{z}_2 + a_3\boldsymbol{z}_3$, given that $\mathrm{COM}_{\overline{\boldsymbol{a}}'}(\boldsymbol{x}'_i) = A_1 + c_i P + c_i^2 A_2$ then we consider

$$\begin{aligned}
\mathrm{COM}_{\overline{\boldsymbol{a}}}(\boldsymbol{w}) &= (\langle \overline{\boldsymbol{a}}, \boldsymbol{w} \rangle) g \\
&= (\langle \overline{\boldsymbol{a}}, a_1\boldsymbol{z}_1 + a_2\boldsymbol{z}_2 + a_3\boldsymbol{z}_3 \rangle) g \\
&= (\langle \overline{\boldsymbol{a}}_L \| \overline{\boldsymbol{a}}_R, a_1(c_1\boldsymbol{x}'_1 \| \boldsymbol{x}'_1) + a_2(c_2\boldsymbol{x}'_2 \| \boldsymbol{x}'_2) + a_3(c_3\boldsymbol{x}'_3 \| \boldsymbol{x}'_3) \rangle) g \\
&= (\langle \overline{\boldsymbol{a}}_L, a_1c_1\boldsymbol{x}'_1 + a_2c_2\boldsymbol{x}'_2 + a_3c_3\boldsymbol{x}'_3 \rangle) g + (\langle \overline{\boldsymbol{a}}_R, a_1\boldsymbol{x}'_1 + a_2\boldsymbol{x}'_2 + a_3\boldsymbol{x}'_3 \rangle) g \\
&= (\langle \overline{\boldsymbol{a}}', a_1c_1\boldsymbol{x}'_1 + a_2c_2\boldsymbol{x}'_2 + a_3c_3\boldsymbol{x}'_3 \rangle) g (\langle \dot{a}_\ell \overline{\boldsymbol{a}}', a_1\boldsymbol{x}'_1 + a_2\boldsymbol{x}'_2 + a_3\boldsymbol{x}'_3 \rangle) g \\
&= (a_1 \langle (c_1 + \dot{a}_\ell)\overline{\boldsymbol{a}}', \boldsymbol{x}'_1 \rangle + a_2 \langle (c_2 + \dot{a}_\ell)\overline{\boldsymbol{a}}', \boldsymbol{x}'_2 \rangle + a_3 \langle (c_3 + \dot{a}_\ell)\overline{\boldsymbol{a}}', \boldsymbol{x}'_3 \rangle) g \\
&= (a_1 \langle \overline{\boldsymbol{a}}', \boldsymbol{x}'_1 \rangle)(g'_1) + (a_2 \langle \overline{\boldsymbol{a}}', \boldsymbol{x}'_2 \rangle)(g'_2) + (a_3 \langle \overline{\boldsymbol{a}}', \boldsymbol{x}'_3 \rangle)(g'_3) \qquad \text{(from Step 4)} \\
&= a_1 \left(A_1 + c_1 P + c_1^2 A_2\right) + a_2 \left(A_1 + c_2 P + c_2^2 A_2\right) + a_3 \left(A_1 + c_3 P + c_3^2 A_2\right) \\
&\qquad\qquad \text{(from last check)} \\
&= (a_1 + a_2 + a_3)A_1 + (a_1c_1 + a_2c_2 + a_3c_3)P + (a_1c_1^2 + a_2c_2^2 + a_3c_3^2)A_2 \quad = P
\end{aligned}$$

Hence, the extracted $\boldsymbol{w}$ is an opening of the commitment P. Similarly we can extract an opening $\boldsymbol{m}$ of the commitment Q. From the binding of the commitment scheme, we have that $\boldsymbol{w} = \boldsymbol{x}$ and $\boldsymbol{m} = \mathsf{rev}(f)$ except with negligible probability.

From the accepting transcripts, we have that $f'_i(\boldsymbol{x}'_i) = y_1 + c_i y + c_i^2 y_2$ for $i = 1, 2, 3$. Now, we consider the following :

$$\begin{aligned}
& f'_i(\boldsymbol{x}'_i) = (c_i f_L + f_R)(\boldsymbol{x}_L + c_i \boldsymbol{x}_R) && \forall i \in \{1, 2, 3\} \\
\implies\ & y_1 + c_i y + c_i^2 y_2 = f_R(\boldsymbol{x}_L) + c_i f(\boldsymbol{x}) + c_i^2 f_L(\boldsymbol{x}_R) && \forall i \in \{1, 2, 3\} \\
\implies\ & y = f(\boldsymbol{x})
\end{aligned}$$

□

Theorem 3.11 *$\Pi_{1\text{-}\mathsf{hom}}$ is a $(k_1, \ldots, k_\ell)$-move protocol for relation $\mathcal{R}_{CH}$, where $k_i = 3$, $\forall i \in$*

Parameters

- **Common parameters** : $(P \in \mathbb{G}_1, Q \in \mathbb{G}_T, y \in \mathbb{G}_2, g \in \mathbb{G}_1)$,
  - $P = \mathrm{COM}_{\overline{\boldsymbol{a}}}(\boldsymbol{x}), Q = \mathrm{COM}^{\mathbb{G}}_{\overline{\boldsymbol{a}}}(\mathsf{rev}(f)), y = f(\boldsymbol{x})$
  - $n = 2^\ell, \dot{\boldsymbol{a}} = (\dot{a}_1, \ldots, \dot{a}_\ell), \overline{\boldsymbol{a}} = \left(\prod_{i=1}^{\ell} \dot{a}_i^{b_i}\right)_{b_i \in \{0,1\}}$
  - $(q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, G, H)$ is a bilinear map.
- $\mathcal{P}$**'s input** : $(\overline{\boldsymbol{a}}g \in \mathbb{G}_1^n, \boldsymbol{x} \in \mathbb{F}_q^n, f \in \mathrm{HOM}(\mathbb{F}_q^n, \mathbb{G}_2))$
- $\mathcal{V}$**'s input** : $\dot{\boldsymbol{a}}H \in \mathbb{G}_2^\ell$

Protocol

1. Let us define k as $k = \mathsf{rev}(f)$. $\mathcal{P}$ parses $\boldsymbol{x} = (\boldsymbol{x}_L \| \boldsymbol{x}_R), f = (f_L \| f_R)$ and $\overline{\boldsymbol{a}}g = (\overline{\boldsymbol{a}}_L g \| (\dot{a}_\ell \overline{\boldsymbol{a}}_L)g)$ and computes and sends the following to $\mathcal{V}$:
   (a) $A_1 = \mathrm{COM}_{\overline{\boldsymbol{a}}_R}(\boldsymbol{x}_L), A_2 = \mathrm{COM}_{\overline{\boldsymbol{a}}_L}(\boldsymbol{x}_R)$
   (b) $B_1 = \mathrm{COM}^{\mathbb{G}}_{\overline{\boldsymbol{a}}_R}(k_L), B_2 = \mathrm{COM}^{\mathbb{G}}_{\overline{\boldsymbol{a}}_L}(k_R)$
   (c) $y_1 = f_R(\boldsymbol{x}_L), y_2 = f_L(\boldsymbol{x}_R)$
2. $\mathcal{V}$ samples $c \leftarrow_R \mathbb{F}_q$ and sends c to $\mathcal{P}$
3. $\mathcal{P}$ sets $\boldsymbol{x}' = \boldsymbol{x}_L + c\boldsymbol{x}_R$, $f' = cf_L \circ f_R$, $g' = (c + \dot{a}_\ell)g$ and implicitly sets $\dot{\boldsymbol{a}}' = (\dot{a}_1, \ldots, \dot{a}_{\ell-1})$ and $\overline{\boldsymbol{a}} = \overline{\boldsymbol{a}}_L$. Note that this also implicity sets $k' = k_L \circ ck_R$.
4. $\mathcal{P}$ sends g' to $\mathcal{V}$ and $\mathcal{V}$ checks the following, proceeds to step 5 if it holds, and aborts otherwise
$$e\left(\frac{g'}{cg}, H\right) = e\left(g, \dot{a}_\ell H\right)$$
5. $\mathcal{P}$ and $\mathcal{V}$ both compute the following :
$$P' = A_1 + cP + c^2 A_2,\ Q' = B_1 + cQ + c^2 B_2,\ y' = y_1 + cy + c^2 y_2$$
6. If $\boldsymbol{x}' \notin \mathbb{F}_q^2$: $\mathcal{P}$ runs PoK Π_1 to prove knowledge of $\boldsymbol{x}', f'$ such that $\mathrm{COM}_{\overline{\boldsymbol{a}}'}(\boldsymbol{x}') = P'$, $\mathrm{COM}^{\mathbb{G}}_{\overline{\boldsymbol{a}}'}(k') = Q'$ and $f'(\boldsymbol{x}') = y'$.
   Hence, $\mathcal{P}$ and $\mathcal{V}$ run the protocol Π_1 with updated common parameters (P', Q', y', g'), prover's input $(\overline{\boldsymbol{a}}'(g'), \boldsymbol{x}', f')$, and verifier's input $(\dot{\boldsymbol{a}}'H)$ for $(P', Q', y'; \boldsymbol{x}') \in \mathcal{R}_{\mathrm{CH}}$

Figure 3.10: Protocol $\Pi_{\mathsf{1\text{-}hom}}$ for relation $\mathcal{R}_{\mathrm{CH}}$

7. If $\boldsymbol{x}' \in \mathbb{F}_q^2$:

   (a) $\mathcal{P}$ sends $\boldsymbol{x}', f'$ to $\mathcal{V}$

   (b) $\mathcal{V}$ computes $k' = \mathsf{rev}(f')$ and checks the following :

$$\mathrm{COM}_{\vec{\boldsymbol{a}}'}(\boldsymbol{x}') = P' \ \wedge \ \mathrm{COM}^{\mathbb{G}}_{\vec{\boldsymbol{a}}'}(k') = Q' \ \wedge \ f'(\boldsymbol{x}') = y'$$

   and outputs 1 if it holds, and outputs 0 otherwise.

Figure 3.10: Protocol $\Pi_{\mathsf{1\text{-}hom}}$ for relation $\mathcal{R}_{\mathrm{CH}}$

$[\ell], \ell = \log n$. *It is perfectly complete and computationally special sound. It incurs total communication of* $3 \log n$ $\mathbb{G}_1$ *elements,* $2 \log n + 2$ $\mathbb{G}_2$ *elements,* $2 \log n$ $\mathbb{G}_T$ *elements, and* $\log n + 2$ *field elements.*

We note that since we run the protocol $\Pi_{\mathsf{1\text{-}hom}}$ as an alternative for steps 4 and 5 of $\Pi_{\mathsf{0\text{-}hom}}$ to avoid having to send a linear-sized vector, $\Pi_{\mathsf{1\text{-}hom}}$ does not require zero-knowledge property as the final message of the protocol $\Pi_{\mathsf{0\text{-}hom}}$ is intended to be sent in clear. The compressed sigma protocol for proving knowledge of homomorphism on a committed vector is given by the compressed protocol $\Pi_{c\text{-}\mathsf{hom}}$, which is defined by $\Pi_{c\text{-}\mathsf{hom}} = \Pi_{\mathsf{1\text{-}hom}} \circ \Pi_{\mathsf{0\text{-}hom}}$. The compressed protocol for relation $\mathcal{R}$ is given by $\Pi_{c\text{-}\mathsf{hom}}$, whose communication and computational complexities are dominated by that of $\Pi_{\mathsf{1\text{-}hom}}$, and hence we obtain a designated verifier succinct argument of knowledge for the relation $\mathcal{R}$.

Proof of Knowledge of k-out-of-n discrete logarithms. The fundamental contribution of [9] of proving Proof of Knowledge of k-out-of-n discrete logarithms (Protocol 3 of [9]) relies on its ability to provide a compressed sigma protocol for opening a general homomorphism as a building block in a black box manner, and our techniques show how to do this with a succinct verifier. We expect that by relying on their techniques to amortize the protocol for opening multiple homomorphisms (which is done by using a challenge provided by the verifier to perform the check on a random linear combination of the homomorphisms) which is then deployed as a black box for the protocol to obtain proof of knowledge of k-out-of-n discrete logarithms, we can obtain a succinct verifier version of the proof in [9].

3.5.4 Compressed Σ-Protocol for Opening General Homomorphisms

We now extend our protocol to opening homomorphisms on committed vectors with coefficients in multiple groups. We believe that using our protocols in applications of CSP to Threshold Signature Schemes and circuit zero-knowledge protocols with bilinear gates [11] will result in

analogs with succinct verifier after an appropriate preprocessing phase.

We first describe the Σ-Protocol of [11], while using our updated commitment scheme with logarithmic verification, for proving knowledge of a witness $\boldsymbol{x}$ which opens a public homomorphism f to a public element y and opens the known commitment $\mathrm{COM}^{\mathbb{G}}$ to a public element P, i.e. $y = f(\boldsymbol{x})$ and $P = \mathrm{COM}^{\mathbb{G}}(\boldsymbol{x}, \gamma)$. Here, we assume $\boldsymbol{x} \in \mathbb{G}_S = \mathbb{F}_q^{n_0} \times \mathbb{G}_1^{n_1} \times \mathbb{G}_k^{n_k}$, and we have access to a homomorphic commitment scheme $\mathrm{COM}^{\mathbb{G}} : \mathbb{G}_S \times \mathbb{F}_q^r \mapsto \mathbb{G}_{\mathcal{C}}$, and a public homomorphism $f : \mathbb{G}_S \mapsto \mathbb{F}_q \times \mathbb{G}_1 \times \cdots \times \mathbb{G}_k$. The relation is given by $\mathcal{R} = \{(P, f, y; \boldsymbol{x}, \gamma) : P = \mathrm{COM}^{\mathbb{G}}(\boldsymbol{x}, \gamma), y = f(\boldsymbol{x})\}$, and the POK $\Pi_{\mathsf{0\text{-}gen\text{-}hom}}$ for $\mathcal{R}$ is in Fig 3.11).

Parameters

- Common parameters : $P = \mathrm{COM}^{\mathbb{G}}(\boldsymbol{x}, \gamma), y = f(\boldsymbol{x})$
- $\mathcal{P}$'s input : $(\boldsymbol{x}, \gamma)$

Protocol

1. $\mathcal{P}$ samples $\boldsymbol{r} \leftarrow_R \mathbb{G}_S, \rho \leftarrow_R \mathbb{F}_q$
2. $\mathcal{P}$ computes $A = \mathrm{COM}^{\mathbb{G}}(\boldsymbol{r}, \rho), t = f(\boldsymbol{r})$ and sends it to $\mathcal{V}$
3. $\mathcal{V}$ samples $c \leftarrow_R \mathbb{F}_q$ and sends it to $\mathcal{P}$
4. $\mathcal{P}$ computes $\boldsymbol{z} = c\boldsymbol{x} + \boldsymbol{r}$ and $\phi = c\gamma + \rho$ and sends it to $\mathcal{V}$
5. $\mathcal{V}$ checks if $\mathrm{COM}^{\mathbb{G}}(\boldsymbol{z}, \phi) = A + cP$ and $f(\boldsymbol{z}) = cy + t$, outputs 1 if it holds, outputs 0 otherwise.

Figure 3.11: PoK $\Pi_{\mathsf{0\text{-}gen\text{-}hom}}$ for relation $\mathcal{R}$ [11]

In protocol $\Pi_{\mathsf{0\text{-}gen\text{-}hom}}$, we note that step 4 renders the communication complexity linear, and that along with step 5 makes the verifier's complexity linear. We now reduce the complexities by running a compressing protocol $\Pi_{\mathsf{1\text{-}gen\text{-}hom}}$ where we compress while relying on the compatibility of compression provided by the compactness of the commitment scheme for committing to the elements of the groups $\mathbb{F}_q, \mathbb{G}_1, \ldots, \mathbb{G}_k$. In $\Pi_{\mathsf{1\text{-}gen\text{-}hom}}$, compress the part of co-domain of $\mathrm{COM}^{\mathbb{G}}$ which is compact, we parse $\mathrm{COM}^{\mathbb{G}}$ as COM_1 and COM_2, where COM_1 contains the compressible (compact) co-domain of the commitment, and COM_2 contains the incompressible (non-compact) co-domain of the commitment. Hence, COM_1 is a compact commitment scheme and COM_2 is not (takes n-dimensional element to $n+1$-dimensional element). Hence, for some r_1 and r_2 such that $r_1 + r_2 = r$, we have

$$\mathrm{COM}^{\mathbb{G}} : \mathbb{F}_q^{n_0} \times \mathbb{G}_1^{n_1} \times \mathbb{G}_k^{n_k} \times \mathbb{F}_q^r \mapsto \mathbb{G}_{\mathcal{C}}$$

$$\mathrm{COM}_1 : \mathbb{F}_q^{n_0} \times \mathbb{G}_1^{n_1} \times \mathbb{G}_{k-1}^{n_{k-1}} \times \mathbb{F}_q^{r_1} \mapsto \mathbb{G}_{\mathcal{C}_1}$$
$$\mathrm{COM}_2 : \mathbb{G}_k^{n_k} \times \mathbb{F}_q^{r_2} \mapsto \mathbb{G}_{\mathcal{C}_2}$$

where size of $\mathbb{G}_{\mathcal{C}_1}$ is independent of the input dimensions in the domain, and size of $\mathbb{G}_{\mathcal{C}_1}$ is dependent on the input dimensions in the domain (increases by one with respect to the input dimensions in the domain).

We now implement the aforementioned idea by parsing the the witness $\boldsymbol{x}$ as $\boldsymbol{x} = (\boldsymbol{x}_S, \boldsymbol{x}_T)$ where $\boldsymbol{x}_S = (\boldsymbol{x}_0, \ldots, \boldsymbol{x}_{k-1})$ contains the compressible co-domain of the commitment, and $\boldsymbol{x}_T = \boldsymbol{x}_k$ contains the incompressible co-domain of the commitment. Since we want the verifier complexity to be sublinear, we also provide commitments to the homomorphism by treating the homomorphism description as a vector containing elements of $\mathbb{F}_q, \mathbb{G}_1, \ldots, \mathbb{G}_k$. While we commit to the homomorphism f, we parse f as $f = (f_S, f_T)$, where f_S contains the compressible co-domain of the commitment, and f_T contains the incompressible co-domain of the commitment.

Notation for $\Pi_{\text{1-gen-hom}}$. We denote the commitment key for COM_1 which commits to elements of $\mathbb{F}_q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_k$ by $\mathsf{ck}_0, \ldots, \mathsf{ck}_{k-1}$. For example, for $(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{x}_S; \gamma) \in \mathbb{F}_q^{n_0} \times \mathbb{G}_1^{n_1} \times \mathbb{G}_2^{n_2}$, we have $\mathrm{COM}_1(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{x}_S) = \langle \boldsymbol{g}, \boldsymbol{x} \rangle + e(\boldsymbol{y}, \mathbf{H}) + e(\mathbf{G}, \boldsymbol{x}_S) \in \mathbb{G}_T^2$, we set $\mathsf{ck}_0 = \boldsymbol{g}$, $\mathsf{ck}_1 = \mathbf{H}$, $\mathsf{ck}_2 = \mathbf{G}$, and $(\dot{\mathsf{ck}}_0, \dot{\mathsf{ck}}_1, \dot{\mathsf{ck}}_2)$ is the verification key. For example, in $\Pi_{\text{1-hom}}$ described in section 3.5.3, ck denotes the commitment key held by the prover $\mathsf{ck} = g^{\overline{\boldsymbol{a}}}$ and $\dot{\mathsf{ck}}$ denotes the randomness encoded in the other group held by the verifier $\dot{\mathsf{ck}} = H^{\dot{\boldsymbol{a}}}$ where $n = 2^\ell, \dot{\boldsymbol{a}} = (\dot{a}_1, \ldots, \dot{a}_\ell)$, $\overline{\boldsymbol{a}} = \left(\prod_{i=1}^{\ell} \dot{a}_i^{b_i}\right)_{b_i \in \{0,1\}}$.

$$\begin{aligned}\mathcal{R}'_{CH} = \{(P, Q, y; \boldsymbol{x}, f) : \boldsymbol{x} = (\boldsymbol{x}_S, \boldsymbol{x}_T), y = (y_1, y_2, y_3, y_4), f = (f_S, f_T),\\ y_1 = f_S(\boldsymbol{x}_S), y_2 = f_S(\boldsymbol{x}_T), y_3 = f_T(\boldsymbol{x}_S), y_4 = f_T(\boldsymbol{x}_T),\\ Q = \mathrm{COM}_1(\mathsf{rev}(f_S)), P = (P_1, P_2), P_1 = \mathrm{COM}_1(\boldsymbol{x}_S), P_2 = \mathrm{COM}_2(\boldsymbol{x}_T)\}\end{aligned}$$

We present the PoK $\Pi_{\text{1-gen-hom}}$ for $\mathcal{R}'_{CH}$ in Fig 3.12.

Theorem 3.12 *$\Pi_{\text{1-hom}}$ is a $(k_1, \ldots, k_\ell)$-move protocol for relation $\mathcal{R}_{CH}$, where $k_i = 3$, $\forall i \in [\ell], \ell = \log m$, $m = \max_{i=0}^{k-1} n_i$. It is perfectly complete and computationally special sound. It incurs total communication of $O(\log m)$ source (compressible) group elements (including $\mathbb{F}_q$), $O(\log m + n_k)$ target group elements.*

Similar to earlier protocols, we aim to run the protocol $\Pi_{\text{1-gen-hom}}$ as an alternative for steps 4 and 5 of $\Pi_{\text{0-gen-hom}}$ to avoid having to send a linear-sized vector, $\Pi_{\text{1-gen-hom}}$ does not require zero-knowledge property as the final message of the protocol $\Pi_{\text{0-gen-hom}}$ is intended to be sent in

Parameters

- **Common parameters** : $P, Q, y, \mathsf{ck}_{0,1}, \mathsf{ck}_{1,1}, \ldots, \mathsf{ck}_{k-1,1}$ (where $\mathsf{ck}_{i,1}$ is the first element of ck_i, $i = 0, \ldots, k-1$)
  - $\boldsymbol{x} = (\boldsymbol{x}_S, \boldsymbol{x}_T), y = (y_1, y_2, y_3, y_4), f = (f_S, f_T)$,
  - $y_1 = f_S(\boldsymbol{x}_S), y_2 = f_S(\boldsymbol{x}_T), y_3 = f_T(\boldsymbol{x}_S), y_4 = f_T(\boldsymbol{x}_T)$,
  - $Q = \mathrm{COM}_1(\mathsf{rev}(f_S)), P = (P_1, P_2), P_1 = \mathrm{COM}_1(\boldsymbol{x}_S), P_2 = \mathrm{COM}_2(\boldsymbol{x}_T)$
- $\mathcal{P}$**'s input** : $\mathsf{ck}_0, \mathsf{ck}_1, \ldots, \mathsf{ck}_{k-1}, \boldsymbol{x} = (\boldsymbol{x}_S, \boldsymbol{x}_T), \boldsymbol{x}_S = (\boldsymbol{x}_0, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_{k-1}), \boldsymbol{x}_T = (\boldsymbol{x}_k), f = (f_S, f_T)$
- $\mathcal{V}$**'s input** : $\dot{\mathsf{ck}}_0, \dot{\mathsf{ck}}_1, \ldots, \dot{\mathsf{ck}}_{k-1}$

Protocol

1. $\mathcal{P}$ parses $\boldsymbol{x}_i = (\boldsymbol{x}_{i,L} \| \boldsymbol{x}_{i,R})$, for $i = 0, \ldots, k$, $\boldsymbol{x}_{S,\alpha} = (\boldsymbol{x}_{0,\alpha}, \ldots, \boldsymbol{x}_{k-1,\alpha})$, and $\boldsymbol{x}_{T,\alpha} = (\boldsymbol{x}_{k,\alpha})$ for $\alpha = L, R$, and $f_S = (f_{S,L} \| f_{S,R}), f_T = (f_{T,L} \| f_{T,R})$.
2. Similarly, $\mathcal{P}$ parses the commitment keys for $\boldsymbol{x}_S$ and f_S as $\mathsf{ck}_S = (\mathsf{ck}_0, \ldots, \mathsf{ck}_{k-1})$ and commitment keys for $\boldsymbol{x}_T$ and f_T as ck_T.
3. $\mathcal{P}$ sets $k = \mathsf{rev}(f_S)$ and computes the following :
   (a) $A_1 = \mathrm{COM}_1(0, \boldsymbol{x}_{S,L}), A_2 = \mathrm{COM}_1(\boldsymbol{x}_{S,R}, 0)$
   (b) $B_1 = \mathrm{COM}_1(0, k_L), B_2 = \mathrm{COM}_1(k_R, 0)$
   (c) $a_1 = f_{S_R}(\boldsymbol{x}_{S,L}), a_2 = f_{S_L}(\boldsymbol{x}_{S,R})$
   (d) $b_1 = f_{S_R}(\boldsymbol{x}_{T,L}), b_2 = f_{S_L}(\boldsymbol{x}_{T,R})$
   (e) $d_1 = f_{T_R}(\boldsymbol{x}_{S,L}), d_2 = f_{T_L}(\boldsymbol{x}_{S,R})$
4. $\mathcal{P}$ sends the computed values a_2, b_2, A_1, A_2, B_1 and B_2 to $\mathcal{V}$
5. $\mathcal{V}$ samples $c \leftarrow_R \mathbb{F}_q$ and sends it to $\mathcal{P}$
6. $\mathcal{P}$ sets the updated commitment key as $\mathsf{ck}'_i = c \cdot \mathsf{ck}_{i,L} + \mathsf{ck}_{i,R}$ for all $i = 0, 1, \ldots, k-1$
7. $\mathcal{P}$ sends the first element of all updated commitment keys $\mathsf{ck}'_{i,1}$ to $\mathcal{V}$, and $\mathcal{V}$ checks the following for each $\mathsf{ck}_{i,1}$, proceeds to step 8 if it holds, and aborts otherwise

$$e\left(\frac{\mathsf{ck}'_{i,1}}{\mathsf{ck}^c_{i,1}}, \mathsf{gen}_i\right) = e\left(\mathsf{ck}_{i,1}, \dot{\mathsf{ck}}_{i,\ell}\right), \quad i \in \{0, 1, \ldots, k-1\}$$

   where gen_i is the generator of the group containing $\dot{\mathsf{ck}}_i$.

Figure 3.12: PoK $\Pi_{\mathsf{1\text{-}gen\text{-}hom}}$ for relation $\mathcal{R}'_{CH}$

8. $\mathcal{P}$ sets $\boldsymbol{x}'_S = \boldsymbol{x}_{S,L} + c\boldsymbol{x}_{S,R}$, $f'_S = cf_{S,L} + f_{S,R}$, and implicitly updates the randomness for each updated commitment key $\dot{\mathsf{ck}}'_i$ by dropping the last element $\dot{\mathsf{ck}}_{i,\ell}$ from $\dot{\mathsf{ck}}_i$.

9. $\mathcal{P}$ and $\mathcal{V}$ both compute the following :

   (a) $P'_1 = A_1 + cP_1 + c^2A_2$, $Q' = B_1 + cQ + c^2B_2$

   (b) $y'_1 = a_1 + cy_1 + c^2a_2, y'_4 = y_4$,

   (c) $y'_2 = b_1 + cy_2 + c^2b_2$, $y'_3 = d_1 + cy_3 + c^2d_2$

10. If $\boldsymbol{x}'_S$ contains more than 2 elements from any group : $\mathcal{P}$ and $\mathcal{V}$ run the protocol $\Pi_{\mathsf{1\text{-}gen\text{-}hom}}$ with updated common parameters $(P', Q', y'), P' = (P'_1, P_2), y' = (y'_1, y'_2, y'_3, y'_4)$, prover's input $(\mathsf{ck}'_0, \mathsf{ck}'_1, \ldots, \mathsf{ck}'_{k-1}, \boldsymbol{x}' = (\boldsymbol{x}'_S, \boldsymbol{x}_T), f' = (f'_S, f_T)$, and verifier's input $(\mathsf{ck}_{0,1}, \mathsf{ck}_{1,1}, \ldots, \mathsf{ck}_{k-1,1}, \dot{\mathsf{ck}}_1, \ldots, \dot{\mathsf{ck}}_{k-1})$ for $(P', Q', y'; \boldsymbol{x}', f') \in \mathcal{R}'_{CH}$

11. Otherwise :

    (a) $\mathcal{P}$ sends $\boldsymbol{x}' = (\boldsymbol{x}'_S, \boldsymbol{x}_T), f' = (f'_S, f_T)$ to $\mathcal{V}$

    (b) $\mathcal{V}$ computes $k' = \mathsf{rev}(f'_S)$ and checks the following where COM'_1 is the commitment with updated commitment keys $\mathsf{ck}'_i, i = 0, \ldots, k-1$

        i. $\mathrm{COM}'_1(\boldsymbol{x}'_S) = P'_1$, $\mathrm{COM}'_1(k') = Q'$, $\mathrm{COM}_2(\boldsymbol{x}_T) = P_2$

        ii. $f'_S(\boldsymbol{x}'_S) = y'_1$, $f_T(\boldsymbol{x}_T) = y_4$

        iii. $(f'_S, cf'_S)(\boldsymbol{x}_T) = y'_2$, $f_T(c\boldsymbol{x}'_S, \boldsymbol{x}'_S) = y'_3$

        and outputs 1 if it holds, and outputs 0 otherwise.

Figure 3.12: PoK $\Pi_{\mathsf{1\text{-}gen\text{-}hom}}$ for relation $\mathcal{R}'_{CH}$

clear. The compressed sigma protocol for proving knowledge of homomorphism on a committed vector is given by the compressed protocol $\Pi_{c\text{-gen-hom}}$, which is defined by :

$$\Pi_{c\text{-gen-hom}} = \Pi_{1\text{-gen-hom}} \circ \Pi_{0\text{-gen-hom}}$$

Hence, the compressed protocol for relation $\mathcal{R}$ is given by $\Pi_{c\text{-gen-hom}}$, whose communication and computational complexities are dominated by that of $\Pi_{1\text{-gen-hom}}$, and we obtain a designated verifier succinct argument of knowledge for the relation $\mathcal{R}$.

Chapter 4

Distributed Proof of Knowledge (DPoK) and its Application in Input Authentication

In this chapter[1], we present a new notion of distributed zero-knowledge proofs, which we call *distributed proof of knowledge* (DPoK), that enables a prover to distribute the proof generation to a set of workers which holds the shares of the witness. We also provide an additional notion of *robustness* that helps maintain security even in the presence of dishonest usage of shares by workers. Next, we provide construction for DPoKs for discrete log relation, and algebraic signature schemes like BBS+ [29, 41] and PS [97]. Finally, using our DPoKs for algebraic signature schemes, we provide a compiler that transforms a linear secret-sharing based honest majority MPC to one with input authentication with negligible overhead.

4.1 Introduction

To motivate our distributed zero-knowledge proof, we first start with its potential application through the lens of well-understood primitive of multiparty computation (MPC). Secure MPC [108, 109, 70, 79, 18] allows two or more parties to jointly compute a function of their private inputs, while ensuring input privacy and output correctness (even in the presence of some corrupt parties). Traditional security notions for MPC ensure *output correctness* and *input privacy*, that is, nothing is leaked about the parties' private inputs beyond the (correct) output of the computation. However, no assurance is given about how the parties choose their

[1]This chapter is based on the joint work [55] with Chaya Ganesh, Sikhar Patranabis and Nitin Singh, that appeared in Asiacrypt 2024.

private inputs.

Unfortunately, certain applications of MPC could be sensitive to "ill-formed inputs". Maliciously chosen inputs could either corrupt the output or reveal the output on arbitrary inputs, thus violating the desired real-world security guarantees of an MPC protocol. Such attacks are not captured by traditional MPC security definitions.

Input Authenticity in MPC. There are several real-world applications of MPC where it is important to ensure that the inputs used by parties are *authentic*. If a set of individuals on a job portal wish to compute "industry average compensation" for their expertise and experience in a privacy preserving manner (e.g., services provided by glassdoor), one would want them to input payslips bearing their employers' signature. Similarly, in applications involving hospitals performing joint computations on patient data for treatment efficacy, it is desirable to ensure that the data used is signed by a regulatory authority. Input validation is also of practical relevance in applications of MPC in computation on genomic data [26]. For all of these applications, the traditional MPC security guarantees are clearly inadequate. A natural question that confronts us then is: *how do we ensure that authentic inputs are used in MPC*?

Authentication via Certification. In the real world, data authentication typically involves the data being attested by a relevant *certifying authority*. In our work, we specifically consider applications where an input bearing a signature is considered *authentic* and we can assume the existence of a relevant certifying authority that provides the signature. For instance, employers can act as the certification authority to digitally sign the payslips when parties wish to compute 'industry average compensation' using services like glassdoor, a financial auditor can act as the certification authority to digitally sign the bills of sale when shipping companies wish to compute aggregate statistics on private data, a regulatory authority (like WHO) act as the certification authority to digitally sign the medical records when hospitals wish to perform joint computation over sensitive patient data, and so on. Since the certifying authority cannot be omnipresent to vouch for authenticity of the data, it is increasingly common for individuals to claim this attestation through *digital signatures* that can be verified efficiently. In fact, there exist several digital signature schemes today [40, 29, 97] that allow establishing attestation by a certifying authority while requiring minimal disclosure of attributes, and while maintaining *unlinkability* (several usages of the same credential cannot be linked to the same individual). Unfortunately, such secure mechanisms for authenticating data in the individual context do not translate when computing over data from *multiple* data owners using vanilla MPC protocols (that do not consider input authentication).

Potential Approaches and Pitfalls. A naïve approach would be to incorporate input authentication *as part of the function* to be computed. However, this is practically inefficient. For example, incorporating signature verification as part of the function would entail performing expensive operations such as hashing inside MPC (typically, most signature schemes hash the message), and would also require expressing the algebraic operations underlying signature verification as arithmetic circuits. This significantly blows up the size of the circuit, rendering the resulting MPC protocol practically inefficient.

A more efficient alternative is to have the certifying authority *sign a commitment* (e.g., a Pedersen commitment [94]) to each input, and then have the parties prove that their inputs are those contained inside the public commitments (using customized zero-knowledge proofs). However, this fails to provide *unlinkability*, which is an essential privacy requirement. In particular, one can use the signed commitment to link different protocols where the same input is reused. The alternative would be to get the certifying authority to sign a different commitment for each protocol execution, which again requires the authority to be omnipresent, and is clearly impractical.

Certain prior works [5, 27] proposed using *authenticated secret-sharing* in order to certify inputs to an MPC protocol. However, authenticated secret-sharing only provides stand-alone guarantees about the shares themselves, and additional techniques would be needed to ensure that malicious parties actually use these authenticated shares in the execution of the actual MPC protocol (the details of such techniques are not specified completely in prior works [5, 27]). Ideally, we want a notion that *ties* input authentication into the underlying MPC, thus preventing malicious parties from using inputs different from the authenticated ones.

Our Goal. We aim to *lift* existing MPC protocols into authenticated ones that ensure that an additional predicate is satisfied by each input (for instance, each input is signed by a common certifying authority). We want to achieve such input authentication (i) without changing the underlying MPC protocol, (ii) without representing the predicate as a circuit, (iii) incurring communication overhead that is succinct in the size of the inputs (which are typically large for the applications we consider), and (iv) maintaining unlinkability. These requirements immediately preclude prior approaches requiring the authentication relation to be expressed as a circuit [30, 76], as well as the natural approach based on signed public commitments outlined above, which lacks unlinkability.

4.1.1 Our Contributions

In this work, we study *authenticated MPC*. We present the first *generic compiler* than efficiently augments existing MPC protocols to additionally ensure that each input has a valid attestation

(in the form of a digital signature) from a relevant certifying authority, while retaining both practical efficiency and unlinkability. We illustrate the compatibility of our proposed approach with popularly used privacy-preserving verifiable attestation mechanisms based on digital signatures such as BBS+ [29, 13] and PS [97]. Towards this goal, we put forth a notion of distributed (zero-knowledge) proof of knowledge that is of independent interest.

Distributed Proof of Knowledge (DPoK). In Section 4.3, we put forth a notion of a *distributed proof of knowledge* (abbreviated as (DPoK)). A DPoK works in a setting with multiple provers and a single verifier, where the witness is secret shared among the provers. Concretely, for a relation $\mathcal{R}$ and an instance-witness pair $(x, w) \in \mathcal{R}$, the verifier holds the (public) instance x, and each prover holds a share w_i of the (secret) witness w such that $w = \mathsf{Reconstruct}(w_1, \ldots, w_n)$. We also assume a restricted communication model: (i) the provers do not communicate with each other, and (ii) the verifier communicates only via a broadcast channel and is public-coin (this facilitates *public verifiability*, which is used crucially in our eventual solution for authenticated MPC). Our definition of DPoK may thus be viewed a natural distributed analogue of honest-verifier public-coin protocols.

Robust Complete DPoK. Our basic DPoK definition does not prevent malicious provers from disrupting protocol execution, and only provides *security with abort*. To tackle this, we introduce a stronger notion of *robust completeness* for a DPoK, which additionally provides tolerance against abort in the presence of (a potentially smaller number of) maliciously corrupt provers. Looking ahead, using robust complete DPoKs allows us to achieve authenticated MPC protocols with stronger security guarantees.

DPoK for Discrete Log. In Section 4.3, we also construct a DPoK for the discrete logarithm relation, where the witness (the discrete log of a publicly known group element) is secret-shared (using Shamir secret sharing) across multiple provers. Notably, our construction achieves: (i) *succinct* communication (logarithmic in the size of the witness), and (ii) robust completeness (which ensures that the protocol accepts even in the presence of up to $n/3$ malicious provers, where provers only holds shares to the correct witness). For succinct communication, we use techniques due to Attema et al. [8] to *compress* the communication complexity of our protocol from linear to logarithmic in the size of the witness. We realize robust completeness via error-correction *in the exponents* of group elements. To this end, we leverage results from low degree testing used in prior works to construct efficient zkSNARKs (such as in [4, 22]). While achieving robust completeness is straightforward if we do not care about succinctness (and vice versa), the main technical novelty of our construction is to achieve *both* properties simultaneously.

In Section 4.5, we present a generalization of the above DPoK for discrete log that works with *any threshold linear secret sharing scheme*. In this generalized version, we characterize the corruption threshold for robust completeness in terms of the minimum distance of the linear code associated with the threshold linear secret sharing scheme. As an example, we derive concrete bounds on the corruption threshold for the popularly used *replicated secret sharing* scheme.

DPoKs for Algebraically Structured Signatures. Our DPoK for discrete log can be used to build a DPoK for any digital signature scheme where the associated proof of knowledge of a signature can be modeled as a proof of knowledge of the opening of a Pedersen commitment. We present specific instances of this general approach for signature schemes that are algebraically compatible, namely BBS+ [29, 13, 41][1] (detailed in Section 4.4) and PS [97] (detailed in Section 4.7.3). These signature schemes are popular candidates for applications such as verifiable credentials for self-sovereign digital identity. While these signature schemes natively support efficient (albeit non-distributed) zero-knowledge proofs of knowledge of a valid message-signature pair, our work introduces the first practically efficient DPoKs for these signature schemes that are both succinct and robust complete. Our techniques are modular, and we believe that they can be extended to yield DPoKs for other algebraically structured signatures such as [38], as well as algebraic relations of interest for other applications.

Round Efficient DPoKs in the ROM. The above definitions and constructions of DPoKs are in the standard model. In Section 4.6, we formally define *round efficient* DPoKs in the random oracle model (ROM). This definition is based on the Fiat-Shamir heuristic [59], using which we transform a DPoK (with number of rounds logarithmic in the size of the witness) into a round efficient DPoK (having constant number of rounds). Under this definition, we present round efficient versions of our DPoK constructions for discrete log and algebraically structured signatures; these protocols achieve the same robust completeness and succinct communication guarantees as the original protocols, albeit in the ROM.

Authenticated MPC. We now expand upon our main contribution, namely *authenticated MPC*. Informally, we consider a notion of input authenticity for MPC where each input is certified using a valid signature from a certification authority. This is standard in applications where a publicly known certifying authority (external to the MPC protocol) signs an input to certify that the input satisfies certain properties[2]. We build upon our DPoKs for BBS+

[1] There are standardization efforts for using BBS+ signatures in verifiable credentials for Web 3.0, leading to a recent RFC draft [87].

[2] Our techniques extend to other notions of authenticity such as proving that the inputs open publicly known commitments.

and PS signatures to propose a generic compiler that transforms any (threshold linear) secret-sharing based maliciously secure honest-majority MPC protocol into its *authenticated* MPC version. Our compiler yields the first practically efficient MPC protocols that satisfy an ideal notion of input authenticity while preserving practical efficiency and unlinkability. We note that our compiler incurs *negligible communication overhead* over the original MPC protocol. For simplicity, our ideal functionality and subsequent protocols are described assuming a common signature authority for all inputs. The more general case involving multiple signing authorities also follows with minor modifications without incurring any loss of efficiency.

Ideal Functionality for Authenticated MPC. In Section 4.8, we formalize the above notion for authenticated MPC via an ideal functionality $\mathcal{F}_{\mathsf{MPC}}^{\mathsf{auth}}$ that works as follows. The parties send their inputs x_i and signature σ_i on x_i to $\mathcal{F}_{\mathsf{MPC}}^{\mathsf{auth}}$ for $i \in [n]$. The functionality $\mathcal{F}_{\mathsf{MPC}}^{\mathsf{auth}}$ then checks if σ_i is a valid signature on x_i for all $i \in [n]$. For each $j \in [n]$ such that σ_j is not a valid signature on x_j, $\mathcal{F}_{\mathsf{MPC}}^{\mathsf{auth}}$ sends $(\mathtt{abort}, P_j)$ to all of the parties. Otherwise it computes $y = f(x_1, \ldots, x_n)$ and outputs y to all of the parties.

We note that our ideal functionality ties input authentication into the underlying MPC, thus preventing malicious parties from using different inputs as compared to the authenticated ones. The prior works [5, 27] only provide stand-alone guarantees about the authenticated shares themselves, and would require additional techniques to ensure that these authenticated shares are then used in the execution of the actual MPC protocol which are currently not considered. We further note that our ideal functionality already captures unlinkability, since the adversary does not learn any additional information about the authenticated input (beyond the function output) that might allow it to correlate the usage of the same input-signature pair across multiple executions. This rules out solutions based on signing public commitments to inputs, which trivially violate unlinkability.

Compiler for Authenticated MPC. In Section 4.8, we present a compiler that transforms any Shamir secret-sharing based maliciously secure honest-majority MPC protocol Π into its *authenticated* MPC version Π' that securely realizes the above ideal functionality $\mathcal{F}_{\mathsf{MPC}}^{\mathsf{auth}}$, where each input is authenticated using a BBS+ signature. Our compiler builds upon our DPoK for BBS+ signatures from Section 4.4. In Section 4.7.3, we present an analogous compiler for input authentication using PS signatures, which builds upon our DPoK for PS signatures. In both cases, the compiled protocol Π' inherits the security of Π as long as the inputs are authentic (by definition, we abort if this is not the case)[1]. If Π guarantees security with identifiable abort,

[1]In some applications, it is acceptable to continue computation on default inputs instead of aborting when authentication fails.

then the same holds for Π'. If Π achieves guaranteed output delivery, then so does Π' (albeit for a corruption threshold $t < n/3$) – this crucially uses the *robust completeness* property of the underlying DPoKs.

Generalization and Extensions. We note that our approach works in general for: (a) any (threshold linear) secret-sharing based MPC protocol, and (b) any signature scheme such that the associated proof of knowledge can be modeled as a proof of knowledge of the opening of a Pedersen commitment (such as CL signatures [38] and PS signatures [97]). Our DPoK-based approach also offers the flexibility of extending our compiler to support other notions of input authentication, beyond proving knowledge of signatures. In particular, one can build upon our approach to prove a wider class of expressive predicates over secret-shared inputs, thus catering to a wide range of applications with diverse proof requirements (e.g., federated learning). For instance, each party can publish a commitment to its input at the beginning of the authenticated MPC protocol, and then use our DPoK-based framework to prove the following simultaneously: (i) the secret-shared input is signed by a certifying authority (this follows from the basic compiler), (ii) the secret-shared input is a valid opening to the published commitment, and (iii) the opening to the commitment satisfies a certain predicate. Note that, if a different application requires new/additional properties to be checked, the aforementioned approach avoids the need to involve the certifying authority each time. Similarly, it maintains unlinkability since a fresh commitment is used for each protocol execution, while the DPoK allows keeping the signature from the certifying authority private.

4.1.2 Technical Overview

In this section, we provide a brief overview of our techniques. We begin by outlining ideas to distribute a well-known protocol for proving knowledge of discrete logarithm of a public group element. This relation will be at the core of expressive algebraic relations that we will consider later.

Proof of Knowledge of Discrete Log. Let $\mathbb{G}$ be a group of prime order p. Given $x \in \mathbb{G}$, recall Schnorr's protocol [100, 101] for proving knowledge of discrete logarithm w such that $x = g^w$ for some generator g (here (g, x) is public and w is the secret witness). Let $(\mathcal{P}^1, \mathcal{P}^2, \mathcal{V})$ be the protocol where we denote by $\mathcal{P}^1$ and $\mathcal{P}^2$ the algorithms that compute, the prover's first message $a = g^\alpha$ for random $\alpha \in \mathbb{F}_p$, and the prover's last message (response) $z = \alpha + cw$, respectively, where c is the challenge from the space $\{0,1\}^l$ for some length l. Let $\mathcal{V}$ be the algorithm that takes x, transcript $\tau = (a, c, z)$ and accepts iff $g^z = ax^c$.

DPoK for Discrete Log. In order to *distribute* the above protocol, we begin by assuming n provers $\mathcal{P}_i$ who each hold a share w_i such that $w = w_1 + \cdots + w_n \pmod{p}$. Now, each prover runs Σ with their respective shares in parallel[1]. That is, $\mathcal{P}_i$ runs $\mathcal{P}^1$, broadcasts $a_i = g^{\alpha_i}$, receives challenge c from $\mathcal{V}$, and runs $\mathcal{P}^2$ and broadcasts z_i. The transcript is $\tau = (a_1, \ldots, a_n, c, z_1, \ldots, z_n)$, and the verifier accepts iff $g^{\Sigma z_i} = \prod a_i x^c = \prod_i a_i x^c$. This holds since $g^{\Sigma z_i} = g^{\Sigma(\alpha_i + c w_i)} = \prod_i a_i x^c$.

This idea generalizes to any linear secret sharing scheme, and also extends to other relations. For instance, to prove knowledge of representation of a vector of discrete logarithms with respect to public generators. In our final construction we use additional ideas like randomization of the first message of each $\mathcal{P}_i$ via a sharing of 0 in order to ensure zero-knowledge. This DPoK has communication complexity linear in the size of the witness. To achieve succinctness, we instead use as a starting point a compressed sigma protocol [8] in order to achieve a distributed protocol with logarithmic communication complexity (see Section 4.3.2 for details).

Robust Completeness. While the ideas described above result in protocols that are zero-knowledge and sound against a malicious adversary controlling up to t parties, completeness is guaranteed only if all the provers follow the protocol. However, in the distributed setting, a stronger, but natural notion is a *robust* completeness property where completeness holds as long as the shares reconstruct a valid witness, even if some provers are malicious. The main technical challenge in achieving robust completeness for a distributed proof is to retain succinctness. Our key technical novelty is to achieve both robustness and succinctness *simultaneously* via ideas from low-degree testing. We achieve this by identifying and discarding corrupt shares. At a high level, the provers commit to their shares and then reveal a certain linear form determined by the challenge over their shares. Given a challenge $\boldsymbol{c} \in \mathbb{F}_p^m$, each $\mathcal{P}_i$ broadcasts $z_i = \langle \boldsymbol{c}, \mathbf{w}_i \rangle$. In the honest case, these opened linear forms are expected to be a sharing of the same linear form on the reconstructed witness: $\mathbf{z} = (z_1, \ldots, z_n)$ recombine to z where $z = \langle \boldsymbol{c}, \mathbf{w} \rangle$. The verifier error-corrects the received $\mathbf{z}'$ to the nearest codeword, and identifies the erroneous positions. By assumption our corruption threshold is smaller than half the minimum distance of the code, so the erroneous positions clearly come from corrupt provers. Can some corrupt provers strategically introduce errors in individual shares so that they "cancel out" in the inner product with $\boldsymbol{c}$? We lean on coding theoretic result (Lemma 4.2) for linear codes to claim that such a prover only succeeds with negligible probability. Finally, having identified the corrupt messages, we can reconstruct the claimed commitment in the exponent using commitments

[1] This is a simplified description; in our actual protocol Π_{dlog} (Section 4.1), there are no parallel sessions, each instance uses a random share, ensuring that we do not reuse the shares, and in the FS-compiled version $\Pi_{\mathsf{dlog}}^{\mathsf{FS}}$ (Section 4.4), parties send non-interactive proofs instead of sending the first-messages separately in parallel. We note that ROS attacks [24] in the context of concurrent signatures are therefore inapplicable in our setting. See also Section 4.1.3.1 for a more detailed discussion.

of honest shares (now identified). We need more details around this core idea to ensure the protocol is zero-knowledge (see Section 4.3.2 for a complete treatment).

DPoKs for Algebraically Structured Signatures. It turns out that the above approach can be naturally generalized to obtain a DPoK for the opening of a Pedersen commitment [95]. We use this observation as a starting point to realize DPoKs for algebraically structured signatures such as BBS+ [29, 13, 41] and PS [97], which naturally admit proofs of knowledge that can be cast as proving knowledge of openings of Pedersen commitments. As a core technical contribution, we introduce a modified proof of knowledge for the BBS+ signature scheme, which leads to a vastly more efficient DPoK as compared to the straightforward approach of distributing prior proofs of knowledge for BBS+ signatures. We refer to Section 4.4 for details. Analogous DPoK for PS signatures is presented in Section 4.7.3.

Compiler for Authenticated MPC. In order to construct an authenticated MPC protocol, we build upon the above DPoKs for BBS+ and PS signatures. Our compiler reuses the input sharing that is already done as part of an honest-majority MPC protocol. Before proceeding with computation on the shares, the distributed zero-knowledge proof is invoked to verify authenticity, and then the rest of the MPC protocol proceeds. Since the shares of the witness come from a party in the MPC protocol, our robustness property guarantees that if the dealer is honest (that is, a valid witness was shared), then even if some parties acting as provers are dishonest, the authenticity proof goes through (see Section 4.8 for details).

We also note that, while we rely on broadcast for our protocols, all relevant related work on Fully Linear Probabilistically Checkable Proofs (FLPCP) [30] and previous works on authenticated MPC [27, 5, 76] also make use of a broadcast channel. A broadcast channel is not a limitation, and can be implemented using point-to-point channels. In the setting where the number of parties is not too large (as in the applications we consider), the communication overhead to realize broadcast is not prohibitive.

4.1.3 Related Work

We summarize some relevant related work, and compare our compiler with prior approaches for authenticated MPC. We refer to Section 4.1.3.2 for some additional discussions.

Certified Inputs. The earlier works of [15, 78, 112] achieve input validation for the special case of *two*-party computation using garbled circuit (GC) based techniques. Another work [27] constructs MPC with certified inputs, albeit using techniques that are specific to certain MPC protocols [50, 49]. A recent work [5] develops techniques for computing bilinear pairings over secret shared data, which aims to enable signature verification inside MPC for the PS signature

scheme [97]. Both works [5, 27] emulate a functionality similar to authenticated secret-sharing protocol, where shares of an input certified by some certification authority are provided at the end of the protocol execution. While the goal of authenticated MPC has been studied, these works would require additional consistency checks to ensure the consistency of shares used across the protocols for authentication of shares and the underlying MPC execution. Although the explicit details are not provided in the protocol description, we expect the requirement of some consistency check on the MACs to ensure the usage of same shares during authentication protocol and original MPC for function computation. In our work, we formalize this notion of authenticated MPC as an ideal functionality which incorporates the consistency checks, and prove that the proposed constructions realize this. For instance, consider the scenario where a malicious party receives the shares of a certified input held by an honest party, which is done via an authenticated secret-sharing protocol, however while running the MPC itself it chooses to not use the shares received during the previously run authenticated secret-sharing protocol and uses an arbitrarily chosen share instead. The current definitions in [5, 27] fails to safeguard against such an attack and would require additional assumptions to ensure the consistency of shares.

To be precise, the current protocol description of $\Pi_{\mathsf{CertInput}}$ in [5] (Section 5.1) emulates the authenticated secret-sharing, such that at the end of the protocol, if an input corresponds to a valid signature, the shares of that input is available to every party. This protocol first secret-shares the input, then using the shares held by everyone as input invokes another protocol Π_{Verify} to ascertain if the shares obtained in the previous phase corresponds to an input for which there is a valid signature. However, note that only Step 3 of Π_{Verify} considers the shares of the input, which need not be the shares used for running the MPC, unless additional consistency checks using the MACs on the shares are in place. Such details do not explicitly appear in the protocols presented in [5].

The protocols in [27] also follow a similar template based on authenticated secret-sharing. Their techniques consider two specific MPC protocols [49, 50] for input certification. Concretely, Theorem 8 for input certification in [27] ensures that a malicious prover cannot feed an input which does not correspond to the valid signature. While it is not explicitly specified in [27] that the commitments to the inputs used for the batch verification of signatures are consistent with the inputs used for the remaining proof of knowledge statements, we assume that this is indeed the case.

In this chapter, we recognize the benefits of having a formal definition to capture the consistency of shares of input used in authentication and the MPC. To this end, we explicitly provide an ideal functionality ensuring the same, and then present a construction satisfying this ideal

functionality. We also avoid the possibility of using different inputs for certification and MPC by enforcing that the honest party shares must completely determine the reconstructed input which is being authenticated. While this observation has not been specified in either of the works, this specific restriction would also ensure that the consistency of shares holds for constructions in [5, 27] as well.

We use efficient compressed DPoKs for signature verification instead of verifying signatures inside the MPC protocol, hence differing from both [5] and [27] in terms of techniques used and properties achieved. In particular, our compiler is modular, fully generic (works in a plug-and-play manner with any threshold linear secret sharing based MPC protocol), and avoids the (potentially expensive) protocol-specific techniques and preprocessing requirements that are inherent to [5, 27]. Our compiler also enables stronger security guarantees as compared to abort security, namely identifiable abort (and even full security/guaranteed output delivery in certain cases), which neither [5] nor [27] achieves.

Distributed Zero-knowledge. Various notions of distributed zero-knowledge have appeared in literature. The notion of distributed interactive proofs appeared in [95], in the context of relations describing the verification of signatures, where the signature is public and the secret key is shared. The notion in [107] considers a distributed prover in order to improve prover efficiency, but the witness is still held by one entity. In Feta [16], the distributed notion is a generalization of designated verifier to the threshold setting where a set of verifiers jointly verify the correctness of the proof. Prio [47] proposes secret shared non-interactive proofs where again, there is a single prover and many verifiers.

Our formulation of DPoKs also differs from recent works on distributed zkSNARKs [102, 91, 51], where the focus is on jointly computing a non-interactive publicly verifiable proof (with specific focus on Groth16 [74], Plonk [62] and Marlin [45]). Their constructions require additional interaction among the workers over private channels. On the other hand, we consider DPoKs where all interaction with the verifier takes place over a public broadcast channel. We also study the notion of *robust completeness* that guarantees completion even in the presence of malicious behavior while ensuring succinct proof size, which was not achieved in prior works. Note that distributed zkSNARKs fundamentally differ in their objective. DPOKs prove that the given shares (e.g., the one used for MPC) reconstruct a valid witness, whereas distributed zkSNARKs do not certify a given sharing.

A recent work on distributed zkSNARKs, called zkSaaS [68], considers a monolithic prover that aims outsources proof generation to (untrusted) servers in a privacy-preserving manner for increased efficiency. However, we target applications that require proving (algebraically structured) relations involving an already secret-shared witness. Plugging it naively does not work

as a replacement for our proposed compiler since it would not ensure that the same input shares are used consistently in the authentication protocol and the core MPC. Additionally, similar to the distributed proofs with multiple verifier, [68] also requires expressing the algebraically structured relations as circuits, which is inefficient for the algebraic relations considered in our work.

Proofs on Secret-shared Data. Notions of zero-knowledge proofs on distributed data is explored in recent works [30, 76, 16]. The former work proposes the abstraction of a fully linear PCP (FLPCP) where each verifier only has access to a share of the statement, and the latter work is based on MPC-in-the-head paradigm. The techniques of distributed verification [30, 76, 16] assumes the relations to be represented as an arithmetic circuit, whereas our DPoKs consider algebraic relations whose circuit respresentation is prohibitively expensive. Additionally, distributed verifier paradigm considers a designated prover who knows entire witness to create a proof oracle, which is verified in distributed fashion, while DPoKs do not require a prover which knows the entire witness. For example for proof of $g^x h^y = C$ wheres x and y belongs to different parties, a DPoK will succeed as long as provers have valid shares of x and y.

Our observation is that algebraic relations like discrete log is naturally distributed witness relation. A public statement and shared witness is better suited for algebraic relations, and our distributed zero-knowledge definition captures such natural relations. Since the focus of our work is on concrete efficiency (prover overhead, communication overhead), we take advantage of the algebraic nature of the relation to design concretely efficient DPoKs by modeling the witness as being distributed and statement being public. In this approach, we expect rich classes of protocols (compressed sigma protocols, Bulletproofs etc that avoid circuit representation for several useful relations) to be amenable to be distributed under our definition. In addition, [30] provides sublinear communication only for special circuits (like degree 2) and the circuits of interest for us are unlikely to have this structure.

We also note that [30] does not consider the robustness property. We put forth the robustness notion that guarantees that the protocol runs to completion even in the presence of malicious parties (when the prover is honest). This property is indeed important for our applications, as this means that the compiled authenticated MPC protocol can identify malicious parties in the authentication stage. The distributed completeness guarantees of [16] considers robustness, however its protocol execution incurs communication cost linear in the size of the circuit in the offline phase. However, [16] does not allow aggregation of multiple instances of authentication of input into one execution of the underlying distributed protocol, which we support efficiently.

Finally, the motivating application for [30] is compiling passive security to active security,

and therefore the statements that show up – like the next message function of the protocol -– have a low degree circuit representation. We consider the authenticated input application where our relations of interest are algebraic in nature (e.g. verification of an algebraic signature scheme) and admit efficient sigma protocols.

4.1.3.1 Resistance to Known Vulnerabilities

Here, we present a discussion on why our proposed DPoK protocols and our compiler for authenticated MPC resist some known attacks and insecurities of ZKP protocols in practice.

Resistance to ROS Attacks. In [24], the authors presented an algorithm for solving ROS (Random inhomogeneities in a Overdetermined Solvable system of linear equations) mod p in polynomial time for $\ell > \log p$ dimensions, which leads to the ROS attack on certain advanced families of digital signatures which involve computations over secret shares. However, the ROS attack does not apply to our proposed DPoK protocols. In particular, note that the ROS attack only works when: (i) there are more than $\log p$ parallel sessions for the same shares, (ii) the adversary chooses its first message after seeing all of the other first messages from the honest parties, (iii) the adversary chooses the challenge.

The ROS attack is not applicable for our protocols as: (i) there are no parallel sessions in our protocols, (ii) each protocol is instantiated using the output of (the randomized) Share algorithm of the underlying secret sharing scheme ($\mathsf{Share}, \mathsf{Reconstruct}$), thereby ensuring that we do not reuse the shares across sessions, and in the round-efficient versions of our proposed protocols: (iii) the parties send non-interactive proofs instead of sending the first-messages separately (see $\Pi^{\mathsf{FS}}_{\mathsf{dlog}}$ in Section 4.6), and finally (iv) the challenge is not chosen by the adversary (verifier); it is determined by performing a hash of the available public transcript.

Resistance to OSNARK-related Vulnerabilities. In [60], the authors provide a study of when SNARKs are insecure in the presence of certain oracles (in particular, the knowledge soundness guarantees do not hold in such settings since the extraction fails). As defined in [60], an OSNARK is a SNARK that guarantees extraction even in presence of an oracle for the prover. We note here that the negative result for the existence of OSNARKs, as outlined in [60], does not provide a general impossibility result, since it only applies either to SNARKs where the prover has access to oracles with secret states (such that the extractor does not have access to these states), and for standard-model SNARKs. We note that the attack does not apply: (i) to SNARKs in the ROM, and (ii) when the extractor is black-box in the adversary. Fiat-Shamir transformed Sigma protocols are also known to satisfy black-box *simulation-extractability*, i.e., knowledge soundness holds even in the presence of proof oracles [65, 66]. Analogously, our Fiat-Shamir transformed round-efficient proofs of knowledge are simulation-extractable in the

random oracle model, as we establish through formal proofs of security. In particular, there are no other oracles with secret states in our setting. We emphasize that signatures are already independently obtained by the parties on their inputs, and signing or signature-oracles are not included as part of our authenticated MPC protocols.

4.1.3.2 Comparison of our approach with Anonymity Sets

In this section, we present some additional discussion on the comparison of our DPoK-based approach with the approach of signing public commitments. Previously we discussed an alternative approach for achieving authenticated MPC based on having the certifying authority sign commitments to the private inputs of the parties, and then having the parties prove during the MPC protocol that their inputs indeed open the public commitments. As discussed earlier, this approach trivially violates the desired property of *unlinkability*, since one can link the usage of the same input across different protocol executions from the public commitments. A possible fix is to use *anonymity sets*: all commitments to the inputs are made publicly available, and instead of explicitly identifying which commitment is linked with each input, the party provides a zero-knowledge proof of knowledge of an opening of one of the several signed commitments, along with a proof of membership of the commitment in the public set.

While this is a plausible solution, we believe that full unlinkability (as modeled implicitly by our ideal functionality and realized by our proposed solution) is a better solution that anonymity. First of all, the anonymity set needs to be large enough for any reasonable notion of unlinkability to hold; however, this is an issue as the size of the statement to prove increases with the size of the set, leading to additional overheads for the proof of knowledge. Additionally, one has to prove that a commitment used is a member of the accumulated set, requiring additional proofs of membership. Finally, in practical applications, it is unclear which entity will create and maintain this set accumulator: for instance, if a new data set to be used as input for a computation is signed by an authority, it must be added to the accumulator. This leads to additional overheads for accumulator maintenance.

4.2 Preliminaries

In this section, we introduce notations and present preliminary background material.

Notation. We write $x \leftarrow_R \chi$ to represent that an element x is sampled uniformly at random from a set/distribution $\mathcal{X}$. The output x of a deterministic algorithm $\mathcal{A}$ is denoted by $x = \mathcal{A}$ and the output x' of a randomized algorithm $\mathcal{A}'$ is denoted by $x' \leftarrow_R \mathcal{A}'$. For $n \in \mathbb{N}$, let $[n]$ denote the set $\{1, \ldots, n\}$. For $a, b \in \mathbb{N}$ such that $a, b \geq 1$, we denote by $[a, b]$ the set of integers lying between a and b (both inclusive). We refer to $\lambda \in \mathbb{N}$ as the security parameter, and denote

by $\mathsf{poly}(\lambda)$ and $\mathsf{negl}(\lambda)$ any generic (unspecified) polynomial function and negligible function in λ, respectively. A function $f : \mathbb{N} \to \mathbb{N}$ is said to be negligible in λ if for every positive polynomial p, $f(\lambda) < 1/p(\lambda)$ when λ is sufficiently large.

Let $\mathbb{G}$ be a group and $\mathbb{F}_p$ denote the field of prime order p. We use boldface to denote vectors. Let $\boldsymbol{g} = (g_1, \ldots, g_n) \in \mathbb{G}^n$ and $\boldsymbol{x} = (x_1, \ldots, x_n) \in \mathbb{F}_p^n$, then $\boldsymbol{g}^{\boldsymbol{x}}$ is defined by $\boldsymbol{g}^{\boldsymbol{x}} = g_1^{x_1} \cdots g_n^{x_n}$. For $\boldsymbol{g} = (g_1, \ldots, g_n) \in \mathbb{G}^n$ and $\boldsymbol{h} = (h_1, \ldots, h_n) \in \mathbb{G}^n$, $\boldsymbol{g} \circ \boldsymbol{h}$ denotes component-wise multiplication, and is defined by $\boldsymbol{g} \circ \boldsymbol{h} = (g_1 h_1, \ldots, g_n h_n)$. For $\boldsymbol{g} = (g_1, \ldots, g_n) \in \mathbb{G}^n$ and $\boldsymbol{x} = (x_1, \ldots, x_n) \in \mathbb{F}_p^n$, $\boldsymbol{g}_L$ (similarly, $\boldsymbol{x}_L$) denotes the left half of the vector $\boldsymbol{g}(\boldsymbol{x})$ and $\boldsymbol{g}_R(\boldsymbol{x}_R)$ denotes the right half, such that $\boldsymbol{g} = \boldsymbol{g}_L \| \boldsymbol{g}_R$ and $\boldsymbol{x} = \boldsymbol{x}_L \| \boldsymbol{x}_R$.

4.2.1 Secure Multiparty Computation

Secure multiparty computation (MPC) enables n mutually distrustful parties to jointly compute a given function f over their private inputs, where i^{th} party P_i holds the private input x_i, while maintaining the properties of *correctness* and *privacy*. Intuitively, the property of *correctness* ensures that the output computed at the end of the MPC execution is correct, and the property of *privacy* ensures that nothing beyond the output of the computation is revealed. The distrust amongst parties is captured by the existence of an adversary $\mathcal{A}$ that can corrupt the behaviour of up to t parties. This work only considers static corruption, where the set of corrupted parties is decided before the protocol execution. Note that we only consider computationally bounded adversaries for MPC execution in this thesis.

In MPC, we can consider semi-honest adversaries, which honestly follows the protocol specification and attempts to learn more information from the protocol execution. In essence, the privacy against such adversaries are formalized by the existence of a PPT simulator that, given only the output and inputs of the corrupt parties, can generate the view of the corrupt parties in a real execution. However, in this work, we consider malicious adversaries that can arbitrarily deviate from the protocol execution. The privacy guarantees against malicious adversaries are formalized in [7] by comparing a real protocol execution to an ideal model, which consists of the parties sending their private inputs to an incorruptible trusted third party and receiving the output.

Let f be a function. Formally, Π is known as a secure MPC protocol for the function f, if for every input $\boldsymbol{x}$ of f, the following properties of correctness and privacy hold.

Definition 16 (Correctness) *Let* $\mathsf{out}_\Pi(\boldsymbol{x})$ *be the output of the protocol execution* Π *on input* $\boldsymbol{x}$*, then we have* $\mathsf{out}_\Pi(\boldsymbol{x}) = f(\boldsymbol{x})$.

Definition 17 (Privacy) *Let* $\boldsymbol{x} = (x_1, \ldots, x_n)$ *and* P_i *denote the* i^{th} *party with input* x_i*. Let*

$\mathcal{A}$ *denote the adversary with auxiliary input* z, $\mathcal{C}$ *denote the set of indices of parties corrupted by the adversary* $\mathcal{A}$, *such that* $|\mathcal{C}| \leq t$. *The executions in the real and ideal models are defined below.*

- ***Execution in the Real Model:*** *In real model, the parties run the protocol* Π. *The adversary* $\mathcal{A}$ *controls* $\{P_i : i \in \mathcal{C}\}$, *and is assumed to be rushing, i.e., in every given round it can see the messages sent by the honest parties before determining the messages sent by the corrupt parties. Let* $\mathsf{Real}_{\Pi,\mathcal{A}(z),\mathcal{C}}(\boldsymbol{x})$ *denote the random variable that consists of the view of the adversary* $\mathcal{A}$ *(which includes the input and the internal randomness of the corrupt parties, along with the messages received by them) and the output of the honest parties.*

- ***Execution in the Ideal Model:*** *The ideal model consists of the honest parties, an incorruptible trusted party and an ideal PPT adversary* Sim, *which controls the same set of corrupted parties* $\{P_i : i \in \mathcal{C}\}$. *The honest parties send their inputs to the trusted party. The ideal adversary* Sim *receives the auxiliary input* z *and sees the input of the corrupted parties* $\{x_i : i \in \mathcal{C}\}$. *Note that* Sim *can substitute any* x_i *with any arbitrary* x'_i, *for* $i \in \mathcal{C}$, *such that* $|x_i| = |x'_i|$ *holds. Let* x'_i *be the inputs received by the trusted party, which it then uses to compute the output* $(y_1, \ldots, y_n) = f(x'_1, \ldots, x'_n)$. *The simulator may send* abort *to the trusted party, in which case the trusted party sends the abort message* $\perp$ *to all parties, otherwise it sends* y_i *to* P_i, *for all* $i \in [n]$. *Let* $\mathsf{Ideal}_{f,\mathsf{Sim}(z),\mathcal{C}}(\boldsymbol{x})$ *denote the output of the ideal adversary* Sim *(which includes its auxiliary input, the initial inputs of the corrupt parties and the messages received by them) and the output of the honest parties.*

The n*-party protocol* Π *is said to satisfy the property of privacy, if for every* $\mathcal{A}$ *in the real model, there exists a PPT simulator* Sim *in the ideal model, such that the following computational indistinguishability holds for every* $\boldsymbol{x}$ *and* z, *where* $z, x_i \in \{0,1\}^*$ *for all* $i \in [n]$.

$$\left\{\mathsf{Ideal}_{f,\mathsf{Sim}(z),\mathcal{C}}(\boldsymbol{x})\right\} \cong_c \left\{\mathsf{Real}_{\Pi,\mathcal{A}(z),\mathcal{C}}(\boldsymbol{x})\right\}$$

Security of MPC. In an MPC execution having security with *abort*, the adversary may abort the protocol upon receiving the output, and deprive the honest parties of the opportunity to obtain the output. In an MPC execution with *identifiable abort* security, although the adversary may abort the protocol upon receiving the output and deprive the honest parties of the opportunity to obtain the output, but in such a scenario a non-empty set of corrupt parties will be identified by the honest parties. In an MPC execution with *guaranteed output delivery* (GOD) security, all the honest parties are guaranteed to receive a correct output irrespective of any adversarial strategy.

Capacity of Corruption. In secure MPC, we generally assume that the adversary can only corrupt atmost t number of parties in the protocol execution, where t is a publicly known threshold. This is known as a *threshold* adversary, which is the focus of our application. The literature of MPC also considers *non-threshold* adversaries that induces corruption based on a publicly known adversarial structure.

If the threshold adversary can corrupt any number of parties during the n-party MPC execution, it is known as the *dishonest majority* setting and is denoted by $t < n$. On the other hand, if the threshold adversary is restricted to corrupt less than half of the total number of parties, it is known as the *honest majority* setting and is denoted by $t < n/2$. Additonally within honest majority, we can restrict the threshold adversary further to $t < n/3$, which requires more than two-third of the total parties to behave honestly.

4.2.2 Threshold Secret Sharing

For ease of exposition we define a special case of *threshold linear secret sharing* scheme below. For concreteness, the reader may assume a (t, n) Shamir Secret Sharing. The more general definition appears in Section 4.5.

Definition 4.1 (Threshold Secret Sharing) *A (t, n) threshold secret sharing over finite field $\mathbb{F}$ consists of algorithms $(\mathsf{Share}, \mathsf{Reconstruct})$ as described below:*

- Share *is a randomized algorithm that on input $s \in \mathbb{F}$ samples a vector $(s_1, \ldots, s_n) \in \mathbb{F}^n$, which we denote as $(s_1, \ldots, s_n) \leftarrow_R \mathsf{Share}(s)$.*

- $\mathsf{Reconstruct}$ *is a deterministic algorithm that takes a set $\mathcal{I} \subseteq [n]$, $|\mathcal{I}| \geq t$, a vector $(s_1, \ldots, s_{|\mathcal{I}|})$ and outputs*
  $s = \mathsf{Reconstruct}((s_1, \ldots, s_{|\mathcal{I}|}), \mathcal{I}) \in \mathbb{F}$. We will often omit the argument $\mathcal{I}$ when it is clear from the context.

A threshold secret sharing scheme satisfies the following properties:

- **Correctness**: *For every $s \in \mathbb{F}$, any $(s_1, \ldots, s_n) \leftarrow_R \mathsf{Share}(s)$ and any subset $\mathcal{I} = \{i_1, \ldots, i_q\} \subseteq [n]$ with $q > t$, we have $\mathsf{Reconstruct}((s_{i_1}, \ldots, s_{i_q}), \mathcal{I}) = s$.*

- **Privacy**: *For every $s \in \mathbb{F}$, any $(s_1, \ldots, s_n) \leftarrow_R \mathsf{Share}(s)$ and any subset $\mathcal{I} = \{i_1, \ldots, i_q\} \subseteq [n]$ with $q \leq t$, the tuple $(s_{i_1}, \ldots, s_{i_q})$ is information-theoretically independent of s.*

A concrete (t, n) sharing scheme over a finite field $\mathbb{F}$, known as the Shamir Secret Sharing is realized by choosing a set of distinct points $\boldsymbol{\eta} = \{\eta_1, \ldots, \eta_n\}$ in $\mathbb{F} \setminus \{0\}$. Then given $s \in \mathbb{F}$, the

Share algorithm uniformly samples a polynomial p of degree at most t such that $p(0) = s$ and outputs $(p(\eta_1), \ldots, p(\eta_n))$ as the shares. The Reconstruct algorithm essentially reconstructs the value $s = p(0)$ using Lagrangian interpolation. We canonically extend the Share and Reconstruct algorithms to vectors by applying them component-wise.

Definition 4.2 (Linear Code) *An $[n, k, d]$-linear code $\mathcal{L}$ over field $\mathbb{F}$ is a k-dimensional subspace of $\mathbb{F}^n$ such that $d = \min\{\Delta(\boldsymbol{x}, \boldsymbol{y}) : \boldsymbol{x}, \boldsymbol{y} \in \mathcal{L}, \boldsymbol{x} \neq \boldsymbol{y}\}$. Here Δ denotes the hamming distance between two vectors.*

We say that an $m \times n$ matrix $\mathbf{P} \in \mathcal{L}^m$ if each row of $\mathbf{P}$ is a vector in $\mathcal{L}$. We also overload the distance function Δ over matrices; for matrices $\mathbf{P}, \mathbf{Q} \in \mathbb{F}^{m \times n}$, we define $\Delta(\mathbf{P}, \mathbf{Q})$ to be the number of columns in which $\mathbf{P}$ and $\mathbf{Q}$ differ. For a matrix $\mathbf{P} \in \mathbb{F}^{m \times n}$ and an $[n, k, d]$ linear code $\mathcal{L}$ over $\mathbb{F}$, we define $\Delta(\mathbf{P}, \mathcal{L}^m)$ to be minimum value of $\Delta(\mathbf{P}, \mathbf{Q})$ where $\mathbf{Q} \in \mathcal{L}^m$.

Definition 4.3 (Reed Solomon code) *For any finite field $\mathbb{F}$, any n-length vector $\boldsymbol{\eta} = (\eta_1, \ldots, \eta_n) \in \mathbb{F}^n$ of distinct elements of $\mathbb{F}$ and integer $k < n$, the* Reed Solomon Code *$\mathcal{RS}_{n,k,\boldsymbol{\eta}}$ is an $[n, k, n - k + 1]$ linear code consisting of vectors $\big(p(\eta_1), \ldots, p(\eta_n)\big)$ where p is a polynomial of degree at most $k - 1$ over $\mathbb{F}$.*

We note that shares output by (t, n) Shamir secret sharing are vectors in $[n, t + 1, n - t]$ Reed Solomon code. We can leverage tests for membership of a vector in a linear code (based on parity-check matrix) to check if a set of shares $\{s_i\}_{i \in \mathcal{H}}$ for $\mathcal{H} \subseteq [n]$ and $|\mathcal{H}| > t$ uniquely determine a shared value s for Shamir Secret Sharing scheme. Below, we formalise the notion of consistent shares and state a lemma to check such shares. In the interest of space, we directly state the results for general $m \in \mathbb{N}$, i.e. when vectors $\boldsymbol{s} \in \mathbb{F}^m$ are shared.

Definition 4.4 (Consistent Shares) *Let $\mathcal{L}$ be the linear code determined by a (t, n) Shamir secret sharing scheme over finite field $\mathbb{F}$. For $m \in \mathbb{N}$, we call a set of shares $\{\boldsymbol{s}_i\}_{i \in \mathcal{H}}$ for $\mathcal{H} \subseteq [n]$ with $|\mathcal{H}| \geq t + 1$ to be $\mathcal{L}^m$-*consistent *if there exists $(\mathbf{v}_1, \ldots, \mathbf{v}_n) \in \mathcal{L}^m$ such that $\boldsymbol{s}_i = \mathbf{v}_i$ for $i \in \mathcal{H}$. In this case $\boldsymbol{s} = \mathsf{Reconstruct}(\mathbf{v}_1, \ldots, \mathbf{v}_n) \in \mathbb{F}^m$ is the unique shared value determined by the shares $\{\boldsymbol{s}_i\}_{i \in \mathcal{H}}$.*

We define the predicate $\mathsf{Consistent} : \mathbb{F}^{\mathcal{H}+1} \to \{0, 1\}$ *as*

$$\mathsf{Consistent}(\{\boldsymbol{s}_i\}_{i \in \mathcal{H}}, \boldsymbol{s}) = \begin{cases} 1, & |\mathcal{H}| \leq t \\ 1, & |\mathcal{H}| > t \wedge \{\boldsymbol{s}_i\}_{i \in \mathcal{H}} \text{ is } \mathcal{L}^m\text{-consistent} \\ & \quad \bigwedge \mathsf{Reconstruct}(\{\boldsymbol{s}_i\}_{i \in \mathcal{H}}) = \boldsymbol{s} \\ 0, & \text{otherwise.} \end{cases}$$

We use this Consistent(.) predicate to determine if a vector $\boldsymbol{s}$ can be a possible candidate which could have been used to generate the set of shares held by the honest parties $\{\boldsymbol{s}_i\}_{i\in\mathcal{H}}$.

Lemma 4.1 *Let $\mathcal{L}$ be the linear code determined by a (t,n) Shamir secret sharing scheme over finite field $\mathbb{F}$. Then for $m\in\mathbb{N}$ and all $\mathcal{H}\subseteq[n]$ with $q=|\mathcal{H}|\geq t+1$, there exists $q\times(n-t)$ matrix $\mathbf{H}_{\mathcal{H}\mathcal{H}}$ over $\mathbb{F}$ such that shares $\{\boldsymbol{s}_i\}_{i\in\mathcal{H}}$ are $\mathcal{L}^m$-consistent and determine the value $\boldsymbol{s}\in\mathbb{F}^m$ if and only if $\boldsymbol{X}\mathbf{H}_{\mathcal{H}}=(\boldsymbol{s},\mathbf{0}^{n-t-1})$ where $\boldsymbol{X}=(\boldsymbol{x}_1,\ldots,\boldsymbol{x}_q)$ is some canonical ordering of $\{\boldsymbol{s}_i\}_{i\in\mathcal{H}}$.*

Proof: We sketch the proof. For a matrix $\mathbf{P}\in\mathcal{L}^m$, we have $\mathbf{PH}=\mathbf{0}^{n-t-1}$ where $\mathbf{H}$ is the parity check matrix for the $[n,t+1,n-t]$ code $\mathcal{L}$. Now for $\mathcal{H}\subseteq[n]$ with $|\mathcal{H}|\geq t+1$, and matrix $\boldsymbol{X}$ determined by $\mathcal{L}^m$-consistent shares $(\boldsymbol{s}_i)_{i\in\mathcal{H}}$, there exists a matrix $\mathbf{T}_{\mathcal{H}}$ such that $\boldsymbol{X}\mathbf{T}_{\mathcal{H}}\in\mathcal{L}^m$, and hence $\boldsymbol{X}\mathbf{T}_{\mathcal{H}}\mathbf{H}=\mathbf{0}^{n-t-1}$. Thus for $\mathbf{H}_{\mathcal{H}}=[\boldsymbol{k},\mathbf{T}_{\mathcal{H}}\mathbf{H}]$ where $\boldsymbol{k}$ is the column of reconstruction coefficients for the set $\mathcal{H}$, we have $\boldsymbol{X}\mathbf{H}_{\mathcal{H}}=(\boldsymbol{s},\mathbf{0}^{n-t-1})$. □

4.2.3 Proofs of Knowledge

Let $\mathcal{R}$ be a NP-relation and $\mathcal{L}$ be the corresponding NP-language, where $\mathcal{L}=\{x:\exists\, w \text{ such that } (x,w)\in\mathcal{R}\}$. Here, x is called an *instance or statement* and w is called a *witness*. An *interactive proof system* consists of a pair of PPT algorithms $(\mathcal{P},\mathcal{V})$. $\mathcal{P}$, known as the prover algorithm, takes as input an instance $x\in\mathcal{L}$ and its corresponding witness w, and $\mathcal{V}$, known as the verifier algorithm, takes as input an instance x. Given a public instance x, the prover $\mathcal{P}$, convinces the verifier $\mathcal{V}$, that $x\in\mathcal{L}$. At the end of the protocol, based on whether the verifier is convinced by the prover's claim, $\mathcal{V}$ outputs a decision bit. A stronger *proof of knowledge* (PoK)[1] property says that if the verifier is convinced, then the prover knows a witness w such that $(x,w)\in\mathcal{R}$. In this thesis, we consider POKs that satisfy two security properties, namely, *honest-verifier zero-knowledge* (HVZK) and *special-soundness*.

A protocol is said to be *honest-verifier zero-knowledge* (HVZK) if the transcript of messages resulting from a run of the protocol can be simulated by an efficient algorithm without knowledge of the witness. A protocol is said to have *k-special-soundness*, if given k accepting transcripts, an extractor algorithm can output a w' such that $(x,w')\in\mathcal{R}$. Furthermore, a protocol is said to have $(k_1,\ldots,k_\mu)$-*special-soundness* [32], if given a tree of $\prod_{i=1}^{\mu}k_i$ accepting transcripts, the extractor can extract a valid witness. Here, each vertex in the tree of $\prod_{i=1}^{\mu}k_i$ accepting transcripts corresponds to the prover's messages and each edge in the tree corresponds the verifier's challenge, and each root-to-leaf path is a transcript. An interactive protocol is

[1]Note that throughout this thesis, we use *proof* and *argument* interchangeably, but we are only concerned with arguments (proofs with computational soundness) in this thesis.

said to be *public-coin* if the verifier's messages are uniformly random strings. Public-coin protocols can be transformed into non-interactive arguments using the Fiat-Shamir [59] heuristic by deriving the verifier's messages as the output of a Random Oracle. In this work, we consider public-coin protocols.

We refer to Section 4.2.5 for a detailed treatment of non-interactive zero-knowledge (NIZK) proof systems.

4.2.4 BBS+ Signatures and PoK for BBS

In this section, we recall the BBS+ signature scheme [29, 87, 41], and its proof of knowledge. We use the variant of BBS+ signatures and the proof of knowledge from [41], which is the currently adopted variant in the IETF standard for verifiable crendentials [87]. Later, we also describe a slight variant of the BBS+ proof of knowledge from [41], which leads to corresponding distributed proofs with better amortized complexity (i.e, when several DPoKs are required at a time).

Definition 4.5 (BBS+ Signature Scheme [29, 87]) *The BBS+ signature scheme to sign a message of the form* $\boldsymbol{m} = (m_1, \ldots, m_\ell) \in \mathbb{F}_p^\ell$ *consists of a tuple of PPT algorithms* (Setup, KeyGen, Sign, V *described as follows :*

- $\mathsf{Setup}(1^\lambda)$ *: For security parameter* λ*, this algorithm outputs groups* $\mathbb{G}_1, \mathbb{G}_2$*, and* $\mathbb{G}_T$ *of prime order* p*, with an efficient bilinear map* $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ *as part of the public parameters* pp*, along with* g_1 *and* g_2*, which are the generators of groups* $\mathbb{G}_1$ *and* $\mathbb{G}_2$ *respectively.*

- $\mathsf{KeyGen}(\mathsf{pp})$ *: This algorithm samples* $(h_0, \ldots, h_\ell) \leftarrow_R \mathbb{G}_1^{\ell+1}$ *and* $x \leftarrow_R \mathbb{F}_p^*$*, computes* $w = g_2^x$ *and outputs* $(\mathsf{sk}, \mathsf{pk})$*, where* $\mathsf{sk} = x$ *and* $\mathsf{pk} = (g_1, w, h_0, \ldots, h_\ell)$*.*

- $\mathsf{Sign}(\mathsf{sk}, m_1, \ldots, m_\ell)$ *: This algorithm samples* $\beta, s \leftarrow_R \mathbb{F}_p$*, computes* $A = \left(g_1 h_0^s \prod_{i=1}^{\ell} h_i^{m_i}\right)^{\frac{1}{\beta+x}}$ *and outputs* $\sigma = (A, \beta, s)$*.*

- $\mathsf{Verify}(\mathsf{pk}, (m_1, \ldots, m_\ell), \sigma)$ *: This algorithm parses* σ *as* $(\sigma_1, \sigma_2, \sigma_3)$*, and checks*

$$e\left(\sigma_1, w g_2^{\sigma_2}\right) = e\left(g_1 h_0^{\sigma_3} \prod_{i=1}^{\ell} h_i^{m_i},\ g_2\right).$$

If yes, it outputs 1, and outputs 0 otherwise.

Original PoK for BBS+ Signature Scheme. Here, we first recall the proof of knowledge for BBS+ signatures, which was originally proposed in [41], and then present our modified version next.

– **Common Input**: Public Key $\mathsf{pk} = (w, h_0, \ldots, h_\ell)$

– $\mathcal{P}$**'s inputs**: Message $\boldsymbol{m} \in \mathbb{F}_p^\ell$ and signature $\sigma = (A, \beta, s)$ on $\boldsymbol{m}$, with $A = \left(g_1 h_0^s \prod_{i=1}^{\ell} h_i^{m_i}\right)^{\frac{1}{\beta+x}}$.

1. $\mathcal{P}$ samples $r_1 \leftarrow_R \mathbb{F}_p^*$ and computes $A' = A^{r_1}$ and $r_3 = r_1^{-1}$
2. $\mathcal{P}$ computes $\bar{A} = (A')^{-\beta} \cdot b^{r_1}$, where $b = g_1 h_0^s \prod_{i=1}^{\ell} h_i^{m_i}$.
3. $\mathcal{P}$ samples $r_2 \leftarrow_R \mathbb{F}_p$ and computes $d = b^{r_1} \cdot h_0^{-r_2}$ and $s' = s - r_2 \cdot r_3$
4. $\mathcal{P}$ sends $(A', \bar{A}, d)$ to $\mathcal{V}$, and they run a ZKPoK for the relation:

$$(A')^{-\beta} h_0^{r_2} = \bar{A}/d \ \wedge \ d^{-r_3} h_0^{s'} \prod_{i=1}^{\ell} h_i^{m_i} = g_1^{-1}$$

   where $(\boldsymbol{m}, r_2, r_3, \beta, s')$ is the witness.

5. $\mathcal{V}$ checks that $A' \neq 1_{\mathbb{G}_1}, e\left(A', w\right) = e\left(\bar{A}, g_2\right)$, verifies the ZKPoK proof and outputs 1 if all the checks pass, and 0 otherwise.

Modified PoK for BBS+ Signature Scheme. We present our modified proof of knowledge (PoK) for BBS+ signatures, building on the PoK originally proposed in [41], wherein we split the relation $d^{-r_3} h_0^{s'} \prod_{i=1}^{\ell} h_i^{m_i} = g_1^{-1}$ by requiring the prover to equivalently show:

$$d^{-r_3} h_0^{s'-\eta} = C \ \wedge \ h_0^{\eta} \prod_{i=1}^{\ell} h_i^{m_i} = D \ \wedge \ C \cdot D = g_1^{-1}$$

The above decomposition has advantage that the (long) message $\boldsymbol{m}$ appears only with public generators which leads to better aggregation of DPoKs over several messages. The complete modified protocol appears below.

– **Common Input**: Public Key $\mathsf{pk} = (w, h_0, \ldots, h_\ell)$

– $\mathcal{P}$**'s inputs**: Message $\boldsymbol{m} \in \mathbb{F}_p^\ell$ and signature $\sigma = (A, \beta, s)$ on $\boldsymbol{m}$, with $A = \left(g_1 h_0^s \prod_{i=1}^{\ell} h_i^{m_i}\right)^{\frac{1}{\beta+x}}$.

1. $\mathcal{P}$ samples $r_1 \leftarrow_R \mathbb{F}_p^*$ and computes $A' = A^{r_1}$ and $r_3 = r_1^{-1}$
2. $\mathcal{P}$ computes $\bar{A} = (A')^{-\beta} \cdot b^{r_1}$, where $b = g_1 h_0^s \prod_{i=1}^{\ell} h_i^{m_i}$.

3. $\mathcal{P}$ samples $r_2 \leftarrow_R \mathbb{F}_p$ and computes $d = b^{r_1} \cdot h_0^{-r_2}$ and $s' = s - r_2 \cdot r_3$
4. $\mathcal{P}$ samples $\eta \leftarrow_R \mathbb{F}_p$ and sets $C = d^{-v} h_0^{s'-\eta}$, and $D = h_0^{\eta} \prod_{i=1}^{\ell} h_i^{m_i}$.
5. $\mathcal{P}$ sends $(A', \bar{A}, d, C, D)$ to $\mathcal{V}$.
6. $\mathcal{P}$ and $\mathcal{V}$ run a ZKPoK for the discrete-logarithm relation:

$$(A')^{-\beta} h_0^{r_2} = \bar{A}/d \ \wedge \ d^{-r_3} h_0^{s'-\eta} = C \ \wedge \ h_0^{\eta} \prod_{i=1}^{\ell} h_i^{m_i} = D$$

   where $(\boldsymbol{m}, r_2, r_3, \beta, s', \eta)$ is the witness.
7. $\mathcal{V}$ checks that $A' \neq 1_{\mathbb{G}_1}, C \cdot D = g_1^{-1}, e(A', w) = e\left(\bar{A}, g_2\right)$, verifies the ZKPoK proof and outputs 1 if all the checks pass, and 0 otherwise.

4.2.5 Non-Interactive Zero-Knowledge Proofs in the Random Oracle Model

The Fiat-Shamir heuristic [59] transforms a public-coin interactive proof into an non-interactive version in the random oracle model. Given a public-coin proof system $\Pi = (\mathcal{P}, \mathcal{V})$ with r rounds and Ch_i is the challenge space for the ith round. The corresponding non-interactive proof system $\Pi_{\mathsf{FS}} = (\mathsf{Setup}_{\mathsf{FS}}, \mathcal{P}_{\mathsf{FS}}, \mathcal{V}_{\mathsf{FS}})$ is defined as follows.

- $\mathsf{H} \leftarrow_R \mathsf{Setup}_{\mathsf{FS}}(1^{\lambda})$ The setup algorithm for $i \in [1, r]$ samples a function H_i uniformly from a set of all functions that map $\{0,1\}^*$ to Ch_i. Note that this is equivalent to instantiating H_i from a single random oracle via domain separation. We denote by H the set $\{\mathsf{H}_i\}_{i \in [1,r]}$.
- $\pi \leftarrow_R \mathcal{P}_{\mathsf{FS}}{}^{\mathsf{H}}(x, w)$ The prover produces a proof string π on input statement x, and witness w. For each round $i \in [1, r]$, $\mathcal{P}_{\mathsf{FS}}{}^{\mathsf{H}}$ invokes the next message function of the interactive prover $\mathcal{P}(x, w)$ on prior challenge c_{i-1} to get a_i, and obtains the ith round challenge by computing $c_i = \mathsf{H}_i(x, a_1, c_1, \ldots, a_{i-1}, c_{i-1}, a_i)$. Then $\mathcal{P}_{\mathsf{FS}}{}^{\mathsf{H}}$ outputs $\pi = (a_1, c_1, \ldots, a_r, c_r, a_{r+1})$.
- $b \leftarrow_R \mathcal{V}_{\mathsf{FS}}{}^{\mathsf{H}}(x, \pi)$ The verifier on input statement x, and proof string π, outputs a decision bit. $\mathcal{V}_{\mathsf{FS}}{}^{\mathsf{H}}$ outputs $b = 1$, meaning the verifier accepts the proof, iff $\mathcal{V}(x, \pi) = 1$ *and* $c_i = \mathsf{H}_i(x, a_1, c_1, \ldots, c_{i-1}, a_i)$ for all $i \in [1, r]$.

Definition 4.6 (Knowledge soundness in the ROM) *Consider a non-interactive proof system* $\Pi_{\mathsf{FS}} = (\mathsf{Setup}_{\mathsf{FS}}, \mathcal{P}_{\mathsf{FS}}, \mathcal{V}_{\mathsf{FS}})$ *for relation* $\mathcal{R}$. Π_{FS} *is* extractable *with knowledge error* κ :

$\mathbb{N} \times \mathbb{N} \to [0,1]$ *in the random oracle model, if there exists an extractor* Ext *and some polynomial* poly*, such that for any* PPT *adversary* $\mathcal{P}$ *that makes at most* q *queries to* H*, it holds that*

$$\mathrm{ext}(\mathcal{P}, \mathsf{Ext}) \geq \frac{\mathrm{acc}(\mathcal{P}) - \kappa(\lambda, q)}{\mathsf{poly}(\lambda)}$$

and Ext *halts in an expected number of steps that is polynomial in* λ *and* q*, where the probabilities* acc *and* ext *are defined as follows.*

$$\mathrm{acc}(\mathcal{P}) = \Pr\left[\, b = 1 \;\middle|\; \begin{array}{r} \mathsf{H} \leftarrow_R \mathsf{Setup}_{\mathsf{FS}}(1^\lambda); \\ (x, \pi) \leftarrow_R \mathcal{P}^{\mathsf{H}}(\rho); \\ b \leftarrow_R \mathcal{V}_{\mathsf{FS}}{}^{\mathsf{H}}(x, \pi) \end{array} \right]$$

$$\mathrm{ext}(\mathcal{P}, \mathsf{Ext}) = \Pr\left[\, \begin{array}{c} b = 1 \wedge \\ (x, w) \in \mathcal{R} \end{array} \;\middle|\; \begin{array}{r} \mathsf{H} \leftarrow_R \mathsf{Setup}_{\mathsf{FS}}(1^\lambda); \\ (x, \pi) \leftarrow_R \mathcal{P}^{\mathsf{H}}(\rho); \\ b \leftarrow_R \mathcal{V}_{\mathsf{FS}}{}^{\mathsf{H}}(x, \pi); \\ w \leftarrow_R \mathsf{Ext}^{\mathcal{P}}(x, \pi, \rho, \mathcal{Q}_1) \end{array} \right]$$

where $\mathcal{Q}_1 = \{\mathcal{Q}_{1,i}\}_{i \in [1,r]}$ *is the set consisting of pairs corresponding to queries to the random oracle* H *with index* i*, and the response. In the experiment* ext*,* Ext *has oracle access to the next-message function of* $\mathcal{P}$*.*

Zero-knowledge for non-interactive proofs is defined in the explicitly programmable random oracle model where the simulator is allowed to program the random oracle. The zero-knowledge simulator $\mathcal{S}_{\mathsf{FS}}$ is modeled as a stateful algorithm that operates in two modes. In the first mode, $(c_i, \mathsf{st}') \leftarrow \mathcal{S}_{\mathsf{FS}}(1, \mathsf{st}, x, i)$ handles random oracle calls to H_i on input x. In the second mode, $(\tilde{\pi}, \mathsf{st}') \leftarrow \mathcal{S}_{\mathsf{FS}}(2, \mathsf{st}, x)$ simulates a valid proof string. We define stateful wrapper oracles.

- $\mathcal{S}_1(t, i)$ denotes the oracle that returns the first output of $\mathcal{S}_{\mathsf{FS}}(1, \mathsf{st}, t, i)$;
- $\mathcal{S}_2(x, w)$ returns the first output of $\mathcal{S}_{\mathsf{FS}}(2, \mathsf{st}, x)$ if $(\mathsf{pp}, x, w) \in \mathcal{R}$ and $\perp$ otherwise; (This is because ZK is defined only for true statements.)

Definition 4.7 (Non-interactive Zero-Knowledge) *A NIZK* $\Pi_{\mathsf{FS}} = (\mathsf{Setup}_{\mathsf{FS}}, \mathcal{P}_{\mathsf{FS}}{}^{\mathsf{H}}, \mathcal{V}_{\mathsf{FS}}{}^{\mathsf{H}})$ *for relation* $\mathcal{R}$ *is* non-interactive zero-knowledge *in the random oracle model, if there exists a PPT simulator* $\mathcal{S}_{\mathsf{FS}} = (\mathcal{S}_1, \mathcal{S}_2)$ *such that for all PPT distinguisher* $\mathcal{D}$*, the following is negligible in* λ

$$\left| \Pr\left[\mathcal{D}^{\mathsf{H}, \mathcal{P}_{\mathsf{FS}}{}^{\mathsf{H}}}(1^\lambda) = 1 \;:\; \mathsf{H} \leftarrow_R \mathsf{Setup}_{\mathsf{FS}}(1^\lambda)\right] - \Pr\left[\mathcal{D}^{\mathcal{S}_1, \mathcal{S}_2}(1^\lambda) = 1 \;:\; \mathsf{H} \leftarrow_R \mathsf{Setup}_{\mathsf{FS}}(1^\lambda)\right] \right|$$

where both $\mathcal{P}_{\mathsf{FS}}{}^{\mathsf{H}}(x,w)$ *and* $\mathcal{S}_2$ *return* $\perp$ *if* $(x,w) \notin \mathcal{R}$.

Additionally, given a HVZK simulator $\mathcal{S}$ for Π, we can construct a NIZK simulator $\mathcal{S}_{\mathsf{FS}}$ for Π_{FS} as follows.

- On query (x,i) with mode 1, $\mathcal{S}_{\mathsf{FS}}(1,\mathsf{st},x,i)$ lazily samples a lookup table $\mathcal{Q}_{1,i}$ maintained in state st. It checks whether $\mathcal{Q}_{1,i}[x]$ is already defined; if yes, it returns the previously assigned value; otherwise it returns and sets a fresh random value c_i sampled from Ch_i.
- On query x with mode 2, $\mathcal{S}_{\mathsf{FS}}(2,\mathsf{st},x)$ calls the HVZK simulator $\mathcal{S}$ of Π to obtain a simulated transcript $\tilde{\pi} = (a_1, c_1, \ldots, a_r, c_r, a_{r+1})$. Then, it programs the tables such that $\mathcal{Q}_{1,1}[x, a_1] := c_1, \ldots, \mathcal{Q}_{1,r}[x, a_1, c_1, \ldots, a_r] := c_r$. If any of the table entries has been already defined $\mathcal{S}_{\mathsf{FS}}$ aborts, which happens with negligible probability under the assumption that a_1 has high min-entropy.

4.2.6 Compressed Sigma Protocols

We recall the sigma protocol for vectors, for proving knowledge of discrete log $\boldsymbol{s} \in \mathbb{F}_p^\ell$ of a vector of group elements $\boldsymbol{g}$, such that $\boldsymbol{g}^{\boldsymbol{s}} = z$. Here, a prover $\mathcal{P}$ with knowledge of the secret vector $\boldsymbol{s}$, samples a random vector of scalars $\boldsymbol{r} \leftarrow_R \mathbb{F}_p^\ell$, and sends $\alpha = \boldsymbol{g}^{\boldsymbol{r}}$ to the verifier $\mathcal{V}$. $\mathcal{V}$ then samples a challenge $c \leftarrow_R \mathbb{F}_p$ and sends it to $\mathcal{P}$ and in the next round $\mathcal{P}$ replies with $\boldsymbol{x} = c\boldsymbol{s} + \boldsymbol{r}$ where $\mathcal{V}$ checks if $\boldsymbol{g}^{\boldsymbol{x}} = z^c\alpha$. Here, the size of the last message of $\mathcal{P}$ is linear in input size, and hence it makes the proof size linear. We note that, for the proof to be succeed, it suffices to convince the verifier $\mathcal{V}$ that $\mathcal{P}$ knows $\boldsymbol{x}$ such that $\boldsymbol{g}^{\boldsymbol{x}} = z^c\alpha$. Here, we recall the $\log_2 m - 1$ round protocol using the *split and fold* technique [8], which has logarithmic proof size, for proving knowledge of $\boldsymbol{x} \in \mathbb{F}_p^\ell$ such that $\boldsymbol{g}^{\boldsymbol{x}} = y$ where $y = z^c\alpha$:

- **Common input :** $\boldsymbol{g} \in \mathbb{G}^m$, $z \in \mathbb{G}$
- $\mathcal{P}$**'s input :** $\boldsymbol{x} \in \mathbb{F}_p^\ell$

1. $\mathcal{P}$ computes $A = \boldsymbol{g}_R^{\boldsymbol{x}_L}$, $B = \boldsymbol{g}_L^{\boldsymbol{x}_R}$ and sends them to $\mathcal{V}$.
2. $\mathcal{V}$ samples $c \leftarrow_R \mathbb{F}_p$ and sends it to $\mathcal{P}$.
3. $\mathcal{P}$ comutes $\boldsymbol{x}' = \boldsymbol{x}_L + c\boldsymbol{x}_R$.
4. $\mathcal{P}$ and $\mathcal{V}$ independently computes $\boldsymbol{g}' = \boldsymbol{g}_L^c \circ \boldsymbol{g}_R \in \mathbb{G}^{\ell/2}$ and $z' = Ay^cB^{c^2}$.
5. If size$(\boldsymbol{g}') = 2$, $\mathcal{P}$ sends $\boldsymbol{x}'$ to $\mathcal{V}$, else $\mathcal{P}$ and $\mathcal{V}$ repeat the protocol from step 1 with $\boldsymbol{x} = \boldsymbol{x}'$, $\boldsymbol{g} = \boldsymbol{g}'$ and $y = z'$.

where for a vector $\boldsymbol{s}$, $\boldsymbol{s}_L$ denotes the left half of the vector and $\boldsymbol{s}_R$ denote the right half.

The underlying sigma protocol has perfect completeness, special honest-verifier zero-knowledge (SHVZK) and 2-special soundness, and the later protocol has perfect completeness and 3-special soundness at each step of the recursion. Hence, the overall protocol $\mathsf{CSP}\{(z, \boldsymbol{x}) : \boldsymbol{g}^{\boldsymbol{x}} = z\}$ has perfect completeness, SHVZK which comes from the underlying sigma protocol and $(2, k_1, \ldots, k_{(\log_2 \ell - 1)})$-special soundness, where $k_i = 3 \ \forall i \in [\log_2 \ell - 1]$. The protocol can be compiled into a non-interactive argument of knowledge using Fiat-Shamir heuristic [59] in the random oracle model, which we denote by $\mathsf{NIPK}.\mathcal{P}^{\mathsf{RO}}_{\mathsf{FS}}\{(z, \boldsymbol{x}) : \boldsymbol{g}^{\boldsymbol{x}} = z\}$ for the random oracle RO.

4.2.7 Coding Theory

The following coding theoretic result is used to identify malicious behaviour in the distributed proof of knowledge protocol in Section 4.3.2. It has been previously used in construction of zero-knowledge proofs in the interactive oracle setting (e.g [4, 22]), to check that the oracle represents "low degree polynomials".

Lemma 4.2 ([23], Theorem 1.2) *Let $\mathcal{L}$ be an $[n, k, d]$-linear code over finite field $\mathbb{F}$ and let $\mathbf{S}$ be an $m \times n$ matrix over $\mathbb{F}$. Let $e = \Delta(\mathbf{S}, \mathcal{L}^m)$ be such that $e < d/2$. Then for any codeword $\boldsymbol{r} \in \mathcal{L}$, and $\boldsymbol{\gamma}$ sampled uniformly from $\mathbb{F}^m$, we have $\Delta(\boldsymbol{r} + \boldsymbol{\gamma}^T\mathbf{S}, \mathcal{L}) = e$ with probability at least $1 - n/|\mathbb{F}|$. Furthermore, if E denotes the column indices where $\mathbf{S}$ differs from the nearest matrix $\mathbf{Q}$ in $\mathcal{L}^m$, with probability $1 - n/|\mathbb{F}|$ over choice of $\boldsymbol{\gamma}$, the vector $\boldsymbol{r} + \boldsymbol{\gamma}^T\mathbf{S}$ differs from the closest codeword $\mathbf{v} \in \mathcal{L}$ at precisely the positions in E.*

4.3 Distributed Proof of Knowledge

In this section, we formalize the notion of *distributed* proof of knowledge (DPoK) in which multiple provers, each having a share of the witness engage in an interactive protocol with a verifier to convince it that their shares determine a valid witness. The provers do not directly interact with each other, and all the interaction with the verifier takes place over a public broadcast channel.

4.3.1 Defining a DPoK

Definition 4.8 (Distributed Proof of Knowledge) We define n-party *distributed proof of knowledge* for relation generator RGen and a secret-sharing scheme $\mathsf{SSS} = (\mathsf{Share}, \mathsf{Reconstruct})$ by the tuple $\mathsf{DPoK}_{\mathsf{SSS},\mathsf{RGen}} = (\mathsf{Setup}, \Pi)$ where Setup is a PPT algorithm and Π is an interactive

protocol between PPT algorithms $\mathcal{P}$ (prover), $\mathcal{V}$ (verifier) and $\mathcal{W}_1, \ldots, \mathcal{W}_n$ (workers) defined as follows:

- **Setup Phase**: For relation $\mathcal{R} \leftarrow_R \mathsf{RGen}(1^{1^\lambda})$, $\mathsf{Setup}(\mathcal{R})$ outputs public parameters pp as $\mathsf{pp} \leftarrow_R \mathsf{Setup}(\mathcal{R})$. The setup phase is required to be executed only once for a given relation $\mathcal{R}$. We assume $\mathcal{R}$ consists of pairs $(\boldsymbol{x}, \boldsymbol{w})$ where $\boldsymbol{w}$ is parsed as $(\boldsymbol{s}, \boldsymbol{t})$) with $\boldsymbol{s} \in \mathbb{F}^m$. Looking ahead, we partition the witness as $(\boldsymbol{s}, \boldsymbol{t})$ to explicitly specify which parts of the witness later needs to be shared [1].

- **Input Phase**: The prover $\mathcal{P}$ receives $(\boldsymbol{x}, (\boldsymbol{s}, \boldsymbol{t})) \in \mathcal{R}$ as input, while the worker $\mathcal{W}_i$, $i \in [n]$ receives $(\boldsymbol{x}, \boldsymbol{s}_i)$ as input, where $(\boldsymbol{s}_1, \ldots, \boldsymbol{s}_n) \leftarrow_R \mathsf{Share}(\boldsymbol{s})$. All parties receive $\boldsymbol{x}$ as input.

- **Preprocessing Phase**: This is (an optional) phase where the prover $\mathcal{P}$ sends some auxiliary information aux_i to worker $\mathcal{W}_i$ using secure private channels.

- **Interactive Phase**: In this phase, the parties interact using a public broadcast channel according to the protocol Π. The protocol Π is a k-round protocol for some $k \in \mathbb{N}$, with $(\mathsf{pp}, \boldsymbol{x}, \boldsymbol{s}, \boldsymbol{t})$ as $\mathcal{P}$'s input, $(\mathsf{pp}, \boldsymbol{x}, \boldsymbol{s}_i, \mathsf{aux}_i)$ as the input of $\mathcal{W}_i$ and $(\mathsf{pp}, \boldsymbol{x})$ as the input of $\mathcal{V}$. The verifier's message in each round $j \in [k]$ consists of a uniformly sampled challenge $\boldsymbol{c}_j \in \mathbb{F}^{\ell_j}$ for $\ell_j \in \mathbb{N}$. In each round $j \in [k]$, the worker $\mathcal{W}_i$ (resp. the prover $\mathcal{P}$) broadcasts a message $\boldsymbol{m}_{ij}$ (resp., $\boldsymbol{m}_i$) which depends on it's random coins and the messages received in prior rounds (including preprocessing phase).

- **Output Phase**: At the conclusion of k rounds, verifier outputs a bit $b \in \{0, 1\}$ indicating accept (1) or reject (0).

A distributed proof of knowledge $\mathsf{DPoK}_{\mathsf{SSS},\mathsf{RGen}}$ as described above is said to be *t-private*, *ℓ-robust* if the following hold:

- **Completeness**: We say that completeness holds if for all $\mathcal{R} \leftarrow_R \mathsf{RGen}(1^{1^\lambda})$ and $(\boldsymbol{x}, \boldsymbol{s}) \in \mathcal{R}$, the honest execution of all the phases results in 1 being output in the output phase with probability 1.

- **Knowledge-Soundness**: We say that knowledge soundness holds if for any PPT adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, where $\mathcal{A}_2$ corrupts the prover $\mathcal{P}$ and subset of workers $\{\mathcal{W}_i\}_{i \in \mathsf{C}}$ for some

[1] We specify $\boldsymbol{s} \in \mathbb{F}^m$ since our secret sharing works over a finite field. The witness component $\boldsymbol{t}$ need not, in general, be a field element. In fact, in our application, the witness is a message signature pair where the message is in $\mathbb{F}^m$ and the signature is a group element. This group element is not secret shared, yet, the DPOK guarantees extraction of a valid signature message pair.

$\mathsf{C} \subseteq [n]$, there exists an extractor Ext with oracle access to $\mathcal{A}_2$ (recall that the prover and the set of corrupt $\mathcal{W}_i$ are controlled by $\mathcal{A}_2$) such the following probability is negligible.

$$\Pr\left[\begin{array}{c|r} \mathcal{V}_{\mathcal{A},\Pi}(\mathsf{pp},\boldsymbol{x}) = 1 \wedge & \mathcal{R} \leftarrow_R \mathsf{RGen}(1^\lambda) \\ ((\boldsymbol{x},(\boldsymbol{s},\boldsymbol{t})) \notin \mathcal{R} \vee & \mathsf{pp} \leftarrow_R \mathsf{Setup}(\mathcal{R}) \\ \mathsf{Consistent}(\{\boldsymbol{s}_i\}_{i\notin\mathsf{C}},\boldsymbol{s}) = 0) & (\boldsymbol{x},\{\boldsymbol{s}_i\}_{i\notin\mathsf{C}}) \leftarrow_R \mathcal{A}_1(\mathsf{pp}) \\ & (\boldsymbol{s},\boldsymbol{t}) \leftarrow_R \mathsf{Ext}^{\mathcal{A}_2}(\mathsf{pp},\boldsymbol{x},\{\boldsymbol{s}_i\}_{i\notin\mathsf{C}}) \end{array}\right]$$

In the above, $\mathcal{V}_{\mathcal{A},\Pi}(\mathsf{pp},\boldsymbol{x})$ denotes the verifier's output in the protocol Π with its input as $(\mathsf{pp},\boldsymbol{x})$ and $\mathcal{A}$ being the adversary. The extractor takes as input the shares of the honest parties specified by the adversary $\mathcal{A}_1$, and with all but negligible probability extracts a valid witness.

– **Honest Verifier Zero-Knowledge**: We say that a DPoK is honest verifier zero-knowledge if for all $\mathcal{R} \leftarrow_R \mathsf{RGen}(1^{1^\lambda})$, $(\boldsymbol{x},\boldsymbol{s}) \in \mathcal{R}$ and any PPT adversary $\mathcal{A}$ corrupting a set of workers $\{\mathcal{W}_i\}_{i\in\mathsf{C}}$, where $|\mathsf{C}| \leq t$, there exists a PPT simulator Sim such that $\mathsf{View}_{\mathcal{A},\Pi}(\mathsf{pp},\boldsymbol{x})$ is indistinguishable from $\mathsf{Sim}(\mathsf{pp},\boldsymbol{x})$ for $\mathsf{pp} \leftarrow_R \mathsf{Setup}(\mathcal{R})$. Here, the view is given by $\mathsf{View}_{\mathcal{A},\Pi} = \{\boldsymbol{r},(\mathbf{M}_i)_{i\in\mathsf{C}}\}$ where $\boldsymbol{r}$ denotes the internal randomness of $\mathcal{A}$ and $\mathbf{M}_i$ is the set of all messages received by $\mathcal{W}_i$ in Π. We remark that we define honest-verifier zero-knowledge as is standard for public-coin interactive protocols. After Fiat-Shamir compilation into a non-interactive proof, we get full zero-knowledge against a malicious verifier.

– **Robust-Completeness**: We say that robust-completeness holds if for all $\mathcal{R} \leftarrow_R \mathsf{RGen}(1^{1^\lambda})$, $(\boldsymbol{x},\boldsymbol{s}) \in \mathcal{R}$ and any PPT adversary $\mathcal{A}$ corrupting a set of workers $\{\mathcal{W}_i\}_{i\in\mathsf{C}}$, where $|\mathsf{C}| \leq \ell$, $\mathcal{V}_{\mathcal{A},\Pi}(\mathsf{pp},\boldsymbol{x}) = 1$ with overwhelming probability where $\mathsf{pp} \leftarrow_R \mathsf{Setup}(\mathcal{R})$.

Remark 1 *Robust completeness is a stronger notion of completeness in the sense that it holds even if some corrupt workers deviate maliciously from the protocol, as opposed to the standard notion of completeness which only holds if all the workers follow the protocol. Looking ahead, we use robust complete DPoKs to design authenticated MPC protocols that preserve the underlying protocol's resilience against malicious behavior.*

Remark 2 *We assume that the sharing phase is executed before the onset of DPoK, hence the knowledge soundness extractor of DPoK expects honest party shares in order to extract the witness. Since knowledge soundness is supposed to hold against a corrupt prover and some corrupt workers, it is meaningful to say that the adversary breaks knowledge soundness if no extractor can construct corrupt party shares that* ***together with the honest party shares***

determine a valid witness. Note that extractor is required to produce shares of corrupt parties which "explain" the successful outcome of the protocol in conjunction with the shares used by honest parties. Hence, DPoK enables us to certify a given sharing.

Remark 3 *We assume an honest verifier $\mathcal{V}$ for ease of exposition. In Section 4.6, we relax this assumption by transforming any* $\mathsf{DPoK}_{\mathsf{SSS},\mathsf{RGen}}$ *protocol that uses only public-coins and communication over broadcast channels between the workers and the verifier (with no communication among the workers), into a round-efficient version* $\mathsf{RE\text{-}DPoK}_{\mathsf{SSS},\mathsf{RGen}}$ *in the random oracle model, wherein the verifier's challenge is computed using the Fiat-Shamir heuristic [59].*

4.3.2 Robust Complete DPoK for Discrete Log

In this section, we provide a $\mathsf{DPoK}_{\mathsf{SSS},\mathsf{DlogGen}}$ for the discrete log relation based on Shamir Secret Sharing (SSS) [104]. Let DlogGen be a relation generator that on input $(1^{1^\lambda}, 1^\ell)$ outputs $(\mathbb{G}, \boldsymbol{g}, p)$ where p is a 1^λ-bit prime, $\mathbb{G}$ is a cyclic group of order p and $\boldsymbol{g} = (g_1, \ldots, g_\ell) \leftarrow_R \mathbb{G}^\ell$ is a uniformly sampled set of generators. The associated relation $\mathcal{R}^{\mathrm{DL}}$ is defined by $(z, \boldsymbol{s}) \in \mathcal{R}^{\mathrm{DL}}$ if $\boldsymbol{g}^{\boldsymbol{s}} = z$. Let $\mathsf{SSS} = (\mathsf{Share}, \mathsf{Reconstruct})$ denote (t, n) Shamir secret sharing over $\mathbb{F}_p$. Our protocol Π_{dlog} realizing $\mathsf{DPoK}_{\mathsf{SSS},\mathsf{DlogGen}}$ is presented in Figure 4.1.

However, for ease of exposition, we first explain a simpler non-robust version of the protocol, before explaining the robust version. We use an instantiation of compressed sigma protocols (CSP) due to Attema et al. [8] as a black-box (please refer to Section 4.2.6 for more details). We run CSP protocol instances over a broadcast channel, meaning that each worker $\mathcal{W}_i$ (playing the role of the prover of that instance) broadcasts its messages as part of the CSP protocol, and the verifier broadcasts all challenges as well.

Warm-up: Non-robust DPoK for DLOG. We begin by describing a simpler, non-robust version of Π_{dlog} outlined above, which we call $\Pi_{\mathsf{nr\text{-}dlog}}$. Let us consider the scenario where the parties $\mathcal{W}_i$, $i \in [n]$, holds the shares $\boldsymbol{s}_i$ for a secret $\boldsymbol{s}$ such that $(z, \boldsymbol{s}) \in \mathcal{R}^{\mathrm{DL}}$, i.e. $z = \boldsymbol{g}^{\boldsymbol{s}}$. Now note that since $(\boldsymbol{s}_1, \ldots, \boldsymbol{s}_n) \leftarrow_R \boldsymbol{s}$, there exists some publicly known k_i such that $\sum_i k_i \boldsymbol{s}_i = \boldsymbol{s}$. In particular, the protocol $\Pi_{\mathsf{nr\text{-}dlog}}$ executes the following steps:

- **Input Phase:** The prover holds $(z, \boldsymbol{s})$ and each worker $\mathcal{W}_i$ $(i \in [n])$ holds $(z, \boldsymbol{s}_i)$, where $\boldsymbol{s}_i$ are shares of $\boldsymbol{s}$ i.e. $(\boldsymbol{s}_1, \ldots, \boldsymbol{s}_n) \leftarrow_R \mathsf{Share}(\boldsymbol{s})$.

Interactive Phase

- Each worker $\mathcal{W}_i$ $(i \in [n])$ broadcasts a commitment $A_i = \boldsymbol{g}^{\boldsymbol{s}_i}$ to their shares $\boldsymbol{s}_i$, along with a proof of knowledge π_i of its exponent $\boldsymbol{s}_i$ with respect to the associated commitment A_i.

– Thereafter, the verifier checks the following:

  · The proofs π_i (with respect to the broadcast commitment A_i) are valid for all $i \in [n]$.

  · The broadcast A_i and the publicly known z satisfies the relation $z = \prod_i A_i^{k_i}$ for the publicly known reconstruction coefficients $\{k_i : i \in [n]\}$.

Robust DPoK for DLOG. Note that the previously described protocol $\Pi_{\mathsf{nr\text{-}dlog}}$ achieves completeness only if all of the parties participating to produce the proof are honest. To achieve completeness even in the presence of corrupt parties, known as the stronger guarantee of robust completeness, we require error-correction. However the shares that requires error-correction are in the exponent of a publicly known group element and it is known from [96] that error correction is not possible in the exponent. To ensure that error correction is possible in the exponent, we leverage the coding theoretic lemma that states that a random linear combination of a set of error-correcting codes (e.g., Reed-Solomon code) retains the position of errors as long as the number of errors are 'small'. In particular, the protocol Π_{dlog} executes the following steps:

– **Input Phase:** The prover holds $(z, \boldsymbol{s})$ and each worker $\mathcal{W}_i$ $(i \in [n])$ holds $(z, \boldsymbol{s}_i)$, where $\boldsymbol{s}_i$ are shares of $\boldsymbol{s}$ i.e. $(\boldsymbol{s}_1, \ldots, \boldsymbol{s}_n) \leftarrow_R \mathsf{Share}(\boldsymbol{s})$.

– **Preprocessing:** We need an additional preprocessing step for providing robustness. In this phase, before the onset of the interactive phase of the protocol, the prover samples $r \leftarrow_R \mathbb{F}_p$, computes $(r_1, \ldots, r_n) \leftarrow_R \mathsf{Share}(r)$ and sends the share r_i to the worker $\mathcal{W}_i$.

Interactive Phase

– **Commit to Shares:** In the interactive phase, each worker $\mathcal{W}_i$ $(i \in [n])$ first commit to their respective shares by

  · broadcasting $A_i = \boldsymbol{g}^{\boldsymbol{s}_i}$ and running its associated proof of knowledge $\mathsf{CSP}\{(A_i, \boldsymbol{s}_i) : \boldsymbol{g}^{\boldsymbol{s}_i} = A_i\}$ over broadcast to obtain π_{i1}.

  · broadcasting $B_i = h_1^{r_i} h_2^{\omega_i}$ for $\omega_i \leftarrow_R \mathbb{F}_p$ and running its its associated proofs of knowledge
  $\mathsf{CSP}\{(B_i, (r_i, \omega_i)) : h_1^{r_i} h_2^{\omega_i} = B_i\}$ over broadcast to obtain π_{i2}.

– **Reveal Linear Form over Shares:** The verifier samples a challenge $\boldsymbol{\gamma} \leftarrow_R \mathbb{F}_p^{\ell}$ and broadcasts it. Thereafter, the workers broadcast the linear form $v_i = \langle \boldsymbol{\gamma}, \boldsymbol{s}_i \rangle + r_i$. Recall that, we know that random linear combination of a codeword is also a codeword (recalled

in Lemma 4.2). Using Lemma 4.2, since $\{(\boldsymbol{s}_i, r_i) : i \in [n]\}$ are codewords respectively, the linear combination of those codewords $(v_1, \ldots, v_n)$ using the randomly sampled γ is also a codeword.

Additionally, to ensure that corrupt workers use $\boldsymbol{s}_i, r_i$ consistent with earlier commitments A_i, B_i we additionally require them to run the following proof of knowledge CSP over broadcast to obtain π_{i3}:

$$\pi_{i3} = \mathsf{CSP}\{((A_i B_i, \boldsymbol{\gamma} \| \mathbf{1} \| \mathbf{0}, v_i), (\boldsymbol{s}_i, r_i, \omega_i)) : \boldsymbol{g}^{\boldsymbol{s}_i} h_1^{r_i} h_2^{\omega_i} = A_i B_i \wedge \langle \boldsymbol{\gamma}, \boldsymbol{s}_i \rangle + r_i = v_i\}.$$

– **Verifier Determines Honest Commitments:** Let $\mathbf{v} = (v_1, \ldots, v_n)$, defined by $v_i = \langle \boldsymbol{\gamma}, \boldsymbol{s}_i \rangle + r_i$, be the vector of honestly computed values, and $\mathbf{v}' = (v'_1, \ldots, v'_n)$ be the respective broadcast values received by the workers in the previous step. If one of the proofs π_{i1}, π_{i2} or π_{i3} is invalid, the verifier set $b_i = 0$ else it sets $b_i = 1$. Since $\Delta(\mathbf{v}', \mathbf{v}) \leq d < (n-t)/2$, $\mathcal{V}$ can compute $\mathbf{v}$ from $\mathbf{v}'$ by decoding algorithm (e.g. Berlekamp-Welch) for Reed-Solomon codes. Set $\mathsf{C} = \{i \in [n] : v_i \neq v'_i \vee b_i = 0\}$ and let $\mathbf{H}_Q = (h_{jk})$ denote the matrix guaranteed by Lemma 4.1 for $Q = [n] \backslash \mathsf{C} = \{i_1, \ldots, i_q\}$ for $q \in \mathbb{N}$.

Informally, C is the set consisting of the position of 'errors' noted by the verifier and the new reconstruction coefficient k'_i is computed for the set $[n] \setminus \mathsf{C} = \{i_1, \ldots, i_q\}$. Thereafter the verifier proceeds with the final check with the non-error positions in $\{i_1, \ldots, i_q\}$ by using the new reconstruction coefficients and the corresponding commitments sent in the previous round. Also, we rely on the fact that we use shares of a codeword $(\boldsymbol{s}, r)$ in the proof of knowledge π_{i3} to ensure that the received values $(v_1, \ldots, v_n)$, if correctly computed, would also be a codeword and error-correction can be used on the new codeword $(v_1, \ldots, v_n)$.

– **Output using Honest Messages:** $\mathcal{V}$ outputs $(1, \mathsf{C})$ if $\left(\prod_{j \in [q]} A_{i_j}^{h_{jk}}\right)_{k=1,\ldots,n-t} = (z, \mathbf{0}^{n-t-1})$, and $(0, \{\mathcal{P}\})$ otherwise.

This is achieved via the additional steps (4b) through (6) in Π_{dlog} outlined in the figure above. We subsequently present a formal proof that Π_{dlog} achieves d-robust completeness for $d < \mathsf{dist}/2$, where $\mathsf{dist} = (n-t)$ is the minimum distance of the Reed-Solomon code induced by (t, n)-SSS.

Remark 4 *The final step of protocol Π_{dlog} checks $(n-t)$ equations over exponents and not just the reconstruction equation. This is to ensure that we extract the witness consistent with honest party shares of the witness. This is crucial in the security proof of our compiler for honest*

majority protocols where honest party shares determine a unique consistent witness, and this ensures that corrupt parties use the same inputs in both the DPoK *protocol and the associated MPC protocol.*

Theorem 4.1 *Assuming that* CSP *satisfies completeness, knowledge-soundness and zero-knowledge with* $O(\log \ell)$*-communication overhead,* Π_{dlog} *is a* $\mathsf{DPoK}_{\mathsf{SSS},\mathsf{DlogGen}}$ *(as per definition 4.8) for relation generator* DlogGen *and* (t,n)*-*SSS *with the following properties:*

- ***Security****: t-private and d-robust, for* $d < \mathsf{dist}/2$*, where* $\mathsf{dist} = (n-t)$ *is the minimum distance of the Reed-Solomon code induced by* (t,n)*-*SSS*.*
- ***Efficiency****:* $O(n)$ *communication over point-to-point channels and* $O(n \log \ell)$ *communication over broadcast channels.*

Proof:

Proof. We provide the proof of security and efficiency below.

Proof of Security. In order to prove security, we prove robust completeness, knowledge-soundness and zero-knowledge.

Robust Completeness. We show that when the prover is honest, and has a correct witness $\boldsymbol{s}$, the verifier outputs 1 and identifies the corrupt workers with overwhelming probability. Let $\mathcal{A}$ be an adversary corrupting set C' of workers with $|\mathsf{C}'| = d < (n-t)/2$. Let $\mathbf{S}$ denote the matrix with i^{th} column as $(\boldsymbol{s}_i, r_i)$ for $i \in [n]$. Clearly $\mathbf{S} \in \mathcal{L}^m$ for $m = \ell + 1$. We construct a matrix $\mathbf{S}'$ as follows: for $i \in \mathsf{C}'$ where the adversary's proofs π_{i1}, π_{i2} and π_{i3} are valid, we extract $\boldsymbol{s}'_i$ and r_i from the proofs π_{i1} and π_{i2} respectively, and set $(\boldsymbol{s}'_i, r'_i)$ as the i^{th} column of $\mathbf{S}'$. For $i \in \mathsf{C}'$ where one of the proofs is not valid, we set i^{th} column of $\mathbf{S}'$ as $(\boldsymbol{s}'_i, r'_i)$ for $\boldsymbol{s}'_i, r'_i$ sampled uniformly. Finally for $i \notin \mathsf{C}'$, we set the i^{th} column of $\mathbf{S}'$ as $(\boldsymbol{s}_i, r_i)$ (i.e. it is identical to the corresponding column in $\mathbf{S}$). Intuitively, the matrix $\mathbf{S}'$ is the corrupted version of honest matrix $\mathbf{S}$ in which columns corresponding to corrupt provers consist of shares $(\boldsymbol{s}'_i, r'_i)$ the adversary had in its "head". Looking ahead, we force the adversary to reveal a linear combination over the shares in its "head", and if they are inconsistent with $\mathbf{S}$, the resulting message v'_i will differ from honestly computed v_i (Lemma 4.2), which will identify the corrupt messages. We now proceed with the formal proof. Let E denote the set of column indices where $\mathbf{S}$ and $\mathbf{S}'$ differ. Let $\mathbf{v}' = (v'_1, \ldots, v'_n)$ be the vector where v'_i is sent by $\mathcal{W}_i$ in Step (5). Clearly, as $\Delta(\mathbf{v}', \mathcal{L}) \leq |\mathsf{C}'| < (n-t)/2$, we can decode $\mathbf{v}'$ to vector $\mathbf{v} = (v_1, \ldots, v_n) \in \mathcal{L}$. By uniqueness

[1]Note that here the witness is $\boldsymbol{s} \in \mathbb{F}_p^\ell$, and we do not have any component $\boldsymbol{t}$ which is not being secret-shared.

1. **Public Parameters**: Let $(\mathbb{G}, \boldsymbol{g}, p) \leftarrow_R \mathsf{DlogGen}(1^{1^\lambda}, 1^\ell)$. Let $\mathcal{R}^{\mathrm{DL}}$ denote the relation consisting of pairs $(z, \boldsymbol{s})$ such that $\boldsymbol{g}^{\boldsymbol{s}} = z$. Let $(h_1, h_2) \leftarrow_R \mathsf{Setup}(\mathcal{R}^{\mathrm{DL}})$ be two independent generators of $\mathbb{G}$.

2. **Input Phase**: The prover gets $(z, \boldsymbol{s})$ while workers $\mathcal{W}_i$, $i \in [n]$ are given $(z, \boldsymbol{s}_i)$ where $(\boldsymbol{s}_1, \ldots, \boldsymbol{s}_n) \leftarrow_R \mathsf{Share}(\boldsymbol{s})$. [1]

3. **Preprocessing:** Prover samples $r \leftarrow_R \mathbb{F}_p$, computes $(r_1, \ldots, r_n) \leftarrow_R \mathsf{Share}(r)$ and sends r_i to $\mathcal{W}_i$ for $i \in [n]$.

4. **Commit to Shares:** In the interactive phase, each worker $\mathcal{W}_i$, for $i \in [n]$, does the following:

   (a) $\mathcal{W}_i$ broadcasts $A_i = \boldsymbol{g}^{\boldsymbol{s}_i}$ and runs its associated proofs of knowledge $\mathsf{CSP}\{(A_i, \boldsymbol{s}_i) : \boldsymbol{g}^{\boldsymbol{s}_i} = A_i\}$ over broadcast to obtain π_{i1}.

   (b) $\mathcal{W}_i$ broadcasts $B_i = h_1^{r_i} h_2^{\omega_i}$ for $\omega_i \leftarrow_R \mathbb{F}_p$ and runs its associated proofs of knowledge $\mathsf{CSP}\{(B_i, (r_i, \omega_i)) : h_1^{r_i} h_2^{\omega_i} = B_i\}$ over broadcast to obtain π_{i2}.

5. **Reveal Linear Form over Shares:**

   (a) $\mathcal{V}$ samples $\boldsymbol{\gamma} \leftarrow_R \mathbb{F}_p^\ell$ and broacasts it.

   (b) For all $i \in [n]$, $\mathcal{W}_i$ computes $v_i = \langle \boldsymbol{\gamma}, \boldsymbol{s}_i \rangle + r_i$ and broadcasts v_i.

   (c) For all $i \in [n]$, $\mathcal{W}_i$ also runs the associated proof of knowledge to obtain π_{i3}, i.e.

$$\pi_{i3} = \mathsf{CSP}\{((A_i B_i, \boldsymbol{\gamma} \| \mathbf{1} \| \mathbf{0}, v_i), (\boldsymbol{s}_i, r_i, \omega_i)) : \boldsymbol{g}^{\boldsymbol{s}_i} h_1^{r_i} h_2^{\omega_i} = A_i B_i \wedge \langle \boldsymbol{\gamma}, \boldsymbol{s}_i \rangle + r_i = v_i\}.$$

6. **Verifier Determines Honest Commitments:**

   (a) Let $\mathbf{v}' = (v_1', \ldots, v_n')$ be the received values in the previous step by the workers, instead of the honestly computed valyes $(v_1, \ldots, v_n)$.

   (b) If one of the proofs π_{i1}, π_{i2} or π_{i3} is invalid, the verifier set $b_i = 0$ else it sets $b_i = 1$.

   (c) Since $\Delta(\mathbf{v}', \mathbf{v}) \leq d < (n-t)/2$ from assumption, $\mathcal{V}$ computes $\mathbf{v}$ from $\mathbf{v}'$ by decoding algorithm (e.g. Berlekamp-Welch) for Reed-Solomon codes. Set $\mathsf{C} = \{i \in [n] : v_i \neq v_i' \vee b_i = 0\}$ and let $\mathbf{H}_Q = (h_{jk})$ denote the matrix guaranteed by Lemma 4.1 for $Q = [n] \backslash \mathsf{C} = \{i_1, \ldots, i_q\}$ for $q \in \mathbb{N}$.

7. **Output using Honest Messages:** $\mathcal{V}$ outputs $(1, \mathsf{C})$ if

$$\left(\prod_{j \in [q]} A_{i_j}^{h_{jk}} \right)_{k=1,\ldots,n-t} = (z, \mathbf{0}^{n-t-1})$$

   and $(0, \{\mathcal{P}\})$ otherwise.

Figure 4.1: Protocol Π_{dlog}

of decoding, we must have $v'_i = v_i$ for $i \notin \mathsf{C}'$. We will prove that with overwhelming probability we must have $(\boldsymbol{s}'_i, r'_i) = (\boldsymbol{s}_i, r_i)$ for all $i \in Q$, which from Lemma 4.1 will imply that verifier outputs 1 (this is because verifier simply checks matrix relation in Lemma 4.1 over exponents). For sake of contradiction, assume that $(s'_i, r'_i) \neq (s_i, r_i)$ for $i \in \mathcal{H}$. We can assume that the proofs $\pi_{i1}, \ldots, \pi_{i3}$ were valid, for otherwise $b_i = 0$, which would imply $i \notin \mathcal{H}$, a contradiciton. Now from soundness of the proofs and binding property of the pedersen commitments, with overwhelming probability we must have $v'_i = \langle \boldsymbol{\gamma}, \boldsymbol{s}'_i \rangle + r'_i$. By assumption we have $i \in E$ and thus from Lemma 4.2, with overwhelming probability we have $v'_i \neq v_i$. Thus $i \notin \mathcal{H}$, which is again a contradiction. This proves that $\boldsymbol{s}'_i = \boldsymbol{s}_i$ for $i \in \mathcal{H}$, and thus the vector $(\boldsymbol{s}'_i)_{i \in \mathcal{H}}$ is $\mathcal{L}^m$-consistent. From Lemma 4.1, we conclude that the verifier outputs 1.

Knowledge-Soundness. To prove knowledge-soundness, we describe the extractor Ext which is provided the shares $\boldsymbol{s}_i, i \notin \mathsf{C}$ with C denoting the indices of workers corrupted by adversary $\mathcal{A}$. The extractor Ext runs the adversary $\mathcal{A}$. When $\mathcal{A}$ succeeds, for each $j \in [q]$ in Step (6) the extractor Ext sets $\boldsymbol{s}'_{i_j} = \boldsymbol{s}_{i_j}$ if $i_j \notin \mathsf{C}$; otherwise it invokes the extractor for CSP, which has oracle access to the worker $\mathcal{W}_{i_j}$ acting as the prover for the instantiation of $\mathsf{CSP}\{(A_i, \boldsymbol{s}_i) : \boldsymbol{g}^{\boldsymbol{s}_i} = A_i\}$, to extract $\boldsymbol{s}'_{i_j}$ satisfying $\boldsymbol{g}^{\boldsymbol{s}'_{i_j}} = A_{i_j}$. The verification check in Step (7) implies that the tuple $\left(\boldsymbol{s}'_{i_j}\right)_{j \in [q]}$ is $\mathcal{L}^\ell$-consistent. The extractor outputs the witness $\boldsymbol{s}$, which is reconstructed from the columns of the unique matrix $\mathbf{S} \in \mathcal{L}^\ell$ determined by the tuple $\left(\boldsymbol{s}'_{i_j}\right)_{j \in [q]}$ This completes the proof of knowledge-soundness for Π_{dlog}.

Zero-Knowledge. For proving zero-knowledge, we describe the simulator as follows. Without loss of generaltiy, let us assume that $\mathsf{C} = \{1, \ldots, \epsilon\}$ for $\epsilon \leq t$. The simulator Sim runs the adversary as follows:

- Sim receives $\{A_i, B_i\}_{i \in \mathsf{C}}$ from the adversary.
- Sim simulates messages $\{A_i, B_i, \pi_{i1}, \pi_{i2}\}_{i \notin \mathsf{C}}$ of the honest parties as follows:
  - Sim chooses $A'_i \leftarrow_R \mathbb{G}$ for $1 \leq i \leq t$, and sets $\boldsymbol{a} = (z, A'_1, \ldots, A'_t)$.
  - Sim sets $A'_{t+j} = \boldsymbol{a}^{\boldsymbol{t}_j}$ where $\boldsymbol{t}_j \in \mathbb{F}_p^{t+1}$ is the interpolation vector such that $f(t+j) = \langle (f(0), \ldots, f(t)), \boldsymbol{t}_j \rangle$ for all polynomials $f(x)$ of degree $\leq t$, i.e. $\boldsymbol{t}_j = \{\lambda_0(t+j), \lambda_1(t+j), \ldots, \lambda_t(t+j)\}$ where $\lambda_0(x), \ldots, \lambda_t(x)$ are lagrange polynomials with respect to the set $\{0, \ldots, t\}$.
  - Sim picks B'_i, $i > \epsilon$ uniformly at random from $\mathbb{G}$.
  - Sim invokes the simulator for the CSP to obtain $\pi_{i1} = \mathsf{CSP}\{(A_i, \boldsymbol{s}_i) : \boldsymbol{g}^{\boldsymbol{s}_i} = A_i\}$, $\pi_{i2} = \mathsf{CSP}\{(B_i, (r_i, \omega_i)) : h_1^{r_i} h_2^{\omega_i} = B_i\}$.

· Then Sim sends the messages $\{A'_i, B'_i, \pi_{i1}, \pi_{i2}\}_{i>\epsilon}$ to $\mathcal{A}$.

– Sim simulates the challenge by sampling $\boldsymbol{\gamma} \leftarrow_R \mathbb{F}_p^\ell$.

– Sim receives $\{v_i\}_{i<\epsilon}$ from $\mathcal{A}$, along with the proofs $\{\pi_{i3}\}_{i<\epsilon}$.

– Sim sets $v' \leftarrow_R \mathbb{F}_p$ and computes $(v'_1, \ldots, v'_n) \leftarrow_R \mathsf{Share}(v')$, computes simulated CSP proof $\pi_{i3} = \mathsf{CSP}\{((A_i B_i, \boldsymbol{\gamma}, v_i), (\boldsymbol{s}_i, r_i, \omega_i)) : \boldsymbol{g}^{\boldsymbol{s}_i} h_1^{r_i} h_2^{\omega_i} = A_i B_i \wedge \langle \boldsymbol{\gamma}, \boldsymbol{s}_i \rangle + r_i = v_i\}$, and sends $\{v_i, \pi_{i3}\}_{i>\epsilon}$.

– Sim sends $(v'_i, \pi_{i3})_{i>\epsilon}$ to the adversary $\mathcal{A}$.

To ensure indistinguishability of transcripts, we only need to provide argument for correctness of honest-party's first messages $\{A_j\}_{i \notin \mathsf{C}}$ provided by the simulator, since the other messages are sampled according to the protocol specification. We argue correctness of simulation of honest-party first messages $\{A_j\}_{i \notin \mathsf{C}}$ as follows. In real execution of the protocol, the vector of shares for party j is of the form $(f_1(j), \ldots, f_\ell(j))$, where $f_i : i \in [\ell]$ are the polynomials used to share the values $s_i : i \in [\ell]$ respectively. Let $\boldsymbol{f} = (f_1, \ldots, f_\ell)$ denote the vector of sharing polynomials and let $\boldsymbol{f}(j)$ to denote the vector $(f_1(j), \ldots, f_\ell(j))$. Then for $j > \epsilon$ in the real protocol, $(A_j)_{j>\epsilon}$ are distributed as $(\boldsymbol{g}^{\boldsymbol{f}(j)})_{j>\epsilon}$, subject to constraint that $\boldsymbol{g}^{\boldsymbol{f}(0)} = z$. Sampling such a polynomials f_i, $i \in [\ell]$ corresponds to choosing $f_i(1), \ldots, f_i(t)$ uniformly and then determining $f_i(t+j) = \langle (f_i(0), \ldots, f_i(t)), \boldsymbol{t}_j \rangle$ using the interpolation vector $\boldsymbol{t}_j$. Thus $\boldsymbol{f}(t+j)$ is a $\boldsymbol{t}_j$-linear combination of $\boldsymbol{f}(0), \ldots, \boldsymbol{f}(t)$, which dictates simulator's computation of A_{t+j} from vector $\boldsymbol{a}$. The simulated transcript is an accepting transcript as $\boldsymbol{g}^{\boldsymbol{f}(0)} = z$ and $\boldsymbol{g}^{\boldsymbol{f}(i)} = A_i$ for all $i \notin \mathsf{C}$, and the verification check is satisfied since a known linear combination of $\{\boldsymbol{f}(i)\}_{i \notin \mathsf{C}}$ in the exponent yields the desired value $\boldsymbol{f}(0)$ in the exponent. Additionally, since $\{\boldsymbol{f}(i)\}_{i \notin \mathsf{C}}$ are implicitly set as the honest-party shares, it is identical to the correct distribution of secret shares. This completes the proof of zero-knowledge for Π_{dlog}.

Proof of Efficiency/Succinctness. Assuming that CSP has $O(\log \ell)$-communication overhead [8], it follows by inspection that Π_{dlog} incurs $O(n)$ communication over point-to-point channels (where the prover distributes additional randomness to the workers) and $O(n \log \ell)$ communication over broadcast channels (for n instances of CSP). This completes the proof of efficiency/succinctness for Π_{dlog}, and hence the proof of Theorem 4.1. □

The following corollary of Theorem 4.1 follows immediately and yields the concrete bounds on the corruption threshold tolerated by Π_{dlog}.

Corollary 4.1 *Setting $d = t < n/3$, Π_{dlog} is $n/3$-private and $n/3$-robust.*

Finally, the following corollary also follows immediately from the proof of Theorem 4.1, and formally captures the properties of the non-robust protocol $\Pi_{\mathsf{nr\text{-}dlog}}$.

Corollary 4.2 *Assuming that* CSP *satisfies completeness, knowledge-soundness and zero-knowledge with* $O(\log \ell)$*-communication overhead,* $\Pi_{\mathsf{nr\text{-}dlog}}$ *is a* $\mathsf{DPoK}_{\mathsf{SSS},\mathsf{DlogGen}}$ *for relation generator* DlogGen *and* (t, n)*-*SSS *that satisfies completeness and* t*-privacy, and incurs* $O(n)$ *communication over point-to-point channels and* $O(n \log \ell)$ *communication over broadcast channels.*

Note that $\Pi_{\mathsf{nr\text{-}dlog}}$ retains all properties of its robust counterpart apart from d-robustness as stated in Theorem 4.1.

Generalization to Threshold Linear Secret Sharing. We can generalize the above protocol to work with *any* threshold linear secret sharing (TLSS) scheme. In the generalized version, the corruption threshold for robust completeness depends on the exact distance of the linear code induced by the TLSS scheme. As a corollary, we derive concrete bounds on the corruption threshold for robust completeness when using *replicated secret sharing*. The relevant technical details appear in Section 4.5.

Round Efficient DPoK for Discrete Log. In Section 4.6, we describe a round-efficient version of Π_{dlog} in the random oracle model (obtained using the Fiat-Shamir heuristic), which we call $\Pi^{\mathsf{FS}}_{\mathsf{dlog}}$. We highlight here that, while Π_{dlog} requires a logarithmic (in the size of the witness) number of rounds of interaction, the round-efficient version $\Pi^{\mathsf{FS}}_{\mathsf{dlog}}$ only requires a *constant* number of rounds of interaction. Apart from this, $\Pi^{\mathsf{FS}}_{\mathsf{dlog}}$ satisfies the same robust completeness, knowledge soundness and zero-knowledge properties as Π_{dlog}, albeit in the random oracle model.

4.4 DPoK for BBS+ Signatures over Secret-Shared Inputs

In this section, we build upon our (publicly verifiable) DPoK for the discrete log relation to design a protocol that allows a prover $\mathcal{P}$ to prove knowledge of a BBS+ (or PS) signature on a secret-shared input. Concretely, suppose that the prover $\mathcal{P}$ holds a BBS+ (or PS) signature σ on a message $\boldsymbol{m}$ under a public key pk, where $\boldsymbol{m}$ is secret-shared across n parties $\mathcal{W}_1, \ldots, \mathcal{W}_n$ (i.e. each worker $\mathcal{W}_i$ holds a share $\boldsymbol{m}_i$). The goal of the protocol is to allow the prover $\mathcal{P}$ to convince a designated verifier $\mathcal{V}$ that σ is a valid signature on $\boldsymbol{m}$ under pk, *without* revealing σ in the clear (this helps realize the desired property of signature unlinkability, as explained subsequently). We also present similar PoK protocols for PS signatures [97] over secret-shared inputs in Section 4.7.3. Looking ahead, we use these protocols as building blocks to design our

compiler for upgrading any secret-sharing based MPC protocol into an authenticated version of the same protocol, where the (secret-shared) inputs are authenticated using BBS+(or PS) signatures as above.

We start by defining the relation for BBS+ signature verification.

Definition 4.9 (BBS+ Relation) *Let* BBSGen *denote the relation generator, such that* $\mathsf{BBSGen}(1^{1^\lambda}, \ell)$ *outputs a bilinear group* $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, e, p) \leftarrow_R BBS.\mathsf{Setup}(1^{1^\lambda})$. *The corresponding relation* $\mathcal{R}^{\text{bbs}}$ *is defined by* $(\boldsymbol{x}, (\boldsymbol{m}, \boldsymbol{t})) \in \mathcal{R}^{\text{bbs}}$ *for* $\boldsymbol{x} = \mathsf{pk} = (g_1, w, h_0, \ldots, h_\ell) \in \mathbb{G}_1 \times \mathbb{G}_2 \times \mathbb{G}_1^\ell$, $\boldsymbol{m} = (m_1, \ldots, m_\ell) \in \mathbb{F}_p^\ell$ *and* $\boldsymbol{t} = \sigma = (A, \beta, s) \in \mathbb{G}_1 \times \mathbb{F}_p^2$ *if* $e(A, wg_2^\beta) = e(g_1 h_0^s \prod_{i=1}^\ell h_i^{m_i}, g_2)$.

Our DPoK Protocol $\Pi_{\mathsf{bbs+}}$. We build upon the robust complete DPoK Π_{dlog} for discrete log to propose a DPoK achieving robust completeness for BBS+ signatures, which allows a designated prover $\mathcal{P}$, to show knowledge of a BBS+ signature (A, β, s) over the message $\boldsymbol{m} \in \mathbb{F}_p^\ell$ that is secret-shared amongst the workers $\mathcal{W}_1, \ldots, \mathcal{W}_n$. Recall that this PoK involved the following steps: (i) the prover randomly chooses some auxiliary inputs, and combines them with the signature to output a randomized first message (this randomization ensures unlinkability), and then (ii) the prover shows knowledge of these auxiliary inputs and components of the signature satisfying discrete-log relations determined by the first message. Our BBS+ DPoK over secret-shared inputs follows a similar blueprint, where the prover similarly randomizes the first message using certain auxiliary inputs. In our case, the prover: (i) secret-shares the auxiliary inputs to the workers using point-to-point channels (this step is unique to our protocol and is designed to facilitate distributed proving in the subsequent steps), and (ii) broadcasts the first message to the workers *and* the verifier (this step uses broadcast channels and is conceptually similar to the PoK over non-distributed inputs). At this point, the problem reduces to a DPoK for the discrete log relation. We handle this using our robust complete DPoK Π_{dlog} for discrete log.

We prove the $\Pi_{\mathsf{bbs+}}$ to be a DPoK for the relation generator BBSGen in the following theorem.

Theorem 4.2 *Assuming that* Π_{dlog} *is a* $\mathsf{DPoK}_{\mathsf{SSS},\mathsf{DlogGen}}$ *for relation generator* $\mathsf{DlogGen}$ *and* (t, n)-SSS, $\Pi_{\mathsf{bbs+}}$ *is a* DPoK *for the relation generator* BBSGen *and* (t, n)-SSS *with:*

- ***Security****: t-private and d-robust, for* $d < \mathsf{dist}/2$, *where* $\mathsf{dist} = (n - t)$ *is the minimum distance of the Reed-Solomon code induced by* (t, n)-SSS.
- ***Efficiency****:* $O(n)$ *communication over point-to-point channels and* $O(n \log \ell)$ *communication over broadcast channels.*

– **Public Key:** $\mathsf{pk} = (w, h_0, \ldots, h_\ell)$

– $\mathcal{P}$**'s inputs**: Message $\boldsymbol{m} = (m_1, \ldots, m_\ell) \in \mathbb{F}_p^\ell$ and signature $\sigma = (A, \beta, s)$ on $\boldsymbol{m}$, with $A = \left(g_1 h_0^s \prod_{i=1}^{\ell} h_i^{m_i}\right)^{\frac{1}{\beta+x}}$, such that $(\mathsf{pk}, (\boldsymbol{m}, \sigma)) \in \mathcal{R}^{\text{bbs}}$

– $\mathcal{W}_i$**'s inputs:** $\mathcal{W}_i$ possesses the i^{th} share $\boldsymbol{m}_i$ of the message vector $\boldsymbol{m}$, such that $\mathsf{Reconstruct}(\boldsymbol{m}_1, \ldots, \boldsymbol{m}_n) = \boldsymbol{m}$

– **Preprocessing**: $\mathcal{P}$ samples $u \leftarrow_R \mathbb{F}_p^*, r \leftarrow_R \mathbb{F}_p, \eta \leftarrow_R \mathbb{F}_p$, and computes $d = b^u \cdot h_0^{-r}$ and $t = s - r \cdot v$ where $v = u^{-1}, b = g_1 h_0^s \prod_{i=1}^{\ell} h_i^{m_i}$. $\mathcal{P}$ computes $(r_1, \ldots, r_n) \leftarrow_R \mathsf{Share}(r)$, $(v_1, \ldots, v_n) \leftarrow_R \mathsf{Share}(v)$, $(\beta_1, \ldots, \beta_n) \leftarrow_R \mathsf{Share}(\beta)$, $(t_1, \ldots, t_n) \leftarrow_R \mathsf{Share}(t)$, $(\eta_1, \ldots, \eta_n) \leftarrow_R \mathsf{Share}(\eta)$. $\mathcal{P}$ sends the shares $(r_i, v_i, \beta_i, t_i, \eta_i)$ to $\mathcal{W}_i$, for all $i \in [n]$.

In other words, each $\mathcal{W}_i$ locally holds the i-th share $\boldsymbol{s}_i = (\boldsymbol{m}_i, r_i, v_i, \beta_i, t_i, \eta_i)$ such that

$$\boldsymbol{s} = (\boldsymbol{m}, r, v, \beta, t) = \mathsf{Reconstruct}\left(\{\boldsymbol{s}_i\}_{i \in [n]}\right).$$

– **Interactive Protocol:**

1. $\mathcal{P}$ computes $A' = A^u$, $\bar{A} = (A')^{-\beta} \cdot b^u (= (A')^x)$, where $b = g_1 h_0^s \prod_{i=1}^{\ell} h_i^{m_i}$ and $d = b^u \cdot h_0^{-r}$. $\mathcal{P}$ sets $C = d^{-v} h_0^{t-\eta}$, $D = h_0^\eta \prod_{i=1}^{\ell} h_i^{m_i}$, and broadcasts $(A', \bar{A}, d, C, D)$ to each $\mathcal{W}_i$ and $\mathcal{V}$.
2. The workers $\mathcal{W}_i$, $i \in [n]$ and $\mathcal{V}$ run the DPoK Π_{dlog} for the relation $D = h_0^\eta \prod_{i=1}^{\ell} h_i^{m_i}$, where $(\eta, m_1, \ldots, m_\ell)$ are secret-shared across the workers; and $\boldsymbol{g} = (h_0, \ldots, h_\ell)$, $z = D$ is available to all parties.
3. Simultaneously, the workers $\mathcal{W}_i$, $i \in [n]$ and $\mathcal{V}$ run the DPoK Π_{dlog} for the relation $C = d^{-v} h_0^{t-\eta} \wedge \frac{\bar{A}}{d} = (A')^{-\beta} h_0^r$, where (v, η) and (β, r) are secret-shared; and $\boldsymbol{g} = ((d, h_0), (A', h_0))$, $z = (C, \frac{\bar{A}}{d})$ is available to all parties.
4. $\mathcal{V}$ accepts if $C \cdot D = g_1^{-1}$, *and* $e(A', w) = e\left(\bar{A}, g_2\right)$, *and both* instances of Π_{dlog} accept.

Figure 4.2: Protocol $\Pi_{\mathsf{bbs+}}$

Proof:

We provide the proof of security and efficiency below. In order to prove security, we prove robust completeness, soundness, and zero-knowledge.

Robust Completeness. Robust completeness follows from direct calculation using the robust completeness of the underlying subprotocols $\mathsf{DPoK}\ \Pi_{\mathsf{dlog}}$ for $\mathsf{DlogGen}$, used in step (3) and (4).

Knowledge Soundness. Consider an adversary that corrupts a t-sized subset of the workers in $\Pi_{\mathsf{bbs+}}$. By inspection, for $t < n/3$, an honest verifier detects the corrupt subset of workers, since the underlying protocol Π_{dlog} satisfies d-robust completeness for $d < n/3$.

Consider an adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ which corrupts $\mathcal{P}$ and $\mathcal{W}_i$, $i \in \mathsf{C}$. We show that, given an extractor Ext for Π_{dlog}, it is possible to design an extraction algorithm Ext' that given $\{\boldsymbol{m}_i\}_{i \notin \mathsf{C}}$, where $\boldsymbol{m}_i$ is the share of $\boldsymbol{m}$ provided to $\mathcal{W}_i$, extracts a signature σ on $\boldsymbol{m}$. First Ext runs the adversary $\mathcal{A}$ to obtain the messages $(r_i, v_i, \beta_i, t_i, \eta_i)$ for $i \notin \mathsf{C}$. The extractor Ext' also obtains the message $(A', \bar{A}, d, C, D)$ from $\mathcal{A}$. Next it sets $\boldsymbol{s}'_i = (\eta_i, \boldsymbol{m}_i)$ and $\boldsymbol{s}''_i = (v_i, y_i, \beta_i, r_i)$ for $i \notin \mathsf{C}$ where $y_i = t_i - \eta_i$ for $i \notin \mathsf{C}$. It then invokes the extractor Ext for DPoK sub-protocol Π_{dlog} in steps (2) and (3) respectively and computes the extracted witness as follows:

$$
\begin{aligned}
&(\boldsymbol{s}')_{i \in \mathsf{C}} = (\eta, \boldsymbol{m})_{i \in \mathsf{C}} \leftarrow_R \mathsf{Ext}^{\mathcal{A}}(\{\boldsymbol{s}'_i\}_{i \notin \mathsf{C}}) \\
&(\boldsymbol{s}'')_{i \in \mathsf{C}} = (v, y, \beta, r)_{i \in \mathsf{C}} \leftarrow_R \mathsf{Ext}^{\mathcal{A}}(\{\boldsymbol{s}''_i\}_{i \notin \mathsf{C}}) \\
&\text{where} \\
&\eta = \mathsf{Reconstruct}(\eta_1, \ldots, \eta_n), \quad \boldsymbol{m} = \mathsf{Reconstruct}(\boldsymbol{m}_1, \ldots, \boldsymbol{m}_n) \\
&v = \mathsf{Reconstruct}(v_1, \ldots, v_n), \quad y = \mathsf{Reconstruct}(y_1, \ldots, y_n) \\
&\beta = \mathsf{Reconstruct}(\beta_1, \ldots, \beta_n), \quad r = \mathsf{Reconstruct}(r_1, \ldots, r_n)
\end{aligned}
$$

Using the message $(A', \bar{A}, d, C, D)$ obtained from the adversary $\mathcal{A}$ and the outputs $\eta, \boldsymbol{m}, v, y, \beta, r$ obtained from the extractor Ext for DPoK sub-protocol Π_{dlog}, extracted witness is computed as $(\boldsymbol{m}, \boldsymbol{t})$, where $\boldsymbol{t} = (A'^{v}, \beta, y + \eta + vr)$.

Here, we parse the extracted witness $\boldsymbol{m}$ as $\boldsymbol{m} = (m_1, \ldots, m_\ell)$. From knowledge-soundness of the DPoK sub-protocol Π_{dlog} and verifier's checks, with overwhelming probability we have: $D = h_0^{\eta} \prod_{i=1}^{\ell} h_i^{m_i}$, $C = d^{-v} h_0^{y}$, $(A')^{-\beta} h_0^{r} = \bar{A}/d$, $C \cdot D = g_1^{-1}$ and $\bar{A} = (A')^{x}$. We first note that $v \neq 0$, otherwise substituting C, D in the relation $C \cdot D = g_1^{-1}$ yields a non-trivial discrete-log

relation between the generators $g_1, h_0, \ldots, h_\ell$. From the preceding equations, we can derive:

$$(A'^v)^{\beta+x} = g_1 h_0^{y+\eta+vr} \prod_{i=1}^{\ell} h_i^{m_i}$$

which shows that $(A'^v, \beta, y + \eta + vr)$ is a valid signature on $\boldsymbol{m}$. Hence, the extractor Ext' has computed a valid witness for the BBSGen relation. This completes the proof of knowledge soundness for $\Pi_{\mathsf{bbs+}}$.

Honest Verifier Zero-Knowledge. Finally, consider an adversary $\mathcal{A}$ that corrupts workers $\mathcal{W}_i, i \in \mathsf{C}$ where $|\mathsf{C}| \leq t$. We show that, given a ZK-simulator $\mathsf{Sim}_1^{\mathsf{zk}}$ for Π_{dlog} and a ZK-simulator $\mathsf{Sim}_2^{\mathsf{zk}}$ for the single-prover proof of knowledge for BBS+ signatures from [41], we construct a simulation algorithm Sim' that output a simulated view of an honest verifier in the protocol $\Pi_{\mathsf{bbs+}}$ without the knowledge of the witness $(\boldsymbol{m}, \sigma)$. Using the simulator $\mathsf{Sim}_2^{\mathsf{zk}}$, the simulator Sim' generates the message $(A', \bar{A}, d, C, D)$. As the statements for the DPoKs in steps (2) and (3) depend entirely on the public parameters and the preceding message, the simulation follows by invoking simulator $\mathsf{Sim}_1^{\mathsf{zk}}$ to simulate the transcript for respective DPoKs on the statements derived from the simulated first message. Looking ahead, in the formal proof of security for our compiled MPC protocol, we use this simulation algorithm Sim' to simulate proofs of knowledge of BBS+ signatures on the inputs of the honest parties. This completes the proof of zero-knowledge soundness for $\Pi_{\mathsf{bbs+}}$.

Proof of Efficiency/Succinctness. Recall that Π_{dlog} has $O(n)$ communication over point-to-point channels and $O(n \log \ell)$-communication overhead over broadcast channel. It follows by inspection that $\Pi_{\mathsf{bbs+}}$ also inherit the same communication overheads from Π_{dlog}. This completes the proof of efficiency for $\Pi_{\mathsf{bbs+}}$, and hence the proof of Theorem 4.2.

□

Efficiently Batching BBS+ PoKs. We now present the protocol $\Pi_{\mathsf{bbs\text{-}auth\text{-}opt}}$ which efficiently batches n parallel instances of the protocol $\Pi_{\mathsf{bbs+}}$ with the party $\mathcal{P}_i$ acting as the prover in the i^{th} instance of the protocol. The optimization exploits the fact that each party needs to prove a linear (in exponents) relation over large part of its witness (the message vector), which can be reduced via a random challenge to proving a linear relation over the linearly combined messages. However we lose robustness: we can no longer identify the corrupt parties or a corrupt prover using error-correction as in $\Pi_{\mathsf{bbs+}}$, as the combined witness cannot be attributed to a specific party. Thus, we simply abort if one of the checks in the underlying protocol $\Pi_{\mathsf{nr\text{-}dlog}}$ fails.

– **Public Parameters**: $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, e, p) \leftarrow_R \mathsf{BBSGen}(1^{1^\lambda})$ defining BBS+ relation $\mathcal{R}^{\text{bbs}}$. Let $\mathsf{pk} = (g_1, w = g_2^x, h_0, \ldots, h_\ell)$ be a known public key for secret key $\mathsf{sk} = x \leftarrow_R \mathbb{F}_p$.

– P_i**'s inputs**:

  – Message $\boldsymbol{m}_i \in \mathbb{F}_p^\ell$ and signature $\sigma_i = (A_i, \beta_i, s_i)$ on $\boldsymbol{m}_i$ under pk.

  – i^{th} share of the message $\boldsymbol{m}_j$ of P_j.

– **Preprocessing**: $\mathcal{P}_i$ samples $u_i \leftarrow_R \mathbb{F}_p^*, r_i \leftarrow_R \mathbb{F}_p, \eta_i \leftarrow_R \mathbb{F}_p$, and computes $d_i = b_i^{u_i} \cdot h_0^{-r_i}$ and $t_i = s_i - r_i \cdot v_i$ where $v_i = u_i^{-1}, b_i = g_1 h_0^{s_i} \prod_{i=1}^{\ell} h_i^{m_i}$. and secret shares $r_i, v_i, t_i, \eta_i, \beta_i$ among $P_1, \ldots, P_n$. All parties set $\boldsymbol{g} = (h_0, \ldots, h_\ell)$.

– **Interactive Protocol**

1. $\mathcal{P}_i$, $i \in [n]$ computes $A_i' = A_i^{u_i}$, $\bar{A}_i = (A_i')^{-\beta} \cdot b_i^u (= (A_i')^x)$. $\mathcal{P}$ sets $C_i = d_i^{-v_i} h_0^{t_i - \eta_i}$, $D_i = \boldsymbol{g}^{\eta_i, \boldsymbol{m}_i}$, and broadcasts $(A_i', \bar{A}_i, d_i, C_i, D_i)$.
2. The verifier samples a challenge $\boldsymbol{\gamma} \leftarrow_R \mathbb{F}_p^\ell$ and broacasts it. Each P_i then computes $\boldsymbol{y}_i = \sum_{j \in [n]} \gamma^j (\eta_{ij}, \boldsymbol{m}_{ij})$, where $\eta_{ij}, \boldsymbol{m}_{ij}$ denotes $\mathcal{P}_i$'s share of $\mathcal{P}_j$'s inputs $\boldsymbol{m}_j, \eta_{ij}$.
3. All parties compute $D = \prod_{j \in [n]} D_j^{\gamma^j}$.

$$\text{Parties hold shares } \boldsymbol{y}_i \text{ of } \boldsymbol{y} \text{ satisfying } \boldsymbol{g}^{\boldsymbol{y}} = D$$

4. Parties run the interactive phase of the protocol $\Pi_{\mathsf{nr\text{-}dlog}}$ on statement D with $\boldsymbol{g}$ as the generator. They run the interactive phase of the protocol $\Pi_{\mathsf{nr\text{-}dlog}}$ on statements $C_i = d_i^{-v_i} h_0^{t_i - \eta_i} \wedge \frac{\bar{A}_i}{d_i} = (A_i')^{-\beta_i} h_0^{r_i}$, for each $i \in [n]$ with generators (d_i, h_0) and (A_i', h_0) respectively.
5. Parties also check that $e\left(\prod_{i=1}^n A_i', w\right) = e\left(\prod_{i=1}^n \bar{A}_i, g_2\right)$ holds.

– **Output**: P_j outputs $b_j = 1$ if all the above protocols lead to accept.

Figure 4.3: Protocol $\Pi_{\mathsf{bbs\text{-}auth\text{-}opt}}$

Round Efficient DPoK for BBS+ Signatures. Finally, note that by replacing Π_{dlog} with its round efficient version $\Pi^{\mathsf{FS}}_{\mathsf{dlog}}$ in the random oracle model (obtained using the Fiat-Shamir heuristic, presented in Section 4.6) in steps (2) and (3) of the Interactive Phase, we obtain a round efficient version of the protocol, which we call $\Pi^{\mathsf{FS}}_{\mathsf{bbs+}}$. Observe that $\Pi^{\mathsf{FS}}_{\mathsf{bbs+}}$ requires constant rounds of interaction, as compared to logarithmic (in the size of the message) rounds of interaction for $\Pi_{\mathsf{bbs+}}$, and satisfies the same robust completeness, knowledge soundness and zero-knowledge properties as $\Pi_{\mathsf{bbs+}}$, albeit in the random oracle model.

4.5 Generalization to Threshold Linear Secret Sharing Scheme

In this section, we provide generalization of our technique shown for Shamir Secret Sharing [104] to any Threshold Linear Secret Sharing Scheme. Here we present the definition of Threshold Linear Secret Sharing (TLSS) Scheme, which is a restriction of the definition of Linear Secret Sharing Scheme provided in [48, Chapter 6] to the case when each party receives same number of shares.

Definition 4.10 (Threshold Linear Secret Sharing Scheme) *A (t, n, r) threshold linear secret-sharing* (TLSS) *scheme over a finite field $\mathbb{F}$ consists of algorithms* (Share, Reconstruct) *as described below:*

- Share *is a randomized algorithm that is defined by a $m \times (t+1)$ matrix M (for some $m \geq n$) and a labeling function $\phi : [m] \to [n]$ such that $|\phi^{-1}(i)| = r$ for all $i \in [n]$. On input $s \in \mathbb{F}$,* Share *samples $r_1, \ldots, r_t \leftarrow_R \mathbb{F}$ uniformly and independently and sets $\boldsymbol{r}_s = (s, r_1, \ldots, r_t)$. It sets $\boldsymbol{s}_i = \{(M\boldsymbol{r}_s)_j \ : \ \phi(j) = i\}$ as the i^{th} share for all $i \in [n]$. We denote the output as $(\boldsymbol{s}_1, \ldots, \boldsymbol{s}_n) \leftarrow_R \mathsf{Share}(s)$, where $\boldsymbol{s}_i \in \mathbb{F}^r$ is the share sent to i^{th} party.*

- Reconstruct *is a deterministic algorithm that takes a set $\mathcal{I} \subseteq [n]$, $|\mathcal{I}| > t$, a vector of shares $(\boldsymbol{s}_1, \ldots, \boldsymbol{s}_{|\mathcal{I}|})$ and outputs $s = \mathsf{Reconstruct}((\boldsymbol{s}_1, \ldots, \boldsymbol{s}_{|\mathcal{I}|}), \mathcal{I}) \in \mathbb{F}$. Specifically, for all sets $\mathcal{I} \subseteq [n]$ with $|\mathcal{I}| > t$, there exists a vector $\boldsymbol{k}_{\mathcal{I}} = (k_{11}, \ldots, k_{nr}) \in \mathbb{F}^{nr}$ such that $\boldsymbol{s} = \sum_{i=1}^{n} \sum_{j=1}^{r} k_{ij} s_{ij}$. Here $\boldsymbol{s}_i = (s_{i1}, \ldots, s_{ir})$ for $i \in [n]$.*

A TLSS *scheme satisfies the following properties:*

- **Correctness**: *For every $s \in \mathbb{F}$, any $(\boldsymbol{s}_1, \ldots, \boldsymbol{s}_n) \leftarrow_R \mathsf{Share}(s)$ and any subset $\mathcal{I} = \{i_1, \ldots, i_q\} \subseteq [n]$ with $q > t$, we have $\mathsf{Reconstruct}((\boldsymbol{s}_{i_1}, \ldots, \boldsymbol{s}_{i_q}), \mathcal{I}) = s$.*

- **Privacy**: *For every $s \in \mathbb{F}$, any $(\boldsymbol{s}_1, \ldots, \boldsymbol{s}_n) \leftarrow_R \mathsf{Share}(s)$ and any subset $\mathcal{I} = \{i_1, \ldots, i_q\} \subseteq [n]$ with $q \leq t$, the tuple $(\boldsymbol{s}_{i_1}, \ldots, \boldsymbol{s}_{i_q})$ is information-theoretically independent of s.*

Remark 5 *We focus on Threshold Linear Secret Sharing schemes in this section, and we denote it as* TLSS*. As before we can extend a* TLSS *scheme to secret-share vectors* $\boldsymbol{s} \in \mathbb{F}^\ell$ *by applying* Share, Reconstruct *algorithms component-wise.*

4.5.1 Robust DPoK for Discrete Log for TLSS

In this section we generalize the construction of robust complete protocol for discrete-log relation presented in Section 4.3.2 to the case when (Share, Reconstruct) can be an arbitrary TLSS scheme. We also characterize the robustness threshold for the same in terms of minimum distance of linear code associated with the TLSS scheme. The proof of robust completeness now depends on Lemma 4.3 (below), which generalizes Lemma 4.2 to the case when linear code is over an extension field $\mathbb{F}_{p^r} \cong \mathbb{F}_p^r$ of the field $\mathbb{F} = \mathbb{F}_p$.

Let DlogGen be a relation generator that on input $(1^{1^\lambda}, m)$ outputs $(\mathbb{G}, \boldsymbol{g}, p)$ where p is a 1^λ-bit prime, $\mathbb{G}$ is a cyclic group of order p and $\boldsymbol{g} = (g_1, \ldots, g_m) \leftarrow_R \mathbb{G}^m$ is a uniformly sampled set of generators. The associated relation $\mathcal{R}^{\mathrm{DL}}$ is defined by $(z, \boldsymbol{s}) \in \mathcal{R}^{\mathrm{DL}}$ if $\boldsymbol{g}^{\boldsymbol{s}} = z$. Let TLSS = (Share, Reconstruct) denote (t, n, r) threshold linear secret sharing over finite field of order p $\mathbb{F} = \mathbb{F}_p$. We follow the framework presented for DlogGen; namely Π_{dlog} (Figure 4.1), that is t-private, d-robust and incurs $O(n)$ communication over point-to-point channels and $O(n \log \ell)$ communication over broadcast channels. We present our generalized protocol with the similar guarantees.

Additional Preliminaries and Notation. We setup some useful notation and preliminaries specific to this section to ease the presentation. For $s \in \mathbb{F}$, we will view the output $(s_1, \ldots, s_n) \leftarrow_R \mathsf{Share}(s)$ to consist of n-shares each over $\mathbb{F}_{p^r}$, i.e. we view $s_i \in \mathbb{F}^r$ as an element of $\mathbb{F}_{p^r}$. Applying the sharing component-wise, for a vector $\boldsymbol{s} \in \mathbb{F}^\ell$, we view the output $(\boldsymbol{s}_1, \ldots, \boldsymbol{s}_n) \leftarrow_R \mathsf{Share}(\boldsymbol{s})$ to consist of n-shares, each in $(\mathbb{F}_{p^r})^\ell$, i.e an ℓ-length vector over $\mathbb{F}_{p^r}$. We also veiw a vector $\boldsymbol{s} = (s_1, \ldots, s_\ell) \in (\mathbb{F}_{p^r})^\ell$ as $\ell \times r$ matrix over $\mathbb{F}$, where i^{th} row of the matrix corresponds to $s_i \in \mathbb{F}_{p^r}$ viewed as a vector in $\mathbb{F}^r$. We also introduce the linear code $\mathcal{L}_{\mathsf{TLSS}}$, which is induced by the sharings under the TLSS scheme.

Definition 4.11 (TLSS induced code) *For an* (n, t, r)-TLSS *scheme over* $\mathbb{F}$ *given by algorithms* (Share, Reconstruct)*, we define linear code* $\mathcal{L}_{\mathsf{TLSS}}$ *over the field* $\mathbb{F}_{p^r}$ *as*

$$\mathcal{L}_{\mathsf{TLSS}} = \{(s_1, \ldots, s_n) : \Pr\left[(s_1, \ldots, s_n) \leftarrow_R \mathsf{Share}(s), s \leftarrow_R \mathbb{F}\right] > 0\},$$

consisting of all possible sharings output by the Share *algorithm.*

We now state the generalization of Lemma 4.2 to fields of the form $\mathbb{F}_{p^r}$. The lemma is

proved in [51][Lemma A.5]. We recall that for an $[n, k, *]$ linear code $\mathcal{L}$ over $\mathbb{F}$, $\mathcal{L}^m$ denotes the set of $m \times n$ matrices over $\mathbb{F}$ whose rows are codewords in $\mathcal{L}$.

Lemma 4.3 *Let $\mathcal{L}$ be an $[n, k, d]$-linear code over finite field $\mathbb{F}_{p^k}$ and let $\mathbf{S}$ be an $m \times n$ matrix over $\mathbb{F}_{p^k}$. Let $e = \Delta(\mathbf{S}, \mathcal{L}^m)$ be such that $e < d/3$. Then for any codeword $\boldsymbol{r} \in \mathcal{L}$, and $\boldsymbol{\gamma}$ sampled uniformly from $\mathbb{F}^m$, we have $\Delta(\boldsymbol{r} + \boldsymbol{\gamma}^T\mathbf{S}, \mathcal{L}) = e$ with probability at least $1 - d/|\mathbb{F}|$. Furthermore, if E denotes the column indices where $\mathbf{S}$ differs from the nearest matrix $\mathbf{Q}$ in $\mathcal{L}^m$, with probability $1 - d/|\mathbb{F}|$ over choice of $\boldsymbol{\gamma}$, the vector $\boldsymbol{r} + \boldsymbol{\gamma}^T\mathbf{S}$ differs from the closest codeword $\mathbf{v} \in \mathcal{L}$ at precisely the positions in E.*

We now proceed with the description of the generalised protocol, where we highlight key differences from the protocol Π_{dlog} for the case of Shamir Secret Sharing.

1. *Public Parameters*: The public parameters, as before consists of $(\mathbb{G}, \boldsymbol{g}, p) \leftarrow_R \mathsf{DlogGen}(1^\lambda, \ell)$. Additionally we have $h_1, h_2 \leftarrow_R \mathbb{G}$. The relation $\mathcal{R}^{\mathrm{DL}}$ consists of $(z, \boldsymbol{s})$ satisfying $\boldsymbol{g}^{\boldsymbol{s}} = z$.

2. *Input Phase*: The prover gets $(z, \boldsymbol{s})$ while workers $\mathcal{W}_i, i \in [n]$ are given $(z, \boldsymbol{s}_i)$ where $(\boldsymbol{s}_1, \ldots, \boldsymbol{s}_n) \leftarrow_R \mathsf{Share}(\boldsymbol{s})$.

3. *Preprocessing:* The prover sends δ_i to $\mathcal{W}_i$ for $i \in [n]$ where $(\delta_1, \ldots, \delta_n) \leftarrow_R \mathsf{Share}(\delta)$ for $\delta \leftarrow_R \mathbb{F}_{p^r}$.

4. *Commit to Shares:* In the interactive phase, the worker $\mathcal{W}_i$ proceeds as follows: The worker veiws the share $\boldsymbol{s}_i$ as $\ell \times r$ matrix M_i over $\mathbb{F}$. Then for each $j \in [r]$, it computes $A_{ij} = \boldsymbol{g}^{M_i[j]}$, where $M_i[j]$ denotes the j^{th} column of the matrix. Similarly it views the input δ_i as vector $(\delta_{i1}, \ldots, \delta_{ir})$ over $\mathbb{F}$. It then computes commitments B_{ij} for $j \in [r]$ as $B_{ij} = h_1^{\delta_{ij}} h_2^{\omega_j}$ for $\omega_j \leftarrow_R \mathbb{F}$. Finally $\mathcal{W}_i$ broadcasts $\mathbf{A}_i = (A_{i1}, \ldots, A_{ir})$ and $\mathbf{B}_i = (B_{i1}, \ldots, B_{ir})$.

5. *Reveal Linear Form over Shares:* The verifier sends a challenge vector $\boldsymbol{\gamma} \leftarrow_R \mathbb{F}^\ell$, and the workers broadcast the linear form $v_i = \langle \boldsymbol{\gamma}, \boldsymbol{s}_i \rangle + \delta_i$. In the preceding inner-product, we consider $\boldsymbol{s}_i$ as a vector over $\mathbb{F}_{p^r}$ and v_i, δ_i are considered as elements in the field $\mathbb{F}_{p^r}$. To ensure that corrupt workers use $\boldsymbol{s}_i, \delta_i$ consistent with earlier commitments $\mathbf{A}_i, \mathbf{B}_i$ we additionally require them to provide proofs by running the proof of knowledge CSP for the following relations (viewing $\boldsymbol{s}_i$ as $\ell \times r$ matrix M_i over $\mathbb{F}$):

$$\pi_{i1} = \mathsf{CSP}(M_i) : \boldsymbol{g}^{M_i[j]} = A_{ij} \; \forall \, j \in [r],$$
$$\pi_{i2} = \mathsf{CSP}(\delta_i, \omega_1, \ldots, \omega_r) : h_1^{\delta_{ij}} h_2^{\omega_j} = B_{ij} \; \forall \, j \in [r],$$
$$\pi_{i3} = \mathsf{CSP}\big\{(M_i, \delta_i, \omega_1, \ldots, \omega_r) : \boldsymbol{g}^{M_i[j]} h_1^{\delta_{ij}} h_2^{\omega_j} = A_{ij} B_{ij} \wedge \langle \boldsymbol{\gamma}, M_i[j] \rangle + \delta_{ij} = v_{ij} \; \forall \, j \in [r]\big\}.$$

The NIPK used above can be instantiated with $O(\log \ell)$ communication complexity using compressed sigma protocols (CSPs) of Attema et al. [8], made non-interactive using Fiat-Shamir transformation. We observe that each proof asserts r constraints, which can be reduced to one constraint each using a random challenge. We skip the details here.

6. *Verifier Determines Honest Commitments:* Let $\mathbf{v}' = (v'_1, \ldots, v'_n)$ be the purported values of $(v_1, \ldots, v_n)$ received in the previous step. If one of the proofs π_{i1}, π_{i2} or π_{i3} is invalid, he verifier sets $v'_i \leftarrow_R \mathbb{F}_{p^r}$ (randomly). Here we use $\mathbf{v} = (v_1, \ldots, v_n)$ defined by $v_i = \langle \boldsymbol{\gamma}, \boldsymbol{s}_i \rangle + r_i$ to denote the vector of honestly computed values. We recall that we consider $\mathbf{v}$ to be a vector over $\mathbb{F}_{p^r}^n$. Since $\Delta(\mathbf{v}', \mathbf{v}) \leq d < \mathsf{dist}/2$, with dist being the minimum distance of the code induced by the TLSS, $\mathcal{V}$ can compute $\mathbf{v}$ from $\mathbf{v}'$ by using error correction. Let C denote indices of corrupt workers (who actually deviate from the protocol). From Lemma 4.3 we conclude $\mathsf{C} = \{i \in [n] : v_i \neq v'_i\}$ with overwhelming probability. Let $k'_1, \ldots, k'_q$ denote the reconstruction coefficients for the set $[n] \backslash \mathsf{C}$ where each $k'_i = (k'_{i1}, \ldots, k'_{ir}) \in \mathbb{F}^r$ for each i.

7. *Output using honest messages:* $\mathcal{V}$ outputs $(1, \mathsf{C})$ if $\prod_{j \in [q], t \in [r]} A_{i_j,t}^{k'_{jt}} = z$, and $(0, \{\mathcal{P}\})$ otherwise.

Theorem 4.3 (Robust Distributed Proof of Knowledge for Discrete Log for TLSS) *Assuming that the discrete log assumption holds over the group $\mathbb{G}$, the above protocol is a $\mathsf{DPoK}_{\mathsf{TLSS},\mathsf{DlogGen}}$ for relation generator $\mathsf{DlogGen}$ and (t, n, r)-TLSS scheme which satisfies t-privacy and d-robustness, for $d < \mathsf{dist}/3$, where dist is the minimum distance the linear code induced by the TLSS scheme. Moreover the protocol incurs $O(rn)$ communication over point-to-point channels and $O(rn + \log \ell)$ communication over broadcast channels.*

The proof of the above theorem is similar to that for the protocol Π_{dlog}, except that we use Lemma 4.3 instead of Lemma 4.2 to identify corrupt messages, and appropriately omit them from the verification check. We now discuss implications of the above theorem for specific threshold secret sharing schemes.

4.5.2 (Corollary) Distributed Proof of Knowledge using Replicated Secret Sharing

Our earlier results obtained for Shamir Secret Sharing [104] in Theorem 4.1 can be seen as special case of Theorem 4.3 for $r = 1$ and $\mathsf{dist} = (n - t)$. Here we additionally specialise Theorem 4.3 to the case of *replicated secret sharing*. We recall the definition of Replicated Secret Sharing (RSS) Scheme provided in [57].

Definition 4.12 (Replicated Secret Sharing Scheme) *A* $(t, n, \binom{n-1}{t})$ *replicated linear secret-sharing* (RSS) *scheme over a finite field* $\mathbb{F}$ *consists of algorithms* (Share, Reconstruct) *as described below:*

- Share *is a randomized algorithm that on input* $s \in \mathbb{F}$*, samples* $s_A \in \mathbb{F}$ *for all* $A \in [n], |A| = t$*, such that* $\sum_A s_A = s$*, and sets* $s_i = \{s_A : i \notin A\}$*. We denote the output as* $(\boldsymbol{s}_1, \ldots, \boldsymbol{s}_n) \leftarrow_R \mathsf{Share}(s)$*, where* $\boldsymbol{s}_j \in \mathbb{F}^{\binom{n-1}{t}}$ *is the share sent to party* P_j*.*
- Reconstruct *is a deterministic algorithm that takes a set* $\mathcal{I} \subseteq [n]$*,* $|\mathcal{I}| \geq t$*, a vector* $(s_1, \ldots, s_{|\mathcal{I}|})$ *and outputs* $s = \mathsf{Reconstruct}((s_1, \ldots, s_{|\mathcal{I}|}), \mathcal{I}) \in \mathbb{F}$*.*

A RSS *scheme satisfies the following properties:*

- **Correctness***: For every* $s \in \mathbb{F}$*, any* $(s_1, \ldots, s_n) \leftarrow_R \mathsf{Share}(s)$ *and any subset* $\mathcal{I} = \{i_1, \ldots, i_q\} \subseteq [n]$ *with* $q \geq t$*, we have* $\mathsf{Reconstruct}((s_{i_1}, \ldots, s_{i_q}), \mathcal{I}) = s$*.*
- **Privacy***: For every* $s \in \mathbb{F}$*, any* $(s_1, \ldots, s_n) \leftarrow_R \mathsf{Share}(s)$ *and any subset* $\mathcal{I} = \{i_1, \ldots, i_q\} \subseteq [n]$ *with* $q < t$*, the tuple* $(s_{i_1}, \ldots, s_{i_q})$ *is information-theoretically independent of* s*.*

Remark 6 *We note that* RSS *scheme is a specific instance of* TLSS *scheme discussed in the prior section.*

Let DlogGen be a relation generator that on input $(1^{1^\lambda}, m)$ outputs $(\mathbb{G}, \boldsymbol{g}, p)$ where p is a 1^λ-bit prime, $\mathbb{G}$ is a cyclic group of order p and $\boldsymbol{g} = (g_1, \ldots, g_m) \leftarrow_R \mathbb{G}^m$ is a uniformly sampled set of generators. The associated relation $\mathcal{R}^{\mathrm{DL}}$ is defined by $(z, \boldsymbol{s}) \in \mathcal{R}^{\mathrm{DL}}$ if $\boldsymbol{g}^{\boldsymbol{s}} = z$. Let RSS = (Share, Reconstruct) denote $(t, n, \binom{n-1}{t})$ replicated secret sharing over $\mathbb{F}_p$. In this section, we state the theorems and the threshold bounds for RSS as a specific case of TLSS (Theorem 4.3).

Theorem 4.4 (Robust Distributed Proof of Knowledge for Discrete Log for Replicated Secret Sharing) *Assuming that the discrete log assumption holds over the group* $\mathbb{G}$*, protocol* $\Pi_{\mathsf{rob\text{-}rss}}$ *is a* $\mathsf{DPoK}_{\mathsf{RSS,DlogGen}}$ *for relation generator* DlogGen *and* $(t, n, \binom{n-1}{t})$-RSS *scheme which satisfies* t*-privacy and* d*-robustness, for* $d = t < \mathsf{dist}/3$*, where* $\mathsf{dist} = (n - t)$ *is the minimum distance of two valid codewords of the linear code induced by the* TLSS *scheme.*

Remark 7 *We note that the corruption threshold of* $t < n/3$ *attainable for Shamir Secret Sharing* (SSS) *Scheme and Replicated Secret Sharing* (RSS) *Scheme follows from the fact that the underlying linear code defined by both sharing schemes attain a minimum distance of* $\mathsf{dist} = n - t$ *between any two valid codewords. We note that the linear codes considered for* SSS *scheme lies in* $\mathbb{F}_p$ *(Reed-Solomon Codes), whereas the linear codes considered for* RSS *lies in* $\mathbb{F}_{p^k}$*.*

4.6 Round Efficient Distributed Proof of Knowledge

In this section, we formalize the notion of *distributed* proof of knowledge (DPoK) in the random oracle model (ROM) which multiple provers, each having a share of the witness engage in an interactive protocol with a verifier to convince it that their shares determine a valid witness. The provers do not directly interact with each other, and all the interaction with the verifier takes place over a public broadcast channel.

We define a round efficient DPoK by building upon our original definition for DPoK from Section 4.3. Our definition is based on the Fiat-Shamir heuristic [59], using which we transform a DPoK (with number of rounds logarithmic in the size of the message) into a round efficient DPoK (having constant number of rounds).

Definition 4.13 (Round Efficient DPoK in the ROM) Let $\mathsf{DPoK}_{\mathsf{SSS},\mathsf{RGen}} = (\mathsf{Setup}, \Pi)$ be a DPoK as in Definition 4.8 for relation generator RGen and a secret-sharing scheme $\mathsf{SSS} = (\mathsf{Share}, \mathsf{Reconstruct})$, where Setup is a PPT algorithm, and Π is a k-round interactive protocol between PPT algorithms $\mathcal{P}$ (prover), $\mathcal{V}$ (interactive verifier) and $\mathcal{W}_1, \ldots, \mathcal{W}_n$ (workers), such that all of the interaction with the verifier takes place over a public broadcast channel, and where in each round $j \in [k]$, the verifier $\mathcal{V}$ broadcasts a challenge sampled uniformly from the challenge set Ch_j. We define the corresponding round efficient DPoK for the same $(\mathsf{RGen}, \mathsf{SSS})$ pair as a tuple of the form $\mathsf{RE\text{-}DPoK}_{\mathsf{SSS},\mathsf{RGen}} = (\mathsf{Setup}_{\mathsf{FS}}, \Pi_{\mathsf{FS}}, \mathcal{V}_{\mathsf{FS}})$, where $\mathsf{Setup}_{\mathsf{FS}}$ is a PPT setup algorithm, Π_{FS} is an interactive protocol between PPT algorithms $\mathcal{P}_{\mathsf{FS}}$ (prover) and $(\mathcal{W}^{\mathsf{RO}}_{\mathsf{FS}})_1, \ldots, (\mathcal{W}^{\mathsf{RO}}_{\mathsf{FS}})_n$ (workers), and $\mathcal{V}_{\mathsf{FS}}$ is PPT verification algorithm. These are defined as follows:

- **Setup** $[(\mathsf{pp}, \mathsf{RO}) \leftarrow_R \mathsf{Setup}_{\mathsf{FS}}(\mathcal{R}, 1^{1^\lambda})]$: The setup algorithm takes as input a relation $\mathcal{R} \leftarrow_R \mathsf{RGen}(1^{1^\lambda})$ and outputs a tuple of the form $(\mathsf{pp}, \mathsf{RO})$, where $\mathsf{pp} \leftarrow_R \mathsf{Setup}(\mathcal{R})$, and $\mathsf{RO} = \{\mathsf{RO}_i\}_{i\in[1,r]}$, with each RO_i being a random function sampled uniformly from the set of all functions that maps $\{0,1\}^*$ to the challenge set Ch_i. As in our general definition of DPoK, the setup phase is required to be executed only once for a given relation $\mathcal{R}$. We again assume that $\mathcal{R}$ consists of pairs $(\boldsymbol{x}, \boldsymbol{w})$ where $\boldsymbol{w}$ is parsed as $(\boldsymbol{s}, \boldsymbol{t})$) with $\boldsymbol{s} \in \mathbb{F}^m$; looking ahead, we partition the witness as $(\boldsymbol{s}, \boldsymbol{t})$ to explicitly specify which parts of the witness later needs to be shared. Also, note that sampling each RO_i independently is equivalent to instantiating RO_i from a single random oracle via domain separation.
- **Interactive Protocol** Π_{FS}: executed jointly by the prover $\mathcal{P}^{\mathsf{RO}}_{\mathsf{FS}}$ and the workers $(\mathcal{W}^{\mathsf{RO}}_{\mathsf{FS}})_1, \ldots, (\mathcal{W}^{\mathsf{RO}}_{\mathsf{FS}})_n$ in the following phases:

– **Input Phase**: The prover $\mathcal{P}^{\mathsf{RO}}_{\mathsf{FS}}$ receives $(\mathsf{pp}, \boldsymbol{x}, (\boldsymbol{s}, \boldsymbol{t})) \in \mathcal{R}$ as input, while each worker $(\mathcal{W}^{\mathsf{RO}}_{\mathsf{FS}})_i$, $i \in [n]$ receives $(\boldsymbol{x}, \boldsymbol{s}_i)$ as input, where $(\mathsf{pp}, \boldsymbol{s}_1, \ldots, \boldsymbol{s}_n) \leftarrow_R \mathsf{Share}(\boldsymbol{s})$.

– **Preprocessing Phase**: This is (an optional) phase where the prover $\mathcal{P}^{\mathsf{RO}}_{\mathsf{FS}}$ sends some auxiliary information aux_i to worker $(\mathcal{W}^{\mathsf{RO}}_{\mathsf{FS}})_i$ using secure private channels. This phase is identical to the preprocessing phase (if any) in the underlying DPoK scheme, with the prover $\mathcal{P}^{\mathsf{RO}}_{\mathsf{FS}}$ invoking the prover $\mathcal{P}$ of DPoK to obtain its output in the preprocessing phase, and sending the same to the workers $(\mathcal{W}^{\mathsf{RO}}_{\mathsf{FS}})_1, \ldots, (\mathcal{W}^{\mathsf{RO}}_{\mathsf{FS}})_n$.

– **Interactive Phase**: In this phase, the prover and the workers interact using a public broadcast channel as follows, where all algorithm presented with FS subscript have access to the random oracle RO:

  * The prover $\mathcal{P}^{\mathsf{RO}}_{\mathsf{FS}}$ (resp. each worker $(\mathcal{W}^{\mathsf{RO}}_{\mathsf{FS}})_i$) invokes the prover $\mathcal{P}$ (resp. the corresponding worker $\mathcal{W}_i$ of) of DPoK to produce the same round message as in the protocol Π.

  * Suppose that in round j of the protocol Π (for $j \in [k]$), the verifier $\mathcal{V}$ of the underlying DPoK outputs a challenge $\boldsymbol{c}_j \leftarrow_R \mathsf{Ch}_j$. In Π_{FS}, each worker $(\mathcal{W}^{\mathsf{RO}}_{\mathsf{FS}})_i$ computes $\boldsymbol{c}_j$ locally as

$$\boldsymbol{c}_j = \mathsf{RO}_j\left(\boldsymbol{x}, \{\{\boldsymbol{m}_{i,\ell}\}_{i\in[n]}, \boldsymbol{c}_\ell\}_{\ell\in[j-1]}\right),$$

    where $\boldsymbol{m}_{i,\ell}$ is the prior message of $\mathcal{W}_i$ in round ℓ, and $\boldsymbol{c}_\ell$ is prior challenge in round ℓ.

  Let $\pi = \left(\boldsymbol{x}, \{\{\boldsymbol{m}_{i,\ell}\}_{i\in[n]}, \boldsymbol{c}_\ell\}_{\ell\in[k]}\right)$ denote the transcript of protocol Π_{FS} at the conclusion of k rounds.

– **Verification**: $[b \leftarrow_R \mathcal{V}^{\mathsf{RO}}_{\mathsf{FS}}(\mathsf{pp}, \boldsymbol{x}, \pi)]$: The verifier $\mathcal{V}^{\mathsf{RO}}_{\mathsf{FS}}$ takes as input $(\mathsf{pp}, \boldsymbol{x}, \pi)$ and outputs a decision bit $b \in \{0, 1\}$. It outputs 1 if and only if both of the following hold: (i) $\mathcal{V}(\mathsf{pp}, \boldsymbol{x}, \pi) = 1$ ($\mathcal{V}$ being the verifier of DPoK), and (ii) for each $j \in [k]$, $\boldsymbol{c}_j = \mathsf{RO}_j\left(\boldsymbol{x}, \{\{\boldsymbol{m}_{i,\ell}\}_{i\in[n]}, \boldsymbol{c}_\ell\}_{\ell\in[j-1]}\right)$. Otherwise, the verifier $\mathcal{V}^{\mathsf{RO}}_{\mathsf{FS}}$ outputs 0.

A distributed proof of knowledge $\mathsf{RE\text{-}DPoK}_{\mathsf{SSS},\mathsf{RGen}}$ as described above is said to be *t-private*, *ℓ-robust* if the following hold:

– **Completeness**: We say that completeness holds if for any $\mathcal{R} \leftarrow_R \mathsf{RGen}(1^{1^\lambda})$, for $(\mathsf{pp}, \mathsf{RO}) \leftarrow_R \mathsf{Setup}_{\mathsf{FS}}(\mathcal{R}, 1^{1^\lambda}, 1^k)$, and for any $(\boldsymbol{x}, \boldsymbol{s}) \in \mathcal{R}$, if π denotes the transcript of an honest execution of the protocol Π_{FS}, then we have

$$\Pr[\mathcal{V}^{\mathsf{RO}}_{\mathsf{FS}}(\mathsf{pp}, \boldsymbol{x}, \pi) = 1] = 1$$

– **Knowledge Soundness**: We say that knowledge soundness holds if for any security parameter 1^λ and any PPT adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ that makes at most $Q = \mathsf{poly}(\lambda)$ queries to RO, where $\mathcal{A}_2$ corrupts the prover $\mathcal{P}^{\mathsf{RO}}_{\mathsf{FS}}$ and subset of workers $\{(\mathcal{W}^{\mathsf{RO}}_{\mathsf{FS}})_i\}_{i\in\mathsf{C}}$ for some $\mathsf{C} \subseteq [n]$, there exists an extractor Ext with oracle access to $\mathcal{A}_2$ (which controls $\mathcal{P}^{\mathsf{RO}}_{\mathsf{FS}}$ and the set of corrupt $(\mathcal{W}^{\mathsf{RO}}_{\mathsf{FS}})_i$) such that for any $\mathcal{R} \leftarrow_R \mathsf{RGen}(1^\lambda)$, the following probability is negligible,

$$\Pr\left[\begin{array}{c} \mathcal{V}^{\mathsf{RO}}_{\mathsf{FS}}(\mathsf{pp}, \boldsymbol{x}, \pi) = 1 \wedge \\ ((\boldsymbol{x}, (\boldsymbol{s}, \boldsymbol{t})) \notin \mathcal{R} \vee \\ \mathsf{Consistent}(\{\boldsymbol{s}_i\}_{i\notin\mathsf{C}}, \boldsymbol{s}) = 0) \end{array} \middle| \begin{array}{c} (\mathsf{pp}, \mathsf{RO}) \leftarrow_R \mathsf{Setup}_{\mathsf{FS}}(\mathcal{R}) \\ (\boldsymbol{x}, \{\boldsymbol{s}_i, \mathsf{aux}_i\}_{i\notin\mathsf{C}}) \leftarrow_R \mathcal{A}_1(\mathsf{pp}) \\ \pi := \\ \Pi_{\mathsf{FS}}\big(\mathcal{A}_2(\rho), \{(\mathcal{W}^{\mathsf{RO}}_{\mathsf{FS}})_i(\alpha_i)_{i\notin\mathsf{C}}\}\big) \\ (\boldsymbol{s}, \boldsymbol{t}) \leftarrow_R \\ \mathsf{Ext}^{\mathcal{A}_2}(\mathsf{pp}, \boldsymbol{x}, \{\boldsymbol{s}_i\}_{i\notin\mathsf{C}}, \pi, \mathcal{Q}) \end{array}\right]$$

where π denotes the transcript of an execution of the protocol Π_{FS} between the adversary $\mathcal{A}_2$ (which controls $\mathcal{P}^{\mathsf{RO}}_{\mathsf{FS}}$ and the set of corrupt $(\mathcal{W}^{\mathsf{RO}}_{\mathsf{FS}})_i$), and the honest workers.

– **Zero-Knowledge**: Zero-knowledge for publicly verifiable DPoKs is defined in the explicitly programmable random oracle model where the simulator is allowed to program the random oracle. The zero-knowledge simulator $\mathcal{S}_{\mathsf{FS}}$ is modeled as a stateful algorithm that operates in two modes. In the first mode, $(c_i, \mathsf{st}') \leftarrow \mathcal{S}_{\mathsf{FS}}(1, \mathsf{st}, \boldsymbol{x}, i)$ handles random oracle calls to RO_i on input $\boldsymbol{x}$. In the second mode, $(\tilde{\pi}, \mathsf{st}') \leftarrow \mathcal{S}_{\mathsf{FS}}(2, \mathsf{st}, \boldsymbol{x})$ simulates a valid proof string. We define stateful wrapper oracles.

  – $\mathcal{S}_1(t, i)$ denotes the oracle that returns the first output of $\mathcal{S}_{\mathsf{FS}}(1, \mathsf{st}, t, i)$;
  – $\mathcal{S}_2(x, w)$ returns the first output of $\mathcal{S}_{\mathsf{FS}}(2, \mathsf{st}, \boldsymbol{x})$ if $(\mathsf{pp}, \boldsymbol{x}, \boldsymbol{s}) \in \mathcal{R}$ and $\perp$ otherwise; (This is because ZK is defined only for true statements.)

  We say that a DPoK is zero-knowledge in the random oracle model if for all $\mathcal{R} \leftarrow_R \mathsf{RGen}(1^{1^\lambda})$, $(\boldsymbol{x}, \boldsymbol{s}) \in \mathcal{R}$ and any PPT adversary $\mathcal{A}$ corrupting a set of workers $\{(\mathcal{W}^{\mathsf{RO}}_{\mathsf{FS}})_i\}_{i\in\mathsf{C}}$, where $|\mathsf{C}| \leq t$, there exists a PPT simulator $\mathcal{S}_{\mathsf{FS}}$ such that $\mathsf{View}_{\mathcal{A},\mathsf{RO},\Pi_{\mathsf{FS}}}(\mathsf{pp}, \boldsymbol{x})$ is indistinguishable from $\mathcal{S}_{\mathsf{FS}}(\mathsf{pp}, \boldsymbol{x})$ for $\mathsf{pp} \leftarrow_R \mathsf{Setup}_{\mathsf{FS}}(\mathcal{R})$. Here, the view is given by $\mathsf{View}_{\mathcal{A},\mathsf{RO},\Pi_{\mathsf{FS}}} = \{\boldsymbol{r}, (\mathbf{M}_i)_{i\in\mathsf{C}}\}$ where $\boldsymbol{r}$ denotes the internal randomness of $\mathcal{A}$ and $\mathbf{M}_i$ is the set of all messages received by $(\mathcal{W}^{\mathsf{RO}}_{\mathsf{FS}})_i$ in Π_{FS}.

– **Robust-Completeness**: We say that robust-completeness holds if for all $\mathcal{R} \leftarrow_R \mathsf{RGen}(1^{1^\lambda})$, $(\boldsymbol{x}, \boldsymbol{s}) \in \mathcal{R}$ and any PPT adversary $\mathcal{A}$ corrupting a set of workers $\{(\mathcal{W}^{\mathsf{RO}}_{\mathsf{FS}})_i\}_{i\in\mathsf{C}}$, where $|\mathsf{C}| \leq \ell$, $(\mathcal{V}^{\mathsf{RO}}_{\mathsf{FS}})_{\mathcal{A},\Pi_{\mathsf{FS}}}(\mathsf{pp}, \boldsymbol{x}, \Pi_{\mathsf{FS}}) = 1$ with overwhelming probability where $\mathsf{pp} \leftarrow_R \mathsf{Setup}_{\mathsf{FS}}(\mathcal{R})$.

Robust Complete Round Efficient DPoK for Discrete Log. We now provide a $\mathsf{RE\text{-}DPoK}_{\mathsf{SSS},\mathsf{DlogGen}}$ for the discrete log relation based on Shamir Secret Sharing (SSS) [104]. Let DlogGen be a relation generator that on input $(1^{1^\lambda}, 1^\ell)$ outputs $(\mathbb{G}, \boldsymbol{g}, p)$ where p is a 1^λ-bit prime, $\mathbb{G}$ is a cyclic group of order p and $\boldsymbol{g} = (g_1, \ldots, g_\ell) \leftarrow_R \mathbb{G}^\ell$ is a uniformly sampled set of generators. The associated relation $\mathcal{R}^{\mathrm{DL}}$ is defined by $(z, \boldsymbol{s}) \in \mathcal{R}^{\mathrm{DL}}$ if $\boldsymbol{g}^{\boldsymbol{s}} = z$. Let $\mathsf{SSS} = (\mathsf{Share}, \mathsf{Reconstruct})$ denote (t, n) Shamir secret sharing over $\mathbb{F}_p$. Our protocol Π_{dlog} realizing $\mathsf{RE\text{-}DPoK}_{\mathsf{SSS},\mathsf{DlogGen}}$ is as below. However, for ease of exposition, we first explain a simpler non-robust version of the protocol, before explaining the robust version.

We use the non-interactive proof of knowledge for the discrete logarithm relation, namely $\mathsf{NIPK}_{\mathsf{FS}} = (\mathsf{NIPK.Setup}_{\mathsf{FS}}, \mathsf{NIPK}.\mathcal{P}^{\mathsf{RO}}_{\mathsf{FS}}, \mathsf{NIPK}.\mathcal{V}^{\mathsf{RO}}_{\mathsf{FS}})$, obtained by applying the Fiat-Shamir heuristic (using random oracle $\mathsf{RO} : \{0,1\}^* \to \mathbb{F}_p^\ell$) on the public-coin compressed sigma protocol [8] for proof of knowledge of the discrete logarithm relation. Additionally, we present the protocol Π_{dlog} using Fiat-Shamir heuristic [59] and a random oracle $\mathsf{RO} : \{0,1\}^* \to \mathbb{F}_p^\ell$.

We now state and prove the following theorem for $\Pi^{\mathsf{FS}}_{\mathsf{dlog}}$.

Theorem 4.5 *Assuming that* NIPK *satisfies completeness, knowledge-soundness and zero-knowledge with $O(\log \ell)$-communication overhead, $\Pi^{\mathsf{FS}}_{\mathsf{dlog}}$ is a* $\mathsf{RE\text{-}DPoK}_{\mathsf{SSS},\mathsf{DlogGen}}$ *(as per definition 4.8) for relation generator* DlogGen *and (t, n)-*SSS *with the following properties:*

- ***Security****: t-private and d-robust, for $d < \mathsf{dist}/2$, where $\mathsf{dist} = (n - t)$ is the minimum distance of the Reed-Solomon code induced by (t, n)-*SSS*.*
- ***Efficiency****: $O(n)$ communication over point-to-point channels and $O(n \log \ell)$ communication over broadcast channels.*

Proof sketch. For knowledge-soundness, the intuition behind extraction of a valid witness are the fact that the shares (provided to the extractor via definition) held by the honest parties uniquely determines the output and the adversary succeeds in proving the statement in a protocol where these honest-party shares are used. For zero-knowledge, the key intuition behind the simulation is that the adversial messages can be 'ignored' for providing an accepting transcript as the protocol does 'error-correction' and removes the 'bad shares' from consideration.

Proof: Completeness and robust completeness of $\Pi^{\mathsf{FS}}_{\mathsf{dlog}}$ follows similarly from the completeness and robust completeness of its respective counterpart Π_{dlog}.

Knowledge-Soundness. To prove knowledge-soundness, we describe the extractor Ext for $\Pi^{\mathsf{FS}}_{\mathsf{dlog}}$ as follows. Let C be the set of indices of workers corrupted by adversary $\mathcal{A}$. Additionally,

[1] Note that here the witness is $\boldsymbol{s} \in \mathbb{F}_p^\ell$, and we do not have any component $\boldsymbol{t}$ which is not being secret-shared.

1. **Public Parameters**: Let $(\mathbb{G}, \boldsymbol{g}, p) \leftarrow_R \mathsf{DlogGen}(1^{1^\lambda}, 1^\ell)$. Let $\mathcal{R}^{\mathrm{DL}}$ denote the relation consisting of pairs $(z, \boldsymbol{s})$ such that $\boldsymbol{g}^{\boldsymbol{s}} = z$. Let $(h_1, h_2) \leftarrow_R \mathsf{Setup}(\mathcal{R}^{\mathrm{DL}})$ be two independent generators of $\mathbb{G}$.

2. **Input Phase**: The prover gets $(z, \boldsymbol{s})$ while workers $(\mathcal{W}_{\mathsf{FS}}^{\mathsf{RO}})_i$, $i \in [n]$ are given $(z, \boldsymbol{s}_i)$ where $(\boldsymbol{s}_1, \ldots, \boldsymbol{s}_n) \leftarrow_R \mathsf{Share}(\boldsymbol{s})$. [1]

3. **Preprocessing:** The prover sends r_i to $(\mathcal{W}_{\mathsf{FS}}^{\mathsf{RO}})_i$ for $i \in [n]$ where $(r_1, \ldots, r_n) \leftarrow_R \mathsf{Share}(r)$ for $r \leftarrow_R \mathbb{F}_p$.

4. **Commit to Shares:** In the interactive phase, the workers first commit to their respective shares by broadcasting

   (a) $A_i = \boldsymbol{g}^{\boldsymbol{s}_i}$ and its associated proofs of knowledge $\pi_{i1} = \mathsf{NIPK}.\mathcal{P}_{\mathsf{FS}}^{\mathsf{RO}}\{(A_i, \boldsymbol{s}_i) : \boldsymbol{g}^{\boldsymbol{s}_i} = A_i\}$.

   (b) $B_i = h_1^{r_i} h_2^{\omega_i}$ for $\omega_i \leftarrow_R \mathbb{F}_p$ and its associated proofs of knowledge $\pi_{i2} = \mathsf{NIPK}.\mathcal{P}_{\mathsf{FS}}^{\mathsf{RO}}\{(B_i, (r_i, \omega_i)) : h_1^{r_i} h_2^{\omega_i} = B_i\}$.

5. **Reveal Linear Form over Shares:** Each worker $(\mathcal{W}_{\mathsf{FS}}^{\mathsf{RO}})_i$ computes $\boldsymbol{\gamma}$ as $\boldsymbol{\gamma} = \mathsf{RO}\left(z \| A_1 \| B_1 \| A_2 \| B_2 \| \ldots \| A_n \| B_n\right) \in \mathbb{F}_p^\ell$. Thereafter, the workers broadcast the linear form $v_i = \langle \boldsymbol{\gamma}, \boldsymbol{s}_i \rangle + r_i$. To ensure that corrupt workers use $\boldsymbol{s}_i, r_i$ consistent with earlier commitments A_i, B_i we additionally require them to broadcast proof π_{i3} as:

$$\pi_{i3} = \mathsf{NIPK}.\mathcal{P}_{\mathsf{FS}}^{\mathsf{RO}}\{((A_i B_i, \boldsymbol{\gamma} \| \mathbf{1} \| \mathbf{0}, v_i), (\boldsymbol{s}_i, r_i, \omega_i)) : \boldsymbol{g}^{\boldsymbol{s}_i} h_1^{r_i} h_2^{\omega_i} = A_i B_i \wedge \langle \boldsymbol{\gamma}, \boldsymbol{s}_i \rangle + r_i = v_i\}.$$

6. **Verifier Determines Honest Commitments:** Let $\mathbf{v}' = (v_1', \ldots, v_n')$ be the received values in the previous step by the workers, instead of $(v_1, \ldots, v_n)$. If one of the proofs π_{i1}, π_{i2} or π_{i3} is invalid, the verifier set $b_i = 0$ else it sets $b_i = 1$. Here we use $\mathbf{v} = (v_1, \ldots, v_n)$ defined by $v_i = \langle \boldsymbol{\gamma}, \boldsymbol{s}_i \rangle + r_i$ to denote the vector of honestly computed values. Since $\Delta(\mathbf{v}', \mathbf{v}) \leq d < (n-t)/2$, $\mathcal{V}_{\mathsf{FS}}^{\mathsf{RO}}$ can compute $\mathbf{v}$ from $\mathbf{v}'$ by decoding algorithm (e.g. Berlekamp-Welch) for Reed-Solomon codes. Set $\mathsf{C} = \{i \in [n] : v_i \neq v_i' \vee b_i = 0\}$ and let $\mathbf{H}_Q = (h_{ij})$ denote the matrix guaranteed by Lemma 4.1 for $Q = [n] \backslash \mathsf{C} = \{i_1, \ldots, i_q\}$ for $q \in \mathbb{N}$.

7. **Output using Honest Messages:** $\mathcal{V}$ outputs $(1, \mathsf{C})$ if $\left(\prod_{j \in [q]} A_{i_j}^{h_{jk}}\right)_{k=1,\ldots,n-t} = (z, \mathbf{0}^{n-t-1})$, and $(0, \{\mathcal{P}_{\mathsf{FS}}^{\mathsf{RO}}\})$ otherwise.

Figure 4.4: Protocol $\Pi_{\mathsf{dlog}}^{\mathsf{FS}}$

we assume that there is an extractor Ext_1 for NIPK proof. The extractor Ext runs the adversary $\mathcal{A}$ as follows:

- Ext is provided $(\mathsf{pp}, z, \{\boldsymbol{s}_i\}_{i\notin\mathsf{C}}, \Pi_{\mathsf{FS}}, \mathcal{Q})$ as input at the onset, where $\{\boldsymbol{s}_i\}_{i\notin\mathsf{C}}$ are the honest-party shares and $\mathcal{Q}$ is the set of RO queries made by the adversary $\mathcal{A}$.
- Ext receives A_i, B_i from $\mathcal{A}$ along with the NIPK proofs $\{\pi_{i1}, \pi_{i2}\}$ for all $i \in \mathsf{C}$, such that $\pi_{i1} = \mathsf{NIPK}.\mathcal{P}^{\mathsf{RO}}_{\mathsf{FS}}\{(A_i, \boldsymbol{s}_i) : \boldsymbol{g}^{\boldsymbol{s}_i} = A_i\}$, $\pi_{i2} = \mathsf{NIPK}.\mathcal{P}^{\mathsf{RO}}_{\mathsf{FS}}\{(B_i, (r_i, \omega_i)) : h_1^{r_i} h_2^{\omega_i} = B_i\}$.
- Ext computes $\{A_i = \boldsymbol{g}^{\boldsymbol{s}_i}, B_i = h_1^{r_i} h_2^{\omega_i}\}_{i\notin\mathsf{C}}$ and sends $\{A_i, , \pi_{i1}, B_i, \pi_{i2}\}_{i\notin\mathsf{C}}$ to $\mathcal{A}$, where $\pi_{i1} = \mathsf{NIPK}.\mathcal{P}^{\mathsf{RO}}_{\mathsf{FS}}\{(A_i, \boldsymbol{s}_i) : \boldsymbol{g}^{\boldsymbol{s}_i} = A_i\}$, $\pi_{i2} = \mathsf{NIPK}.\mathcal{P}^{\mathsf{RO}}_{\mathsf{FS}}\{(B_i, (r_i, \omega_i)) : h_1^{r_i} h_2^{\omega_i} = B_i\}$.
- Ext computes $\boldsymbol{\gamma} = \mathsf{RO}\left(z \| A_1 \| B_1 \| A_2 \| B_2 \| \ldots \| A_n \| B_n\right)$
- Ext receives $\{v_i, \pi_{i3}\}_{i\in\mathsf{C}}$ from $\mathcal{A}$
- Ext computes v_i, π_{i3} as $\{v_i = \langle \boldsymbol{\gamma}, \boldsymbol{s}_i\rangle + r_i\}_{i\notin\mathsf{C}}$ and $\pi_{i3} = \mathsf{NIPK}.\mathcal{P}^{\mathsf{RO}}_{\mathsf{FS}}\{((A_i B_i, \boldsymbol{\gamma} \| \mathbf{1} \| \mathbf{0}, v_i), (\boldsymbol{s}_i, r_i, \omega_i)) : \boldsymbol{g}^{\boldsymbol{s}_i} h_1^{r_i} h_2^{\omega_i} = A_i B_i \wedge \langle \boldsymbol{\gamma}, \boldsymbol{s}_i\rangle + r_i = v_i\}$, and sends $\{v_i, \pi_{i3}\}_{i\notin\mathsf{C}}$
- Ext sets $\boldsymbol{s}'_i = \boldsymbol{s}_i$ for all $i \notin \mathsf{C}$ and for all $i \in \mathsf{C}$, it invokes the extractor Ext_1 for the Fiat-Shamir transformed proof π_{i1} to extract $\boldsymbol{s}'_i$ satisfying $\boldsymbol{g}^{\boldsymbol{s}'_i} = A_i$.
- Ext finally computes $\boldsymbol{s}'$ as $\boldsymbol{s}' = \mathsf{Reconstruct}(\{s_i\}_{i\notin\mathsf{C}})$ and outputs $\boldsymbol{s}'$.

Note that by using random oracle RO to obtain the challenge $\boldsymbol{\gamma}$ in Step (iii) described above, we ensure that a 'random linear combination' of the code is considered in Step (6) of the protocol. Now, considering that the adversary $\mathcal{A}$ succeeds, we now argue the correctness of the extracted witness. Since the adversary succeeds, the verification check in Step (7) of the protocol implies that the tuple $\left(\boldsymbol{s}'_i\right)_{i\notin\mathsf{C}}$ is $\mathcal{L}^\ell$-consistent and the reconstructed vector $\boldsymbol{s}'$ satisfies $\boldsymbol{s}' = \mathsf{Reconstruct}(\{s_i\}_{i\notin\mathsf{C}})$ along with $\left(\prod_{j\notin\mathsf{C}} A_j^{h_{jk}}\right)_{k=1,\ldots,n-t} = (z, \mathbf{0}^{n-t-1})$, where $A_j = \boldsymbol{g}^{\boldsymbol{s}_j}$ for all $j \notin \mathsf{C}$. Note that the extractor's output $\boldsymbol{s}'$ is reconstructed from the columns of the unique matrix $\mathbf{S} \in \mathcal{L}^\ell$ determined by the tuple $(\boldsymbol{s}'_i)_{j\notin\mathsf{C}}$. Hence, the extractor output is a valid witness for the given statement. This completes the proof of knowledge-soundness for $\Pi^{\mathsf{FS}}_{\mathsf{dlog}}$.

Knowledge-error. Since there are three non-parallel instances of Fiat-Shamir transformed NIPK protocol from Attema et al. [8] being invoked, if the knowledge-error of the Fiat-Shamir transformed version is κ', then the knowledge-error of $\Pi^{\mathsf{FS}}_{\mathsf{dlog}}$ is $\kappa \leq 3\kappa'$. And we know from [8] that the knowledge-error κ'' of NIPK protocol is negligible, and [12] ensures that the knowledge-error κ' of non-parallel Fiat-Shamir version of the multi-round protocol is still negligible and degrades only linearly with respect to the number of queries to the Random Oracle. Specifically,

if Q is the upper-bound for the number of Random Oracle queries for NIPK protocol, then given that κ'' is the knowledge-error of the interactive NIPK protocol, from [12] we get that $\kappa' = (Q+1).\kappa''$.

Zero-Knowledge. For proving zero-knowledge, we describe the simulator as follows. Without loss of generaltiy, let us assume that $\mathsf{C} = \{1, \ldots, \epsilon\}$ for $\epsilon \leq t$. The simulator $\mathcal{S}_{\mathsf{FS}}$ runs the adversary as follows:

- $\mathcal{S}_{\mathsf{FS}}$ receives $\{A_i, B_i\}_{i \in \mathsf{C}}$ from the adversary.
- $\mathcal{S}_{\mathsf{FS}}$ simulates messages $\{A_i, B_i, \pi_{i1}, \pi_{i2}\}_{i \notin \mathsf{C}}$ of the honest parties as follows:
  - $\mathcal{S}_{\mathsf{FS}}$ chooses $A'_i \leftarrow_R \mathbb{G}$ for $1 \leq i \leq t$, and sets $\boldsymbol{a} = (z, A'_1, \ldots, A'_t)$.
  - $\mathcal{S}_{\mathsf{FS}}$ sets $A'_{t+j} = \boldsymbol{a}^{\boldsymbol{t}_j}$ where $\boldsymbol{t}_j \in \mathbb{F}_p^{t+1}$ is the interpolation vector such that $f(t+j) = \langle (f(0), \ldots, f(t)), \boldsymbol{t}_j \rangle$ for all polynomials $f(x)$ of degree $\leq t$, i.e. $\boldsymbol{t}_j = \{\lambda_0(t+j), \lambda_1(t+j), \ldots, \lambda_t(t+j)\}$ where $\lambda_0(x), \ldots, \lambda_t(x)$ are lagrange polynomials with respect to the set $\{0, \ldots, t\}$.
  - $\mathcal{S}_{\mathsf{FS}}$ picks B'_i, $i > \epsilon$ uniformly at random from $\mathbb{G}$.
  - $\mathcal{S}_{\mathsf{FS}}$ invokes the simulator for the NIPK to obtain $\pi_{i1} = \mathsf{NIPK}.\mathcal{P}^{\mathsf{RO}}_{\mathsf{FS}}\{(A_i, \boldsymbol{s}_i) : \boldsymbol{g}^{\boldsymbol{s}_i} = A_i\}$, $\pi_{i2} = \mathsf{NIPK}.\mathcal{P}^{\mathsf{RO}}_{\mathsf{FS}}\{(B_i, (r_i, \omega_i)) : h_1^{r_i} h_2^{\omega_i} = B_i\}$.
  - Then $\mathcal{S}_{\mathsf{FS}}$ sends the messages $\{A'_i, B'_i, \pi_{i1}, \pi_{i2}\}_{i > \epsilon}$ to $\mathcal{A}$.
- $\mathcal{S}_{\mathsf{FS}}$ queries the random oracle RO to obtain the challenge $\boldsymbol{\gamma} \leftarrow_R \mathbb{F}_p^{\ell}$.
- Thereafter, the simulator receives $\{v_i\}_{i < \epsilon}$ from $\mathcal{A}$, along with the proofs $\{\pi_{i3}\}_{i < \epsilon}$.
- $\mathcal{S}_{\mathsf{FS}}$ sets $v' \leftarrow_R \mathbb{F}_p$ and computes $(v'_1, \ldots, v'_n) \leftarrow_R \mathsf{Share}(v')$, computes simulated $\mathsf{NIPK}.\mathcal{P}^{\mathsf{RO}}_{\mathsf{FS}}$ proof
  $\pi_{i3} = \mathsf{NIPK}.\mathcal{P}^{\mathsf{RO}}_{\mathsf{FS}}\{((A_i B_i, \boldsymbol{\gamma} \| \mathbf{1} \| \mathbf{0}, v_i), (\boldsymbol{s}_i, r_i, \omega_i)) : \boldsymbol{g}^{\boldsymbol{s}_i} h_1^{r_i} h_2^{\omega_i} = A_i B_i \wedge \langle \boldsymbol{\gamma}, \boldsymbol{s}_i \rangle + r_i = v_i\}$, and sends $\{v_i, \pi_{i3}\}_{i > \epsilon}$.
- Finally, $\mathcal{S}_{\mathsf{FS}}$ sends $(v'_i, \pi_{i3})_{i > \epsilon}$ to the adversary $\mathcal{A}$.

We argue correctness of simulation of honest-party first messages $\{A_j\}_{i \notin \mathsf{C}}$ as follows: in the real protocol, the vector of shares for party j is of the form $(f_1(j), \ldots, f_\ell(j))$, where $f_i : i \in [\ell]$ are the polynomials used to share the values $s_i : i \in [\ell]$ respectively. Let $\boldsymbol{f} = (f_1, \ldots, f_\ell)$ denote the vector of sharing polynomials and let $\boldsymbol{f}(j)$ to denote the vector $(f_1(j), \ldots, f_\ell(j))$. Then for $j > \epsilon$ in the real protocol, $(A_j)_{j > \epsilon}$ are distributed as $(\boldsymbol{g}^{\boldsymbol{f}(j)})_{j > \epsilon}$, subject to constraint that $\boldsymbol{g}^{\boldsymbol{f}(0)} = z$. Sampling such a polynomials f_i, $i \in [\ell]$ corresponds to choosing $f_i(1), \ldots, f_i(t)$ uniformly and

then determining $f_i(t+j) = \langle (f_i(0), \ldots, f_i(t)), \boldsymbol{t}_j \rangle$ using the interpolation vector $\boldsymbol{t}_j$. Thus $\boldsymbol{f}(t+j)$ is a $\boldsymbol{t}_j$-linear combination of $\boldsymbol{f}(0), \ldots, \boldsymbol{f}(t)$, which dictates simulator's computation of A_{t+j} from vector $\boldsymbol{a}$. The simulated transcript is an accepting transcript as $\boldsymbol{g}^{\boldsymbol{f}(0)} = z$ and $\boldsymbol{g}^{\boldsymbol{f}(i)} = A_i$ for all $i \notin \mathsf{C}$, and the verification check is satisfied since a known linear combination of $\{\boldsymbol{f}(i)\}_{i \notin \mathsf{C}}$ in the exponent yields the desired value $\boldsymbol{f}(0)$ in the exponent. Additionally, since $\{\boldsymbol{f}(i)\}_{i \notin \mathsf{C}}$ are implicitly set as the honest-party shares, it is identical to the correct distribution of secret shares. This completes the proof of zero-knowledge for $\Pi^{\mathsf{FS}}_{\mathsf{dlog}}$. □

We note that knowledge soundness ensures simulation extractability in the random oracle model [65, 66], and hence, our Fiat-Shamir transformed round efficient DPoK is simulation-extractable. The following corollary of Theorem 4.5 follows immediately and yields the concrete bounds on the corruption threshold tolerated by $\Pi^{\mathsf{FS}}_{\mathsf{dlog}}$.

Corollary 4.3 *Setting $d = t < n/3$, $\Pi^{\mathsf{FS}}_{\mathsf{dlog}}$ is $n/3$-private and $n/3$-robust.*

4.7 PS Signatures

In this section we show the generality of techniques shown above by providing distributed protocols for another pairing-based signature scheme, whose proof of knowledge of signature also reduces to discrete logarithm relation.

We begin by recalling the Pointcheval Sanders (PS) signature scheme from [97], along with the associated proof of knowledge.

Definition 4.14 (PS Signature Scheme [97]) *The PS Signature Scheme to sign a message $\boldsymbol{m} = (m_1, \ldots, m_\ell) \in \mathbb{F}_p^\ell$ consists of a tuple of PPT algorithms* $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ *described as follows :*

- $\mathsf{Setup}(1^\lambda)$ *: For security parameter λ, this algorithm outputs groups $\mathbb{G}_1, \mathbb{G}_2$, and $\mathbb{G}_T$ of prime order p, with an efficient bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$, as part of the public parameters* pp*. Note that the bilinear groups are of type 3, which ensures that there are no homomorphisms between $\mathbb{G}_1$ and $\mathbb{G}_2$ that are efficiently computable.*

- $\mathsf{KeyGen}(\mathsf{pp})$ *: This algorithm samples $\tilde{g} \leftarrow_R \mathbb{G}_2$ and $(x, y_1, \ldots, y_\ell) \leftarrow_R \mathbb{F}_p^{n+1}$, computes $(\tilde{X}, \tilde{Y}_1, \ldots, \tilde{Y}_\ell) = (\tilde{g}^x, \tilde{g}^{y_1}, \ldots, \tilde{g}^{y_\ell})$, and outputs $(\mathsf{sk}, \mathsf{pk})$, where $\mathsf{sk} = (x, y_1, \ldots, y_\ell)$ and D $\mathsf{pk} = (\tilde{g}, \tilde{X}, \tilde{Y}_1, \ldots, \tilde{Y}_\ell)$.*

- $\mathsf{Sign}(\mathsf{sk}, m_1, \ldots, m_\ell)$ *: This algorithm samples $h \leftarrow_R \mathbb{G}_1 \backslash \{0\}$, and outputs $\sigma = (h, h^{x + \sum_j y_j m_j})$.*

– $\mathsf{Verify}(\mathsf{pk}, (m_1, \ldots, m_\ell), \sigma)$ *: This algorithm parses* σ *as* (σ_1, σ_2)*, and first checks if* $\sigma_1 \neq \mathsf{e}_1$*. It then proceeds to check if*

$$e\left(\sigma_1, \tilde{X} \cdot \prod_j \tilde{Y}_j^{m_j}\right) = e(\sigma_2, \tilde{g}).$$

If yes, it outputs 1, and outputs 0 otherwise.

Note that given $\sigma = (\sigma_1, \sigma_2)$, $\sigma' = (\sigma_1^r, \sigma_2^r)$ is also a valid signature if σ is a valid signature. However, it can be seen that the distribution of σ is not independent of the message $\boldsymbol{m}$ in the above scheme.

4.7.1 Proof of Knowledge of PS Signatures

PS signatures support an efficient zero-knowledge proof of knowledge (ZKPoK) wherein a prover holding a valid PS signature σ on a message vector $\boldsymbol{m}$ can efficiently prove knowledge of the signature. A prover $\mathcal{P}$ who owns a PS signature $\sigma = (\sigma_1, \sigma_2)$ on a message $\boldsymbol{m} = (m_1, \ldots, m_\ell) \in \mathbb{F}_p^\ell$ can prove knowledge of such a signature using a slight modification of the signature scheme as described above. At a high level, $\mathcal{P}$ generates a signature on a a pair $(\boldsymbol{m}, t)$ for uniformly sampled $t \leftarrow_R \mathbb{F}_p$ based on the original signature σ; the usage of a random t makes the resulting signature independent of $\boldsymbol{m}$. The complete protocol is as below:

– **Public Key** $\mathsf{pk} = (\tilde{g}, \tilde{X}, \tilde{Y}_1, \ldots, \tilde{Y}_\ell)$

– $\mathcal{P}$**'s inputs**: Message $\boldsymbol{m} \in \mathbb{F}_p^\ell$ and signature $\sigma = (\sigma_1, \sigma_2)$ on $\boldsymbol{m}$

1. $\mathcal{P}$ samples $r, t \leftarrow_R \mathbb{F}_p$ and computes $\sigma' = (\sigma_1^r, (\sigma_2 \cdot \sigma_1^t)^r)$.
2. $\mathcal{P}$ sends the computed value $\sigma' = (\sigma_1', \sigma_2')$ to $\mathcal{V}$.
3. $\mathcal{P}$ and $\mathcal{V}$ run a ZKPoK of $(\boldsymbol{m}, t)$ for the relation:

$$e(\sigma_1', \tilde{X}) \cdot \prod_j e(\sigma_1', \tilde{Y}_j)^{m_j} \cdot e(\sigma_1', \tilde{g})^t = e(\sigma_2', \tilde{g}).$$

4. $\mathcal{V}$ accepts if the ZKPoK is valid.

The proof of knowledge protocol used in Step (3) is a special case of "proof of opening", wherein we can use a protocol for proving the knowledge of $\boldsymbol{s} \in \mathbb{F}_p^\ell$ which opens the commitment $z = \boldsymbol{g}^{\boldsymbol{s}}$ where $\boldsymbol{g} = (g_1, \ldots, g_\ell)$ and $g_1, \ldots, g_\ell$ are public generators of a group $\mathbb{G}$ (of order p), where the discrete log problem is hard. We describe the protocol concretely below.

– $\mathcal{P}$ **and** $\mathcal{V}$**'s common inputs**: $z \in \mathbb{G}$.

– $\mathcal{P}$**'s private inputs**: $\boldsymbol{s} \in \mathbb{F}_p^\ell$.

1. $\mathcal{P}$ samples $\boldsymbol{r} \leftarrow_R \mathbb{F}_p^\ell$ and computes $\alpha = g^{\boldsymbol{r}}$.
2. $\mathcal{P} \rightarrow \mathcal{V}$: α.
3. $\mathcal{V} \rightarrow \mathcal{P}$: $c \leftarrow_R \mathbb{F}_p$.
4. $\mathcal{P} \rightarrow \mathcal{V}$: $\boldsymbol{s}' = c\boldsymbol{s} + \boldsymbol{r}$.
5. $\mathcal{V}$ checks: $g^{\boldsymbol{s}'} = \alpha z^c$.

We also describe another variant of PS Signature Scheme, based on a stronger assumption (Assumption 1 in [97]), that leads to much more efficient distributed prover protocols. This variant is same as the one described in Definition 4.14, with the exception of KeyGen algorithm which includes additional elements in the public key (hence stronger assumption). The modified KeyGen algorithm is described below:

Definition 4.15 (PS Signature: B [97]) *The PS Signature Scheme to sign a message* $\boldsymbol{m} = (m_1, \ldots, m_\ell) \in \mathbb{F}_p^\ell$ *consists of a tuple of PPT algorithms* (Setup, KeyGen, Sign, Verify) *as described in Definition 4.14, except* KeyGen *which is described below:*

– $\mathsf{KeyGen}(\mathsf{pp})$*: The algorithm samples* $g \leftarrow_R \mathbb{G}_1$, $\tilde{g} \leftarrow_R \mathbb{G}_2$, $(x, y_1, \ldots, y_{\ell+1}) \leftarrow_R \mathbb{F}_p^{\ell+1}$ *and computes* $(X, Y_1, \ldots, Y_{\ell+1}) = (g^x, g^{y_1}, \ldots, g^{y_{\ell+1}})$, $(\tilde{X}, \tilde{Y}_1, \ldots, \tilde{Y}_{\ell+1}) = (\tilde{g}^x, \tilde{g}^{y_1}, \ldots, \tilde{g}^{y_{\ell+1}})$*. It then outputs* $(\mathsf{sk}, \mathsf{pk})$ *where* $\mathsf{sk} = (x, y_1, \ldots, y_{\ell+1})$ *and* $\mathsf{pk} = (g, Y_1, \ldots, Y_{\ell+1}, \tilde{g}, \tilde{X}, \tilde{Y}_1, \ldots, \tilde{Y}_{\ell+1})$.

– $\mathsf{Sign}(\mathsf{sk}, (m_1, \ldots, m_\ell))$*: Choose* $h \leftarrow_R \mathbb{G}_1 \backslash \{0\}$ *and output* $(h, h^{x + \sum_{i=1}^{\ell} y_i \cdot m_i})$*. Note that* Sign *still works on the* ℓ*-length message.*

4.7.2 Alternate Proof of Knowledge of PS Signatures

We describe a protocol for showing knowledge of a PS signature (σ_1, σ_2) on a message $\boldsymbol{m} \in \mathbb{F}_p^\ell$ while simultaneously revealing a dynamically sampled commitment C of $\boldsymbol{m}$. The proof of knowledge reduces to the knowledge of opening of C and a short pairing check as described below:

– **Public Key** $\mathsf{pk} = (g, Y_1, \ldots, Y_{\ell+1}, \tilde{g}, \tilde{X}, \tilde{Y}_1, \ldots, \tilde{Y}_{\ell+1})$

– $\mathcal{P}$**'s inputs**: Message $\boldsymbol{m} \in \mathbb{F}_p^\ell$ and signature $\sigma = (\sigma_1, \sigma_2)$ on $\boldsymbol{m}$

1. $\mathcal{P}$ samples $r, t, s \leftarrow_R \mathbb{F}_p$ and computes $\sigma' = (\sigma_1^r, (\sigma_2 \cdot \sigma_1^t)^r \cdot Y_{\ell+1}^s)$, $C = \tilde{g}^t \prod_{i=1}^{l} \tilde{Y}_i^{m_i} \in \mathbb{G}_2$.
2. $\mathcal{P}$ sends the computed value $\sigma' = (\sigma'_1, \sigma'_2)$ and C to $\mathcal{V}$.
3. $\mathcal{P}$ and $\mathcal{V}$ run a ZKPoK showing knowledge of $(m_1, \ldots, m_\ell, t)$ such that $C = \tilde{g}^t \prod_{i=1}^{\ell} \tilde{Y}_i^{m_i}$ and a ZKPoK showing knowledge of s such that $e(Y_{\ell+1}, \tilde{g})^s = e(\sigma'_2, \tilde{g}) e(\sigma'_1, \tilde{X})^{-1} e(\sigma'_1, C)^{-1}$.
4. $\mathcal{V}$ accepts if the ZKPoKs are valid.

Proof: For completeness, notice that $\sigma_2 = \sigma_1^{x+\sum_{i=1}^{\ell} y_i m_i}$ and thus we have $\sigma'_1 = \sigma_1^r$, $\sigma'_2 = Y_{\ell+1}^s \cdot \sigma_1^{r(x+\sum_{i=1}^{\ell} y_i m_i + t)}$ and $C = \tilde{g}^t \prod_{i=1}^{\ell} \tilde{Y}_i^{m_i}$. Thus we have:

$$\begin{aligned} e(\sigma'_2, \tilde{g}) &= e(\sigma_1^r, \tilde{g}^{x+\sum_{i=1}^{\ell} y_i m_i + t}) \cdot e(Y_{\ell+1}, \tilde{g})^s \\ &= e(\sigma'_1, \tilde{X}) \cdot e(\sigma'_1, C) \cdot e(Y_{\ell+1}, \tilde{g})^s \end{aligned}$$

The above is equivalent to the verification relation. Zero-knowledge follows from the fact that σ'_1, σ'_2 and C are distributed uniformly in their respective domains, and from the zero-knowledge property of the ZKPoKs. To show knowledge soundness, we show an extractor $\mathcal{E}$ which extracts a valid signature on a message in $\mathbb{F}_p^\ell$. Using the extractors for the ZKPoKs, $\mathcal{E}$ obtains $(m_1, \ldots, m_\ell, t, s)$ such that

$$C = \tilde{g}^t \prod_{i=1}^{\ell} \tilde{Y}_i^{m_i}, \quad e(\sigma'_2, \tilde{g}) = e(\sigma'_1, \tilde{X}) \cdot e(\sigma'_1, C) \cdot e(Y_{\ell+1}, \tilde{g})^s$$

The extractor $\mathcal{E}$ computes $\left(\sigma_1 = \sigma'_1, \sigma_2 = \sigma'_2 (\sigma'_1)^{-t} (Y_{\ell+1})^{-s}\right)$. To see that (σ_1, σ_2) is a valid signature we verify:

$$\begin{aligned} e(\sigma_2, \tilde{g}) &= e(\sigma'_2, \tilde{g}) \cdot e(\sigma'_1, \tilde{g})^{-t} \cdot e(Y_{\ell+1}, \tilde{g})^{-s} \\ &= e(\sigma'_1, \tilde{X}) \cdot e(\sigma'_1, C) \cdot e(\sigma'_1, \tilde{g})^{-t} \\ &= e(\sigma'_1, \tilde{X}) \cdot e(\sigma'_1, \prod_{i=1}^{\ell} \tilde{Y}_i^{m_i}) \\ &= e(\sigma_1, \tilde{X} \prod_{i=1}^{\ell} \tilde{Y}_i^{m_i}) \end{aligned}$$

The above shows (σ_1, σ_2) is a valid signature for the block $(m_1, \ldots, m_\ell)$ for the public key $(\tilde{g}, \tilde{X}, \tilde{Y}_1, \ldots, \tilde{Y}_\ell)$. □

4.7.3 DPoK for PS Signatures over Secret-Shared Inputs

We now present a DPoK for PS signatures for secret-shared inputs. We start by defining a relation relevant to PS signature verification.

Definition 4.16 (PS Relation) *Let* PSGen *denote the relation generator, such that* $\mathsf{PSGen}(1^{1^\lambda}, \ell)$ *outputs a bilinear group* $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, e, p) \leftarrow_R PS.\mathsf{Setup}(1^{1^\lambda})$. *The corresponding relation* $\mathcal{R}^{\text{ps}}$ *is defined by* $(\boldsymbol{x}, (\boldsymbol{m}, \boldsymbol{u})) \in \mathcal{R}^{\text{ps}}$ *for* $\boldsymbol{x} = \mathsf{pk} = (g, Y_1, \ldots, Y_{\ell+1}, \tilde{g}, \tilde{X}, \tilde{Y}_1, \ldots, \tilde{Y}_{\ell+1}) \in \mathbb{G}_1^{\ell+2} \times \mathbb{G}_2^{\ell+3}$, $\boldsymbol{m} = (m_1, \ldots, m_\ell) \in \mathbb{F}_p^\ell$ *and* $\boldsymbol{u} = (\sigma, t) = ((\sigma_1, \sigma_2), t) \in \mathbb{G}_1^2 \times \mathbb{F}_p$ *if*

$$e(\sigma_1', \tilde{X}) \cdot \prod_j e(\sigma_1', \tilde{Y}_j)^{m_j} \cdot e(\sigma_1', \tilde{g})^t = e(\sigma_2', \tilde{g}).$$

Our Protocol Π_{ps}. Our DPoK protocol Π_{ps} for relation PSGen is described below, which can be invoked from our compiler with input authentication based on PS signatures (instead of BBS+). It builds upon the known PS PoK [97] in the non-distributed setting. The PoK involved the following steps: (i) the prover randomizes the signature using some auxiliary inputs and broadcasts the randomized signature to all other parties (this randomization ensures unlinkability), and then (ii) the prover shows knowledge of these auxiliary inputs and secret-shares of the message satisfying discrete-log relations determined by the first message.

Our PS PoK over secret-shared inputs follows the same blueprint, where the prover similarly randomizes the first message using certain auxiliary inputs. In our case, the problem reduces to a DPoK for the discrete log relation, with the workers holding the shares of the witness (message) and the verifier holding the public statement (public key pk + the randomized signature). We handle this using our robust complete DPoK Π_{dlog} for discrete log.

We note that DPoK protocol Π_{ps} achieves robust completeness, knowledge-soundness and zero-knowledge. The proof is straightforward from the existing proof of knowledge of PS signatures and robust completeness, knowledge-soundness and zero-knowledge properties of our DPoK protocol Π_{dlog} for discrete log.

Theorem 4.6 *Assuming that* Π_{dlog} *is a* $\mathsf{DPoK}_{\mathsf{SSS},\mathsf{DlogGen}}$ *for relation generator* DlogGen *and* (t, n)-SSS, Π_{ps} *is a* DPoK *for the relation generator* PSGen *and* (t, n)-SSS *with the following properties:*

- ***Security****: t-private and d-robust, for* $d < \mathsf{dist}/2$, *where* $\mathsf{dist} = (n - t)$ *is the minimum distance of the Reed-Solomon code induced by* (t, n)-SSS.

– **Public Key** $\mathsf{pk} = (g, Y_1, \ldots, Y_{\ell+1}, \tilde{g}, \tilde{X}, \tilde{Y}_1, \ldots, \tilde{Y}_{\ell+1})$

– $\mathcal{P}$**'s inputs**: Message $\boldsymbol{m} = (m_1, \ldots, m_\ell) \in \mathbb{F}_p^\ell$ and signature $\sigma = (\sigma_1, \sigma_2)$ on $\boldsymbol{m}$

– $\mathcal{W}_i$**'s inputs :** $\mathcal{W}_i$ possesses the i^{th} share $\boldsymbol{m}_i$ of the message vector $\boldsymbol{m}$, such that $\mathsf{Reconstruct}(\boldsymbol{m}_1, \ldots, \boldsymbol{m}_n) = \boldsymbol{m}$

– **Preprocessing** : $\mathcal{P}$ samples $t \leftarrow_R \mathbb{F}_p$, computes $(t_1, \ldots, t_n) \leftarrow_R \mathsf{Share}(t)$. $\mathcal{P}$ sends the shares t_i to $\mathcal{W}_i$, for all $i \in [n]$.

– **Interactive Protocol**

1. $\mathcal{P}$ samples $r, v \leftarrow_R \mathbb{F}_p$ and computes $\sigma' = (\sigma_1^r, (\sigma_2 \cdot \sigma_1^t)^r \cdot Y_{\ell+1}^v)$, $C = \tilde{g}^t \prod_{i=1}^{\ell} \tilde{Y}_i^{m_i}$. P also generates a NIPK π showing knowledge of v such that $e(\sigma_1', \tilde{X}) \cdot e(\sigma_1', C) \cdot e(Y_{\ell+1}, \tilde{g})^v = e(\sigma_2', \tilde{g})$.
2. $\mathcal{P}$ broadcasts the computed value $\sigma' = (\sigma_1', \sigma_2')$, C and π to $\mathcal{V}$.
3. Each $\mathcal{W}_i$ and $\mathcal{V}$ locally set $\boldsymbol{g} = (\tilde{g}, \tilde{Y}_1, \ldots, \tilde{Y}_\ell)$.
4. Each $\mathcal{W}_i$ locally holds the i-th share $\boldsymbol{s}_i = (\boldsymbol{m}_i, t_i)$ such that $\boldsymbol{s} = (\boldsymbol{m}, t) = \mathsf{Reconstruct}\left(\{\boldsymbol{s}_i\}_{i \in [n]}\right)$.
5. All $\mathcal{W}_i$ for $i \in [n]$ and $\mathcal{V}$ run DPoK protocol Π_{dlog} for the relation $\boldsymbol{g}^{\boldsymbol{s}} = C$
6. $\mathcal{V}$ accepts if π is valid and Π_{dlog} accepts.

Figure 4.5: Protocol Π_{ps}

– ***Efficiency***: *$O(n)$ communication over point-to-point channels and $O(n \log \ell)$ communication over broadcast channels.*

Remark 8 (Public Verifiability) *The protocol Π_{ps} was presented and analyzed assuming an honest designated verifier for simplicity. By replacing Π_{dlog} with its publicly verifiable version $\Pi^{\mathsf{pv}}_{\mathsf{dlog}}$ in steps (5) of the Interactive Phase, we obtain a publicly verifiable version of the protocol, which we call $\Pi^{\mathsf{pv}}_{\mathsf{ps}}$. Observe that $\Pi^{\mathsf{pv}}_{\mathsf{ps}}$ requires one less round of interaction, as compared to Π_{ps}, while it retains the properties of robust completeness, knowledge soundness and honest verifier zero-knowledge holds identically for the Π_{ps}.*

4.8 Application of Distributed Proofs of Knowledge in Input Authentication in MPC

In this section we present our compiler for MPC with input authentication that outputs an MPC protocol where each input is authenticated using a BBS+ signature under a common (public) verification key. We can also obtain a compiler for MPC with input authentication using similar techniques for PS Signatures discussed in Section 4.7.3.

Class of MPC Protocols. Our compiler takes advantage of the observation that a large class of secret-sharing based MPC protocols share the following template. (i) There is an input sharing phase where parties secret-share their inputs, and (ii) when using secret sharing schemes with certain thresholds ($t_{\mathsf{sh}} < |H|$), the input of parties is completely determined at the end of the input sharing phase. This means that using inputs inconsistent with this sharing is considered deviating, against which the protocol is secure. This is precisely where our compiler performs well: verification of authenticity (or any other predicate) on the inputs can be done fully outside the MPC by running a DPoK on the shares. (iii) For an MPC protocol of this template, there exists a simulator $\mathsf{Sim} = (\mathsf{Sim}_{\mathsf{sh}}, \mathsf{Sim}_{\mathsf{on}})$, where $\mathsf{Sim}_{\mathsf{sh}}$ deterministically extracts the inputs of corrupt parties, and $\mathsf{Sim}_{\mathsf{on}}$ simulates the protocol view.

Features of Our Compiler. Our compiler allows identification of all (malicious) parties with non-authenticated inputs (this is a consequence of the robust completeness property of Π_{dlog} used inside $\Pi_{\mathsf{bbs+}}$). We further note that our robust protocol Π_{dlog} tolerates a maximum corruption threshold of $t < n/3$ (assuming that the secret-sharing used is Shamir's secret sharing). Hence, our compiled MPC protocol also tolerates a maximum corruption threshold of $t < n/3$. Using the non-robust version will result in a non-robust compiler that retains the $t < n/2$ threshold of the underlying MPC.

Inputs

The ideal functionality receives from each party P_i an input-signature pair of the form $(\boldsymbol{x}_i, \sigma_i)$ under the public verification key pk.

Verify Authenticity

1. If $\mathsf{Ver}(\mathsf{pk}, x_i, \sigma_i) \neq 1$ for some party P_i, then output a set of corrupted parties C and abort.
2. Otherwise, proceed to computation.

Computation

Invoke the ideal functionality $\mathcal{F}_{\mathsf{MPC}}$ for Π_{mpc} on inputs $(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)$.

Figure 4.6: Functionality $\mathcal{F}_{\mathsf{MPC}}^{\mathsf{auth}}$

The Desired Ideal Functionality. We define below the desired ideal functionality $\mathcal{F}_{\mathsf{MPC}}^{\mathsf{authid}}$ for MPC with input authentication.

4.8.1 Our Compiler for Authenticated MPC

We now present a formal description of our compiler. Let $\Pi_{\mathsf{mpc}} = (\Pi_{\mathsf{sh}}, \Pi_{\mathsf{on}})$ be a secret-sharing based MPC protocol that guarantees security with abort against malicious corruptions of a dishonest majority of the parties $\{P_1, \ldots, P_n\}$, where:

- Π_{sh} denotes the secret-sharing phase of Π_{mpc} and consists of the steps used by each party P_i for $i \in [n]$ to secret-share its input $\boldsymbol{x}_i \in \mathbb{F}_p^\ell$ to all of the other parties (throughout, we assume that this sharing is done using a linear secret-sharing scheme $(\mathsf{Share}, \mathsf{Reconstruct})$.
- Π_{on} denotes the remaining steps of the protocol Π_{mpc} where the parties interact to compute $y = f(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)$.

In the description of our compiler, we assume that each party P_i holds a BBS+ signature σ_i on its input $\boldsymbol{x}_i$ with respect to a common public verification key pk. The compiler runs n instances of $\Pi_{\mathsf{bbs+}}$, where for instance i, party P_i acts as the prover and all other parties P_j for $j \neq i$ act as verifiers. Given $\Pi_{\mathsf{mpc}} = (\Pi_{\mathsf{sh}}, \Pi_{\mathsf{on}})$, our robust compiler outputs an authenticated MPC protocol $\Pi_{\mathsf{ampc}} = (\overline{\Pi}_{\mathsf{sh}}, \overline{\Pi}_{\mathsf{on}})$. The compiler Π_{ampc} is described above.

Theorem 4.7 (Security of Π_{ampc}) *Assuming that: (a) the MPC protocol Π_{mpc} securely emulates the ideal functionality $\mathcal{F}_{\mathsf{MPC}}$, and (b) Π_{dlog} is a $\mathsf{DPoK}_{\mathsf{SSS},\mathsf{DlogGen}}$ for relation generator*

– **Inputs:** All parties hold public parameters and the verification key pk of a BBS+ signature scheme. Party P_i has input $\boldsymbol{x}_i \in \mathbb{F}_p^\ell$, together with a signature σ_i, such that $(\mathsf{pk}, (\boldsymbol{x}_i, \sigma_i)) \in \mathcal{R}^{\text{bbs}}$.

– $\overline{\Pi}_{\mathsf{sh}}$: This phase is identical to Π_{sh}, i.e., each party P_i shares its input $\boldsymbol{x}_i$ to all other parties exactly as in Π_{sh}.

– $\overline{\Pi}_{\mathsf{on}}$: In this phase, the parties do the following:

  – For each $j = 1, \ldots, n$, the parties execute an instance of $\Pi_{\mathsf{bbs+}}$ for $(\mathsf{pk}, (\boldsymbol{x}_j, \sigma_j)) \in \mathcal{R}^{\text{bbs}}$ with $\mathcal{P}_j$ acting as the Prover, $\mathcal{P}_1, \ldots, \mathcal{P}_n$ constituting the workers and $\mathcal{P}_i$, $i \neq j$ acting as verifiers, .

    If any party outputs 0 at the end of this phase, the protocol aborts.

  – Otherwise, the parties jointly execute Π_{on}.

Figure 4.7: Protocol $\Pi_{\mathsf{ampc}} = (\overline{\Pi}_{\mathsf{sh}}, \overline{\Pi}_{\mathsf{on}})$

DlogGen *and* (t, n)-SSS *our compiled MPC protocol with input authentication* Π_{ampc} *securely emulates the ideal functionality* $\mathcal{F}_{\mathsf{MPC}}^{\mathsf{auth}}$ *for the same corruption threshold of* $t < n/3$.

Proof. We construct a simulator for the Π_{ampc} protocol, and prove indistinguishability of the simulation from a real-world execution of Π_{ampc}. The underlying MPC protocol Π_{mpc} secure emulates $\mathcal{F}_{\mathsf{MPC}}$, and let $\mathsf{Sim} = (\mathsf{Sim}_{\mathsf{sh}}, \mathsf{Sim}_{\mathsf{on}})$ be the corresponding simulator.

Simulator for Π_{ampc}. We now describe the simulator $\overline{\mathsf{Sim}}$ for the authenticated MPC protocol $\Pi_{\mathsf{ampc}} = \big(\overline{\Pi}_{\mathsf{sh}}, \overline{\Pi}_{\mathsf{on}}\big)$. Let $\mathcal{H} \subseteq [n]$ and $\mathcal{C} \subset [n]$ denote the set of honest and corrupt parties, respectively. The simulator $\overline{\mathsf{Sim}}$ proceeds as follows:

1. Simulate the sharing phase $\overline{\Pi}_{\mathsf{sh}}$ of the underlying MPC Π_{mpc} by invoking $\mathsf{Sim}_{\mathsf{sh}}$ (note that $\mathsf{Sim}_{\mathsf{sh}}$ does not expect any inputs). $\overline{\mathsf{Sim}}$ receives the ith share $\{\boldsymbol{s}_i^j\}_{i \in \mathcal{H}}$ from the adversary (invoked by $\mathsf{Sim}_{\mathsf{sh}}$) corresponding to the input $\boldsymbol{s}^j$ of each corrupt party $P_j, j \in \mathcal{C}$.

2. For each P_j s.t. $j \in \mathcal{C}$, let $(\Pi_{\mathsf{bbs+}})_j$ denote the instance of the protocol $\Pi_{\mathsf{bbs+}}$ used by the parties where P_j acts as the prover, and all of the remaining parties acting as both workers and verifiers. The simulation of the online phase proceeds as follows.

   (a) First, the simulator of the online phase invokes the simulator of the underlying DPoK $\Pi_{\mathsf{bbs+}}$ to simulate the proofs of knowledge of BBS+ signatures on the inputs of the honest parties.

(b) For each instance $\Pi_{\mathsf{bbs+}}$, where a corrupt party P_j, $j \in \mathcal{C}$ is acting as the prover, invoke the extractor Ext' of the DPoK $\Pi_{\mathsf{bbs+}}$ on the shares of the honest parties $(\boldsymbol{s}_i^j)_{i\in\mathcal{H}}$ corresponding to the corrupt party P_j's input to extract the witness $(\boldsymbol{x}_j, \sigma_j)$ from P_j. Note that since we assume honest-majority, the shares $\{\boldsymbol{s}_i^j\}_{i\in\mathcal{H}}$ given as input to the extractor Ext' completely determines the respective inputs of each corrupt party $P_j, j \in \mathcal{C}$. Hence, the compiler aborts if $\mathsf{Consistent}(\boldsymbol{x}_j, \{\boldsymbol{s}_i^j\}_{i\in\mathcal{H}}) = 0$.

(c) Invoke $\mathsf{Sim}_{\mathsf{on}}$ to simulate the online phase of the underlying MPC Π_{mpc}.

3. Send $\{(\boldsymbol{x}_j, \sigma_j)\}_{j\in\mathcal{C}}$ to $\mathcal{F}_{\mathsf{MPC}}^{\mathsf{auth}}$. If $\mathcal{F}_{\mathsf{MPC}}^{\mathsf{auth}}$ aborts by identifying some subset of corrupt parties, abort while identifying the same subset of corrupt parties; otherwise output whatever $\mathcal{F}_{\mathsf{MPC}}^{\mathsf{auth}}$ outputs.

Completing the Security Proof. We now prove the security of Π_{ampc} by using a sequence of hybrids described as follows (for simplicity of exposition, we assume w.l.o.g. that parties $P_1, \ldots, P_{|\mathcal{C}|}$ are corrupt and parties $P_{|\mathcal{C}|+1}, \ldots, P_n$ are honest):

- Hyb_0: This hybrid is identical to the real-world execution of Π_{ampc}.

- Hyb_1: This hybrid is identical to Hyb_0 except that we simulate the sharing phase $\overline{\Pi}_{\mathsf{sh}}$ of the underlying Π_{mpc} protocol by invoking $\mathsf{Sim}_{\mathsf{sh}}$. Receive from $\mathsf{Sim}_{\mathsf{sh}}$ the set of shares $\{\boldsymbol{s}_i^j\}_{i\in\mathcal{H}}$ corresponding to the input $\boldsymbol{s}^j$ of each corrupt party $P_j, j \in \mathcal{C}$.

- $\{\mathsf{Hyb}_{2,j}\}_{j\in[0,n-|\mathcal{C}|]}$: Hybrid $\mathsf{Hyb}_{2,0}$ is identical to hybrid Hyb_1, and for each $j \in [1, n-|\mathcal{C}|]$, hybrid $\mathsf{Hyb}_{2,j}$ is identical to $\mathsf{Hyb}_{2,(j-1)}$ except that proof of knowledge corresponding to the input of honest party $P_{|\mathcal{C}|+j}$ is simulated using Sim' as described in Step 2(a) of the simulator. More concretely, for each honest party $P_{|\mathcal{C}|+j}$, instead of using the real input $\boldsymbol{x}_{|\mathcal{C}|+j}$ and the real BBS+ signature $\sigma_{|\mathcal{C}|+j}$, proof of knowledge of a BBS+ signature is simulated instead of running an instance of the protocol $\Pi_{\mathsf{bbs+}}$ where party $P_{|\mathcal{C}|+j}$ is the prover.

- $\{\mathsf{Hyb}_{3,j}\}_{j\in[0,|\mathcal{C}|]}$: The first of these hybrids, i.e., Hybrid $\mathsf{Hyb}_{3,0}$ is identical to hybrid $\mathsf{Hyb}_{2,n-|\mathcal{C}|}$. Next, for each $j \in [1, |\mathcal{C}|]$, hybrid $\mathsf{Hyb}_{3,j}$ is identical to $\mathsf{Hyb}_{3,(j-1)}$ except that we abort if the following bad event occurs: for the instance of $\Pi_{\mathsf{bbs+}}$ where the corrupt party P_j is the prover, invoke the extractor Ext' (as mentioned in Step 2(b) of the simulator and described in the proof overview) on the shares of the honest parties $(\boldsymbol{s}_i^j)_{i\in\mathcal{H}}$ corresponding to the corrupt party P_j's input to extract the witness $(\boldsymbol{x}_j, \sigma_j)$ from P_j. If $(\mathsf{pk}, (\boldsymbol{x}_j, \sigma_j)) \notin \mathcal{R}^{\mathrm{bbs}}$ or $\mathsf{Consistent}(\boldsymbol{x}_j, \{\boldsymbol{s}_i^j\}_{i\in\mathcal{H}}) = 0$, then abort.

– Hyb_4: This hybrid is identical to $\mathsf{Hyb}_{3,|\mathcal{C}|}$ except for the following: invoke $\mathsf{Sim}_{\mathsf{on}}$ of the underlying Π_{mpc} protocol to simulate the online phase $\overline{\Pi}_{\mathsf{on}}$, and output whatever $\mathsf{Sim}_{\mathsf{on}}$ outputs.

– Hyb_5: This hybrid is identical to Hyb_4 except that after invoking $\mathsf{Sim}_{\mathsf{on}}$ to simulate $\overline{\Pi}_{\mathsf{on}}$, we query $\mathcal{F}_{\mathsf{MPC}}^{\mathsf{auth}}$ with the extracted inputs $\{(\boldsymbol{x}_j, \sigma_j)\}_{j\in\mathcal{C}}$.

$\mathsf{Hyb}_0 \approx_c \mathsf{Hyb}_1$. This follows from the security of the underlying Π_{mpc} protocol. Suppose that there exists a PPT adversary $\mathcal{A}$ that can distinguish between Hyb_0 and Hyb_1. It is easy to use $\mathcal{A}$ to construct a PPT adversary $\mathcal{A}'$ that can distinguish between a real and simulated execution of Π_{sh}, thus breaking security of the underlying Π_{mpc} protocol.

$\mathsf{Hyb}_{2,j-1} \approx_c \mathsf{Hyb}_{2,j}$. This follows from the ZK property of Π_{dlog} and the PoK for single-prover version of BBS+ signatures. In particular, suppose that there exists a PPT adversary $\mathcal{A}$ that can distinguish between $\mathsf{Hyb}_{2,(j-1)}$ and $\mathsf{Hyb}_{2,j}$ for some $j \in [1, n - |\mathcal{C}|]$. Then $\mathcal{A}$ can be used to construct one of the following algorithms: (a) either an adversary $\mathcal{A}'$ that breaks the ZK property of the Π_{dlog} protocol, or (b) an adversary $\mathcal{A}''$ that breaks the ZK property of the PoK for single-prover version of BBS+ signatures.

$\mathsf{Hyb}_{3,j-1} \approx_c \mathsf{Hyb}_{3,j}$. This follows from knowledge soundness of Π_{dlog}. The two hybrids differ only when the bad event occurs, i.e., the extractor Ext' in Step 2(b) of the simulator fails to output a valid witness $(\boldsymbol{m}, \sigma)$ where $\boldsymbol{m}$ is consistent with the honest party shares. However, as described in the proof overview, assuming the knowledge-soundness of Π_{dlog}, the extractor Ext' outputs a valid witness. Hence, assuming knowledge-soundness of Π_{dlog}, the probability of the bad event occurring must be negligible.

$\mathsf{Hyb}_4 \approx_c \mathsf{Hyb}_{3,|\mathcal{C}|}$. This follows from the security of the underlying Π_{mpc} protocol. At the end of Π_{sh}, if abort did not occur, then for each $i \in [n]$, all honest parties hold shares $\langle \boldsymbol{x}'_j \rangle_{j\in\mathcal{H}}$ of some $\boldsymbol{x}'_i \in \mathbb{F}^\ell$. In $\mathsf{Hyb}_{3,|\mathcal{C}|}$, the extractor succeeds in outputting a valid witness $\boldsymbol{x}_i$, and this is the unique $\boldsymbol{x}'_i$ determined at the end of Π_{sh}. Suppose that there exists a PPT adversary $\mathcal{A}$ that can distinguish between Hyb_4 and $\mathsf{Hyb}_{3,|\mathcal{C}|}$. It is easy to use $\mathcal{A}$ to construct a PPT adversary $\mathcal{A}'$ that can distinguish between a real and simulated execution of Π_{on}, thus breaking the security of the underlying Π_{mpc} protocol.

$\mathsf{Hyb}_5 \equiv \mathsf{Hyb}_4$. Hyb_5 and Hyb_4 are identical. In Hyb_4, the output of is given by the output of $\mathsf{Sim}_{\mathsf{on}}$ and in Hyb_5, the output is given by the output of $\mathsf{Sim}_{\mathsf{sh}}$, which are idential by the security of the underlying Π_{mpc}. We also note that Hyb_5 is identical to $\overline{\mathsf{Sim}}$.

This completes the proof of Theorem 4.7. ■

Round Efficient Compiler for Authenticated MPC. Finally, it is easy to see that invoking the round efficient $\mathsf{DPoK}\ \Pi^{\mathsf{FS}}_{\mathsf{bbs+}}$ protocol instead of the $\mathsf{DPoK}\ \Pi_{\mathsf{bbs+}}$ protocol enables us to obtain a round efficient version of our compiler. The round efficient version achieves the same security guarantees as the compiler presented above, albeit in the random oracle model.

Chapter 5

Updatable Lookup Arguments and its Application in Batching-efficient RAM

In this chapter[1], we present *updatable lookup arguments* that enables us to perform lookups on tables that have been changed after the preprocessing of expensive parameters, by removing the rigid dependency of the online phase on the *table-dependent* preprocessing. We also look at our constructions for committed index lookup arguments, which takes a step further from the traditional lookup arguments that prove the sub-vector relations, and *ties* the proof to the indices of the elements being 'looked up'. Finally, using our updatable lookup argument as a building block, along with other primitives, we provide a *batching-efficient* RAM (Random Access Memory) that has constant proof size, constant verification complexity, and prover complexity that is sublinear in the size of the RAM.

5.1 Introduction

To motivate our key ZKP primitive of *updatable lookup argument*, we begin by discussing its application in the well-understood primitive of RAM. We highlight the established importance and the extensive prior work on efficiently proving correctness of RAM updates. General purpose *Succinct Non-interactive Arguments of Knowledge* (SNARKs) enable one to generate succinct proofs of membership of a statement in an NP relation expressed as an arithmetic circuit. These proofs are extremely cheap to verify, which makes them useful for *Verifiable Computation* (VC), where a resource-constrained client (e.g., a mobile phone), can outsource an expensive computation to an untrusted server, and later verify the correctness of the computation at a minimal cost.

[1]This chapter is based on the joint work [54] with Chaya Ganesh, Sikhar Patranabis, Shubh Prakash and Nitin Singh, that appeared in ACM CCS 2024.

Modeling RAM in Verifiable Computation. It turns out that arithmetic circuit-based representations are inefficient in expressing relations involving the result of a program execution on memory/state. Such relations frequently arise in the context of verifiable computation, in scenarios that require proving the correctness of query execution against a database, inference from a decision tree, or updates on a table of account balances (e.g., when a batch of transactions, such as account transfers, is applied to the table).

In the aforementioned examples, objects such as database tables, decision trees, and accounts tables can be naturally modeled as instances of addressable memory, or more generally, random access memory (RAM), where one needs to prove that the RAM has been accessed/updated in accordance with the correct execution of the computation. There exists a rich and expanding body of work on efficiently modeling abstractions of RAM in verifiable computation. While a complete treatment of this vast body of work is beyond the scope of this thesis (a fairly recent survey in [106] is a good starting point), we mention two additional properties that are often demanded of the RAM primitive: *persistence* – the ability to persist the RAM state across several computations, and *batching* – where verifiable update of the RAM state is required for small batches of updates. These properties are also the focus of this work.

Application to Blockchain Rollups. Batching-efficient RAM is especially relevant in the context of blockchain *rollups* [14], an umbrella term for recent efforts to scale blockchains by moving expensive computation off the blockchain to the so-called *layer two* (or L2) chains. The blockchain only needs to verify succinct proofs attesting to the correctness of the off-chain computation. This approach is popularly called *rollup* as it allows verifying the result of several (rolled-up) transactions modifying the L2 state, as part of one transaction verified on the main chain. This simultaneously improves scalability and lowers the cost (e.g., gas fees) per transaction due to succinct verification. We consider improving efficiency of rollups an important motivation for our work, but avoid precise details of a smart-contract based instantiation of our solution.

5.1.1 Our Contribution

We present batching-efficient RAM construction, which advances the efforts towards achieving *verifiable outsourcing of state update* such as in [34] and more recently in [92, 42]. The most popular approaches to succinctly represent state involve accumulators based on Merkle-trees [89], or ones based on groups of unknown order (e.g. RSA, class-groups) [39, 31, 92, 42]. The updates to the state are effected by insertions or deletions in the accumulated set. In this work, we model the state as an addressable memory (RAM) described by vector $\boldsymbol{T}$, which stores value v_i at address i. We denote this as $\boldsymbol{T}[\,i\,] = v_i$. The RAM supports two operations,

viz, *loads* expressed as $v_i := \boldsymbol{T}[\,i\,]$, and *stores* expressed as $\boldsymbol{T}[\,i\,] = v_i$. We think of addresses $i \in [0, N]$ for some $N \in \mathbb{Z}$ while the values $v_i \in \mathbb{F}$ for some finite field $\mathbb{F}$. In our construction, we represent both the RAM and operations on it as polynomials, and use appropriate polynomial commitment schemes to obtain succinct commitments (digests) to them. In this chapter, we do not require commitments to be *hiding*, as our focus is on succinctness.

We summarize our contributions below.

– As our first contribution, we propose *update friendly lookup arguments*, which addresses the strict dependence of recent constructions on table-specific preprocessing parameters. Earlier works relied on preprocessing the quotients of the table which has to be 'looked up', where the online phase only requires computing a linear combination of these preprocessed parameters. However, for computation of the quotient in the online phase, this approach strictly relies on these table-dependent preprocessed quotients, which is rendered unusable in the event of any updates to the table. Our innovation extends the utility of table-specific parameters to enable efficient lookups from tables, which are within certain Hamming distance of the preprocessed table.

– We construct *committed index lookup* arguments via black-box reduction to sub-vector arguments that use homomorphic commitments. A committed index lookup involves three committed vectors $\boldsymbol{t}, \boldsymbol{a}$ and $\boldsymbol{v}$ satisfying $v_i = t_{a_i}$ for all i. Similar definition is also used in recent multi-variate lookup arguments in [103], where a similar reduction to sub-vector arguments is obtained under a more restrictive assumption about the elements of the table.

– We crucially employ the above two contributions to construct a *batching-efficient* RAM, which can prove a batch of m updates with an amortized prover complexity of $O(m \log m + \sqrt{mN})$, with N being the size of the RAM. Our dependence on the RAM size is sublinear, in contrast to the linear complexity inherent in recent works on batching-efficient RAM using RSA accumulators [92, 42] or using generic memory checking techniques [105, 21, 19, 114]. All of our protocols are public-coin, and can be made non-interactive using standard techniques [59].

We consider privacy as an orthogonal goal, one we believe is easily achievable via small adaptations to our construction. To also attain privacy, we first require the commitments to be *hiding*. Furthermore, each polynomial must be padded with sufficient number of random masks to allow multiple 'openings' of the same commitment without compromising privacy. For ensuring privacy in updatable lookup argument, we require privacy in the underlying committed index lookup argument, and for ensuring privacy in batching-efficient RAM, we require

privacy across all of its building blocks. For instance, we can ensure privacy in the committed index lookup argument by plugging in a zero-knowledge lookup argument (eg. zkcq+ [43]), and a privacy-preserving memory consistency check can be performed using a zero-knowledge permutation argument. In this work, since our technique focuses on efficient computation of the quotients required from the prover during the protocol execution (whose computation does not involve the verifier), these techniques are compatible with the standard techniques of using hiding commitment schemes to achieve privacy.

5.1.2 Techniques

We present a brief summary of our techniques below. A more detailed technical overview appears in Section 5.1.2.

Update-friendly Lookup Arguments. Our starting point is the recent line of works on lookup arguments which prove that a vector of size m appears as a sub-vector in a large fixed vector (table) of size N with succinct proof sizes and verification, but most notably ensuring that prover runs in time sublinear in the size of the table (N). The pioneering work [110] obtained prover complexity of $O(m^2 + m \log N)$, which was improved in subsequent works to $O(m^2)$ [98], $O(m \log^2 m)$ [111], and $O(m \log m)$ [56, 43]. However, the sublinear prover complexity requires table-dependent $O(N \log N)$ preprocessing and $O(N)$ storage. This table-dependent preprocessing implies that while the aforementioned lookup arguments can be used to obtain efficient ROM (read only memory) semantics and cannot be used as is for RAM (which supports update operations). Moreover, an update involving even a single index renders the entire $O(N)$ preprocessing unusable for further lookups, thus necessitating entire $O(N \log N)$ re-computation. This work is the first effort towards mitigating this rigid dependence, thereby increasing the applicability of the recent lookup arguments. An important contribution we make here is a new method for computing "encoded quotients" used in several recent lookup constructions such as [110, 98, 56, 43]. Our approach for computing these quotients from pre-computed parameters remains efficient even when the table is updated, and it directly applies to all the aforementioned constructions. For a table δ-hamming distance away from the pre-processed one, we incur $(m + \delta) \log^2(m + \delta)$ additional overhead for proving m lookups. To achieve such a quasi-linear overhead in both m and δ, we rely on novel algebraic algorithms described in Section 5.4. We informally summarize our contribution in this regard below, whereas Theorem 5.3 states the precise result.

Theorem 5.1 (Informal) *There exists a deterministic $O(N \log N)$ time algorithm $\mathsf{Preprocess}(\boldsymbol{T}) \to \mathsf{pp}_T$ which on input $\boldsymbol{T} \in \mathbb{F}^N$, outputs parameters pp_T of size $O(N)$ such that: Given pp_T, vectors*

$\boldsymbol{T}' \in \mathbb{F}^N$, $\boldsymbol{t} \in \mathbb{F}^m$ with $\boldsymbol{t}$ being a sub-vector of $\boldsymbol{T}'$ an argument of knowledge for the same can be computed in time $O((m+\delta)\log^2(m+\delta) + f(m))$ where $\delta = \Delta(\boldsymbol{T}, \boldsymbol{T}')$ is the Hamming distance between $\boldsymbol{T}$ and $\boldsymbol{T}'$ while $f(m)$ depends on the specific lookup protocol.

For the constructions based on [110, 98], we set $f(m) = m^2$ in the above, while for [56, 43], we have $f(m) = m \log m$.

Committed Index Lookup : We augment the sub-vector relation in prior lookup arguments which considers whether each entry of a given vector appears in the target vector to one that also identifies the precise positions where the given vector appears in the target vector. When this relation is checked over commitments of the respective vectors; given vector, the target vector and the position vector, we call it *committed index lookup*. The relation we consider is similar to the one considered in [103]. For lookup arguments with homomorphic commitment schemes, we show that committed index lookup can be obtained using a sub-vector lookup argument (Lemma 5.7, Section 5.3.2). Such a construction was also considered in [103], but under a more restrictive assumption that the size of the elements in the table have to be within a certain bound. Lemma 5.7 yields a construction of committed index lookup that uses (a single instance of) the underlying sub-vector protocol in a black-box manner. This immediately implies efficient constructions of arguments for committed index lookups from [110, 98, 111, 56, 43]. In Section 5.3.1, we also present an explicit (non-black-box) adaptation of [98] to obtain a committed index lookup, which again incurs costs comparable to a single instance of the underlying sub-vector protocol.

Batching-Efficient RAM from Lookup Arguments. Memory checking methods based on *address ordered transcripts* [105, 21, 19, 114], which are popularly used in efficient RAM abstractions, incur a cost linear in the size of the RAM. This is prohibitive for efficient batching. As a key idea in this work, we invoke committed index lookup on the large RAMs, to verifiably extract smaller sub-RAMs, which correspond to indices actually involved in the batch update. Then, we use the linear time memory-checking techniques to argue the consistency of these smaller sub-RAMs.

The idea needs to work through some more details, such as showing that the larger RAMs are identical on positions not referenced by the batch of updates (considered in Section 5.5.5). The overall idea is illustrated in Figure 5.1. We also note that the extracted sub-RAMs can have duplicate records, corresponding to multiple updates referencing the same RAM index; however, memory checking methods can be easily adapted to handle such cases. Finally, we would still hit the "rigidity" of lookup arguments in realizing this plan; once the table has changed, lookups are no longer efficient from it. To circumvent this, we use our first contribution on extending

the utility of table-specific parameters to defer parameter re-computation optimally while still availing efficient lookups. More specifically, if we choose to re-compute the full table-specific parameters after k batches (of m updates each), the average cost per batch is $O(N \log N/k + mk \log^2(mk) + f(m))$. Here, $f(m)$ as earlier denotes complexity of the non-updatable base protocol. Setting $k \approx \sqrt{N/m}$ yields the average cost of m updates as $\widetilde{O}(f(m) + \sqrt{mN})$, which scales sublinearly with the size of the RAM. While the preceding analysis considers the worst case, in specific applications (such as account transactions, where few accounts contribute a large volume of transactions), it may be possible to further delay the computation of table-specific parameters. Thus we have:

Theorem 5.2 (Informal) *Given $m, N \in \mathbb{N}$, there exists an argument for verifiable RAM which proves updates of batch size m on RAM of size N with amortized prover complexity of $\widetilde{O}(f(m) + \sqrt{mN})$.*

Polynomial Protocol for RAM. There are several ways to implement the ordered transcript based memory consistency check on the smaller $O(m)$-sized RAMs, for example by expressing the same as an arithmetic circuit. However, for completeness, we also present an argument for RAM as an interactive *polynomial protocol* [64], which is then compiled into an argument of knowledge using the KZG [77] commitment scheme in the algebraic group model (AGM) [61]. This construction appears in Section 5.6.

As we have alluded to earlier, existing memory-checking based techniques to model RAM computations incur a cost that is linear in the size of the RAM. We are interested in the setting where the number of operations whose execution is to be verified is much smaller than the size of the RAM. Thus, our goal is to achieve prover complexity which is *sublinear* in the size of the RAM. Before we proceed, we establish a working definition of RAM for the rest of the chapter. Informally, a RAM maps indices (addresses) to values, where we assume that values come from a finite field $\mathbb{F}$, while indices come from a subset I of $\mathbb{F}$. For us, I will generally be the set $\{1, \ldots, k\}$ for some integer k (which may be different from size of the RAM n). Finally, for an index, there should be at most one value in the RAM, i.e., the association is unambiguous. The formal definition of RAM is as follows:

Definition 5.1 (RAM) *Given $n \in \mathbb{N}$, finite field $\mathbb{F}$ and a set $\mathcal{I} \subseteq \mathbb{F}$, a RAM of size n over indices $\mathcal{I}$ is a tuple $\boldsymbol{T} = (\boldsymbol{a}, \boldsymbol{v}) \in \mathcal{I}^n \times \mathbb{F}^n$ such that $\forall\, i, j \in [n]$ $v_i = v_j$ whenever $a_i = a_j$. We think of $\boldsymbol{T}$ as a table with vectors $\boldsymbol{a}$ and $\boldsymbol{v}$ denoting its columns. The set of all such tables will be denoted by $\mathsf{RAM}_{\mathcal{I},n}$.*

For a table $\boldsymbol{T} = (\boldsymbol{a}, \boldsymbol{v}) \in \mathsf{RAM}_{\mathcal{I},n}$, we refer to tuples (a_i, v_i), $i \in [n]$ as records of the table $\boldsymbol{T}$. We use the access notation $v = \boldsymbol{T}[a]$ to mean that (a, v) is a record of $\boldsymbol{T}$ (note there can be

multiple such records according to our definition). When we consider RAMs where the first column (of indices) is of the form $\boldsymbol{I}_n = (1, 2, \ldots, n)$, we simply denote such RAMs by $\boldsymbol{T} \in \mathbb{F}^n$. For a RAM $\boldsymbol{T} \in \mathsf{RAM}_{\mathcal{I},n}$, a RAM operation is a three tuple (op, a, v) with $op \in \{0, 1\}$, $a \in I$ and $v \in \mathbb{F}$. An operation with $op = 0$ is called a *load* operation which denotes reading a value v mapped to index a in the RAM. Similarly, an operation with $op = 1$ is called a *store* operation, which denotes associating the value v with index a in the RAM. We use $\mathcal{O}_{\mathcal{I}}$ to denote the set of all RAM operations with index set I.

Component	Protocol	Prover Complexity	Verifier Complexity	Communication Complexity
Committed Sub-vector Lookup	CQ [56]	$O(m \log m)\,\mathbb{F}$ $O(m)\,\mathbb{G}_1$	$5P$	$8\mathbb{G}_1$, $3\mathbb{F}$
Committed Index Lookup	Figure 5.2	$O(m \log m)\,\mathbb{F}$ $O(m)\,\mathbb{G}_1$	$5P$	$8\mathbb{G}_1$, $3\mathbb{F}$
Localized Update in RAM	Figure 5.3	$O(m \log^2 m)\,\mathbb{F}$ $O(m)\,\mathbb{G}_1$	$8P$	$19\mathbb{G}_1$, $1\mathbb{G}_2$, $10\mathbb{F}$
Table Specific Preprocessing	Fast KZG [58]	$O(N \log N)\,\mathbb{F}, \mathbb{G}$	-	-
Lookup from Approximate Setup	Section 5.4	$O((m+\delta) \log^2(m+\delta))\,\mathbb{F}$ $O(m+\delta)\,\mathbb{G}_1$	-	-
Polynomial Protocol for RAM	Figure 5.9	$O(m \log m)\,\mathbb{F}$, $O(m)\,\mathbb{G}$	$7P$	$36\mathbb{G}_1$, $30\mathbb{F}$
Batching-Efficient RAM	Figure 5.4	$\widetilde{O}(\sqrt{mN}), \mathbb{F}, \mathbb{G}$	$9P$	$65\mathbb{G}_1$, $1\mathbb{G}_2$, $43\mathbb{F}$

Table 5.1: Asymptotic efficiency of the component protocols for our scheme. Here, N denotes the size of the RAM, m denotes the number of operations, and δ denotes Hamming distance of table for which pre-computed parameters are available from the current table. As before, we use $(\mathbb{F}, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2, g_t)$ to denote a bilinear group, and P to denote a pairing evaluation. The performance figures reported here correspond to our batching-efficient RAM scheme which uses the lookup argument of CQ [56] as a building block.

5.1.3 Batching-Efficient RAM: Blueprint

We will use vectors in $\mathbb{F}^N$ to denote the "large" RAMs, where index column is implicitly assumed to be $(1, \ldots, N)$. Let $\boldsymbol{T}, \boldsymbol{T'} \in \mathbb{F}^N$ denote the initial and final RAM states, and let $\boldsymbol{o}$ be a sequence of m operations $(m < N)$ which updates $\boldsymbol{T}$ to $\boldsymbol{T}'$. Let $\boldsymbol{a} \in \mathbb{F}^m$ denote the vector of RAM indices referenced by the operations in $\boldsymbol{o}$, i.e, a_i is the index referenced by the i^{th} operation. To prove the transformation of $\boldsymbol{T}$ to $\boldsymbol{T}'$ via operation sequence $\boldsymbol{o}$, we proceed as follows:

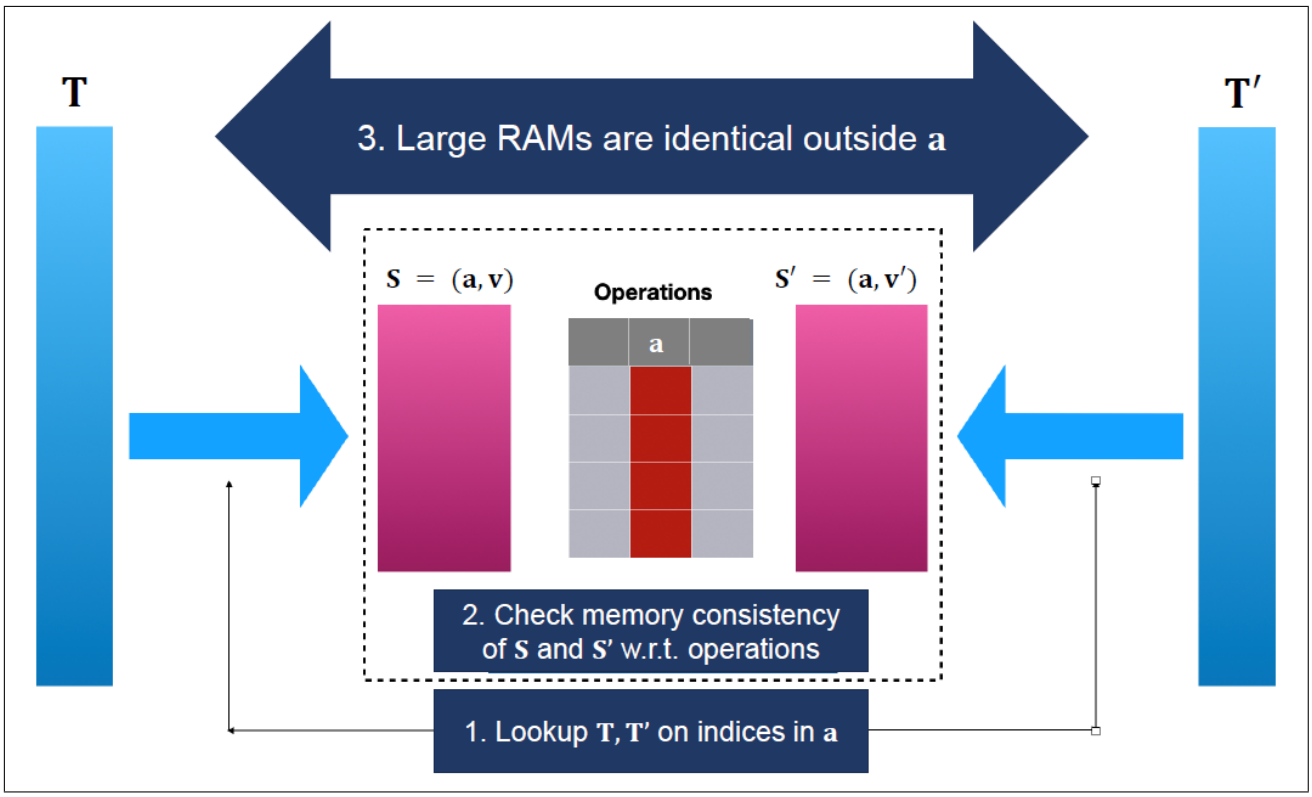

Figure 5.1: Illustrating different steps of sublinear lookup protocol between large RAMs $\boldsymbol{T}$ and $\boldsymbol{T}'$.

- We isolate sub-tables $\mathbf{S} = (\boldsymbol{a}, \boldsymbol{v})$ and $\mathbf{S}' = (\boldsymbol{a}, \boldsymbol{v'})$ of $\boldsymbol{T}$ and $\boldsymbol{T}'$ consisting of rows corresponding to indices in $\boldsymbol{a}$. This requires proving $\boldsymbol{v} = \boldsymbol{T}[\boldsymbol{a}]$ and $\boldsymbol{v'} = \boldsymbol{T}'[\boldsymbol{a}]$, which we show using *committed index lookup* argument discussed in Section 5.3.2.

- On the isolated sub-tables $\mathbf{S}$ and $\mathbf{S}'$ of size m, we use the standard memory checking arguments (c.f. argument presented in Section 5.6) to prove that sequence $\boldsymbol{o}$ correctly updates $\mathbf{S}$ to $\mathbf{S}'$ with prover complexity of $\widetilde{O}(m)$.

- Finally, we show that the RAMs $\boldsymbol{T}$ and $\boldsymbol{T}'$ are identical outside indices in $\boldsymbol{a}$. We describe the protocol for proving the same in Section 5.5.5.

The blueprint for the above approach is illustrated in Figure 5.1.

5.1.4 Batching-Efficient RAM: Components

We now elaborate on the key technical components in realizing the above blueprint.

Committed Index Lookup. To limit the size of the RAM on which we use memory-checking techniques, our first step is to isolate sub-tables of RAMs $\boldsymbol{T}$ and $\boldsymbol{T}'$ corresponding to addresses

which are involved in the operations. This is achieved by looking up RAMs $\boldsymbol{T}$ and $\boldsymbol{T}'$ at indices in the committed vector $\boldsymbol{a}$. We could leverage the recent work on efficient lookup arguments to verifiably extract m indices from a table of size N, in time dependent only on m. However, there are two technical challenges here. First, the aforementioned lookup arguments only prove the sub-vector relation, without linking the extracted vector to the indices in $\boldsymbol{a}$. This is easily solved, as there is an efficient realization of a *committed index lookup* from a *committed sub-vector* argument, where the commitment scheme is homomorphic. The details appear in Section 5.3.2, with the complete protocol presented in Figure 5.2. The second challenge is much more formidable: the efficiency of sub-vector arguments (and the committed index lookup argument derived from them) depends on *expensive* table-specific preprocessing. This is acceptable when the table in question is static, but is infeasible in our setting requiring updatable tables. This motivates our next technical component.

Fast Lookup from Approximate Setup. We build upon the rich body of work on polynomial protocols enabling efficient lookups from static tables [110, 98, 111, 56], which rely on expensive table-dependent pre-computation to optimise online proving performance. We make the first attempt towards breaking this rigid dependence. Our key idea is to extend the utility of pre-computed parameters for a table $\boldsymbol{T}$, to proving lookups from tables $\boldsymbol{T}' \neq \boldsymbol{T}$. We show that for $\delta = \Delta(\boldsymbol{T}, \boldsymbol{T}')$, an argument for m lookups from $\boldsymbol{T}'$ incurs an additional prover overhead of $(m+\delta)\log^2(m+\delta)$ over the lookup argument for static tables. We note that the overhead is *quasi*-linear in both m and δ. Our competitive overhead rests on several innovative applications of algebraic algorithms, which are summarised in Section 5.2.4. We then leverage this ability to use "approximate" setup into a *base + cache* strategy; where at all times we maintain pre-computed parameters corresponding to a base table $\boldsymbol{T}_{\mathsf{b}}$, and use this setup to prove lookups from the current table $\boldsymbol{T}$. We achieve optimal prover effort on average by using parameters for $\boldsymbol{T}_{\mathsf{b}}$ till the current table is at a hamming distance at most $\sqrt{mN}$ from $\boldsymbol{T}_{\mathsf{b}}$, beyond which we recompute full parameters for the current table with $O(N \log N)$ prover effort. The cycle then repeats with current table as the base table.

Naive Approaches are Inadequate. We notice that the aforementioned constructions of lookup arguments require linear combination of encoded quotients of the form $[(T(X) - T(\xi^i))/(X - \xi^i)]_g$ for upto m values of i during the proof generation. While constructions [110, 98] consider quotients encoded in the group $\mathbb{G}_2$, the protocol in [56] encodes them in $\mathbb{G}_1$. We use a generic $[\cdot]_g$ to account for protocol-specific choices. We also see that even a small change to the table requires one to update all the quotients (the polynomial $T(X)$ is common to all quotients). Updating all the quotients after each batch is clearly infeasible. One

could consider delaying the updation of the quotients, till the time they are actually required in a proof, which happens when the corresponding index in the table is involved in lookup. However, each of the m quotients is now potentially "lagging" by δ updates, so we would need $\Omega(m\delta)$ group operations to refresh all of them. This gives us multiplicative degradation with δ, and is clearly unsustainable for reasonable values of δ. In Section 5.4, we present an efficient method to directly compute linear combination of upto $O(m)$ encoded quotients of the form $[(T(X) - T(\xi^i))/(X - \xi^i)]_g$.

Localizing changes in RAMs. While the above two components allow us to reliably extract sub-RAMs corresponding to indices in vector $\boldsymbol{a}$, we still need to prove that RAMs are identical outside indices in $\boldsymbol{a}$. Looking ahead, in terms of polynomials this requires proving that $T(\xi^i) = T^*(\xi^i)$ for $i \notin \{a_i : i \in [m]\}$. Assuming $Z_I(X)$ to be the vanishing polynomial of the set $\{\xi^{a_i} : i \in [m]\}$, this is equivalent to proving that $Z_I(X)(T(X) - T^*(X)) = D(X)Z_{\mathbb{H}}(X)$ for some polynomial D. However, naively this involves working with polynomials with degree $O(N)$, which is expensive. In Section 5.5.5 we show a polynomial protocol for the above relation which requires only $O(m \log^2 m)$ prover effort. The protocol appears in Figure 5.3.

Polynomial Protocol for Memory Checking. To complete the verification, we need to show that the smaller RAMs, $\mathbf{S} = (\boldsymbol{a}, \boldsymbol{v})$ and $\mathbf{S}' = (\boldsymbol{a}, \boldsymbol{v}')$ extracted from larger RAMs $\boldsymbol{T}, \boldsymbol{T}'$ are consistent with respect to the operations. This can be accomplished using standard memory checking techniques based on address ordered transcripts, which we formalize in Section 5.5.1. Later in Section 5.6 and 5.7, we assemble known techniques to present a polynomial protocol for memory consistency based on address ordered transcripts. This involves encoding several artefacts such as operations, transcripts etc., as polynomials and relations among them such as concatenation, permutation and monotonicity as polynomial identities. Our modelling is simple and implementation friendly, and helps in realizing a "circuit-free" overall construction. Complete polynomial protocol for memory checking appears in Figure 5.9, while constituent protocols appear in Figures 5.7, 5.6 and 5.8.

Efficiency. We conclude the overview with a discussion of efficiency achieved by our scheme, and how different components discussed in this section contribute to the overall efficiency. The asymptotic performance of our scheme using CQ [56] is summarized in Table 5.1, with efficiency of the overall scheme highlighted in gray. The table also serves as a ready-reckoner for component protocols involved in the overall scheme. A more detailed discussion and break-up of the polynomial protocol for RAM appears in Table 5.2 in Section 5.6. We note that the verification complexity of the overall solution is substantially less than the aggregate of component protocols; this is due to the fact that several pairing checks required for KZG

verification proofs can be batched together.

Continuity. To support applications such as rollups, we also consider it imperative to ensure that online proof generation does not halt during offline parameter re-generation. In other words, offline parameter re-generation should not hinder the operational continuity of the system. In our scheme, we can ensure this by carefully overlapping the offline computation with online proof generation such that the system can *instantly* switch to using the more recently generated parameters before the online proving time becomes prohibitive.

5.2 Preliminaries

This section presents notations and preliminary background material used in this section.

Notation. Throughout this section, we assume a bilinear group generator BG which on input λ outputs parameters for the protocols. Specifically $\mathsf{BG}(1^\lambda)$ outputs $(\mathbb{F}, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2, g_t)$ where:

- $\mathbb{F} = \mathbb{F}_p$ is a prime field of super-polynomial size in λ, with $p = \lambda^{\omega(1)}$.
- $\mathbb{G}_1, \mathbb{G}_2$ and $\mathbb{G}_T$ are groups of order p, and e is an efficiently computable non-degenerate bilinear pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$.
- Generators g_1, g_2 are uniformly chosen from $\mathbb{G}_1$ and $\mathbb{G}_2$ respectively and $g_t = e(g_1, g_2)$.

We write groups $\mathbb{G}_1$ and $\mathbb{G}_2$ additively, and use the shorthand notation $[x]_1$ and $[x]_2$ to denote group elements $x \cdot g_1$ and $x \cdot g_2$ respectively for $x \in \mathbb{F}$. We implicitly assume that all the setup algorithms for the protocols invoke BG to generate descriptions of groups and fields over which the protocol is instantiated. We use $[n]$ to denote the set of integers $\{1, \ldots, n\}$.

Lagrange Polynomials. We denote the Nth root of unity by ξ and define the subgroup $\mathbb{H}$ as $\mathbb{H} = \{\xi, \ldots, \xi^N\}$. Let $\{\mu_i(X)\}_{i=1}^N$ be the associated Lagrange basis polynomials over the set $\mathbb{H}$; that is, $\mu_i(X) = \prod_{j \neq i} \frac{X - \xi^j}{\xi^i - \xi^j}$. We denote by $Z_\mathbb{H}$ the vanishing polynomial of $\mathbb{H}$; $Z_\mathbb{H}(X) = X^N - 1$.

Formal Derivatives of Polynomials. For a polynomial $f(X) = \sum_{i=0}^d a_i X^i \in \mathbb{F}[X]$, we define its formal derivative to be the polynomial $f'(X) = \sum_{i=1}^d i a_i X^{i-1}$.

5.2.1 Succinct Arguments of Knowledge

Let $\mathcal{R}$ be a NP-relation and Ł be the corresponding NP-language, where Ł $= \{x : \exists\ w$ such that $(x, w) \in \mathcal{R}\}$. A *succinct argument of knowledge* consists of a pair of PPT algorithms $(\mathcal{P}, \mathcal{V})$. Given a public instance x, the prover $\mathcal{P}$, convinces the verifier $\mathcal{V}$, that $x \in$ Ł, where the prover additionally has as a witness w. We use the notation $b \leftarrow_R \langle \mathcal{P}(w), \mathcal{V} \rangle(x)$ to denote $\mathcal{V}$'s

output in the interactive protocol involving $\mathcal{P}$ and $\mathcal{V}$ with w as $\mathcal{P}$'s input and x as the common input. The *knowledge-soundness* property says that if the verifier is convinced, then an efficient extractor algorithm given oracle access to the prover outputs a witness w such that $(x, w) \in \mathcal{R}$. An argument system is *succinct* if the communication complexity and the complexity of $\mathcal{V}$ is polylogarithmic in the size of the witness.

Fiat-Shamir. An interactive protocol is *public-coin* if the verifier's messages are uniformly random strings. Public-coin protocols can be transformed into non-interactive arguments in the Random Oracle Model (ROM) by using the Fiat-Shamir [59] heuristic to derive the verifier's messages as the output of a Random Oracle.

Modular Approach for Succinct Arguments using PIOP. A modular approach for designing efficient succinct arguments consists of two steps; constructing an information theoretic protocol in an idealized model, and then compiling the information-theoretic protocol via a cryptographic compiler to obtain an argument system. Informally, the prover and the verifier interact where the prover provides oracle access to a set of polynomials, and the verifier accepts or rejects by checking certain identities over the polynomials output by the prover and possibly public polynomials known to the verifier. Such a protocol can be compiled into a succinct argument of knowledge by realizing the polynomial oracles using a *polynomial commitment scheme*. A polynomial commitment scheme allows a prover to commit to polynomials, and later verifiably open evaluations at chosen points by giving evaluation proofs. This enables the verifier to probabilistically check polynomial identities at random points of $\mathbb{F}$. Many recent constructions of zkSNARKs [37, 45, 64] follow this approach where the information theoretic object is a polynomial interactive oracle proof (also referred to as PIOP or a polynomial protocol), and the cryptographic primitive in the compiler is a polynomial commitment scheme. Informally, a polynomial interactive oracle proof (also abbreviated as polynomial IOP or PIOP) consists of a prover sending polynomials and the verifier is not required to read the received polynomials, and instead it queries the polynomial at some chosen points to ensure its consistency. We formally define the semantics of a PIOP below (following [67]). Section 2.3 formally introduces a polynomial commitment scheme, and we refer to Section 5.2.2 for the relevant polynomial commitment scheme used throughout this chapter.

Definition 18 (Polynomial Interactive Oracle Proof) *A polynomial IOP is a public-coin interactive proof for a relation $\mathcal{R} = \{(x, w)\}$. $\mathcal{R}$ is an oracle relation which consists of oracles to polynomials over $\mathbb{F}$ with a degree bound d. These oracles can be queried at arbitrary points in $\mathbb{F}$ to evaluate the polynomials at these points. In every round in the protocol, the prover sends polynomial oracles to the verifier. The verifier in every round sends a random challenge. At*

the end of the protocol, the verifier (with oracle access to all the polynomial oracles sent so far) and given its own randomness, outputs accept/reject. A PIOP as an interactive proof system satisfies completeness and knowledge-soundness.

Structured Reference String model. We describe public-coin interactive protocols in the structured reference string (SRS) model where both the parties have access to a SRS. The SRS in our protocols consists of encodings of monomials of the form $\{[x^i]_1\}_{a\leq i\leq b}$, $\{[x^i]_2\}_{c\leq i\leq d}$ for x chosen uniformly from $\mathbb{F}$ and a, b, c, d are bounded by some polynomial in λ. It then follows from [33] that such an SRS can be generated using a universal and updatable setup [75] requiring only one honest participant. In practice, this is a superior security model compared to requiring a fully trusted setup. We use $\mathsf{srs} = (\mathsf{srs}_1, \mathsf{srs}_2)$ to denote the structured reference string of the above form. We say that the srs has degree Q if all the elements of srs_i, $i = 1, 2$ are of the form $[f(x)]_i$ for a polynomial $f \in \mathbb{F}_{<Q}[X]$.

Algebraic Group Model. We analyze security of our protocols in the Algebraic Group Model (AGM) introduced in [61]. An adversary $\mathcal{A}$ is called *algebraic* if every group element output by $\mathcal{A}$ is accompanied by a representation of that group element in terms of all the group elements that $\mathcal{A}$ has seen so far (input and output). In the AGM, an adversary $\mathcal{A}$ is restricted to be *algebraic*, which in our SRS-based protocol means a PPT algorithm satisfying the following: for $i \in \{1, 2\}$, whenever $\mathcal{A}$ outputs an element $A \in \mathbb{G}_i$, it is accompanied by its representation, $\mathcal{A}$ also outputs a vector $\boldsymbol{v}$ over $\mathbb{F}$ such that $A = \langle \boldsymbol{v}, \mathsf{srs}_i \rangle$.

Real and Ideal Pairing Checks. For an algebraic adversary $\mathcal{A}$ interacting in a protocol with a degree Q SRS over a bilinear group, the verifier can use the pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ to perform "ideal check" of the form $(R_1 \cdot T_1) \cdot (R_2 \cdot T_2) \equiv 0$, where R_1, R_2 are vectors of polynomials over $\mathbb{F}$ and T_1, T_2 are public matrices over $\mathbb{F}$. Under the Q-DLOG assumption stated below, the aforementioned ideal check is equivalent (except with a negligible probability) to a real pairing check $(a \cdot T_1) \cdot (T_2 \cdot b) = 0$ with a and b denoting vectors in $\mathbb{F}$ encoding polynomials in R_1 and R_2 in groups $\mathbb{G}_1$ and $\mathbb{G}_2$ respectively (see [64, Lemma 2.2]).

Definition 5.2 (Q-DLOG Assumption [61]) *Fix an integer Q. The Q-DLOG assumption for $(\mathbb{G}_1, \mathbb{G}_2)$ states that given $[1]_1, [x]_1, \ldots, [x^Q]_1$, $[1]_2, [x]_2, \ldots, [x^Q]_2$ for uniformly chosen $x \leftarrow_R \mathbb{F}$, the probability of an efficient $\mathcal{A}$ outputting x is $\mathsf{negl}(\lambda)$.*

5.2.2 KZG Commitment Scheme

The notion of a polynomial commitment scheme (PCS) that allows the prover to open evaluations of the committed polynomial succinctly was introduced in [77] who gave a construction

under the trusted setup assumption. A polynomial commitment scheme over $\mathbb{F}$ is a tuple $\mathsf{PC} = (\mathsf{Setup}, \mathsf{Commit}, \mathsf{open}, \mathsf{eval})$ where:

- $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, D)$. On input security parameter λ, and an upper bound $D \in \mathbb{N}$ on the degree, Setup generates public parameters pp.
- $(C, \tilde{\mathbf{c}}) \leftarrow \mathsf{Commit}(\mathsf{pp}, f(X), d)$. On input the public parameters pp, and a univariate polynomial $f(X) \in \mathbb{F}[X]$ with degree at most $d \leq D$, Commit outputs a commitment to the polynomial C, and additionally an opening hint $\tilde{\mathbf{c}}$.
- $b \leftarrow \mathsf{open}(\mathsf{pp}, f(X), d, C, \tilde{\mathbf{c}})$. On input the public parameters pp, the commitment C and the opening hint $\tilde{\mathbf{c}}$, a polynomial $f(X)$ of degree $d \leq D$, open outputs a bit indicating accept or reject.
- $b \leftarrow \mathsf{eval}(\mathsf{pp}, C, d, x, v; f(X))$. A public-coin interactive protocol $\langle P_{\mathsf{eval}}(f(X)), V_{\mathsf{eval}}\rangle(\mathsf{pp}, C, d, z, v)$ between a PPT prover and a PPT verifier. The parties have as common input public parameters pp, commitment C, degree d, evaluation point x, and claimed evaluation v. The prover has, in addition, the opening $f(X)$ of C, with $\deg(f) \leq d$. At the end of the protocol, the verifier outputs 1 indicating accepting the proof that $f(x) = v$, or outputs 0 indicating rejecting the proof.

A polynomial commitment scheme must satisfy completeness, binding and extractability.

Definition 5.3 (Completeness) *For all polynomials $f(X) \in \mathbb{F}[X]$ of degree $d \leq D$, for all $x \in \mathbb{F}$,*

$$\Pr\left[b = 1 \ : \ \begin{array}{c} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, D) \\ (C, \tilde{\mathbf{c}}) \leftarrow \mathsf{Commit}(\mathsf{pp}, f(X), d) \\ v \leftarrow f(x) \\ b \leftarrow \mathsf{eval}(\mathsf{pp}, C, d, x, v; f(X)) \end{array} \right] = 1.$$

Definition 5.4 (Binding) *A polynomial commitment scheme PC is binding if for all PPT $\mathcal{A}$, the following probability is negligible in λ:*

$$\Pr\left[\begin{array}{c} \mathsf{open}(\mathsf{pp}, f_0, d, C, \tilde{\mathbf{c}}_\mathbf{0}) = 1 \wedge \\ \mathsf{open}(\mathsf{pp}, f_1, d, C, \tilde{\mathbf{c}}_\mathbf{1}) = 1 \wedge \\ f_0 \neq f_1 \end{array} \ : \ \begin{array}{c} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, D) \\ (C, f_0, f_1, \tilde{\mathbf{c}}_\mathbf{0}, \tilde{\mathbf{c}}_\mathbf{1}, d) \leftarrow \mathcal{A}(\mathsf{pp}) \end{array} \right].$$

Definition 5.5 (Knowledge Soundness) *For any PPT adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$*, there exists a PPT algorithm* Ext *such that the following probability is negligible in* λ*:*

$$\Pr\left[\begin{array}{c} b = 1 \wedge \\ \mathcal{R}_{\mathsf{eval}}(\mathsf{pp}, C, x, v; \tilde{f}, \tilde{\mathbf{c}}) = 0 \end{array} : \begin{array}{c} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, D) \\ (C, d, x, v, \mathsf{st}) \leftarrow \mathcal{A}_1(\mathsf{pp}) \\ (\tilde{f}, \tilde{\mathbf{c}}) \leftarrow \mathsf{Ext}^{\mathcal{A}_2}(\mathsf{pp}) \\ b \leftarrow \langle \mathcal{A}_2(\mathsf{st}), V_{\mathsf{eval}} \rangle(\mathsf{pp}, C, d, x, v) \end{array}\right].$$

where the relation $\mathcal{R}_{\mathsf{eval}}$ *is defined as follows:*

$$\mathcal{R}_{\mathsf{eval}} = \{((\mathsf{pp}, C \in \mathbb{G},\ x \in \mathbb{F},\ v \in \mathbb{F});\ (f(X), \tilde{\mathbf{c}})) : (\mathsf{open}(\mathsf{pp}, f, d, C, \tilde{\mathbf{c}}) = 1) \wedge v = f(x)\}$$

We denote by $\mathsf{Prove}, \mathsf{Verify}$, the non-interactive prover and verifier algorithms obtained by applying FS to the eval public-coin interactive protocol, giving a non-interactive PCS scheme $(\mathsf{Setup}, \mathsf{Commit}, \mathsf{Prove}, \mathsf{Verify})$.

Definition 5.6 (Succinctness) *We require the commitments and the evaluation proofs to be of size independent of the degree of the polynomial, that is the scheme is* proof succinct *if* $|C|$ *is* $\mathsf{poly}(\lambda)$*,* $|\pi|$ *is* $\mathsf{poly}(\lambda)$ *where* π *is the transcript obtained by applying FS to* eval*. Additionally, the scheme is* verifier succinct *if* eval *runs in time* $\mathsf{poly}(\lambda) \cdot \mathsf{log}(d)$ *for the verifier.*

In this work, we use the KZG commitment scheme introduced in [77] which satisfies succinctness, completeness and knowledge-soundness (extractability) in the algebraic group model, while additionally featuring a universal and updatable setup. We denote the KZG scheme by the tuple of PPT algorithms $(\mathsf{KZG.Setup}, \mathsf{KZG.Commit}, \mathsf{KZG.Prove}, \mathsf{KZG.Verify})$ as defined below.

Definition 5.7 (KZG Polynomial Commitment Scheme) *Let* $(\mathbb{F}, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2, g_t)$ *be output of bilinear group generator* $\mathsf{BG}(1^\lambda)$ *for security parameter* λ*. The KZG polynomial commitment scheme is defined as follows:*

- $\mathsf{KZG.Setup}$ *on input* $(1^\lambda, d)$*, where* d *is the degree bound, outputs* $\mathsf{srs} = (\{[\tau]_1, \ldots, [\tau^d]_1\}, \{[\tau]_2, \ldots, [\tau^d]_2\})$*.*
- $\mathsf{KZG.Commit}$ *on input* $(\mathsf{srs}, p(X))$*, where* $p(X) \in \mathbb{F}_{\leq d}[X]$*, outputs* $C = [p(\tau)]_1$
- $\mathsf{KZG.Prove}$ *on input* $(\mathsf{srs}, p(X), \alpha)$*, where* $p(X) \in \mathbb{F}_{\leq d}[X]$ *and* $\alpha \in \mathbb{F}$*, outputs* (v, π) *such that* $v = p(\alpha)$ *and* $\pi = [q(\tau)]_1$*, for*

$$q(X) = \frac{p(X) - p(\alpha)}{X - \alpha}$$

– KZG.Verify *on input* $(\mathsf{srs}, C, v, \alpha, \pi)$*, outputs* 1 *if the following equation holds, and* 0 *otherwise.*

$$e(C - v[1]_1 + \alpha\pi, [1]_2) = e(\pi, [\tau]_2)$$

Note that both sides of the verification equation involve a fixed generator, and hence several proof verifications can be batched together to reduce the number of pairing computations. We also assume (w.l.o.g) analogues of KZG.Commit, KZG.Prove and KZG.Verify defined over the group $\mathbb{G}_2$. We shall use the (non-standard) notation $[p(X)]_i$ to denote $[p(\tau)]_i$ for $i \in \{1, 2\}$. This allows us a convenient shorthand for referring to "commitment of the polynomial $p(X)$" in group $\mathbb{G}_i$. Our protocols also use batched KZG proofs to show that polynomial $p(X)$ satisfies $p(\alpha_i) = v_i$ for $i \in [n]$. Let $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_n)$ denote the vector of evaluation points and $\boldsymbol{v} = (v_1, \ldots, v_n)$ denote the vector of claimed evaluations. Then the batched version of KZG.Prove is described as follows:

– KZG.Prove on input $(\mathsf{srs}, p(X), \boldsymbol{\alpha})$, where $p(X) \in \mathbb{F}_{\leq d}[X]$ and $\boldsymbol{\alpha} \in \mathbb{F}^n$, outputs $(\boldsymbol{v}, \pi)$ with $\boldsymbol{v} \in \mathbb{F}^n$ such that $v_i = p(\alpha_i)$ and $\pi = [q(\tau)]_1$ where

$$q(X) = \frac{p(X) - r(X)}{a(X)}$$

In the above equation, $a(X) = (X - \alpha_1) \cdots (X - \alpha_n)$, while $q(X)$ and $r(X)$ are the quotient and remainder polynomials when $p(X)$ is divided by $a(X)$.

– KZG.Verify on input $(\mathsf{srs}, C, \boldsymbol{v}, \boldsymbol{\alpha}, \pi)$, outputs 1 if the following equation is satisfied, and 0 otherwise.

$$e(C - [r(\tau)]_1, [1]_2) = e(\pi, [a(\tau)]_2)$$

Here, the verifier interpolates the polynomial $r(X) \in \mathbb{F}_{<n}[X]$ such that $r(\alpha_i) = v_i$.

KZG for Vectors. For $\boldsymbol{f} \in \mathbb{F}^N$, let $\mathsf{Encode}_{\mathbb{H}}(\boldsymbol{f})$ denote the polynomial encoding of $\boldsymbol{f}$ over $\mathbb{H}$ given by $\sum_{i=1}^{N} f_i \mu_i(X)$. We use KZG to commit to *vectors* by committing to its polynomial encoding. In general a vector $\boldsymbol{g}$ of size m is encoded by a polynomial $g(X) \in \mathbb{F}_{<m}[X]$ which interpolates $\boldsymbol{g}$ over a subgroup $\mathbb{V}$ consisting of m^{th} roots of unity in some canonical order. We will explicitly state the subgroups for all sizes of vectors that we consider.

5.2.3 Lookup Arguments

Prior works on lookup arguments [110, 98, 111, 56] consider proving sub-vector relation over committed vectors, i.e, given commitments c_t and c_v to vectors $\boldsymbol{t} \in \mathbb{F}^N$ and $\boldsymbol{v} \in \mathbb{F}^m$, one proves

that for all $i \in [m]$, there exists $j \in [N]$ such that $v_i = t_j$. We will use $\boldsymbol{v} \preceq \boldsymbol{t}$ to denote that $\boldsymbol{v}$ is a sub-vector of $\boldsymbol{t}$. The definition below summarizes the sub-vector relation as defined in prior works.

Definition 5.8 *We define the* committed sub-vector *relation* $\mathcal{R}^{\mathsf{subvec}}_{\mathsf{srs},N,m}$ *to consist of tuples* $((c_t, c_v), (\boldsymbol{t}, \boldsymbol{v}))$ *where* $c_t, c_v \in \mathbb{G}_1$, $\boldsymbol{t} \in \mathbb{F}^N$, $\boldsymbol{v} \in \mathbb{F}^m$ *such that* $\boldsymbol{v} \preceq \boldsymbol{t}$ *and* $c_t = \mathsf{KZG.Commit}(\mathsf{srs}, \mathsf{Encode}_{\mathbb{H}}(\boldsymbol{t}))$ *and* $c_v = \mathsf{KZG.Commit}(\mathsf{srs}, \mathsf{Encode}_{\mathbb{V}}(\boldsymbol{v}))$.

A committed sub-vector argument is an argument of knowledge for the relation $\mathcal{R}^{\mathsf{subvec}}_{\mathsf{srs},N,m}$. Next, we consider a slightly modified relation that we call *committed index lookup* (called indexed lookup in [103]) where there is a commitment to the indices where $\boldsymbol{v}$ appears in $\boldsymbol{t}$. Formally, we define it as below:

Definition 5.9 *We define the* committed index lookup *relation* $\mathcal{R}^{\mathsf{lookup}}_{\mathsf{srs},N,m}$ *to consist of tuples of the form* $((c_t, c_a, c_v), (\boldsymbol{t}, \boldsymbol{a}, \boldsymbol{v}))$ *where* $c_t, c_a, c_v \in \mathbb{G}_1$, $\boldsymbol{t} \in \mathbb{F}^N$, $\boldsymbol{a}, \boldsymbol{v} \in \mathbb{F}^m$ *such that* $v_i = \boldsymbol{t}[a_i] = t_{a_i}$ *for all* $i \in [m]$ *and* $c_t = \mathsf{KZG.Commit}(\mathsf{srs}, \mathsf{Encode}_{\mathbb{H}}(\boldsymbol{t}))$, $c_a = \mathsf{KZG.Commit}(\mathsf{srs}, \mathsf{Encode}_{\mathbb{V}}(\boldsymbol{a}))$ *and* $c_v = \mathsf{KZG.Commit}(\mathsf{srs}, \mathsf{Encode}_{\mathbb{V}}(\boldsymbol{v}))$.

A committed index lookup argument is a succinct argument of knowledge for the relation $\mathcal{R}^{\mathsf{lookup}}_{\mathsf{srs},N,m}$.

5.2.4 Computational Algebra Preliminaries

Let $\mathbb{F}$ be a finite field of prime order p and $\mathbb{G}$ be a cyclic additive group of order p with generator g. For $s \in \mathbb{F}$, we use the notation $[\,s\,]$ to denote the group element $s \cdot g$. We assume that $\mathbb{F}$ contains the n^{th} root of unity ξ satisfying $\xi^n = 1$ for a large n, and the degrees of all polynomials are less than n.

Fact 5.1 (Fast Evaluation) *Let* $f \in \mathbb{F}[X]$ *be a polynomial of degree* $< d$ *and* $(\xi_1, \ldots, \xi_r) \in \mathbb{F}^r$ *be distinct points in* $\mathbb{F}$. *Then the vector* $(f(\xi_1), \ldots, f(\xi_r))$ *can be computed in* $O((d+r)\log(d+r))$ $\mathbb{F}$ *operations if* $\xi_1, \ldots, \xi_r$ *form roots of unity, and in* $O((d+r)\log^2(d+r))$ $\mathbb{F}$ *operations otherwise.*

Fact 5.2 (Fast Interpolation) *Let* $\xi_1, \ldots, \xi_d$ *be distinct points in* $\mathbb{F}$ *and* $(v_1, \ldots, v_d) \in \mathbb{F}^d$. *Then* $(f_0, \ldots, f_{d-1}) \in \mathbb{F}^d$ *can be computed in* $O(d \log^2 d)$ *operations in* $\mathbb{F}$ *such that* $f(\xi_i) = v_i$ *for all* $i \in [d]$ *where* $f(X) = \sum_{i=0}^{d-1} f_i X^i$.

Fact 5.3 (Fast Multiplication) *Let* $\xi_1, \ldots, \xi_r$ *be distinct points in* $\mathbb{F}$. *Then coefficients of* $f(X) = \prod_{i=1}^{r}(X - \xi_i)$ *can be computed in* $O(r \log^2 r)$ *operations in* $\mathbb{F}$.

Fact 5.4 (Multi KZG proofs [58]) *Let $\{[\,x^i\,]\}_{i=1}^{d}$ be given for some $x \in \mathbb{F}$. Then for set of r distinct points $\xi_1, \ldots, \xi_r$, and a polynomial $f(X) \in \mathbb{F}[X]$ of degree $< d$,the vector $([\,h_1(x)\,], \ldots, [\,h_r(x)\,])$, where $h_i(X) = (f(X) - f(\xi_i))/(X - \xi_i)$ can be computed in $O((r + d)\log(r + d))$ group and field operations when $\xi_1, \ldots, \xi_r$ are roots of unity, and in $O(rlog^2 r + d \log d)$ group and field operations otherwise.*

Fact 5.5 (Lagrange Polynomials) *Let $\mathbb{S} = \{\xi_1, \ldots, \xi_r\}$ be a set of r distinct points and let $\tau_1(X), \ldots, \tau_r(X)$ be the corresponding Lagrange polynomials of degree $r-1$ each. Let $Z_{\mathbb{S}}(X) = \prod_{i=1}^{r}(X - \xi_r)$ denote the vanishing polynomial for $\mathbb{S}$. Then we have:*

$$\sum_{i=1}^{r} \tau_i(X) = 1$$

$$\tau_i(X) = \frac{Z_{\mathbb{S}}(X)}{Z'_{\mathbb{S}}(\xi_i)(X - \xi_i)} \text{ for all } i \in [r]$$

Formal Derivative. For a polynomial $p(X) \in \mathbb{F}[X]$, we define the formal derivative of $p(X)$ as the polynomial $u(X, X)$ where $u(X, Y) = \frac{p(X)-p(Y)}{X-Y}$. It can be seen that $u(X, X)$ is equal to the polynomial $p'(X)$ obtained by differentiating $p(X)$ according to regular rules of calculus. Thus, this definition agrees with the one given earlier in the preliminaries.

Some Useful Results. We now state and prove some facts that are used later throughout the proof.

Lemma 5.1 *For $K \subset [N]$, define H_K to be $\{\xi^i : i \in K\}$. Let $p(X)$ be the vanishing polynomial of H_K. Let $p'(X)$ and $p''(X)$ denote the formal first derivative and second derivative of $p(X)$, respectively. Then, $p''(\xi^i)/p'(\xi^i) = 2 \cdot \sum_{j \in K\setminus\{i\}} 1/(\xi^i - \xi^j)$ for all $i \in K$*

Proof: Observe that $p'(X) = \sum_{i \in K} \prod_{j \in K\setminus\{i\}} (X - \xi^j)$ and $p''(X) = \sum_{i \in K} \sum_{j \in K\setminus\{i\}} \prod_{k \in K\setminus\{i,j\}} (X - \xi^k)$. Thus for $r \in K$, we have:

$$p'(\xi^r) = \prod_{j \in K\setminus\{r\}} (\xi^r - \xi^j),$$

$$p''(\xi^r) = \sum_{j \in K\setminus\{r\}} \prod_{k \in K\setminus\{r,j\}} (\xi^r - \xi^k) + \sum_{i \in K\setminus\{r\}} \prod_{k \in K\setminus\{r,i\}} (\xi^r - \xi^k)$$

Note that only non-zero products in the expansion of $p''(\xi^r)$ occur when $i = r$ or $j = r$, resulting in the two summands for the same in the above equation. Moreover, we notice that both summands are the same, giving us $p''(\xi^r) = 2\sum_{i \in K\setminus\{r\}} \prod_{k \in \{r,i\}} (\xi^r - \xi^k)$. One may now verify that $p'(\xi^r)/p''(\xi^r)$ gives the desired result claimed in the lemma. □

Lemma 5.2 (Sumcheck) *Let $u(X,Y)$ be a bi-variate polynomial over a finite field $\mathbb{F}$ with degree less than N in each of the variables and $\mathbb{H}$ be defined as the group of N^{th} roots of unity ($N << |\mathbb{F}|$) with generator $\xi \in \mathbb{F}$. Then $\sum_{i\in[N]} u(X,\xi^i) = Nu(X,0)$*

Proof: For some $d < N$, we write $u(X,Y) = a_0 + a_1 Y + a_2 Y^2 + \cdots + a_d Y^d$ where each a_i is a polynomial in X of degree less than N. Now we write the sum:

$$\sum_{i\in[N]} u(X,\xi^i) = Na_0 + a_1(\xi + \xi^2 + \cdots + \xi^N) + a_2(\xi^2 + \xi^4 + \cdots + \xi^{2N}) + \cdots + a_d(\xi^d + \cdots + \xi^{Nd})$$

But for any $\alpha = \xi^k$ for $k < N$, $\alpha + \alpha^2 + \cdots \alpha^N = 0$. Thus, all terms vanish except the first term , and hence $\sum_{i\in[N]} u(X,\xi^i) = Na_0$. The lemma follows by observing $a_0 = u(X,0)$. □

We use the following standard observation for our next lemma:

Fact 5.6 *If polynomials f, g of degree $< N$ agree on N points, then they are equal as polynomials, that is, $f(X) = g(X)$*

Lemma 5.3 *Let $Z_{\mathbb{H}}(X)$ be the vanishing polynomial for $\mathbb{H}$, let $\widehat{Z}_K(X)$ and $Z_K(X)$ be the vanishing polynomials for $H_{[N]\setminus K}$ and H_K respectively. Let $\mu_1(X), \ldots, \mu_N(X)$ be Lagrange polynomials for the set $\mathbb{H} = \{\xi, \ldots, \xi^N\}$. Then:*

$$\widehat{Z}_K(X) = \sum_{j\in K} \frac{Z'_{\mathbb{H}}(\xi^j)}{Z'_K(\xi^j)} \mu_j(X), \tag{5.1}$$

$$\widehat{Z}'_K(X) = \sum_{j\in K} \frac{Z'_{\mathbb{H}}(\xi^j)}{Z'_K(\xi^j)} \mu'_j(X) \tag{5.2}$$

Proof: Note that the second equation follows from the first by linearity of derivatives, so it suffices to prove the first equation. Both sides of the identity are polynomials of degree $< N$, so it suffices to show their evaluations are identical over N distinct points. In particular, we show their evaluations are identical over $\mathbb{H}$. Consider evaluating LHS and RHS at ξ^i for $i \in [N] \setminus K$. The left side is 0 by definition of $\hat{Z}_K(X)$, while the right-hand side is zero by the properties of Lagrange polynomials. Now let us consider evaluations LHS and RHS at ξ^i for $i \in K$. The RHS is $\frac{Z'_{\mathbb{H}}(\xi^i)}{Z'_K(\xi^i)}$ by properties of Lagrange polynomials, while the LHS is $\prod_{j\in[N]\setminus K}(\xi^i - \xi^j)$ Multiplying dividing by $\prod_{j\in K\setminus\{i\}}(\xi^i - \xi^j)$ gives:

$$LHS = \frac{\prod_{j\in[N]\setminus\{i\}}(\xi^i - \xi^j)}{\prod_{j\in K\setminus\{i\}}(\xi^i - \xi^j)}$$

Which is $\frac{Z'_{\mathbb{H}}(\xi^i)}{Z'_K(\xi^i)}$, the same as the right hand side. This proves the claim. $\square$

Lemma 5.4 *Let $\mu_1, \ldots, \mu_N$ be the lagrange polynomials for the set $\mathbb{H} = \{\xi^i : i \in [N]\}$ of the N^{th} roots of unity. Then we have:*

$$\mu'_i(\xi^j) = \begin{cases} \frac{(N-1)}{2\xi^i} \text{ if } j = i \\ \frac{\xi^i}{\xi^j(\xi^j - \xi^i)} \text{ otherwise} \end{cases}$$

Proof: Let us first consider the case where $i \neq j$. We know that $\mu_i(X) = \frac{Z_H(X)}{Z'_H(\xi^i)(X-\xi^i)}$. Thus, by applying quotient rule (note that μ_i is defined at ξ^j as $j \neq i$):

$$\mu'_i(X) \cdot Z'_H(\xi^i) = \frac{(X - \xi^i)(N \cdot X^{N-1}) - (X^N - 1)}{(X - \xi^i)^2}$$

Substituting X by ξ^j, we get:

$$\mu'_i(\xi^j) \cdot \frac{N}{\xi^i} = \frac{N(\xi^j - \xi^i)}{\xi^j(\xi^j - \xi^i)^2}$$

Thus, we get:

$$\mu'_i(\xi^j) = \frac{\xi^i}{\xi^j(\xi^j - \xi^i)}.$$

Now, for the second case where $i = j$, we have:

$$\mu_i(X) = \frac{\prod_{j \in [N] \setminus \{i\}}(X - \xi^j)}{Z'_H(\xi^i)}$$
$$\text{or, } \mu_i(X) \cdot Z'_H(\xi^i) = \prod_{j \in [N] \setminus \{i\}} (X - \xi^j)$$

Differentiating the above equation on both sides, we get:

$$\mu'_i(X) \cdot \frac{N}{\xi^i} = \sum_{j \in [N] \setminus \{i\}} \prod_{k \in [N] \setminus \{i,j\}} (X - \xi^k)$$

Substituting $X = \xi^i$ in the above equation yields:

$$\mu'_i(\xi^i) \cdot \frac{N}{\xi^i} = \sum_{j \in [N] \setminus \{i\}} \prod_{k \in [N] \setminus \{i,j\}} (\xi^i - \xi^k)$$

$$
\begin{aligned}
&= \sum_{j\in[N]\setminus\{i\}} \frac{\prod_{k\in[N]\setminus\{i\}}(\xi^i-\xi^k)}{\xi^i-\xi^j} \\
&= \prod_{k\in[N]\setminus\{i\}}(\xi^i-\xi^k) \sum_{j\in[N]\setminus\{i\}} \frac{1}{\xi^i-\xi^j} \\
&= Z'_H(\xi^i) \sum_{j\in[N]\setminus\{i\}} \frac{1}{\xi^i-\xi^j} \\
&= N/\xi^i \sum_{j\in[N]\setminus\{i\}} \frac{1}{\xi^i-\xi^j}
\end{aligned}
$$

We divide on both sides by N/ξ^i in the above, and use Lemma 5.1 to obtain:

$$
\mu'_i(\xi^i) = \sum_{j\in[N]\setminus\{i\}} \frac{1}{\xi^i-\xi^j} = \frac{Z''_H(\xi^i)}{2Z'_H(\xi^i)} = \frac{N-1}{2\xi^i}
$$

□

Lemma 5.5 *Let $K \subseteq \mathbb{N}$ be a set of cardinality k and $\mathcal{X} = \{x_j : j \in K\}$ be a set where x_j for $j \in K$ are distinct elements of $\mathbb{F}$. Let $Z_{\mathcal{X}}(X) = z_k X^k + \cdots + z_0$ denote the vanishing polynomial of $\mathcal{X}$ and $\{\tau_j(X)\}_{j\in K}$ denote the Lagrange polynomials such that $\tau_i(x_j) = \delta_{ij}$ for $i, j \in K$. Then for all $j \in K$, we have $\tau'_j(x_j) = F_K(x_j)/Z'_{\mathcal{X}}(x_j)$ where the polynomial $F_K(X)$ is defined as*

$$
F_K(X) = \binom{k}{2} z_k X^{k-2} + \cdots + \binom{2}{2} z_2 = \sum_{j=2}^{k} z_j \binom{j}{2} X^{j-2}
$$

Proof: For $j \in K$, by definition of Lagrange polynomials, we have:

$$
\tau_j(X) = \frac{Z_{\mathcal{X}}(X)}{(X-x_j)Z'_{\mathcal{X}}(x_j)} = \frac{1}{Z'_{\mathcal{X}}(x_j)} \frac{Z_{\mathcal{X}}(X)}{X-x_j}
$$

By long division of $Z_{\mathcal{X}}(X)$ by $(X - x_j)$, we have:

$$
\begin{aligned}
\tau_j(X) &= \frac{1}{Z'_{\mathcal{X}}(x_j)} \left(z_k X^{k-1} + (x_j z_k + z_{k-1}) X^{k-2} + \cdots + (x_j^{k-1} z_k + \cdots + z_1) \right) \\
&= \frac{1}{Z'_{\mathcal{X}}(x_j)} \sum_{p=0}^{k-1} \left(\sum_{q=p+1}^{k} z_q x_j^{q-p-1} \right) X^p
\end{aligned}
$$

Differentiating both sides, we have:

$$\begin{aligned}\tau_j'(X) &= \frac{1}{Z_{\mathfrak{X}}'(x_j)} \sum_{p=0}^{k-1} \left(\sum_{q=p+1}^{k} z_q x_j^{q-p-1} \right) p X^{p-1} \\ &= \frac{1}{Z_{\mathfrak{X}}'(x_j)} \sum_{p=1}^{k-1} p \sum_{q=p+1}^{k} z_q x_j^{q-p-1} X^{p-1}\end{aligned}$$

Substituting $X = x_j$, we get:

$$\begin{aligned}\tau_j'(x_j) &= \frac{1}{Z_{\mathfrak{X}}'(x_j)} \sum_{p=1}^{k-1} p \sum_{q=p+1}^{k} z_q x_j^{q-2} \\ &= \frac{1}{Z_{\mathfrak{X}}'(x_j)} \sum_{q=2}^{k} z_q x_j^{q-2} \sum_{p=1}^{q-1} p \\ &= \frac{1}{Z_{\mathfrak{X}}'(x_j)} \sum_{q=2}^{k} z_q \binom{q}{2} x_j^{q-2} \\ &= \frac{F_K(x_j)}{Z_{\mathfrak{X}}'(x_j)}\end{aligned}$$

This completes the proof. □

5.3 Committed Index Lookup Arguments

In this section, we explore how to obtain committed index lookup arguments, where we first start with a discussion on extending the lookup arguments of Caulk+ [98] to support committed index lookup arguments, and thereafter we discuss a generic blackbox method to "lift" any lookup argument to a committed index lookup argument.

Let $m, N \in \mathbb{N}$ be fixed parameters with $m < N$ and let srs denote a KZG setup of degree $d \geq N$ over bilinear group $(\mathbb{F}, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, [1]_1, [1]_2, [1]_t)$. Recall that the committed index lookup relation in Definition 5.9 involves the prover showing knowledge of vectors $\boldsymbol{T} \in \mathbb{F}^N$, $\boldsymbol{a} \in \mathbb{F}^m$ and $\boldsymbol{v} \in \mathbb{F}^m$ corresponding to public commitments c_T, c_a and c_v such that they satisfy $v_i = \boldsymbol{T}[\,a_i\,] = T_{a_i}$.

5.3.1 Committed Index Lookup from Caulk+

In this section, we present an explicit (non-black-box) adaptation of Caulk+ [98] to obtain a committed index lookup, which again incurs costs comparable to a single instance of the underlying sub-vector protocol. We present a polynomial protocol for the same, which is an

adaptation of the lookup protocol from Caulk+ [98]. However, here we do not aim for zero-knowledge. Let $T(X) = \mathsf{Encode}_{\mathbb{H}}(\boldsymbol{t})$, $a(X) = \mathsf{Encode}_{\mathbb{V}}(\boldsymbol{a})$ and $v(X) = \mathsf{Encode}_{\mathbb{V}}(\boldsymbol{v})$ denote the polynomials encoding the vectors $\boldsymbol{t}, \boldsymbol{a}$ and $\boldsymbol{v}$ respectively. The verifier knows commitments to these polynomials at the start of the protocol. Now $v_i = \boldsymbol{t}[a_i]$ for $i \in [m]$ is equivalent to $v(\nu^i) = T(\xi^{a(\nu^i)})$ for $i \in [m]$. To obtain a polynomial protocol, the prover interpolates a polynomial $h(X) = \sum_{i=1}^{m} \xi^{a_i} \tau_i(X)$, which satisfies $h(\nu^i) = \xi^{a(\nu^i)}$. To show that polynomial h correctly "exponentiates" evaluations of $a(X)$, we consider the inverting polynomial $\ell(X) = \sum_{i=1}^{N} i\mu_i(X)$ which behaves like "log" over $\mathbb{H}$ by evaluating to i on ξ^i. Now, we see that all constraints are encoded as polynomial identities below:

$$\begin{aligned} \ell(h(X)) &= a(X) \mod Z_{\mathbb{V}} \\ T(h(X)) &= v(X) \mod Z_{\mathbb{V}} \\ Z_{\mathbb{H}}(h(X)) &= 0 \mod Z_{\mathbb{V}} \end{aligned} \tag{5.3}$$

The last polynomial identity ensures that evaluations of h on $\mathbb{V}$ lie in $\mathbb{H}$ (the set of roots of $Z_{\mathbb{H}}$). Since the polynomial ℓ is one-one over $\mathbb{H}$, the first equation implies $h(\nu^i) = \xi^{a_i}$ for all $i \in [m]$. The desired relation $v_i = T_{a_i}$ now follows from the second identity. The above formulation involves composition with polynomials ℓ, T and $Z_{\mathbb{H}}$ of degree $O(N)$, which is inefficient. We use the trick from [98], where we work with low-degree restrictions of $O(N)$-degree polynomials such as T, ℓ over the set $\mathbb{H}_I = \{h(\nu^i) : i \in [m]\} = \{\xi^{a_i} : i \in I\} \subseteq \mathbb{H}$, where $I = \{a_i : i \in [m]\}$. The prover commits to the polynomials $Z_I(X) = \prod_{i \in I}(X - \xi^i)$, $h(X)$ and low degree ($< m$) restrictions T_I, ℓ_I of T and ℓ on the $\mathbb{H}_I$ respectively. The polynomial protocol then checks the following:

$$\begin{aligned} T(X) - T_I(X) &= 0 \mod Z_I, \quad T_I(h(X)) &&= v(X) \mod Z_{\mathbb{V}} \\ \ell(X) - \ell_I(X) &= 0 \mod Z_I, \quad \ell_I(h(X)) &&= a(X) \mod Z_{\mathbb{V}} \\ Z_{\mathbb{H}}(X) &= 0 \mod Z_I, \quad Z_I(h(X)) &&= 0 \mod Z_{\mathbb{V}} \end{aligned} \tag{5.4}$$

It must be noted that the above identities imply the earlier polynomial identities in (5.3). This is so because evaluations of h on $\mathbb{V}$ are roots of Z_I, which implies $T_I(h(\nu^i)) = T(h(\nu^i))$, $\ell_I(h(\nu^i)) = \ell(h(\nu^i))$ and $Z_{\mathbb{H}}(h(\nu^i)) = 0$ over $\mathbb{V}$. While the identities on the left still involve a degree N polynomial, we can use the srs to check the polynomial identity at the point τ encoded in the srs. For example, we can evaluate the encoded quotient $[Q(X)]_2 = \left[\frac{(T(X)-T_I(X)}{Z_I(X)}\right]_2$ using the relation:

$$\left[\frac{T(X) - T_I(X)}{Z_I(X)}\right]_2 = \sum_{i \in I} \frac{1}{Z_I'(\xi^i)} \left[\frac{T(X) - t_i}{X - \xi^i}\right]_2$$

By pre-computing the KZG proofs $W_1^i = \left[\frac{T(X)-t_i}{X-\xi^i}\right]_2$ for all $i \in [N]$, the encoded quotient can be evaluated using $O(m)$ $\mathbb{G}_2$-operations and $O(m \log^2 m)$ $\mathbb{F}$-operations. The identity is then checked using a real pairing check

$$e([T(X)]_1 - [T_I(X)]_1, [1]_2) = e([Z_I(X)]_1, [Q(X)]_2).$$

Similarly, we also pre-compute the encoded quotients $W_2^i = \left[\frac{\ell(X)-i}{X-\xi^i}\right]_2$ and $W_3^i = \left[\frac{Z_{\mathbb{H}}(X)}{X-\xi^i}\right]_2$ for all $i \in [N]$. The quotients can be computed in time $O(N \log N)$ using the techniques in [58]. Using KZG commitment scheme the polynomial relations over $Z_{\mathbb{V}}$ can be checked in a standard manner by having the prover send evaluation proofs for the committed polynomials at a random point chosen by the verifier. The total prover effort incurred is $O(m^2)$ group and field operations. Thus, we have:

Lemma 5.6 *Assuming* KZG *is extractable polynomial commitment scheme, there exists a succinct argument of knowledge for the relation* $\mathcal{R}^{\mathsf{lookup}}_{\mathsf{srs},N,m}$ *with prover complexity of* $O(m^2)$*, given access to pre-computed parameters of size* $O(N)$*.*

5.3.2 Blackbox Committed Index Lookup Arguments from Lookup Arguments

In this section, we "lift" any committed sub-vector argument to a committed index lookup argument, where the latter makes a black-box use of the former. We use the trick of random linear combination of vectors to infer indexed lookup relation among them from sub-vector relation over the aggregated vectors.

Lemma 5.7 *Let* $\boldsymbol{t} \in \mathbb{F}^n$ *and let* $\boldsymbol{a}, \boldsymbol{v} \in \mathbb{F}^m$ *for some positive integers* m, n*. Let* $\boldsymbol{I}_n$ *denote the vector* $(1, \ldots, n)$*. Then for* $\gamma \leftarrow_R \mathbb{F}$*,* $(\boldsymbol{v} + \gamma \boldsymbol{a}) \preceq (\boldsymbol{t} + \gamma \boldsymbol{I}_n)$ *implies* $\boldsymbol{v} = \boldsymbol{t}[\,\boldsymbol{a}\,]$ *except with probability* $mn/|\mathbb{F}|$*.*

Proof: We define vectors of linear polynomials $\boldsymbol{p} = (p_1, \ldots, p_m)$ and $\boldsymbol{q} = (q_1, \ldots, q_n)$ where $p_i(X) = v_i + a_i X$, $i \in [m]$ and $q_i(X) = t_i + iX$, $i \in [n]$. Now, we see that $\boldsymbol{v} = \boldsymbol{t}[\,\boldsymbol{a}\,]$ if and only if $\boldsymbol{p} \preceq \boldsymbol{q}$. For $\gamma \in F$, let $\boldsymbol{p}_\gamma$ and $\boldsymbol{q}_\gamma$ denote the vectors $(p_1(\gamma), \ldots, p_m(\gamma))$ and $(q_1(\gamma), \ldots, q_n(\gamma))$ respectively. It is obvious that $\boldsymbol{p} \preceq \boldsymbol{q}$ implies $\boldsymbol{p}_\gamma \preceq \boldsymbol{q}_\gamma$ for all $\gamma \in \mathbb{F}$. Using Schwartz-Zippel Lemma, it can also be seen that $\Pr_{\gamma \leftarrow_R \mathbb{F}}[\boldsymbol{p} \not\preceq \boldsymbol{q} \,|\, \boldsymbol{p}_\gamma \preceq \boldsymbol{q}_\gamma] \leq mn/|\mathbb{F}|$. The bound follows from the observation that the event occurs only when γ is a common root of at least one pair of linear polynomials $\{(p_i(X), q_j(X)) : i \in [m], j \in [n]\}$. □

In Figure 5.2, we invoke Lemma 5.7 to construct a committed index lookup argument using a

Common Input: $\mathsf{srs}, c_t, c_a, c_v, c_I = [I(X)]_1$ where $I(X) = \mathsf{Encode}_{\mathbb{H}}(\boldsymbol{I})$ encodes the vector $\boldsymbol{I} = (1, \ldots, N) \in \mathbb{F}^N$.
Prover's Input: Vectors $\boldsymbol{t} \in \mathbb{F}^N$, $\boldsymbol{a}, \boldsymbol{v} \in \mathbb{F}^m$.

1. $\mathcal{V}$ samples $\gamma \leftarrow_R \mathbb{F}$ and sends γ to $\mathcal{P}$.
2. $\mathcal{P}$ and $\mathcal{V}$ compute: $\tilde{c}_t = \gamma c_I + c_t$, $\tilde{c}_v = \gamma c_a + c_v$.
3. $\mathcal{P}$ computes: $\tilde{\boldsymbol{t}} = \gamma \boldsymbol{I} + \boldsymbol{t}$, $\tilde{\boldsymbol{v}} = \gamma \boldsymbol{a} + \boldsymbol{v}$.
4. $\mathcal{P}$ and $\mathcal{V}$ run sub-vector argument $(\mathcal{P}_{\mathsf{sv}}, \mathcal{V}_{\mathsf{sv}})$ with $(\mathsf{srs}, \tilde{c}_t, \tilde{c}_v)$ as the common input and $(\tilde{\boldsymbol{t}}, \tilde{\boldsymbol{v}})$ as $\mathcal{P}_{\mathsf{sv}}$'s input.
5. $\mathcal{V}$ outputs $b \leftarrow_R \langle \mathcal{P}_{\mathsf{sv}}(\tilde{\boldsymbol{t}}, \tilde{\boldsymbol{v}}), \mathcal{V}_{\mathsf{sv}} \rangle (\mathsf{srs}, \tilde{c}_t, \tilde{c}_v)$.

Figure 5.2: Committed Index Lookup Argument

committed sub-vector argument $(\mathcal{P}_{\mathsf{sv}}, \mathcal{V}_{\mathsf{sv}})$. We formally state the following lemma, whose proof essentially follows from Lemma 5.7.

Lemma 5.8 *Assuming that $(\mathcal{P}_{\mathsf{sv}}, \mathcal{V}_{\mathsf{sv}})$ is an argument of knowledge for the relation $\mathcal{R}^{\mathsf{subvec}}_{\mathsf{srs},N,m}$ in the AGM, the interactive protocol in Figure 5.2 is an argument of knowledge for the relation $\mathcal{R}^{\mathsf{lookup}}_{\mathsf{srs},N,m}$ in the AGM.*

5.4 Updatable Lookup Arguments: Fast Lookups from Approximate Preprocessing

In this section, we provide details of the algorithm to construct lookup argument for a table $\boldsymbol{T}$, using pre-computed parameters of a table which is a small hamming distance away. The dependence on pre-computed parameters in several recent lookup arguments such as [110, 98, 111, 56] stems from the need to compute an encoded quotient of the form:

$$[Q]_g = \sum_{i \in I} c_i \left[\frac{T(X) - T(\xi^i)}{X - \xi^i} \right]_g \tag{5.5}$$

for some $O(m)$ sized set I. The quotient in Equation (5.5) can be computed in $O(m)$ cost when the quotients $[(T(X) - T(\xi^i))/(X - \xi^i)]_g$ are available for all $i \in [N]$. In this section we exhibit an algorithm which computes the above with $O((m+\delta)\log^2(m+\delta))$ cost, given access to similar quotients for a table at hamming distance δ from $\boldsymbol{T}$. We now describe our approach.

5.4.1 Base + Cache approach

The key idea we employ is to express the current table $\boldsymbol{T} \in \mathbb{F}^N$ as $\boldsymbol{T}_{\mathsf{b}} + \boldsymbol{T}_{\mathsf{ch}}$, where $\boldsymbol{T}_{\mathsf{b}}$ is the table for which we assume that the encoded quotients are available (via the $O(N \log N)$ computation), and $\boldsymbol{T}_{\mathsf{ch}}$ captures the changes to the table since. We will periodically update (say after s batch updates) $\boldsymbol{T}_{\mathsf{b}}$ to current table state, and re-compute all the quotients (we call it the *offline* phase). We will revisit the question on choosing s optimally later. Let $I \subseteq [N]$ denote the set of indices in the current batch of m lookups. The *online* phase of our proof generation involves computing the sum in Equation (5.5) for the table $\boldsymbol{T}$. The following Theorem determines the efficiency of the online phase of our prover.

Theorem 5.3 *Let N, ξ be as defined previously. Given* KZG *proofs $\{W_i\}_{i=1}^N$ with*

$$W_i = \left[T_{\mathsf{b}}(X) - T_{\mathsf{b}}(\xi^i)/(X - \xi^i)\right]_g,$$

where $T_{\mathsf{b}}(X) = \mathsf{Encode}_{\mathbb{H}}(\boldsymbol{T}_{\mathsf{b}})$ encodes a vector $\boldsymbol{T}_{\mathsf{b}} \in \mathbb{F}^N$, for any $I \subseteq [N]$, there exists an algorithm to compute $[Q]_g$ as given in Equation (5.5) for polynomial $T(X) = \mathsf{Encode}_{\mathbb{H}}(\boldsymbol{T})$ encoding the vector $\boldsymbol{T} \in \mathbb{F}^N$ using $O((\delta + |I|)\log^2(\delta + |I|))$ $\mathbb{F}$-operations and $O(\delta + |I|)$ $\mathbb{G}$-operations. Here, δ denotes the hamming distance between vectors $\boldsymbol{T}_{\mathsf{b}}$ and $\boldsymbol{T}$.

Proof: Let $\boldsymbol{T} = \boldsymbol{T}_{\mathsf{b}} + \boldsymbol{T}_{\mathsf{ch}}$ and thus $T(X) = T_{\mathsf{b}}(X) + T_{\mathsf{ch}}(X)$. Define $K = I \cup \{j \in [N] : \boldsymbol{T}_{\mathsf{ch}}[\,j\,] \neq 0\}$ as a set which captures the indices where the current table $\boldsymbol{T}$ differs from the base $\boldsymbol{T}_{\mathsf{b}}$, where we explicitly also include the lookup indices I in K. For $j \in K$, let $\boldsymbol{T}_{\mathsf{ch}}[j] = \Delta t_j$. Then $T_{\mathsf{ch}}(X) = \sum_{j \in K} \Delta t_j \mu_j(X)$. We write the quotient $Q(X)$ as:

$$Q(X) = \sum_{i \in I} c_i \left(\frac{T_{\mathsf{b}}(X) - T_{\mathsf{b}}(\xi^i)}{X - \xi^i}\right) + \sum_{i \in I} c_i \left(\frac{T_{\mathsf{ch}}(X) - T_{\mathsf{ch}}(\xi^i)}{X - \xi^i}\right)$$

From above, we have $[Q(x)]_g = [Q_{\mathsf{b}}(x)]_g + [Q_{\mathsf{ch}}(x)]_g$ where

$$Q_{\mathsf{b}}(X) = \sum_{i \in I} c_i (T_{\mathsf{b}}(X) - T_{\mathsf{b}}(\xi^i))/(X - \xi^i)$$
$$Q_{\mathsf{ch}}(X) = \sum_{i \in I} c_i (T_{\mathsf{ch}}(X) - T_{\mathsf{ch}}(\xi^i))/(X - \xi^i)$$

We can compute $[Q_{\mathsf{b}}(X)]_g$ from the pre-computed KZG openings of $T_{\mathsf{b}}(X)$ at points $\xi^i, i \in I$ using $O(|I|)$ group operations and $O(|I| \log^2 |I|)$ field operations. Therefore, it suffices to compute $[Q_{\mathsf{ch}}(X)]_g$ efficiently. Using $T_{\mathsf{ch}}(X) = \sum_{j \in K} \Delta t_j \mu_j(X)$ we write $Q_{\mathsf{ch}}(X)$ as linear

combination of table-independent polynomials:

$$\begin{aligned} Q_{\mathsf{ch}}(X) &= \sum_{i\in I} c_i \sum_{j\in K} \Delta t_j \frac{\mu_j(X) - \mu_j(\xi^i)}{X - \xi^i} \\ &= \sum_{i\in I} c_i \Delta t_i \frac{\mu_i(X) - 1}{X - \xi^i} + \sum_{i\in I} \sum_{j\in K\setminus\{i\}} c_i \Delta t_j \frac{\mu_j(X)}{X - \xi^i} \end{aligned}$$

Now, we can write $[Q_{\mathsf{ch}}(X)]_g = [Q^{(1)}_{\mathsf{ch}}(X)]_g + [Q^{(2)}_{\mathsf{ch}}(X)]_g$ where:

$$Q^{(1)}_{\mathsf{ch}}(X) = \sum_{i\in I} c_i \Delta t_i \frac{\mu_i(X) - 1}{X - \xi^i},\ Q^{(2)}_{\mathsf{ch}}(X) = \sum_{i\in I} \sum_{j\in K\setminus\{i\}} c_i \Delta t_j \frac{\mu_j(X)}{X - \xi^i}$$

The term $\left[Q^{(1)}_{\mathsf{ch}}(X)\right]_g$ can be computed using $O(|I|)$ group operations by augmenting the setup with pre-computed KZG opening proofs of polynomials $\mu_i(X)$ at ξ^i for $i \in [N]$. This adds $O(N)$ to the setup parameters, while the computation can be done in $O(N \log N)$ time with methods similar to existing pre-computed parameters. This eventually leaves us with $[Q^{(2)}_{\mathsf{ch}}(X)]_g$. Next, we synthesize the polynomial $Q^{(2)}_{\mathsf{ch}}(X)$ in a form that reduces group operations required to compute its encoding.

$$\begin{aligned} Q^{(2)}_{\mathsf{ch}}(X) &= \sum_{i\in I} c_i \sum_{j\in K\setminus\{i\}} \Delta t_j \mu_j(X)/(X - \xi^i) \\ &= \sum_{i\in I} c_i \sum_{j\in K\setminus\{i\}} \frac{\Delta t_j}{Z'_{\mathbb{H}}(\xi^j)} \frac{Z_{\mathbb{H}}(X)}{(X - \xi^i)(X - \xi^j)} \\ &= N^{-1} \sum_{i\in I} c_i \sum_{j\in K\setminus\{i\}} \frac{\xi^j \Delta t_j}{\xi^i - \xi^j} \left(\frac{Z_{\mathbb{H}}(X)}{X - \xi^i} - \frac{Z_{\mathbb{H}}(X)}{X - \xi^j} \right) \\ &= N^{-1} \sum_{i\in I} \left(c_i \cdot \sum_{j\in K\setminus\{i\}} \frac{\xi^j \Delta t_j}{\xi^i - \xi^j} \right) \frac{Z_{\mathbb{H}}(X)}{X - \xi^i} \\ &\quad + N^{-1} \sum_{j\in K} \left(\xi^j \Delta t_j \cdot \sum_{i\in I\setminus\{j\}} \frac{c_i}{\xi^j - \xi^i} \right) \frac{Z_{\mathbb{H}}(X)}{X - \xi^j} \end{aligned} \tag{5.6}$$

In the first step, we substituted $\mu_j(X)$, while in the final step we re-arranged the summation to accumulate the scalar factor for each distinct polynomial of the form $Z_{\mathbb{H}}(X)/(X - \xi^i)$. Define

scalars a_i, $i \in I$ and b_j, $j \in K$ as below:

$$a_i = \sum_{j \in K \setminus \{i\}} \frac{\xi^j \Delta t_j}{\xi^i - \xi^j}, i \in I \quad b_j = \sum_{i \in I \setminus \{j\}} \frac{c_i}{\xi^j - \xi^i}, j \in K \tag{5.7}$$

Now, define $W_3^j := [Z_{\mathbb{H}}(X)/(X - \xi^j)]_g$. We see that W_3^j is just the KZG opening proof of the polynomial $Z_{\mathbb{H}}(X)$ evaluated at ξ^j for $j \in [N]$. These can be precomputed one time and it adds $O(N)$ to the setup parameters and the computation can be done in $O(N \log N)$ time.
Now, we see that $[Q_{\mathsf{ch}}^{(2)}(X)]_g$ can be written as linear combination of $O(|K|+|I|)$ group elements.

$$\left[Q_{\mathsf{ch}}^{(2)}(X)\right]_g = N^{-1} \left(\sum_{i \in I} (c_i a_i) \cdot W_3^i + \sum_{j \in K} (\xi^j \Delta t_j b_j) \cdot W_3^j \right) \tag{5.8}$$

Now, c_i are known constants depending on the specific lookup scheme. So, given $\{a_i\}_{i \in I}, \{b_j\}_{j \in K}$, $\left[Q_{\mathsf{ch}}^{(2)}(X)\right]_g$ can be computed in $O(|I|+|K|)$ group operations. While we have diligently reduced the group operations, we still seem to need $O(|I||K|) = O(m\delta)$ field operations. We clearly need better than naive way of computing the scalars in (5.7) to obtain additive overhead in δ. This is what we consider next. Let $d_j := \xi^j \Delta t_j$. Then we have from Eq (5.7):

$$a_i = \sum_{j \in K \setminus \{i\}} \frac{d_j}{\xi^i - \xi^j}, i \in I \quad b_j = \sum_{i \in I \setminus \{j\}} \frac{c_i}{\xi^j - \xi^i}, j \in K \tag{5.9}$$

So, to compute a_i and b_j, it suffices to compute *reciprocal sums* efficiently. Our next lemma claims that such reciprocal sums can be computed efficiently. Using our next lemma (Lemma 5.9), we conclude that the scalars $a_i, i \in I$ and $b_j, j \in K$ can be computed in time $O(|K| \log^2 |K|)$, which proves the bound in Theorem 5.3. □

Lemma 5.9 *Let $I \subset K \subset [N]$ and let a_i for all $i \in I$ and b_j for all $j \in K$ be as described above. Then, a_i for all $i \in I$ and b_j for all $j \in K$ can be computed in $O(|K| \log^2 |K|)\,\mathbb{F}$ operations.*

Proof sketch. We sketch the proof here for a_i. First, we mention that the special case of the lemma when $d_j = 1$ for all $j \in K$ admits an efficient computation due to the following identity proved in Lemma 5.1.

$$\frac{Z_K''(\xi^i)}{Z_K'(\xi^i)} = 2 \sum_{j \in K \setminus \{i\}} \frac{1}{\xi^i - \xi^j}$$

for $Z_K(X) = \prod_{i \in K}(X - \xi^i)$. The polynomial Z_K can be computed in $O(|K| \log^2 |K|)$ and subsequent evaluations of its first two derivatives can also be evaluated on the set $\{\xi^i : i \in I\}$

with the same complexity. However, to deal with arbitrary values of d_j we need more ingenuity. We will *imagine* d_j to be $p(\xi^j)$ for some polynomial $p(X)$. Moreover, we demand that $p(\xi^j) = 0$ for $j \notin K$. We will not compute such a polynomial p, as it has degree $O(N)$, but view it as an "oracle" which we can hopefully query at the points we need. Then it can be seen that $a_i = g_i(\xi^i) - r_i(\xi^i)$ for rational functions $g_i(X)$ and $r_i(X)$ defined by:

$$g_i(X) = \sum_{j \in [N] \setminus i} \frac{p(X)}{X - \xi^j}, \quad r_i(X) = \sum_{j \in [N] \setminus i} \frac{p(X) - p(\xi^j)}{X - \xi^j} \tag{5.10}$$

Now, $g_i(\xi^i)$ for $i \in I$ turns out to be (using the special case above):

$$p(\xi^i) \sum_{j \in K \setminus \{i\}} 1/(\xi^i - \xi^j) = d_i(Z''_K(\xi^i)/Z'_K(\xi^i))/2$$

Defining $u(X, Y) = (p(X) - p(Y))/(X - Y)$, we can write $r_i(\xi^i)$ as:

$$r_i(X) = \sum_{j \in [N]} u(X, \xi^j) - u(X, \xi^i) \tag{5.11}$$

Observe that $u(X, X) = p'(X)$ and so $u(X, X)$ gives the formal derivative of polynomial $p(X)$. We get $r_i(\xi^i) = r(\xi^i) - p'(\xi^i)$ for all $i \in I$, where $r(X) = \sum_{j \in [N]} u(X, \xi^j)$. Fortunately, $r(X)$ is simply $Nu(X, 0) = N(p(X) - p(0))/X$, a fact that follows from uni-variate sum-check. The problem thus reduces to being able to compute derivatives $p'(\xi^i)$ for $i \in I$ and the value $p(0)$. Before concluding the proof-sketch, we briefly highlight the structure of the polynomial $p(X)$. Since $p(X)$ vanishes for $p(\xi^i)$ for $i \notin K$, it can we written as the product $\widehat{Z}_K(X)q(X)$ where $\widehat{Z}_K$ is the vanishing polynomial of "complementary" roots of unity $\{\xi^i : i \notin K\}$ and q is a low-degree ($< K$) polynomial. Assuming we can interpolate $q(X)$, we can write:

$$p'(\xi^i) = \widehat{Z}_K(\xi^i)q'(\xi^i) + \widehat{Z}'_K(\xi^i)q(\xi^i)$$

In the above expression, we require evaluations of high-degree polynomials $\widehat{Z}_K(X)$ and $\widehat{Z}'_K(X)$ at ξ^i, $i \in I$. This is discussed in Lemma 5.3 and other related lemmas in Section 5.2.4, and motivates the at times tedious algebra there. We will now discuss the detailed proof.

Proof: Now, we present the detailed proof for the computation of a_i for all $i \in I$. Thereafter, we briefly discuss the modifications needed to compute b_j for all $j \in K$.

Computing a_i: Recall that for each $i \in I$, we have:

$$a_i = \sum_{j \in K \setminus \{i\}} \frac{d_j}{\xi^i - \xi^j} \tag{5.12}$$

Also recall that $I \subset K$ in this case. To compute a_i, we first define a polynomial $p(X)$ of degree at most $N-1$ such that $p(\xi^j) = d_j$ for $j \in K$ and $p(\xi^j) = 0$ for $j \in [N] \setminus K$. Then, the vanishing polynomial of $H_{[N]\setminus K}$ divides $p(X)$ and there exists a polynomial $q(X)$ of degree at most $|K| - 1$ such that:

$$p(X) = \hat{Z}_K(X) \cdot q(X) \tag{5.13}$$

where $\hat{Z}_K(X) = \prod_{i \in [N]\setminus K}(X - \xi^i)$ is the vanishing polynomial of $H_{[N]\setminus K}$. Now, we introduce the rational functions:

$$f_i(X) = \sum_{j \in [N]\setminus\{i\}} \frac{p(\xi^j)}{X - \xi^j}, i \in I \tag{5.14}$$

$$g_i(X) = \sum_{j \in [N]\setminus\{i\}} \frac{p(X)}{X - \xi^j}, i \in I \tag{5.15}$$

$$r_i(X) = \sum_{j \in [N]\setminus\{i\}} \frac{p(X) - p(\xi^j)}{X - \xi^j}, i \in I \tag{5.16}$$

Note that, by the definition of $p(X)$, $f_i(\xi^i) = a_i \, \forall i$. Thus, it suffices to compute $f_i(\xi^i)$ for all $i \in I$. Since $f_i(X) = g_i(X) - r_i(X)$ for $i \in I$, we have that $a_i = g_i(\xi^i) - r_i(\xi^i)$. Thus, we need to compute $g_i(\xi^i)$ and $r_i(\xi^i)$ for all $i \in I \subset K$.

$$\begin{aligned} g_i(\xi^i) &= p(\xi^i) \sum_{j \in [N]\setminus\{i\}} \frac{1}{\xi^i - \xi^j} \\ &= \frac{p(\xi^i) Z''_{\mathbb{H}}(\xi^i)}{2 Z'_{\mathbb{H}}(\xi^i)} \quad \text{(from Lemma 5.1)} \\ &= \frac{(N-1) d_i}{2\xi^i} \end{aligned}$$

In the above, we used $Z_H(X) = X^N - 1$ and that $p(\xi^i) = d_i$. In other words, $g_i(\xi^i)$ for all i can be obtained in $O(|I|)$ operations. Therefore, it suffices to compute $r_i(\xi^i)$ for all $i \in I$ efficiently. To this end, we write $r_i(X)$ as:

$$r_i(X) = \sum_{j \in [N]} \frac{p(X) - p(\xi^j)}{X - \xi^j} - \frac{p(X) - p(\xi^i)}{X - \xi^i}$$

By defining the bivariate polynomial $u(X,Y) = (p(X) - p(Y))/(X - Y)$, we get

$$r_i(X) = \sum_{j \in [N]} u(X, \xi^j) - u(X, \xi^i)$$

Defining $r(X) = \sum_{j \in [N]} u(X, \xi^j)$, we have:

$$r_i(X) = r(X) - u(X, \xi^i)$$

Substituting $X = \xi^i$ in the above, we have:

$$r_i(\xi^i) = r(\xi^i) - u(\xi^i, \xi^i) = r(\xi^i) - p'(\xi^i)$$

where $p'(\xi^i) = u(\xi^i, \xi^i)$ by the definition of formal derivative. Now, using $r(X) = Nu(X, 0)$ (Lemma 5.2), we have:

$$r(X) = N\frac{(p(X) - p(0))}{X}$$

Finally, substituting $X = \xi^i$ above, we have:

$$r(\xi^i) = N\frac{(d_i - p(0))}{\xi^i}$$

Thus, it remains to compute $p(0)$ and $p'(\xi^i)$ efficiently for each $i \in I$.

Computing the polynomial $q(X)$: Recall from Equation (5.13) that

$$q(\xi^j) = \frac{p(\xi^j)}{\widehat{Z}_K(\xi^j)}$$

for all $j \in K$. Furthermore, by Lemma 5.3, we have:

$$\hat{Z}_K(\xi^j) = \frac{Z'_{\mathbb{H}}(\xi^j)}{Z'_K(\xi^j)} = \frac{N/\xi^j}{Z'_K(\xi^j)}$$

for each $j \in K$. Observe that, given the set K, we can compute the polynomial $Z_K(X)$ in $O(|K| \log^2 |K|)$ operations using the fast multiplication, and we can then obtain $Z'_K(X)$ in additional $O(|K|)$ operations. Finally, $Z'_K(\xi^j)$ can be evaluated for $j \in K$ in additional $O(|K| \log^2 |K|)$ operations. Thus we can efficiently compute $q(\xi^j)$ for all $j \in K$ $O(|K| \log^2 |K|)$ operations. Since degree of $q(X)$ is strictly less than $|K|$, we can further interpolate to obtain the polynomial $q(X)$ in $O(|K| \log^2 |K|)$ field operations.

Computing $p(0)$: From Equation (5.13), we have

$$p(0) = \widehat{Z}_K(0) \cdot q(0)$$

Additionally, since we have

$$\hat{Z}_K(0) = \frac{Z_H(0)}{Z_K(0)} = \frac{-1}{Z_K(0)}$$

this enables us to compute $p(0)$ since $q(0)$ and $Z_K(0)$ are just the constant terms of the known polynomials $q(X)$ and $Z_K(X)$.

Computing $p'(\xi^i)$: We now show how to compute $p'(\xi^i)$ for each $i \in I$. Using the product rule for derivatives, we have:

$$p'(X) = q(X)\hat{Z}'_K(X) + q'(X)\hat{Z}_K(X)$$

We have shown how to compute $q(\xi^i)$ and $\hat{Z}_K(\xi^i)$ in $O(|K|\log^2|K|)$ field operations. By differentiating the polynomial $q(X)$ from earlier, we obtain $q'(X)$. Then, by fast evaluation, we get evaluations of $q'(X)$ at ξ^i for all $i \in I$, again in $O(|K|\log^2|K|)$ field operations. So it only remains to evaluate $\hat{Z}'_K(\xi^i)$ for each $i \in I$, which we show next. From the second equation of Lemma 5.3, we have:

$$\widehat{Z}'_K(\xi^i) = \sum_{j \in K \setminus \{i\}} \frac{Z'_{\mathbb{H}}(\xi^j)}{Z'_K(\xi^j)}\mu'_j(\xi^i) + \frac{Z'_{\mathbb{H}}(\xi^i)}{Z'_K(\xi^i)}\mu'_i(\xi^i)$$

Using Lemma 5.4, this becomes:

$$\widehat{Z}'_K(\xi^i) = N\xi^{-i} \sum_{j \in K \setminus \{i\}} \frac{1}{Z'_K(\xi^j)(\xi^i - \xi^j)} + \frac{N(N-1)}{2\xi^{2i}Z'_K(\xi^i)}$$

In other words, it suffices to efficiently compute φ_i for all $i \in I$, where

$$\varphi_i = \sum_{j \in K \setminus \{i\}} \frac{1}{Z'_K(\xi^j)(\xi^i - \xi^j)}$$

To this end, we define the following polynomial:

$$\Phi_i(X) = \sum_{j \in K \setminus \{i\}} \frac{1}{Z'_K(\xi^j)(X - \xi^j)}$$

Let $\{\eta_i(X)\}_{i\in K}$ be the set of Lagrange polynomials for the set $H_K = \{\xi^i : i \in K\}$. Then, since $\frac{\eta_j(X)}{Z_K(X)} = \frac{1}{Z'_K(\xi^j)(X-\xi^j)}$, $\Phi_i(X)$ can be rewritten as:

$$\begin{aligned}\Phi_i(X) &= \sum_{j\in K\setminus\{i\}} \frac{\eta_j(X)}{Z_K(X)} \\ &= \sum_{j\in K\setminus\{i\}} \frac{\eta_j(X)/(X-\xi^i)}{Z_K(X)/(X-\xi^i)}\end{aligned}$$

Substituting $X = \xi^i$ in the above, we have:

$$\begin{aligned}\varphi_i = \Phi_i(\xi^i) &= \left(\sum_{j\in K\setminus\{i\}} \frac{\eta_j(X)/(X-\xi^i)}{\prod_{k\in K\setminus\{i\}}(X-\xi^k)}\right)(\xi^i) \\ &= \sum_{j\in K\setminus\{i\}} \left(\frac{\eta_j(X)/(X-\xi^i)}{\prod_{k\in K\setminus\{i\}}(X-\xi^k)}\right)(\xi^i) \\ &= \sum_{j\in K\setminus\{i\}} \frac{\left(\eta_j(X)/(X-\xi^i)\right)(\xi^i)}{\left(\prod_{k\in K\setminus\{i\}}(X-\xi^k)\right)(\xi^i)} \\ &= \frac{1}{Z'_K(\xi^i)} \sum_{j\in K\setminus\{i\}} \left(\eta_j(X)/(X-\xi^i)\right)(\xi^i)\end{aligned}$$

Now, note that for all $j \neq i$, $\left(\eta_j(X)/(X-\xi^i)\right)(\xi^i)$ is just the evaluation of the polynomial $\frac{\eta_j(X)-\eta_j(\xi^i)}{X-\xi^i}$ at the point ξ^i. This is just $\eta'_j(\xi^i)$ by definition of formal derivative of the polynomial $\eta_j(X)$. Thus, we get:

$$\varphi_i = \Phi_i(\xi^i) = \frac{1}{Z'_K(\xi^i)} \sum_{j\in K\setminus\{i\}} \eta'_j(\xi^i)$$

Using that fact that $\sum_{j\in K} \eta_j(X) = 1$ (and hence, $\sum_{j\in K} \eta'_j(X) = 0$), we have

$$\sum_{j\in K\setminus\{i\}} \eta'_j(\xi^i) = \sum_{j\in K} \eta'_j(\xi^i) - \eta'_i(\xi^i) = -\eta'_i(\xi^i)$$

Thus, we get:

$$\varphi_i = \frac{-\eta'_i(\xi^i)}{Z'_K(\xi^i)}$$

At this point, it suffices to efficiently compute $\eta'_i(\xi^i)$ for $i \in I$. For this, we can use Lemma 5.5,

with $\mathcal{X} = H_K = \{\xi^j : j \in K\}$ and $Z_{\mathcal{X}}(X) = Z_K(X)$ as the vanishing polynomial of $\mathcal{X}$, to obtain:

$$\eta_i'(\xi^i) = \frac{F_K(\xi^i)}{Z_K'(\xi^i)}$$

where $F_K(X) = \sum_{j=2}^{k} z_j \binom{j}{2} X^{j-2}$ as defined in Lemma 5.5. Hence, it suffices to compute $F_K(\xi^i)$ for all $i \in I$, where $z_0, \ldots, z_k$ are the coefficients of the polynomial $Z_K(X)$ computed earlier. This concludes the proof of computation of a_i for $i \in I$.

Modifications for Computing b_j for $j \in K$: For computing b_j, we proceed as in the case of a_i, with the roles of sets I and K swapped (all of the corresponding lemmas can be modified accordingly). The only additional technical subtlety arises when we need to compute $\varphi_j = \Phi_j(\xi^j)$ for all $j \in K$, where the polynomial $\Phi_j(X)$ is defined as:

$$\Phi_j(X) = \sum_{i \in I \setminus \{j\}} \frac{\eta_i(X)}{Z_I(X)}$$

Now, we consider two cases: $j \in I$ and $j \in K \setminus I$. We handle the second case first. For each $j \in K \setminus I$, we can very easily compute $\varphi_j = \Phi_j(\xi^j)$ as

$$\begin{aligned}
\Phi_j(\xi^j) &= \sum_{i \in I \setminus \{j\}} \frac{\eta_i(\xi^j)}{Z_I(\xi^j)} \\
&= \frac{1}{Z_I(\xi^j)} \sum_{i \in I} \eta_i(\xi^j) \\
&= \frac{1}{Z_I(\xi^j)}
\end{aligned}$$

This is efficiently computed by evaluating $Z_I(\xi^j)$ for each $j \in K$ in $O(|K| \log^2 |K|)$ operations. Next, we consider the case where $j \in I$. For this, we can again proceed as in the analysis for computing a_i (with the roles of sets I and K swapped) till we need to compute

$$\varphi_j = \Phi_j(\xi^j) = \frac{-\eta_j'(\xi^j)}{Z_I'(\xi^j)}$$

for all $j \in I$. First of all, note that during the prior computation to reach this stage, we would have already computed $Z_I'(\xi^j)$ for all $j \in K$, and thus, for all $j \in I \subset K$. Next, observe that we also computed $\eta_i'(\xi^i)$ for $i \in I$ during the computation of a_i. This completes the computation of b_j for all $j \in K$, and finishes the proof of lemma 5.9. $\square$

5.4.2 Amortized Sublinear Batching

We now return to the question of how frequently should we run the offline phase to compute full parameters. For concrete analysis, let s be the period after which the rebasing takes place; i.e., after s batches of m operations each, we set the base table $\boldsymbol{T}_{\mathsf{b}}$ to the current table, setting $\boldsymbol{T}_{\mathsf{ch}} = \mathbf{0}$. At this point we also compute all encoded quotients for $\boldsymbol{T}_{\mathsf{b}}$ using the $O(N \log N)$ algorithm of [58]. Consider $\delta \leq ms$ as the upper-bound on δ, and distributing the cost of re-basing, the amortized overhead for the batch of m operations is: $O(ms \log^2(ms) + \frac{N \log N}{s})$ $\mathbb{F}$-operations and $O(ms + \frac{N \log N}{s})$ $\mathbb{G}$-operations. Ignoring the logarithmic factors, the cost is minimized by setting $s \approx \sqrt{N/m}$, resulting in amortized prover overhead of $\widetilde{O}(\sqrt{mN})$. We note that the above analysis considers the worst case scenario, where each update affects a distinct position in the table. In settings, where frequency of updates is non-uniform across positions in the table (e.g, in the blockchain example, if bulk of transactions come from small number of clients), we may be able to defer the offline phase even longer. Same is also true for settings where updates to the table are infrequent.

5.5 Batching-efficient RAM using Updatable Lookup Arguments

5.5.1 Memory Consistency for RAM

In this section, we briefly review and formalize existing memory-checking techniques to ensure correctness of RAM operations. The formal definitions for various relations involved in memory checking will be used to describe polynomial protocol for RAM in Section 5.6.

5.5.2 Correctness of RAM Update

The versatility of the RAM primitive stems from its updatability. While a load operation leaves the RAM unchanged, the store operation updates the value in the RAM associated with the referenced index. We model the update via the function $\mathsf{Upd}_{\mathcal{I}}$ which takes RAM $\boldsymbol{T} \in \mathsf{RAM}_{\mathcal{I},n}$, operation $o = (op, a, v) \in \mathcal{O}_{\mathcal{I}}$ as inputs and returns an updated RAM $\boldsymbol{T}' \in \mathsf{RAM}_{\mathcal{I},n}$. The updated RAM $\boldsymbol{T}' = \mathsf{Upd}_{\mathcal{I}}(\boldsymbol{T}, o)$ satisfies $\boldsymbol{T}' = \boldsymbol{T}$ if $op = 0$ while for $op = 1$ it satisfies $\boldsymbol{T}'[\,a\,] = v$ and $\boldsymbol{T}'[\,x\,] = \boldsymbol{T}[\,x\,]$ for $x \neq a$. The central problem in verifiable RAM protocols is to establish that a sequence of operations $\boldsymbol{o} = (o_1, \ldots, o_m)$ are correct with respect to the initial RAM state $\boldsymbol{T}$ and the final RAM state $\boldsymbol{T}'$. This involves ensuring that all load operations read the value which is consistent with updates to the RAM as a result of preceding store operations, and that $\boldsymbol{T}'$ is the final state. We say that an operation $o = (op, a, v)$ is *load-consistent* with respect to

RAM $\boldsymbol{T}$ if $v = \boldsymbol{T}[a]$ whenever o is a load operation (store operations are vacuously defined to be load-consistent). We formally define the notion of consistency below:

Definition 5.10 (Consistent Operations) *Let $n \in \mathbb{N}$ and $\boldsymbol{T}, \boldsymbol{T}' \in \mathsf{RAM}_{\mathcal{I},n}$ for some index set I. We say that a sequence of operations $\boldsymbol{o} = (o_1, \ldots, o_k) \in \mathcal{O}_{\mathcal{I}}^k$ over I is* consistent *with RAM states $\boldsymbol{T}, \boldsymbol{T}'$ if for all $i \in [k]$, $\boldsymbol{T}_i = \mathsf{Upd}_{\mathcal{I}}(\boldsymbol{T}_{i-1}, o_i)$ and operation o_i is load-consistent with respect to $\boldsymbol{T}_{i-1}$. Here we assume $\boldsymbol{T}_0 = \boldsymbol{T}$ and $\boldsymbol{T}_k = \boldsymbol{T}'$.*

For $m, n \in \mathbb{N}$, let $\mathsf{LRAM}_{\mathcal{I},m,n}$ denote the language consisting of tuples $(\boldsymbol{T}, \boldsymbol{o}, \boldsymbol{T}')$ with $\boldsymbol{T}, \boldsymbol{T}' \in \mathsf{RAM}_{\mathcal{I},n}$ and $\boldsymbol{o} \in (\mathcal{O}_{\mathcal{I}})^m$ such that $\boldsymbol{o}$ is consistent with $\boldsymbol{T}, \boldsymbol{T}'$. Next, we formalize the folklore technique of checking correctness of RAM operations using *address-ordered transcripts*.

5.5.3 Consistency Check via Transcripts

A *transcript* is time-stamped sequence of operations executed on a RAM. More formally, given a RAM $\boldsymbol{T} = (\boldsymbol{a}, \boldsymbol{v}) \in \mathsf{RAM}_{\mathcal{I},n}$, operation sequence $\boldsymbol{o} = (o_1, \ldots, o_m)$ with $o_i = (\bar{op}_i, \bar{a}_i, \bar{v}_i) \in \mathcal{O}_{\mathcal{I}}$ and RAM $\boldsymbol{T}' = (\boldsymbol{a}', \boldsymbol{v}') \in \mathsf{RAM}_{\mathcal{I},n}$, the *time ordered transcript* for the tuple $(\boldsymbol{T}, \boldsymbol{o}, \boldsymbol{T}')$ is given by the table tr with $k = 2n + m$ rows and four columns $\mathsf{tr} = (\boldsymbol{t}, \boldsymbol{op}, \boldsymbol{A}, \boldsymbol{V})$ defined as follows: (i) $\boldsymbol{t} = I_k = (1, \ldots, k)$, (ii) $\boldsymbol{op} = 0^n || (\bar{op}_1, \ldots, \bar{op}_m) || 0^n$, (iii) $\boldsymbol{A} = \boldsymbol{a} || (\bar{a}_1, \ldots, \bar{a}_m) || \boldsymbol{a}'$ and (iv) $\boldsymbol{V} = \boldsymbol{v} || (\bar{v}_1, \ldots, \bar{v}_m) || \boldsymbol{v}'$. The i^{th} row of the table tr is (t_i, op_i, A_i, V_i) for $i \in [k]$. The first n records in tr correspond to the contents of $\boldsymbol{T}$, the next m records correspond to the operations in $\boldsymbol{o}$ and final n records correspond to contents of $\boldsymbol{T}'$. The timestamp column $\boldsymbol{t}$ is added to order operations with the same index. Notationally, we write $\mathsf{tr} = \mathsf{TimeTr}(\boldsymbol{T}, \boldsymbol{o}, \boldsymbol{T}')$.

We call a transcript $\mathsf{tr} = (\boldsymbol{t}, \boldsymbol{op}, \boldsymbol{A}, \boldsymbol{V})$ to be *address ordered* if $A_i \leq A_{i+1}$ for $i \in [k-1]$ and $t_i < t_{i+1}$ whenever $A_i = A_{i+1}$. For a transcript $\mathsf{tr} = (\boldsymbol{t}, \boldsymbol{op}, \boldsymbol{A}, \boldsymbol{V})$ with k records and a permutation $\sigma : [k] \to [k]$, we use $\sigma(\mathsf{tr})$ to denote the transcript $(\sigma(\boldsymbol{t}), \sigma(\boldsymbol{op}), \sigma(\boldsymbol{A}), \sigma(\boldsymbol{V}))$ obtained by permuting the records of tr according to the permutation σ. An address ordered transcript for tuple $(\boldsymbol{T}, \boldsymbol{o}, \boldsymbol{T}')$ is defined as $\mathsf{tr}^* = \sigma(\mathsf{tr})$ where $\mathsf{tr} = \mathsf{TimeTr}(\boldsymbol{T}, \boldsymbol{o}, \boldsymbol{T}')$ and σ is a permutation such that tr^* is address ordered. We denote it by $\mathsf{tr}^* = \mathsf{AddrTr}(\boldsymbol{T}, \boldsymbol{o}, \boldsymbol{T}')$. We say that an address ordered transcript $\mathsf{tr} = (\boldsymbol{t}, \boldsymbol{op}, \boldsymbol{A}, \boldsymbol{V})$ satisfies *load-store correctness* if for all pairs of consecutive records (t_i, op_i, A_i, V_i) and $(t_{i+1}, op_{i+1}, A_{i+1}, V_{i+1})$ we have $V_{i+1} = V_i$ whenever $op_{i+1} = 0$ (load operation) and $A_i = A_{i+1}$, i.e, a load operation does not change the value at an index. We formally state the folklore technique for enforcing memory consistency in our setting.

Lemma 5.10 *Let $\mathbb{F}$ be a finite field, $m, n \in \mathbb{N}$ be positive integers and $I \subseteq \mathbb{F}$. Then $(\boldsymbol{T}, \boldsymbol{o}, \boldsymbol{T}') \in \mathsf{LRAM}_{\mathcal{I},n,m}$ if and only if the address ordered transcript $\mathsf{tr}^* = \mathsf{AddrTr}(\boldsymbol{T}, \boldsymbol{o}, \boldsymbol{T}')$ satisfies load-store correctness.*

The consistency check in Lemma 5.10 can be encoded as an arithmetic circuit of size $\widetilde{O}(m+n)$, thus yielding an argument of knowledge for the language $\mathsf{LRAM}_{\mathcal{I},n,m}$ with prover complexity quasi-linear in $m+n$. For completeness, we present a self-contained argument of knowledge for $\mathsf{LRAM}_{\mathcal{I},m,m}$ ($m=n$) based on the "polynomial protocol" framework defined in [64].

5.5.4 Improved Batching-Efficient RAM

We now detail the steps required to realize batching efficient RAM outlined in the technical overview. We first recall the techniques presented in Section 5.3.2 to obtain a committed index lookup argument. Here, we leverage the random linear combination technique to simultaneously check two equations at correlated points of evaluation. We restate the Lemma 5.7 here for reference.

Lemma 5.11 (Restated) *Let $\boldsymbol{t} \in \mathbb{F}^n$ and let $\boldsymbol{a}, \boldsymbol{v} \in \mathbb{F}^m$ for some positive integers m, n. Let $\boldsymbol{I}_n$ denote the vector $(1, \ldots, n)$. Then for $\gamma \leftarrow_R \mathbb{F}$, $(\boldsymbol{v} + \gamma \boldsymbol{a}) \preceq (\boldsymbol{t} + \gamma \boldsymbol{I}_n)$ implies $\boldsymbol{v} = \boldsymbol{t}[\boldsymbol{a}]$ except with probability $mn/|\mathbb{F}|$.*

Thereafter, we invoke Lemma 5.7 to construct a committed index lookup argument (Figure 5.2) using a committed sub-vector argument $(\mathcal{P}_{\mathsf{sv}}, \mathcal{V}_{\mathsf{sv}})$. We now restate the Lemma 5.8 here.

Lemma 5.12 (Restated) *Assuming that $(\mathcal{P}_{\mathsf{sv}}, \mathcal{V}_{\mathsf{sv}})$ is an argument of knowledge for the relation $\mathcal{R}^{\mathsf{subvec}}_{\mathsf{srs},N,m}$ in the AGM, the interactive protocol in Figure 5.2 is an argument of knowledge for the relation $\mathcal{R}^{\mathsf{lookup}}_{\mathsf{srs},N,m}$ in the AGM.*

5.5.5 Almost Identical RAM States

For a vector $\boldsymbol{a} \in [N]^m$, let $\mathsf{uniq}(\boldsymbol{a}) = \{a_i : i \in [m]\}$ denote the subset of unique values in $\boldsymbol{a}$. We call two RAM states $\boldsymbol{T}, \boldsymbol{T}' \in \mathbb{F}^N$ to be $\boldsymbol{a}$-*identical* if $\boldsymbol{T}[i] = \boldsymbol{T}'[i]$ for all $i \notin \mathsf{uniq}(\boldsymbol{a})$. As before, let $T(X), T^*(X)$ and $a(X)$ be polynomials encoding the vectors $\boldsymbol{T}, \boldsymbol{T}'$ (over $\mathbb{H}$) and $\boldsymbol{a}$ (over $\mathbb{V}$). Let c_T, c'_T and c_a be the commitments to vectors $\boldsymbol{T}, \boldsymbol{T}'$ and $\boldsymbol{a}$ respectively in the group $\mathbb{G}_1$. The polynomial protocol to prove that $\boldsymbol{T}, \boldsymbol{T}' \in \mathbb{F}^N$ and $\boldsymbol{a} \in \mathbb{F}^m$ are $\boldsymbol{a}$-identical requires proving the relation $Z_I(X)(T(X) - T^*(X)) = 0$ over the set $Z_{\mathbb{H}}$ where $I = \mathsf{uniq}(\boldsymbol{a})$ and $Z_I(X) = \prod_{i \in I}(X - \xi^i)$ is the vanishing polynomial for the set $\mathbb{H}_I = \{\xi^i : i \in I\}$. To proceed, the honest prover commits to polynomial $Z_I(X)$ and proves (i) $Z_I(X) \cdot (T(X) - T^*(X)) = 0 \bmod Z_{\mathbb{H}}$ and (ii) the zeroes of $Z_I(X)$ form a subset of zeroes of $\mathbb{H}_I(X)$ as defined. Together, the two conditions imply that $T(\xi^i) = T^*(\xi^i)$ for $i \notin \mathsf{uniq}(\boldsymbol{a})$. To prove the first relation, the prover

computes the polynomial $D(X)$ as below:

$$\begin{aligned} D(X) &= \frac{(T(X) - T^*(X)) \cdot Z_I(X)}{Z_{\mathbb{H}}(X)} \\ &= \sum_{i \in I} \frac{(T(\xi^i) - T^*(\xi^i))\mu_i(X)}{Z_{\mathbb{H}}(X)} Z_I(X) \end{aligned}$$

Substituting $\Delta_i = T(\xi^i) - T^*(\xi^i)$, $\mu_i(X) = Z_{\mathbb{H}}(X)/(Z'_{\mathbb{H}}(\xi^i)(X - \xi^i))$, we get

$$D(X) = \sum_{i \in I} \frac{\Delta_i}{Z'_{\mathbb{H}}(\xi^i)} \left(\frac{Z_I(X)}{X - \xi^i} \right) = \sum_{i \in I} \frac{\Delta_i Z'_I(\xi^i)}{Z'_{\mathbb{H}}(\xi^i)} \kappa_i(X) \tag{5.17}$$

In the above, the summation only runs over indices in I, as $\Delta_i = 0$ for $i \notin I$. In the final equality, we use $\kappa_i(X) = Z_I(X)/(Z'_I(\xi^i)(X - \xi^i))$ for $i \in I$ which we recognize as the Lagrange basis polynomials for the set $\{\xi^i : i \in I\}$. Thus, Equation (5.17) implies that $D(X)$ is at most degree $|I| - 1$ polynomial, with $D(\xi^i) = \Delta_i Z'_I(\xi^i)/Z'_{\mathbb{H}}(\xi^i)$ for $i \in I$. The prover can therefore interpolate $D(X)$ (in power basis) in $O(|I| \log^2 |I|)$ $\mathbb{F}$-operations and compute $[D(X)]_1$ in $O(|I|)$ $\mathbb{G}_1$-operations. The prover sends the commitment $[D(X)]_1$ to the verifier. Finally, the verifier can check the identity $Z_I(X) \cdot (T(X) - T^*(X)) = D(X) \cdot Z_{\mathbb{H}}(X)$ by a pairing check. For this, since the tables are committed in $\mathbb{G}_1$, prover will need to send $[Z_I(X)]_2$.

Next, the prover needs to show that zeroes of Z_I are indeed in the set $\mathbb{H}_I = \{\xi^i : i \in I\} = \{\xi^{a_i} : i \in [m]\}$. Clearly, it suffices to show that $Z_I(X)$ divides the polynomial $\prod_{i \in [m]}(X - \xi^{a_i})$. To obtain a polynomial protocol, the prover commits to an auxiliary polynomial $h(X) = \sum_{i=1}^m \xi^{a_i} \tau_i(X)$ which interpolates the vector $\boldsymbol{h} = (\xi^{a_1}, \ldots, \xi^{a_m})$. The correctness of h polynomial can be established by showing that the interpolated vector $\boldsymbol{h}$ satisfies committed index lookup relation $\boldsymbol{h} = \boldsymbol{T}_{\text{exp}}[\boldsymbol{a}]$ where $\boldsymbol{T}_{\text{exp}} = (\xi^1, \ldots, \xi^N)$. Moreover, we notice that the polynomial interpolating the table $\boldsymbol{T}_{\text{exp}}$ is particularly simple, i.e, $T_{\text{exp}}(X) = X$, and thus the setup need not be augmented with table-specific parameters for $\boldsymbol{T}_{\text{exp}}$. Finally, it remains to show that $Z_I(X)$ divides $K(X) = \prod_{i=1}^m (X - h(\nu^i))$. To do so, the prover commits to $K(X)$ and the quotient polynomial $q(X) = K(X)/Z_I(X)$. The verifier checks the polynomial identities at α, i.e $K(\alpha) = q(\alpha)Z_I(\alpha)$ and $K(\alpha) = \prod_{i=1}^m (\alpha - h(\nu^i))$. The former is easily accomplished using evaluation proofs for K, q and Z_I at α. For checking the latter, the prover commits to another polynomial $u(X)$ satisfying $u(\nu^i) = \prod_{j=1}^{i-1} \left((\alpha - h(\nu^j))/(1 + \beta\tau_1(\nu^j))\right)$ for $i \in [m]$ where

$\beta = K(\alpha) - 1$. The verifier ensures the correctness of $u(X)$ by checking:

$$\begin{aligned} \tau_1(X)(u(X) - 1) &= 0 \bmod Z_{\mathbb{V}} \\ u(\nu X)(1 + \beta\tau_1(X)) - u(X)(\alpha - h(X)) &= 0 \bmod Z_{\mathbb{V}}. \end{aligned} \tag{5.18}$$

We prove that the above constraints imply that $K(\alpha) = \prod_{i\in[m]}(\alpha - h(\nu^i))$ in Lemma 5.13. Note that in this protocol we require commitment to the polynomial Z_I in both $\mathbb{G}_1$ and $\mathbb{G}_2$, and thus another pairing check is required to show that the $Z_I(X)$ committed in $\mathbb{G}_1$ is the same as the $Z_I(X)$ committed in $\mathbb{G}_2$ (used for the real pairing check). The complete protocol for checking that RAMs $\boldsymbol{T}$ and $\boldsymbol{T}'$ are identical outside indices in $\boldsymbol{a}$ is given in Figure 5.3.

Lemma 5.13 *There exists a polynomial $u(X) \in \mathbb{F}[X]$ satisfying the identities in Equation (5.18) if and only if $K(\alpha) = 1 + \beta = \prod_{i\in[m]}(\alpha - h(\nu^i))$.*

Proof: Assume that the identitites hold for some polynomial $u(X)$. The first identity implies $u(\nu) = 1$. From the second identity, we conclude that for all $i \in [m]$, we have $u(\nu^{i+1}) = u(\nu^i) \cdot ((\alpha - h(\nu^i))/(1 + \beta\tau_1(\nu^i)))$, and thus:

$$1 = u(\nu^{m+1})/u(\nu) = \prod_{i\in[m]} \left(\frac{\alpha - h(\nu^i)}{1 + \beta\tau_1(\nu^i)} \right).$$

We observe that the product of denominators in the above equation is simply $1 + \beta$ as $\tau_1(\nu^i)$ is 0 for all $i \neq 1$, and thus $1 + \beta = \prod_{i=1}^{m}(\alpha - h(\nu^i))$. In the other direction, it is easy to check that $u(X)$ as defined for an honest prover, satisfies the identities in Equation 5.18. □

5.5.6 Batching-Efficient RAM: Combined Protocol

We put the entire protocol together now. Let I denote the set of indices $\{1, \ldots, N\}$, and $\boldsymbol{I}_N$ denote the vector $(1, \ldots, N)$. We formally define the committed RAM relation for which we present an argument of knowledge in this section.

Definition 5.11 *We define the* committed RAM *relation $\mathcal{R}^{\mathsf{ram}}_{\mathsf{srs},N,m}$ to consist of tuples $((c_T, c'_T, c_{op}, c_a, c_w), (\boldsymbol{T}, \boldsymbol{T}', \boldsymbol{op}, \boldsymbol{a}, \boldsymbol{w}))$ such that:*

- *$(\boldsymbol{T}, \boldsymbol{o}, \boldsymbol{T}') \in \mathsf{LRAM}_{\mathcal{I},N,m}$ for $\boldsymbol{o} = (o_1, \ldots, o_m)$ where we have $o_i = (op_i, a_i, w_i) \in \mathcal{O}_{\mathcal{I}}$ for all $i \in [m]$ (here we implicitly view vectors $\boldsymbol{T}$ and $\boldsymbol{T}'$ as RAMs with index column I_N).*
- *$c_T = \mathsf{KZG.Commit}(\mathsf{srs}, T(X))$, $c'_T = \mathsf{KZG.Commit}(\mathsf{srs}, T^*(X))$, $c_{op} = \mathsf{KZG.Commit}(\mathsf{srs}, op(X))$, $c_a = \mathsf{KZG.Commit}(\mathsf{srs}, a(X))$, $c_w = \mathsf{KZG.Commit}(\mathsf{srs}, w(X))$ where polynomials $T(X), T^*(X)$*

Common Input: $\mathsf{srs}, c_T, c'_T, c_a$.
Prover's Input: Vectors $\boldsymbol{T}, \boldsymbol{T}' \in \mathbb{F}^N$, $\boldsymbol{a} \in \mathbb{F}^m$. Polynomials $T(X), T^*(X)$ and $a(X)$ encoding $\boldsymbol{T}, \boldsymbol{T}'$ and $\boldsymbol{a}$ respectively.

Round 1: Prover commits to auxiliary polynomials

1. $\mathcal{P}$ computes the following:
   - $I = \mathsf{uniq}(\boldsymbol{a})$, $Z_I(X) = \prod_{i \in I}(X - \xi^i)$.
   - $D(X) = Z_I(X)(T(X) - T^*(X))/Z_{\mathbb{H}}(X)$.
   - $h(X)$ such that $h(\nu^i) = \xi^{a_i}$ for $i \in [m]$.
   - $K(X) = \prod_{i=1}^{m}(X - h(\nu^i))$, $q(X) = K(X)/Z_I(X)$.
   - $c_z = [Z_I(X)]_1$, $c'_z = [Z_I(X)]_2$, $c_d = [D(X)]_1$, $c_h = [h(X)]_1$, $c_k = [K(X)]_1$, $c_q = [q(X)]_1$.
2. $\mathcal{P}$ sends $c_z, c'_z, c_d, c_h, c_k, c_q$ to $\mathcal{V}$.
3. $\mathcal{V}$ samples $\alpha \leftarrow_R \mathbb{F}$ and sends α to $\mathcal{P}$.

Round 2: Prover commits to polynomial $u(X)$.

1. $\mathcal{P}$ sets $\beta = K(\alpha) - 1$ and interpolates $u(X)$ on $\mathbb{V}$ such that $u(\nu^i) = \prod_{j=1}^{i-1}\big((\alpha - h(\nu^j))/(1 + \beta\tau_1(\nu^j))\big)$ for $i \in [m]$.
2. $\mathcal{P}$ computes $c_u = [u(X)]_1$ and sends c_u to $\mathcal{V}$.
3. $\mathcal{V}$ samples $r \leftarrow_R \mathbb{F}$ and sends r to $\mathcal{P}$.

Round 3: Prover batches checks in Eq (5.18).

1. $\mathcal{P}$ computes $Q(X) = \big(u(\nu X)(1 + \beta\tau_1(X)) - u(X)(\alpha - h(X)) + r\tau_1(X)(u(X) - 1)\big)/Z_{\mathbb{V}}(X)$
2. $\mathcal{P}$ sends $c_Q = [Q(X)]_1$ to $\mathcal{V}$.
3. $\mathcal{V}$ samples $s \leftarrow_R \mathbb{F}$ and sends s to $\mathcal{P}$.

Round 4: Prover sends evaluations.

1. $\mathcal{P}$ computes and sends the following evaluations to $\mathcal{V}$: $\langle z \rangle_\alpha = Z_I(\alpha)$, $\langle q \rangle_\alpha = q(\alpha)$, $\langle K \rangle_\alpha = K(\alpha)$, $\langle Q \rangle_s = Q(s)$, $\langle u \rangle_s = u(s)$, $\langle u \rangle_{\nu s} = u(\nu s)$, and $\langle h \rangle_s = h(s)$.
2. $\mathcal{V}$ samples $r_\alpha, r_s \leftarrow_R \mathbb{F}$ and sends r_α, r_s to $\mathcal{P}$.

Figure 5.3: Argument for showing RAMs are identical outside small set of indices.

Round 5: Prover batches evaluation proofs.

1. $\mathcal{P}$ computes the following:
   - $p_\alpha(X) = Z_I(X) + r_\alpha q(X) + r_\alpha^2 K(X)$.
   - $p_s(X) = Q(X) + r_s u(X) + r_s^2 h(X)$.
   - $\Pi_\alpha = \mathsf{KZG.Prove}(\mathsf{srs}, p_\alpha, \alpha)$.
   - $\Pi_s = \mathsf{KZG.Prove}(\mathsf{srs}, p_s, s)$, $\Pi_{\nu s} = \mathsf{KZG.Prove}(\mathsf{srs}, u, \nu s)$.
2. $\mathcal{P}$ sends $\Pi_\alpha, \Pi_s, \Pi_{\nu s}$ to $\mathcal{V}$.

Round 6: Verifier checks identities.

1. $\mathcal{V}$ computes $[p_\alpha]_1 = c_z + r_\alpha c_q + r_\alpha^2$, $[p_z]_1 = c_Q + r_s c_u + r_s^2 c_h$.
2. $\mathcal{V}$ checks the following:
   - $\langle z \rangle_\alpha \cdot \langle q \rangle_\alpha = \langle K \rangle_\alpha$.
   - $\langle u \rangle_{\nu s}(1 + \beta\tau_1(s)) - \langle u \rangle_s(\alpha - \langle h \rangle_s) + r\tau_1(s)(\langle u \rangle_s - 1) = \langle Q \rangle_s Z_{\mathbb{V}}(s)$.
   - $e(c_T - c'_T, c'_z) = e(c_d, [Z_{\mathbb{H}}(X)]_2)$.
   - $e([1]_1, c'_z) = e(c_z, [1]_2)$.
   - $\mathsf{KZG.Verify}(\mathsf{srs}\ , [p_\alpha]_1, \langle z \rangle_\alpha + r_\alpha \langle q \rangle_\alpha + r_\alpha^2 \langle K \rangle_\alpha, \alpha, \Pi_\alpha)$.
   - $\mathsf{KZG.Verify}(\mathsf{srs}, [p_z]_1, \langle Q \rangle_s + r_s \langle u \rangle_s + r_s^2 \langle K \rangle_s, s, \Pi_s)$.
   - $\mathsf{KZG.Verify}(\mathsf{srs}, c_u, \langle u \rangle_{\nu s}, \nu s, \Pi_{\nu s})$.

Round 7: Check correctness of polynomial $h(X)$.

1. $\mathcal{P}$ and $\mathcal{V}$ execute committed index lookup argument (Fig 5.2) to check $([X]_1, c_a, c_h) \in \mathcal{R}^{\mathsf{lookup}}_{\mathsf{srs},N,m}$.
2. $\mathcal{V}$ accepts if the above argument accepts and all the preceding checks succeed.

Figure 5.3: Argument for showing RAMs are identical outside small set of indices.

encode vectors $\boldsymbol{T}, \boldsymbol{T}'$ *over* $\mathbb{H}$*, while* $op(X), a(X)$ *and* $w(X)$ *encode vectors* $\boldsymbol{op} = (op_1, \ldots, op_m)$*,* $\boldsymbol{a}$ *and* $\boldsymbol{w}$ *over* $\mathbb{V}$*.*

As outlined in the blueprint, the prover first commits to "smaller" RAMs $\mathbf{S} = (\boldsymbol{a}, \boldsymbol{v})$ and $\mathbf{S}' = (\boldsymbol{a}, \boldsymbol{v}')$ where $\boldsymbol{v} = \boldsymbol{T}[\boldsymbol{a}]$ and $\boldsymbol{v}' = \boldsymbol{T}'[\boldsymbol{a}]$. The prover commits to $\mathbf{S}$ and $\mathbf{S}'$ by sending commitments c_v and c'_v to $\boldsymbol{v}$ and $\boldsymbol{v}'$. Then the prover and verifier execute the committed index lookup protocol to prove:

$$(c_T, c_a, c_v) \in \mathcal{R}^{\mathsf{lookup}}_{\mathsf{srs},N,m} \wedge (c'_T, c_a, c'_v) \in \mathcal{R}^{\mathsf{lookup}}_{\mathsf{srs},N,m} \tag{5.19}$$

The verifier uses a random challenge $\chi \leftarrow_R \mathbb{F}$ to reduce two instances of $\mathcal{R}^{\mathsf{lookup}}_{\mathsf{srs},N,m}$ to one instance $(c_T + \chi c'_T, c_a, c_v + \chi c'_v) \in \mathcal{R}^{\mathsf{lookup}}_{\mathsf{srs},N,m}$. Then, we show that RAMs $\boldsymbol{T}$ and $\boldsymbol{T}'$ are $\boldsymbol{a}$-identical using the protocol in Figure 5.3, described in Section 5.5.5. All that remains is to prove is that the operation sequence $\boldsymbol{o}$ is consistent with small RAMs $\mathbf{S}$ and $\mathbf{S}'$. We check this using the argument in Section 5.6, which is obtained by compiling the polynomial protocol for RAM in Section 5.7 into an argument of knowledge in the AGM. Specifically, the prover and the verifier set $c_S = (c_a, c_v)$, $c'_S = (c_a, c'_v)$ and $c_o = (c_{op}, c_a, c_w)$, and execute the argument of knowledge for showing $(c_S, c_o, c'_S) \in \mathcal{R}^{\mathsf{LRAM}}_{\mathsf{srs},m}$ (see Definition 5.12). We provide the complete protocol listing in Figure 5.4. The protocol in Figure 5.4 assumes pre-computed parameters for the tables $\boldsymbol{T}$ and $\boldsymbol{T}'$. The maintenance of these pre-computed parameters in the presence of updates is detailed in Section 5.4.

Theorem 5.4 *The protocol in Figure 5.4 is a succinct argument of knowledge for the relation* $\mathcal{R}^{\mathsf{ram}}_{\mathsf{srs},N,m}$ *in the AGM, under the Q-DLOG assumption for the bilinear group* $(\mathbb{F}, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2)$*.*

5.6 Argument for RAM From Polynomial Protocols

In this section, we give a self-contained argument of knowledge for membership in the language $\mathsf{LRAM}_{\mathcal{I},m,m}$ introduced in Section 5.5.1. We first consider the polynomial encoding of different RAM artefacts.

5.6.1 Polynomial Encoding

Let $k = 3m$ and let ω be a primitive k^{th} root of unity in $\mathbb{F}$. Let $\nu = \omega^3$, and thus ν is a primitive m^{th} root of unity in $\mathbb{F}$ (We assume, these roots exist in $\mathbb{F}$). We recall $\mathbb{V}$ as the subgroup consisting of m^{th} roots of unity with associated Lagrange basis polynomials $\{\tau_i(X)\}_{i\in[m]}$, while we additionally introduce the set $\mathbb{K}$ of k^{th} roots of unity with $\{\lambda_i(X)\}_{i\in[k]}$ as the associated

Setup $(1^\lambda, N, m, \boldsymbol{T}, \boldsymbol{T}')$:

- $\mathsf{srs} = (\{[\tau^i]_1\}_{i=0}^N, \{[\tau^i]_2\}_{i=0}^N)$ for $\tau \leftarrow_R \mathbb{F}$
- Both $\mathcal{P}$ and $\mathcal{V}$ precomputes $[Z_{\mathbb{H}}(X)]_1, [Z_{\mathbb{H}}(X)]_2$
- $\mathcal{P}$ precomputes the following:
  - $W_2^i = [Z_{\mathbb{H}}(X)/(X - \xi^i)]_2$, $i \in [N]$
- $\mathcal{P}$ precomputes the following (with respect to $(\boldsymbol{T}, \boldsymbol{T}')$):
  - $W_1^i = [(T(X) - T(\xi^i))/(X - \xi^i)]_2$, $i \in [N]$,
  - $W_1^{i'} = [(T^*(X) - T^*(\xi^i))/(X - \xi^i)]_2$, $i \in [N]$.

Common Input: srs, $c_T, c'_T, c_{op}, c_a, c_w \in \mathbb{G}_1$.
Prover's Input: Vectors $\boldsymbol{T}, \boldsymbol{T}', \boldsymbol{op}, \boldsymbol{a}, \boldsymbol{w}$ and their encoding polynomials.

Round 1: Commit to sub RAMs.

1. $\mathcal{P}$ computes $\boldsymbol{v} = \boldsymbol{T}[\boldsymbol{a}]$, $\boldsymbol{v}' = \boldsymbol{T}'[\boldsymbol{a}]$ and the encoding polynomials $v(X)$ and $v^*(X)$.
2. $\mathcal{P}$ computes $c_v = [v(X)]_1$, $c'_v = [v^*(X)]_1$, and sends c_v, c'_v to $\mathcal{V}$.
3. $\mathcal{V}$ samples $\chi \leftarrow_R \mathbb{F}$ and sends χ to $\mathcal{P}$.

Round 2: Execute committed index lookup.

1. $\mathcal{P}$ and $\mathcal{V}$ compute $\hat{c}_T = c_T + \chi c'_T$, $\hat{c}_v = c_v + \chi c'_v$.
2. $\mathcal{P}$ computes $\hat{\boldsymbol{T}} = \boldsymbol{T} + \chi \boldsymbol{T}'$, $\hat{\boldsymbol{v}} = \boldsymbol{v} + \chi \boldsymbol{v}'$.
3. $\mathcal{P}$ and $\mathcal{V}$ execute committed index lookup argument in Fig 5.2, with $(\hat{c}_T, c_a, \hat{c}_v)$ as the common input and $(\hat{\boldsymbol{T}}, \boldsymbol{a}, \hat{\boldsymbol{v}})$ as prover's input.

Round 3: Prove RAMs are $\boldsymbol{a}$-identical.

1. $\mathcal{P}$ and $\mathcal{V}$ execute argument in Fig 5.3 with common input (c_T, c'_T, c_a) and prover's input as $(\boldsymbol{T}, \boldsymbol{T}', \boldsymbol{a})$.

Round 4: Prove sub RAMs are memory-consistent with update.

1. $\mathcal{P}$ and $\mathcal{V}$ execute argument in Fig 5.9 to check $(c_S, c_o, c'_S) \in \mathcal{R}_{\mathsf{srs},m}^{\mathsf{LRAM}}$ with $c_S = (c_a, c_v)$, $c'_S = (c_a, c'_v)$ and $c_o = (c_{op}, c_a, c_w)$.
2. $\mathcal{V}$ accepts if all sub-protocols accept.

Figure 5.4: Our batching-efficient RAM protocol

Lagrange polynomials.

$$\mathbb{K} = \{\omega, \ldots, \omega^k\}, \quad \mathbb{V} = \{\nu, \ldots, \nu^m\} \tag{5.20}$$

As before, we define the encoding of vectors in $\boldsymbol{f} \in \mathbb{F}^k$ as $\mathsf{Encode}_{\mathbb{K}}(\boldsymbol{f}) = \sum_{i\in[k]} f_i\lambda_i(X)$. We canonically extend the encoding of vectors to encode RAM, operations and transcripts by encoding their component vectors. Thus, for a RAM $\boldsymbol{T} = (\boldsymbol{a}, \boldsymbol{v}) \in \mathsf{RAM}_{\mathbb{I},m}$, we define its encoding $\widetilde{T} = (a(X), v(X))$ where $a(X), v(X) \in \mathbb{F}_{<m}[X]$ encode vectors $\boldsymbol{a}, \boldsymbol{v}$ respectively. Given an operation sequence $\boldsymbol{o} = (o_1, \ldots, o_m)$ with $o_i = (\bar{op}_i, \bar{a}_i, \bar{v}_i)$ we encode $\boldsymbol{o}$ as $\widetilde{O} = (\bar{op}(X)$,$\bar{a}(X)$,$\bar{v}(X))$ where $\bar{op}(X)$ encodes the vector $\boldsymbol{op} = (\bar{op}_1, \ldots, \bar{op}_m)$, $\bar{a}(X)$ encodes the vector $(\bar{a}_1, \ldots, \bar{a}_m)$ and $\bar{v}(X)$encodes the vector $(\bar{v}_1, \ldots, \bar{v}_m)$. Finally, a transcript $\mathsf{tr} = (\boldsymbol{t}, \boldsymbol{op}, \boldsymbol{A}, \boldsymbol{V})$ for tuples $(\boldsymbol{T}, \boldsymbol{o}, \boldsymbol{T}')$ where $\boldsymbol{T}, \boldsymbol{T}'$ are RAMs of size m, and $\boldsymbol{o}$ is an operation sequence of size m is encoded as $\widetilde{\mathsf{tr}} = (t(X),\ op(X),\ A(X),\ V(X))$ where the polynomials $t(X), op(X), V(X)$ and $A(X)$ encode the respective vectors in $\mathbb{F}^k$ (See Section 5.5.1).

5.6.2 Relations over Polynomial Encodings

In this section, we describe polynomial checks for two important relations we need in subsequent sections, viz, (i) checking concatenation of vectors and (ii) checking monotonicity and load-store consistency of a transcript. The lemma below specifies the polynomial identities for verifying that vector $\boldsymbol{v} \in \mathbb{F}^k$ is concatenation of vectors $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}$ in $\mathbb{F}^m$.

Lemma 5.14 *Let $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c} \in \mathbb{F}^m$ and $v \in \mathbb{F}^k$ be vectors encoded by polynomials $a(X), b(X), c(X)$ and $v(X)$ respectively. Then,*

$$a(X^3) - v(X) = 0 \quad mod\ Z(X) \tag{A1}$$
$$b(X^3) - v(\omega^m X) = 0 \quad mod\ Z(X) \tag{A2}$$
$$c(X^3) - v(\omega^{2m} X) = 0 \quad mod\ Z(X) \tag{A3}$$

for $Z(X) = \prod_{i=1}^{m}(X - \omega^i)$ if and only if $\boldsymbol{v} = \boldsymbol{a}||\boldsymbol{b}||\boldsymbol{c}$.

Proof: Assume that the polynomial identities hold. Substituting $X = \omega^i$ for $i \in [m]$ in above equations implies for $i \in [m]$: $a_i = v_i$ (Eq (A1)), $b_i = v_{m+i}$ (Eq (A2)) and $c_i = v_{2m+i}$ (Eq (A3)), which together imply $\boldsymbol{v} = \boldsymbol{a}||\boldsymbol{b}||\boldsymbol{c}$. Converse follows by observing that $\boldsymbol{v} = \boldsymbol{a}||\boldsymbol{b}||\boldsymbol{c}$ implies that $v(X) = a(X^3)$, $v(\omega^m X) = b(X^3)$ and $v(\omega^{2m} X) = c(X^3)$ holds for all $X = \omega^i, i \in [m]$. Thus, the equalities hold modulo the polynomial $Z(X)$ as defined above. □

Next, we specify polynomial checks on the encoding of a transcript to ensure it satisfies address-ordering and load-store consistency. Let N be an upper bound on the values

of $\boldsymbol{A}$, i.e, the index set $I \subseteq [N]$. Let $\mathsf{tr} = (\boldsymbol{t}, \boldsymbol{op}, \boldsymbol{A}, \boldsymbol{V})$ be a transcript encoded as $\widetilde{\mathsf{tr}} = (t(X), op(X), A(X), V(X))$. Recall that we need to check two conditions on tr, viz, (i) *monotonicity*: the transcript is sorted by address and timestamp respectively, i.e, $A_i \leq A_{i+1}$ for all $i < k$ and $t_i < t_{i+1}$ whenever $A_i = A_{i+1}$, (ii) *load-store consistency*: whenever $op_{i+1} = 0$ and $A_i = A_{i+1}$, we have $V_i = V_{i+1}$. To do so, we exhibit disjoint sets I_1, I_2 with $I_1 \uplus I_2 = [k-1]$ such that: (i) for all $i \in I_1$, $A_i < A_{i+1}$, (ii) for all $i \in I_2$, $(A_i = A_{i+1}) \wedge (t_i < t_{i+1})$ and (iii) for all $i \in I_2$, $(op_i = 1) \vee (V_i = V_{i+1})$. Note that the conditions on the sets I_1 and I_2 ensures monotonicity. Moreover, it can be seen that load-store consistency requirements are satisfied for all $i \in I_1$ (as $A_i \neq A_{i+1}$). Similarly, load-store consistency also holds for all $i \in I_2$. It remains to exhibit the sets and show that they satisfy the above invariants using polynomials, as in the following lemma:

Lemma 5.15 *Let $\widetilde{\mathsf{tr}}$ be a polynomial encoding of transcript tr of size k, given by polynomials $t(X)$, $op(X)$, $A(X)$ and $V(X)$, with index set $[N]$. Then assuming $kN < |\mathbb{F}|$, tr is address ordered and satisfies load-store consistency if and only if there exist polynomials $Z_1, Z_2, \delta_T, \delta_A$ such that the following hold:*

$$A(\omega X) - A(X) - \delta_A(X) = 0 \ mod \ \mathbb{Z}_1(X) \tag{C1}$$
$$A(\omega X) - A(X) = 0 \ mod \ Z_2(X) \tag{C2}$$
$$t(\omega X) - t(X) - \delta_T(X) = 0 \ mod \ Z_2(X) \tag{C3}$$
$$(op(X) - 1)(V(\omega X) - V(X)) = 0 \ mod \ Z_2(X) \tag{C4}$$
$$Z_1(X) \cdot Z_2(X) \cdot (X - 1) = \mathbb{Z}_{\mathbb{K}}(X) \tag{C5}$$
$$1 \leq A(\omega^i) \leq N \tag{C6}$$
$$1 \leq t(\omega^i) \leq N, 1 \leq \delta_A(\omega^i) \leq N, \ 1 \leq \delta_T(\omega^i) \leq N \ for \ i \in [k] \tag{C7}$$

Proof: Suppose there exist polynomials $Z_1(X), Z_2(X), \delta_T(X)$ and $\delta_A(X)$ satisfying above identities. From Equation (C5), we conclude that their exist sets I_1, I_2 with $I_1 \uplus I_2 = [k-1]$ such that $Z_b(X)$, $b \in \{1, 2\}$ is the vanishing polynomial of the set $\{\omega^i : i \in I_b\}$. We now note that the following are true for $i \in I_1$:

– $A(\omega^{i+1}) - A(\omega^i) = \delta_A(\omega^i)$. Since $1 \leq \delta_A(\omega^i) \leq N$, this ensures $A_i < A_{i+1}$ for the vector $\boldsymbol{A}$ encoded by $A(X)$. We note that $kN < |\mathbb{F}|$ implies there is no overflow modulo the field characteristic.

Similarly, it can be seen that for $i \in I_2$, we must have (i) $A_i = A_{i+1} \wedge t_i < t_{i+1}$ and (ii) $op_i = 1 \vee V_i = V_{i+1}$. Together these imply that the encoded transcript is address-ordered. □

Protocols facilitating the checks mentioned in Lemma 5.14 and Lemma 5.15 are presented in Figure 5.5 and 5.6 respectively.

5.7 Succinct Argument for Verifiable RAM

The polynomial encodings in the previous section can be used to construct a polynomial protocol for checking the membership in the language $\mathsf{LRAM}_{\mathcal{I},m,m}$ for $m \in \mathbb{N}$. The polynomial protocol can be subsequently compiled into a succinct argument using an extractable polynomial commitment scheme. In this section, we use KZG polynomial commitment scheme to obtain a succinct argument for checking membership in $\mathsf{LRAM}_{\mathcal{I},m,m}$ in the Algebraic Group Model (AGM). At a high level, to prove $(\boldsymbol{T}, \boldsymbol{o}, \boldsymbol{T}') \in \mathsf{LRAM}_{\mathcal{I},m,m}$, the prover constructs time ordered transcript tr and then permutes it to obtain the address sorted transcript tr^*. It then sends the polynomial encodings of $\boldsymbol{T}, \boldsymbol{o}, \boldsymbol{T}', \mathsf{tr}$ and tr^* to the verifier, who verifies that:

1. The time ordered transcript is correctly constructed, i.e, $\mathsf{tr} = \mathsf{TimeTr}(\boldsymbol{T}, \boldsymbol{o}, \boldsymbol{T}')$. This is achieved using the protocol in Figure 5.7.

2. The transcript tr^* is a permutation of the transcript tr, i.e, $\mathsf{tr}^* = \sigma(\mathsf{tr})$ for some permutation σ of $[k]$. The protocol for this check appears in Figure 5.8.

3. The transcript tr^* is address ordered and satisfies load-store consistency. We describe the protocol to check this property of transcripts in Figure 5.6.

We check above conditions over commitments. Let srs denote a KZG setup over a bilinear group, with prime order groups $\mathbb{G}_1, \mathbb{G}_2$ and $\mathbb{G}_T$. We canonically commit to RAM, operation sequences and transcripts by committing to their polynomial encodings. Commitment of an encoding represented as tuple of polynomials is simply the tuple consisting of commitments of the component polynomials. We now define the relation $\mathcal{R}^{\mathsf{LRAM}}_{\mathsf{srs},m}$ below, and present a succinct argument for the same.

Definition 5.12 *Let* $\mathcal{R}^{\mathsf{LRAM}}_{\mathsf{srs},m}$ *consist of tuples* $((c_T, c_o, c'_T), (\boldsymbol{T}, \boldsymbol{o}, \boldsymbol{T}'))$ *where* $c_T = \mathsf{KZG.Commit}(\mathsf{srs}, \widetilde{T})$, $c'_T = \mathsf{KZG.Commit}(\mathsf{srs}, \widetilde{T'})$, $c_o = \mathsf{KZG.Commit}(\mathsf{srs}, \widetilde{O})$ *commit to* $\boldsymbol{T}$, $\boldsymbol{T}'$ *and* $\boldsymbol{o}$ *with* $(\boldsymbol{T}, \boldsymbol{o}, \boldsymbol{T}') \in \mathsf{LRAM}_{\mathcal{I},m,m}$.

In the above definition we have $c_T = (c_a, c_v)$ where c_a and c_v are KZG commitments to polynomials $a(X)$ and $v(X)$ in the encoding $\widetilde{T} = (a(X), v(X))$. Similarly we parse $c'_T = (c'_a, c'_v)$ and $c_o = (\bar{c}_{op}, \bar{c}_a, \bar{c}_v)$ (see Section 5.6.1 for polynomial encodings). For proving relation 5.12, prover's input consists of initial RAM state $\boldsymbol{T} = (\boldsymbol{a}, \boldsymbol{v})$, final RAM state $\boldsymbol{T}' = (\boldsymbol{a'}, \boldsymbol{v'})$, operation sequence $\boldsymbol{o} = (o_1, \ldots, o_m)$ with $o_i = (\bar{op}_i, \bar{a}_i, \bar{v}_i)$, time-ordered transcript $\mathsf{tr} = (\boldsymbol{t}, \boldsymbol{op}, \boldsymbol{A}, \boldsymbol{V})$

Common Input: Commitments c_a, c_b, c_c, c_v, and $[Z]_1$ (to the polynomial $Z(X) = \prod_{i=1}^{m}(X - \omega^i)$).
Prover's Input: Vectors $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c} \in \mathbb{F}^m$ and $\boldsymbol{v} \in \mathbb{F}^k$.

1. $\mathcal{V}$ samples $\gamma \leftarrow_R \mathbb{F}$, and sends γ to $\mathcal{P}$.
2. $\mathcal{P}$ computes the following:
   - $h(X) = a(X) + \gamma b(X) + \gamma^2 c(X)$.
   - $Q(X) = (h(X^3) - v(X) - \gamma v(\omega^m X) - \gamma^2 v(\omega^{2m} X))/Z(X)$.
3. $\mathcal{P}$ computes and sends the commitment $[Q]_1 = [Q(X)]_1$ to $\mathcal{V}$.
4. $\mathcal{V}$ samples $s \leftarrow_R \mathbb{F}$, and sends s to $\mathcal{P}$.
5. $\mathcal{P}$ computes and sends the following evaluations to $\mathcal{V}$: $\langle v \rangle_s = v(s)$, $\langle v \rangle_{\omega^m s} = v(\omega^m s)$, $\langle v \rangle_{\omega^{2m} s} = v(\omega^{2m} s)$, $\langle h \rangle_{s^3} = h(s^3)$, $\langle Q \rangle_s = Q(s)$ and $\langle Z \rangle_s = Z(s)$.
6. $\mathcal{V}$ samples $r \leftarrow_R \mathbb{F}$ and sends r to $\mathcal{P}$.
7. $\mathcal{P}$ computes the following KZG proofs:
   - $\Pi_v = \mathsf{KZG.Prove}(\mathsf{srs}, v, (s, \omega^m s, \omega^{2m} s))$.
   - $\Pi_h = \mathsf{KZG.Prove}(\mathsf{srs}, h, s^3)$.
   - $\Pi_f = \mathsf{KZG.Prove}(\mathsf{srs}, f, s)$ where $f(X) = Z(X) + rQ(X)$.
8. $\mathcal{P}$ sends Π_v, Π_h and Π_f to $\mathcal{V}$.
9. $\mathcal{V}$ computes commitments $[h]_1$ and $[f]_1$. [a]
10. $\mathcal{V}$ checks:
    - $\mathsf{KZG.Verify}(\mathsf{srs}, [v]_1, \boldsymbol{e}_v, \boldsymbol{p}_v, \Pi_v)$ where $\boldsymbol{p}_v = (s, \omega^m s, \omega^{2m} s)$ and $\boldsymbol{e}_v = (\langle v \rangle_s, \langle v \rangle_{\omega^m s}, \langle v \rangle_{\omega^{2m} s})$.
    - $\mathsf{KZG.Verify}(\mathsf{srs}, [h]_1, \langle h \rangle_{s^3}, s^3, \Pi_h)$.
    - $\mathsf{KZG.Verify}(\mathsf{srs}, [f]_1, \langle Z \rangle_s + r\langle Q \rangle_s, s, \Pi_f)$.
    - $\langle Q \rangle_s \cdot \langle Z \rangle_s = \langle h \rangle_{s^3} - \langle v \rangle_s - \gamma \langle v \rangle_{\omega^m s} - \gamma^2 \langle v \rangle_{\omega^{2m} s}$.
11. $\mathcal{V}$ outputs accept if all the above checks succeed, else it outputs reject.

[a] This can be done locally by leveraging the linearity of the operation

Figure 5.5: Check concatenation over committed vectors.

Common Input: Commitments c_t, c_{op}, c_A and c_V to $\boldsymbol{t}, \boldsymbol{op}, \boldsymbol{A}$ and $\boldsymbol{V}$ constituting the transcript tr.
Prover's Input: $\mathsf{tr} = (\boldsymbol{t}, \boldsymbol{op}, \boldsymbol{A}, \boldsymbol{V})$ and its polynomial encoding $\widetilde{\mathsf{tr}} = (t(X), op(X), A(X), V(X))$.

1. Prover determines sets I_1, I_2 as described in Section 5.6.2.
2. Prover computes polynomials $Z_1(X)$, $Z_2(X)$, $\delta_T(X)$, $\delta_A(X)$.
3. $\mathcal{P}$ sends $[Z_1(X)]_1$, $[Z_2(X)]_1$, $[\delta_T(X)]_1$, $[\delta_A(X)]_1$ to $\mathcal{V}$.
4. $\mathcal{V}$ samples $\gamma \leftarrow_R \mathbb{F}$, and sends γ to $\mathcal{P}$.
5. $\mathcal{P}$ computes the following polynomials:
   - $Q_1(X) = (A(\omega X) - A(X) - \delta_A(X))/\mathbb{Z}_1(X)$.
   - $Q_2(X) = [(A(\omega X) - A(X)) + \gamma(t(\omega X) - t(X) - \delta_T(X)) + \gamma^2(op(X) - 1)(V(\omega X) - V(X))]/Z_2(X)$
6. $\mathcal{P}$ sends commitments $[Q_1(X)]_1$, $[Q_2(X)]_1$ to $\mathcal{V}$.
7. $\mathcal{V}$ sends $s \leftarrow_R \mathbb{F}$, and sends s to $\mathcal{P}$.
8. $\mathcal{P}$ computes and sends the following evaluations to $\mathcal{V}$: $\langle A\rangle_s = A(s)$, $\langle A\rangle_{\omega s} = A(\omega s)$, $\langle \delta_A\rangle_s = \delta_A(s)$, $\langle t\rangle_s = t(s)$, $\langle t\rangle_{\omega s} = t(\omega s)$, $\langle \delta_T\rangle_s = \delta_T(s)$, $\langle op\rangle_s = op(s)$, $\langle V\rangle_s = V(s)$, $\langle V\rangle_{\omega s} = V(\omega s)$, $\langle Q_1\rangle_s = Q_1(s)$, $\langle Q_2\rangle_s = Q_2(s)$, $\langle Z_1\rangle_s = Z_1(s)$, $\langle Z_2\rangle_s = Z_2(s)$.
9. $\mathcal{V}$ checks the following:
   - $\langle Q_1\rangle_s \cdot \langle Z_1\rangle_s = (\langle A\rangle_{\omega s} - \langle A\rangle_s - \langle \delta_A\rangle_s)$.
   - $\langle Q_2\rangle_s \cdot \langle Z_2\rangle_s = (\langle A\rangle_{\omega s} - \langle A\rangle_s) + \gamma(\langle t\rangle_{\omega s} - \langle t\rangle_s - \langle \delta_T\rangle_s) + \gamma^2(\langle op\rangle_s - 1)(\langle V\rangle_{\omega s} - \langle V\rangle_s)$.
   - $\langle Z_1\rangle_s \cdot \langle Z_2\rangle_s = s^{3m} - 1$.
10. $\mathcal{V}$ samples $r_1, r_2 \leftarrow_R \mathbb{F}$ and sends r_1, r_2 to $\mathcal{P}$.
11. $\mathcal{P}$ computes the following:
    - $\Phi_{ws}(X) = A(X) + r_1 t(X) + r_1^2 V(X)$.
    - $\Phi_s(X) = A(X) + r_2\delta_A(X) + r_2^2 t(X) + r_2^3\delta_T(X) + r_2^4 op(X) + r_2^5 V(X) + r_2^6 Q_1(X) + r_2^7 Q_2(x) + r_2^8 Z_1(X) + r_2^9 Z_2(X)$.
    - $\Pi_{\omega s} = \mathsf{KZG.Prove}(\mathsf{srs}, \Phi_{\omega s}(X), \omega s)$.
    - $\Pi_s = \mathsf{KZG.Prove}(\mathsf{srs}, \Phi_s(X), s)$.
12. $\mathcal{P}$ sends $\Pi_{\omega s}, \Pi_s$ to $\mathcal{V}$.

Figure 5.6: Check that transcript is address ordered and load-store consistent.

14. $\mathcal{V}$ computes the following:

    - $[\Phi_{\omega s}(X)]_1 = c_A + r_1 c_t + r_1^2 c_V$.
    - $[\Phi_s(X)]_1 = c_A + r_2[\delta_A(X)]_1 + r_2^2 c_t + r_2^3[\delta_T(X)]_1 + r_2^4 c_{op} + r_2^5 c_V + r_2^6[Q_1(X)]_1 + r_2^7[Q_2(X)]_1 + r_2^8[Z_1(X)]_1 + r_2^9[Z_2(X)]_1$.
    - $V_{ws} = \langle A \rangle_{\omega s} + r_1 \langle t \rangle_{\omega s} + r_1^2 \langle V \rangle_{\omega s}$.
    - $V_s = \langle A \rangle_s + r_2 \langle \delta_A \rangle_s + r_2^2 \langle t \rangle_s + r_2^3 \langle \delta_T \rangle_s + r_2^4 \langle op \rangle_s + r_2^5 \langle V \rangle_s + r_2^6 \langle Q_1 \rangle_s + r_2^7 \langle Q_2 \rangle_s + r_2^8 \langle Z_1 \rangle_s + r_2^9 \langle Z_2 \rangle_s$.

15. $\mathcal{V}$ checks the following:

    - $\mathsf{KZG.Verify}(\mathsf{srs}, [\Phi_{ws}]_1, V_{ws}, \omega s, \Pi_{\omega s})$.
    - $\mathsf{KZG.Verify}(\mathsf{srs}, [\Phi_s]_1, V_s, s, \Pi_s)$.

16. $\mathcal{P}$ and $\mathcal{V}$ invoke sub-vector arguments $(\mathcal{P}_{\mathsf{sv}}, \mathcal{V}_{\mathsf{sv}})$ (eg. [56]) to prove that (srs, c_A, c_I), (srs, c_t, c_I), $(\mathsf{srs}, [\delta_A(X)]_1, c_I)$ and $(\mathsf{srs}, [\delta_T(X)]_1, c_I)$ are in $\mathcal{R}^{\mathsf{subvec}}_{\mathsf{srs},N,m}$.

17. $\mathcal{V}$ outputs accept if all checks succeed and the sub-vector arguments outputs accept. Otherwise it outputs reject.

Figure 5.6: Check that transcript is address ordered and load-store consistent.

Component	Protocol	Prover Complexity	Verifier Complexity	Communication Complexity
Concatenation of transcripts	Fig 5.7	$O(m \log m)\,\mathbb{F}$ $O(m)\,\mathbb{G}_1$	$2P$	$4\mathbb{G}_1$, $6\mathbb{F}$
Permutation of transcripts	Fig 5.8	$O(m \log m)\,\mathbb{F}$ $O(m)\,\mathbb{G}_1$	$2P$	$4\mathbb{G}_1$, $5\mathbb{F}$
Memory consistency & Address ordering of transcripts	Fig 5.6	$O(m \log m)\,\mathbb{F}$ $O(m)\,\mathbb{G}_1$	$6P$	$20\mathbb{G}_1$,$19\mathbb{F}$
Polynomial Protocol for RAM	Fig 5.9	$O(m \log m)\,\mathbb{F}$ $O(m)\,\mathbb{G}_1$	$7P$	$36\mathbb{G}_1$, $30\mathbb{F}$

Table 5.2: Efficiency parameters for components of polynomial protocol for RAM. Here m denotes both the size of the RAM and number of operations (the special case we consider). P denotes a pairing evaluation, while $\mathbb{G}_1$ $\mathbb{G}_2$ and $\mathbb{F}$ denote the groups and the scalar field of the bilinear group used for instantiating the protocol.

and address-ordered transcript $\mathsf{tr}^* = (\boldsymbol{t}^*, \boldsymbol{op}^*, \boldsymbol{A}^*, \boldsymbol{V}^*)$ obtained from tr using a permutation $\sigma : [k] \to [k]$. Verifier's input consists of the commitments c_T, c_o and c'_T as described above.

The prover starts the protocol by sending commitments c_{tr} and c^*_{tr} to the transcripts tr and tr^* respectively. To show that tr is correctly formed, the prover needs to prove the concatenations: (i) $\boldsymbol{op} = 0^m||(\bar{op}_1, \ldots, \bar{op}_m)||0^m$, (ii) $\boldsymbol{A} = \boldsymbol{a}||(\bar{a}_1, \ldots, \bar{a}_m)||\boldsymbol{a'}$ and (iii) $\boldsymbol{V} = \boldsymbol{v}||(\bar{v}_1, \ldots, \bar{v}_m)||\boldsymbol{v'}$. Note that the time-stamp column $\boldsymbol{t}$ is implicitly assumed to be $(1, \ldots, k)$. The verifier checks the concatenations using Lemma 5.14. It uses a random challenge β to reduce the three concatenations to one concatenation, and uses another challenge γ to reduce the three polynomial checks in Lemma 5.14 to a single check. The complete polynomial protocol is detailed in Figure 5.7.

Next, we show a polynomial protocol for proving that the transcript tr^* is a permutation of the transcript tr. We first recall the permutation argument for vectors from [62].

Lemma 5.16 (Permutation Check [62]) *Let $f(X), g(X)$ be polynomials in $\mathbb{F}[X]$. Then, the vectors $\boldsymbol{f}, \boldsymbol{g} \in \mathbb{F}^k$ encoded by the polynomials are permutations of each other if and only if with overwhelming probability over the choice of $\alpha \leftarrow_R \mathbb{F}$, there exists a polynomial $z(X)$ satisfying the polynomial constraints:*

$$\lambda_1(X)(z(X) - 1) = 0 \ mod\ Z_{\mathbb{K}}(X) \tag{B1}$$

$$(\alpha - g(X))z(\omega X) = (\alpha - f(X))z(X) \ mod\ Z_{\mathbb{K}}(X) \tag{B2}$$

The polynomial protocol in Figure 5.8 essentially invokes the above argument on the random

Common Input: Commitments $c_T = (c_a, c_v)$, $c_o = (\bar{c}_{op}, \bar{c}_a, \bar{c}_v)$, $c'_T = (c'_a, c'_v)$ and $c_{\mathsf{tr}} = (c_t, c_{op}, c_A, c_V)$ to $\boldsymbol{T}, \boldsymbol{o}, \boldsymbol{T}'$ and tr (which is supposed to be the time ordered transcript) respectively. Commitment $[Z(X)]_1$ to the polynomial $Z(X) = \prod_{i=1}^{m}(X - \omega^i)$.
Prover's Input: $\mathsf{tr}, \boldsymbol{T}, \boldsymbol{T}', \boldsymbol{o}$ and their polynomial encodings, $Z(X)$.

1. $\mathcal{V}$ samples $\beta, \gamma \leftarrow_R \mathbb{F}$ and sends β, γ to $\mathcal{P}$.
2. $\mathcal{P}$ computes the following:
   - $G_1(X) = a(X) + \beta v(X)$, $G_2(X) = \bar{a}(X) + \beta \bar{v}(X) + \beta^2 \bar{op}(X)$
   - $G_3(X) = a^*(X) + \beta v^*(X)$, $G(X) = A(X) + \beta V(X) + \beta^2 op(X)$
   - $H(X) = G_1(X) + \gamma G_2(X) + \gamma^2 G_3(X)$
   - $Q(X) = [(H(X^3) - G(X) - \gamma G(\omega^m X) - \gamma^2 G(\omega^{2m} X))]/Z(X)$
3. $\mathcal{P}$ sends commitment $[Q]_1$ of $Q(X)$ to $\mathcal{V}$.
4. $\mathcal{V}$ samples $s \leftarrow_R \mathbb{F}$ and sends s to $\mathcal{P}$.
5. $\mathcal{P}$ computes and send the following evaluations to $\mathcal{V}$: $\langle G \rangle_s = G(s)$, $\langle G \rangle_{\omega^m s} = G(\omega^m s)$, $\langle G \rangle_{\omega^{2m} s} = G(\omega^{2m} s)$, $\langle H \rangle_{s^3} = H(s^3)$, $\langle Q \rangle_s = Q(s)$ and $\langle Z \rangle_s = Z(s)$.
6. $\mathcal{V}$ samples $r \leftarrow_R \mathbb{F}$ and sends r to $\mathcal{P}$.
7. $\mathcal{P}$ computes and sends the following KZG proofs:
   - $\Pi_G = \mathsf{KZG.Prove}(\mathsf{srs}, G(X), (s, \omega^m s, \omega^{2m} s))$.
   - $\Pi_H = \mathsf{KZG.Prove}(\mathsf{srs}, H(X), s^3)$.
   - $\Pi_F = \mathsf{KZG.Prove}(\mathsf{srs}, F(X), s)$ where $F(X) = Z(X) + rQ(X)$.
8. $\mathcal{V}$ computes $[G(X)]_1, [H(X)]_1$ and $[F(X)]_1$. [a]
9. $\mathcal{V}$ checks the following:
   - $\mathsf{KZG.Verify}(\mathsf{srs}, [G]_1, (\langle G \rangle_s, \langle G \rangle_{\omega^m s}, \langle G \rangle_{\omega^{2m} s}), (s, \omega^m s, \omega^{2m} s), \Pi_G)$.
   - $\mathsf{KZG.Verify}(\mathsf{srs}, [H]_1, \langle H \rangle_{s^3}, s^3, \Pi_H)$.
   - $\mathsf{KZG.Verify}(\mathsf{srs}, [F]_1, \langle Z \rangle_s + r\langle Q \rangle_s, s, \Pi_F)$.
   - $\langle Q \rangle_s \cdot \langle Z \rangle_s = \langle H \rangle_{s^3} - \langle G \rangle_s - \gamma \langle G \rangle_{\omega^m s} - \gamma^2 \langle G \rangle_{\omega^{2m} s}$.
10. $\mathcal{V}$ outputs accept if all the above checks succeeds, otherwise it outputs reject.

[a] This can be done locally by leveraging the linearlity of the operation

Figure 5.7: Check the correctness of time-ordered transcript.

linear combination of the columns of the respective transcripts.

Common Input: Commitments $c_{\mathsf{tr}} = (c_t, c_{op}, c_A, c_V)$ and $c^*_{\mathsf{tr}} = (c^*_t, c^*_{op}, c^*_A, c^*_V)$ of transcripts tr and tr^* respectively.
Prover's Input: Transcripts $\mathsf{tr}, \mathsf{tr}^*$ and their polynomial encodings, permutation σ such that $\mathsf{tr}^* = \sigma(\mathsf{tr})$.

1 $\mathcal{V}$ samples $\alpha, \beta, \chi \leftarrow_R \mathbb{F}$ and sends α, β, χ to $\mathcal{P}$.

2 $\mathcal{P}$ computes the following:

- $f(X) = t(X) + \beta op(X) + \beta^2 A(X) + \beta^3 V(X)$.
- $g(X) = t^*(X) + \beta op^*(X) + \beta^2 A^*(X) + \beta^3 V^*(X)$.

3 $\mathcal{P}$ computes polynomials $z(X), q(X)$ as follows:

- Interpolate polynomial $z(X)$ of degree $k-1$ such that $z(\omega) = 1$ and$z(\omega^{i+1}) = \prod_{j=1}^{i}(\alpha - f(\omega^j))/(\alpha - g(\omega^j))$ for $1 \leq i \leq k-1$.
- $q(X) = ((\alpha - g(X))z(\omega X) - (\alpha - f(X))z(X) + \chi\lambda_1(X)(z(X) - 1))/\mathbb{Z}_{\mathbb{K}}(X)$.

4 $\mathcal{P}$ computes commitments $[z(X)]_1$ and $[q(X)]_1$ to polynomials $z(X)$ and $q(X)$ respectively, and sends $[z(X)]_1, [q(X)]_1$ to $\mathcal{V}$.

5 $\mathcal{V}$ computes commitments $[f]_1, [g]_1$. [a]

6 $\mathcal{V}$ checks that $q(X)Z_{\mathbb{K}}(X) = (\alpha - g(X))z(\omega X) - (\alpha - f(X))z(X) + \chi\lambda_1(X)(z(X) - 1)$ by requesting evaluations and KZG proofs of polynomials f, g, q, z at a random point, say s and evaluation and KZG proof of z at ωs.

7 $\mathcal{V}$ outputs accept if all the checks succeed, else it outputs reject.

[a] This can be done locally by leveraging the linearlity of the operation

Figure 5.8: Check that transcripts are permutations of each other.

Finally, we see that Lemma 5.15 implies a polynomial protocol to check that the transcript tr^* is address ordered and satisfies load-store consistency, which essentially involves the prover identifying sets I_1, I_2 as described in Section 5.6.2 and sending auxiliary polynomials $Z_1(X), Z_2(X), \delta^*_A(X)$ and $\delta^*_T(X)$ to the verifier. The verifier then checks the identities (C1)-(C6) in Lemma 5.15. The range checks in (C7) can be checked using polynomial protocols in sub-vector lookup arguments such as [98, 56, 43, 113]. The protocol (compiled using KZG commitments in AGM) can be found in Figure 5.6. The overall protocol for $\mathcal{R}^{\mathsf{LRAM}}_{\mathsf{srs},m}$ which combines invokes protocols in Figures 5.7,5.8 and 5.6 as sub-protocols is presented in Figure 5.9.

Common Input: Commitments $c_T = (c_a, c_v)$, $c_o = (\bar{c}_{op}, \bar{c}_a, \bar{c}_v)$, $c'_T = (c'_a, c'_v)$.
Prover's Input: $\boldsymbol{T}, \boldsymbol{T}', \boldsymbol{o}$ and their polynomial encodings.

1. $\mathcal{P}$ computes the following:
   - tr (time ordered transcript corresponding to $\boldsymbol{T}, \boldsymbol{o}, \boldsymbol{T}'$), its polynomial encoding, and its commitment $c_{\mathsf{tr}} = (c_t, c_{op}, c_A, c_V)$.
   - $Z(X) = \prod_{i=1}^{m}(X - \omega^i)$ and its commitment $[Z(X)]_1$.
2. $\mathcal{P}$ sends $c_{\mathsf{tr}} = (c_t, c_{op}, c_A, c_V)$ and $[Z(X)]_1$ to $\mathcal{V}$.
3. $\mathcal{P}$ and $\mathcal{V}$ run the protocol for checking correctness of time ordered transcript (Figure 5.7).
4. $\mathcal{P}$ computes the address ordered transcript tr^* (along with its polynomial encoding) and the permutation σ from the time ordered transcript tr, such that $\mathsf{tr}^* = \sigma(\mathsf{tr})$.
5. $\mathcal{P}$ computes the commitment $c^*_{\mathsf{tr}} = (c^*_t, c^*_{op}, c^*_A, c^*_V)$ of tr^* and sends c^*_{tr}.
6. $\mathcal{P}$ and $\mathcal{V}$ run the protocol for checking that the two transcripts are permutations of each other (Figure 5.8).
7. $\mathcal{P}$ and $\mathcal{V}$ run the protocol for checking the constraints given in Lemma 5.15 (Figure 5.6.)
8. $\mathcal{V}$ outputs accept if all the three sub-protocols lead to accept, else it outputs reject.

Figure 5.9: Overall protocol for the relation $\mathcal{R}^{\mathsf{LRAM}}_{\mathsf{srs},m}$

Efficiency. We provide a break-up of costs incurred by different components involved in construction of RAM based on memory-checking techniques in Table 5.2. To reduce pairing checks we use standard technique of batching pairing checks involving common generators. In addition, to reduce communication, instead of naively invoking four instances of sub-vector argument in Step 15 of the protocol in Figure 5.6, we concatenate the four vectors using a variant of protocol for concatenation of vectors in Figure 5.5, and then use the sub-vector argument to show that the concatenated vector is a sub-vector of the vector $(1, \ldots, N)$. For CQ [56] based instantiation, this reduces the total communication of this check from $4 \times (8\mathbb{G}_1 + 3\mathbb{F})$ to $(4\mathbb{G}_1 + 6\mathbb{F}) + (8\mathbb{G}_1 + 3\mathbb{F})$, a saving of $\approx 20\mathbb{G}_1$. The reported overheads in Table 5.2 take into account such optimizations.

Chapter 6

Conclusion

This thesis presents advancements in the efficiency of some theoretical primitives in the realm of zero-knowledge proofs (ZKP) and explores some of their applications in real-life scenarios. The dimensions of efficiency in the context of ZKPs consist of proof size, round complexity, verification complexity and prover complexity.

Succinct Verification. To start with, we discussed the well-understood theoretical primitive of sigma protocols in the literature of ZKPs which has attractive real-world applications (eg. blockchain). Sigma protocols are 3-round public-coin *proof of knowledge* protocols that have linear proof size, verification complexity, and prover complexity. Attema and Cramer [8], by casting Bulletproofs [36] in the framework of sigma protocols, provided compressed sigma protocols with logarithmic proof size, but still incur linear verification. They also incur logarithmic round complexity - which is not prohibitive since these protocols are public-coin and can be made non-interactive in the Random Oracle Model by using Fiat-Shamir transformation [59].

To ensure efficient verification, we constructed a compressed sigma protocol (CSP) that has logarithmic proof size and logarithmic verification complexity by moving from transparent setup to updatable setup, which only requires one honest update during the setup phase to provide security guarantees against malicious prover strategies. We first constructed CSP for inner-product argument under discrete log assumption, using which we then provided CSP for arithmetic circuit satisfiability with logarithmic proof size and logarithmic verification complexity. Additionally, we constructed CSP for opening homomorphism in the designated verifier setting with logarithmic proof size and logarithmic verification complexity.

Distributed Proofs. Next, we looked at the usage of ZKPs to enable input authentication in secure multiparty computation (MPC) based on linear secret-sharing, and realized the need for distributed ZKPs to enhance efficiency and security while maintaining privacy. In particular,

we put forward a notion of *distributed proof of knowledge* (DPoK) that enables a prover to distribute the proof generation to a set of *workers* holding the shares of the input, such that (i) workers do not require any private interaction among each other, and (ii) interaction with the verifier is over the broadcast channel where the verifier is public-coin (which helps us achieve public verifiability). We also considered *robustness* in these DPoKs, which ensured security even in the presence of dishonest usage of shares by workers during proof generation, and referred to the protocols with such guarantees as *robust DPoKs*.

We presented constructions of DPoK (and robust DPoKs) for discrete log relation and DPoK for algebraic signature schemes like BBS+ [29, 41] and PS [97]. We also provided constructions of round efficient versions of these DPoKs that are secure in the Random Oracle Model. Using our DPoKs for algebraic signature schemes, we provided a compiler that can lift any threshold linear secret-sharing based honest majority MPC protocol to also have input authentication, while incurring negligible overhead over the underlying MPC.

Lookup Arguments. Finally, we discussed the primitive of lookup arguments that enables us to prove that the vector $\mathbf{S}$ of size m is 'looking up' elements in $\mathbf{T}$ of size N in the indices specified by the m-size vector $\boldsymbol{a} \subset [N]$, i.e. $S_i = T_{a_i}$ for all $i \in m$, where $m << N$. We referred to this class of protocols as *committed index lookup arguments*. Recent works in lookup arguments present improved efficiency in the preprocessing paradigm, where the heavy computation is deferred to the offline phase to make the online phase faster. We removed the rigid dependency of the online phase on the table-dependent parameters computed in the offline phase, and presented *updatable lookup arguments* that enables us to provide efficient proofs for $\mathbf{S} \subset \mathbf{T}'$ when $\mathbf{T}'$ is within a certain Hamming distance of the preprocessed table $\mathbf{T}$.

Using our *updatable lookup argument*, we then presented our *batching-efficient RAM* that enables us to prove that a RAM of size N has undergone m updates (i.e. read/write operations) with constant communication complexity, constant verification complexity, and sublinear (in N) prover complexity. This has applications in providing efficient rollups in blockchain by offloading expensive computation to L2 layer and verifying the off-chain computation.

Open Questions. This thesis explores the theoretical ZKP primitives in the context of various dimensions of efficiency, and raises interesting questions that offer significant scope for future exploration. We briefly state some of the potential directions below.

The techniques of *compressed sigma protocols* (CSP) have been extended to achieve logarithmic proof size for various relations [9, 10, 11]. Exploration of our technique [53] of achieving succinct verification in the compression framework in the context of CSP for lattices [10] or k-out-of-n partial knowledge [9] remains open. Additionally, there is scope to further improve our

construction in [53] that achieves $O(\log n)$ communication and verification complexity, to attain even better communication, verification and/or prover complexities. While our *inner product argument* in [53] considers an updatable setup and relies solely on the discrete log assumption, follow-up works have achieved improved verifier (cubic root) in the transparent setup under the same assumption [82], and the pursuit of better complexities in the transparent setup under the discrete log assumption remains open.

In our work [55], the proof generation for proof of knowledge of discrete log relation (and some algebraic signature schemes [29, 97]) is distributed. Achieving distributed proof of knowledge for interesting relations like arithmetic circuit satisfiability or post-quantum signature schemes still remains open.

Furthermore, although delegation of proof-generation has been explored in recent works for general relations like arithmetic circuit satisfiability [46], it would be interesting to explore delegation of proof-generation for constructions of *lookup arguments* and *batching-efficient RAM* that admit algebraic verification (for protocols based on KZG polynomial scheme [77]) and currently incur high prover complexity (whereas the proof size and verification complexity are constant).

Post-Quantum Vulnerability. The cryptographic constructions discussed throughout this thesis are based on discrete-logarithm based assumptions and bilinear pairings, since our primary concern was improving efficiency of the prior works under similar assumptions. The hardness assumptions considered are extensively studied in the classical setting, and are known to be vulnerable to attacks from quantum computers. Achieving similar efficiency using post-quantum secure hardness assumptions (eg. based on lattices) is a challenging and interesting future direction.

In particular, we can achieve a postquantum secure version of our work on compressed sigma protocols (Chapter 3) by transitioning to lattice-based constructions. As a promising step in this direction, a recent work of Attema et al. [10] already achieves succinct proof size for compressed sigma protocols for lattices, albeit with linear verification complexity. Achieving succinct verification complexity with the existing succinct proof size remains an interesting open question. This would require novel post-quantum secure techniques, since our work relies on the symmetric structure of bilinear pairings.

The construction of distributed proof of knowledge (DPoK) in the next chapter (Chapter 4) extends the notion of compressed sigma protocol to support distributed proof generation, and similarly relies on the assumptions of discrete logarithm (as well as bilinear pairings, to prove knowledge of BBS+ and PS signatures). The primary application of our DPoK is considered in 'input authentication in MPC using BBS+/PS signatures', where the signature schemes under

consideration are not resistant to attacks from post-quantum computers. The primary aim of our work was to improve the overhead in attaining input authentication for MPC, without relying on MPC-specific techniques, and we leverage the efficiency of classical signature schemes like BBS+/PS[1]. In a post-quantum secure version of this application, we would need to consider a post-quantum secure MPC and post-quantum signature scheme.

The recent works on lookup arguments primarily rely on the KZG commitment scheme and the relevant prior works on batching-efficient RAM primarily rely on RSA groups, which makes them vulnerable to post-quantum attacks. With a primary focus on efficiency, our work in Chapter 5 aims to (a) improve the efficiency of batching-efficient RAM to incur only sublinear dependence of the prover complexity on the RAM size, and (b) support efficient lookups on tables undergoing updates, both in the classical setting. As a promising step in the direction of obtaining a post-quantum secure construction, a recent work on *power of polynomial preprocessing* [44] provides lookup arguments relying only on a black-box usage of vector commitments and generic assumptions like collision-resistant hash function, where using a post-quantum vector commitment scheme would yield a post-quantum lookup argument. Further achieving post-quantum updatable lookup arguments and subsequently achieving post-quantum batching-efficient RAM are interesting open problems for future research.

[1]The RFC draft [87] signifies the standardization efforts for using BBS+ signatures in verifiable credentials.

Bibliography

[1] URL https://www.zellic.io/blog/zk-friendly-hash-functions. 20

[2] Masayuki Abe, Georg Fuchsbauer, Jens Groth, Kristiyan Haralambiev, and Miyako Ohkubo. Structure-preserving signatures and commitments to group elements. In Tal Rabin, editor, *CRYPTO 2010*, volume 6223 of *LNCS*, pages 209–236. Springer, Berlin, Heidelberg, August 2010. doi: 10.1007/978-3-642-14623-7_12. 57

[3] Martin R. Albrecht, Lorenzo Grassi, Christian Rechberger, Arnab Roy, and Tyge Tiessen. MiMC: Efficient encryption and cryptographic hashing with minimal multiplicative complexity. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *ASIACRYPT 2016, Part I*, volume 10031 of *LNCS*, pages 191–219. Springer, Berlin, Heidelberg, December 2016. doi: 10.1007/978-3-662-53887-6_7. 19

[4] Scott Ames, Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. Ligero: Lightweight sublinear arguments without a trusted setup. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 2087–2104. ACM Press, October / November 2017. doi: 10.1145/3133956.3134104. 79, 100

[5] Diego F. Aranha, Anders P. K. Dalskov, Daniel Escudero, and Claudio Orlandi. Improved threshold signatures, proactive secret sharing, and input certification from LSS isomorphisms. In Patrick Longa and Carla Ràfols, editors, *LATINCRYPT 2021*, volume 12912, pages 382–404, 2021. 6, 78, 81, 84, 85, 86

[6] Arasu Arun, Chaya Ganesh, Satya Lokam, Tushar Mopuri, and Sriram Sridhar. Dew: A transparent constant-sized polynomial commitment scheme. In *Public-Key Cryptography – PKC 2023: 26th IACR International Conference on Practice and Theory of Public-Key Cryptography, Atlanta, GA, USA, May 7–10, 2023, Proceedings, Part II*, page 542–571, Berlin, Heidelberg, 2023. Springer-Verlag. ISBN 978-3-031-31370-7. doi:

10.1007/978-3-031-31371-4_19. URL https://doi.org/10.1007/978-3-031-31371-4_19. 19

[7] Gilad Asharov and Yehuda Lindell. A full proof of the BGW protocol for perfectly secure multiparty computation. *Journal of Cryptology*, 30(1):58–151, January 2017. doi: 10.1007/s00145-015-9214-4. 90

[8] Thomas Attema and Ronald Cramer. Compressed Σ-protocol theory and practical application to plug & play secure algorithmics. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part III*, volume 12172 of *LNCS*, pages 513–543. Springer, Cham, August 2020. doi: 10.1007/978-3-030-56877-1_18. 4, 12, 15, 16, 20, 21, 28, 30, 38, 39, 42, 50, 79, 83, 99, 103, 109, 119, 124, 126, 193

[9] Thomas Attema, Ronald Cramer, and Serge Fehr. Compressing proofs of k-out-of-n partial knowledge. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part IV*, volume 12828 of *LNCS*, pages 65–91, Virtual Event, August 2021. Springer, Cham. doi: 10.1007/978-3-030-84259-8_3. 4, 70, 194

[10] Thomas Attema, Ronald Cramer, and Lisa Kohl. A compressed Σ-protocol theory for lattices. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part II*, volume 12826 of *LNCS*, pages 549–579, Virtual Event, August 2021. Springer, Cham. doi: 10.1007/978-3-030-84245-1_19. 4, 16, 194, 195

[11] Thomas Attema, Ronald Cramer, and Matthieu Rambaud. Compressed Σ-protocols for bilinear group arithmetic circuits and application to logarithmic transparent threshold signatures. In Mehdi Tibouchi and Huaxiong Wang, editors, *ASIACRYPT 2021, Part IV*, volume 13093 of *LNCS*, pages 526–556. Springer, Cham, December 2021. doi: 10.1007/978-3-030-92068-5_18. ix, 4, 16, 18, 19, 21, 23, 57, 58, 66, 70, 71, 194

[12] Thomas Attema, Serge Fehr, and Michael Klooß. Fiat–shamir transformation of multi-round interactive proofs (extended version). *J. Cryptol.*, 36(4), aug 2023. ISSN 0933-2790. doi: 10.1007/s00145-023-09478-y. URL https://doi.org/10.1007/s00145-023-09478-y. 126, 127

[13] Man Ho Au, Willy Susilo, and Yi Mu. Constant-size dynamic k-TAA. In Roberto De Prisco and Moti Yung, editors, *SCN 06*, volume 4116 of *LNCS*, pages 111–125. Springer, Berlin, Heidelberg, September 2006. doi: 10.1007/11832072_8. 79, 80, 84

[14] barryWhiteHat. rollup. https://github.com/barryWhiteHat/roll_up. 141

[15] Carsten Baum. On garbling schemes with and without privacy. In Vassilis Zikas and Roberto De Prisco, editors, *SCN 16*, volume 9841 of *LNCS*, pages 468–485. Springer, Cham, August / September 2016. doi: 10.1007/978-3-319-44618-9_25. 84

[16] Carsten Baum, Robin Jadoul, Emmanuela Orsini, Peter Scholl, and Nigel P. Smart. Feta: Efficient threshold designated-verifier zero-knowledge proofs. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, CCS '22, page 293–306, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450394505. doi: 10.1145/3548606.3559354. URL https://doi.org/10.1145/3548606.3559354. 5, 86, 87

[17] Stephanie Bayer and Jens Groth. Efficient zero-knowledge argument for correctness of a shuffle. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 263–280. Springer, Berlin, Heidelberg, April 2012. doi: 10.1007/978-3-642-29011-4_17. 49, 52

[18] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *20th ACM STOC*, pages 1–10. ACM Press, May 1988. doi: 10.1145/62212.62213. 76

[19] Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, Eran Tromer, and Madars Virza. SNARKs for C: Verifying program executions succinctly and in zero knowledge. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 90–108. Springer, Berlin, Heidelberg, August 2013. doi: 10.1007/978-3-642-40084-1_6. 142, 144

[20] Eli Ben Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *2014 IEEE Symposium on Security and Privacy*, pages 459–474, 2014. doi: 10.1109/SP.2014.36. 1

[21] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. Succinct non-interactive zero knowledge for a von neumann architecture. In Kevin Fu and Jaeyeon Jung, editors, *USENIX Security 2014*, pages 781–796. USENIX Association, August 2014. 3, 142, 144

[22] Eli Ben-Sasson, Alessandro Chiesa, Michael Riabzev, Nicholas Spooner, Madars Virza, and Nicholas P. Ward. Aurora: Transparent succinct arguments for R1CS. In Yuval Ishai

and Vincent Rijmen, editors, *EUROCRYPT 2019, Part I*, volume 11476 of *LNCS*, pages 103–128. Springer, Cham, May 2019. doi: 10.1007/978-3-030-17653-2_4. 79, 100

[23] Eli Ben-Sasson, Dan Carmon, Yuval Ishai, Swastik Kopparty, and Shubhangi Saraf. Proximity gaps for reed-solomon codes. In *61st FOCS*, pages 900–909. IEEE Computer Society Press, November 2020. doi: 10.1109/FOCS46700.2020.00088. 100

[24] Fabrice Benhamouda, Tancrède Lepoint, Julian Loss, Michele Orrù, and Mariana Raykova. On the (in)security of ROS. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part I*, volume 12696 of *LNCS*, pages 33–53. Springer, Cham, October 2021. doi: 10.1007/978-3-030-77870-5_2. 83, 88

[25] Nir Bitansky, Alessandro Chiesa, Yuval Ishai, Rafail Ostrovsky, and Omer Paneth. Succinct non-interactive arguments via linear interactive proofs. In Amit Sahai, editor, *TCC 2013*, volume 7785 of *LNCS*, pages 315–333. Springer, Berlin, Heidelberg, March 2013. doi: 10.1007/978-3-642-36594-2_18. 3

[26] Marina Blanton and Fattaneh Bayatbabolghani. Efficient server-aided secure two-party function evaluation with applications to genomic computation. *PoPETs*, 2016(4):144–164, October 2016. doi: 10.1515/popets-2016-0033. 77

[27] Marina Blanton and Myoungin Jeong. Improved signature schemes for secure multi-party computation with certified inputs. In Javier López, Jianying Zhou, and Miguel Soriano, editors, *ESORICS 2018, Part II*, volume 11099 of *LNCS*, pages 438–460. Springer, Cham, September 2018. doi: 10.1007/978-3-319-98989-1_22. 6, 78, 81, 84, 85, 86

[28] Manuel Blum, Paul Feldman, and Silvio Micali. Non-interactive zero-knowledge and its applications (extended abstract). In *20th ACM STOC*, pages 103–112. ACM Press, May 1988. doi: 10.1145/62212.62222. 11

[29] Dan Boneh, Xavier Boyen, and Hovav Shacham. Short group signatures. In Matthew Franklin, editor, *CRYPTO 2004*, volume 3152 of *LNCS*, pages 41–55. Springer, Berlin, Heidelberg, August 2004. doi: 10.1007/978-3-540-28628-8_3. 6, 76, 77, 79, 80, 84, 95, 194, 195

[30] Dan Boneh, Elette Boyle, Henry Corrigan-Gibbs, Niv Gilboa, and Yuval Ishai. Zero-knowledge proofs on secret-shared data via fully linear PCPs. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part III*, volume 11694 of *LNCS*, pages 67–97. Springer, Cham, August 2019. doi: 10.1007/978-3-030-26954-8_3. 78, 84, 87

[31] Dan Boneh, Benedikt Bünz, and Ben Fisch. Batching techniques for accumulators with applications to IOPs and stateless blockchains. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part I*, volume 11692 of *LNCS*, pages 561–586. Springer, Cham, August 2019. doi: 10.1007/978-3-030-26948-7_20. 141

[32] Jonathan Bootle, Andrea Cerulli, Pyrros Chaidos, Jens Groth, and Christophe Petit. Efficient zero-knowledge arguments for arithmetic circuits in the discrete log setting. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 327–357. Springer, Berlin, Heidelberg, May 2016. doi: 10.1007/978-3-662-49896-5_12. 16, 19, 20, 21, 94

[33] Sean Bowe, Ariel Gabizon, and Ian Miers. Scalable multi-party computation for zk-SNARK parameters in the random beacon model. Cryptology ePrint Archive, Report 2017/1050, 2017. URL https://eprint.iacr.org/2017/1050. 152

[34] Benjamin Braun, Ariel J. Feldman, Zuocheng Ren, Srinath Setty, Andrew J. Blumberg, and Michael Walfish. Verifying computations with state (extended version). Cryptology ePrint Archive, Report 2013/356, 2013. URL https://eprint.iacr.org/2013/356. 141

[35] Emmanuel Bresson, Yassine Lakhnech, Laurent Mazaré, and Bogdan Warinschi. A generalization of DDH with applications to protocol analysis and computational soundness. In Alfred Menezes, editor, *CRYPTO 2007*, volume 4622 of *LNCS*, pages 482–499. Springer, Berlin, Heidelberg, August 2007. doi: 10.1007/978-3-540-74143-5_27. 64

[36] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *2018 IEEE symposium on security and privacy (SP)*, pages 315–334. IEEE, 2018. 4, 16, 19, 193

[37] Benedikt Bünz, Ben Fisch, and Alan Szepieniec. Transparent SNARKs from DARK compilers. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part I*, volume 12105 of *LNCS*, pages 677–706. Springer, Cham, May 2020. doi: 10.1007/978-3-030-45721-1_24. 19, 151

[38] Jan Camenisch and Anna Lysyanskaya. An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In Birgit Pfitzmann, editor, *EUROCRYPT 2001*, volume 2045 of *LNCS*, pages 93–118. Springer, Berlin, Heidelberg, May 2001. doi: 10.1007/3-540-44987-6_7. 80, 82

[39] Jan Camenisch and Anna Lysyanskaya. Dynamic accumulators and application to efficient revocation of anonymous credentials. In Moti Yung, editor, *CRYPTO 2002*, volume 2442 of *LNCS*, pages 61–76. Springer, Berlin, Heidelberg, August 2002. doi: 10.1007/3-540-45708-9_5. 141

[40] Jan Camenisch and Els Van Herreweghen. Design and implementation of the idemix anonymous credential system. In Vijayalakshmi Atluri, editor, *ACM CCS 2002*, pages 21–30. ACM Press, November 2002. doi: 10.1145/586110.586114. 77

[41] Jan Camenisch, Manu Drijvers, and Anja Lehmann. Anonymous attestation using the strong diffie hellman assumption revisited. In *TRUST 2016*, volume 9824, pages 1–20. Springer, 2016. 6, 76, 80, 84, 95, 96, 114, 194

[42] Matteo Campanelli, Dario Fiore, Semin Han, Jihye Kim, Dimitris Kolonelos, and Hyunok Oh. Succinct zero-knowledge batch proofs for set accumulators. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *ACM CCS 2022*, pages 455–469. ACM Press, November 2022. doi: 10.1145/3548606.3560677. 8, 141, 142

[43] Matteo Campanelli, Antonio Faonio, Dario Fiore, Tianyu Li, and Helger Lipmaa. Lookup arguments: Improvements, extensions and applications to zero-knowledge decision trees. In Qiang Tang and Vanessa Teague, editors, *PKC 2024, Part II*, volume 14602 of *LNCS*, pages 337–369. Springer, Cham, April 2024. doi: 10.1007/978-3-031-57722-2_11. 7, 143, 144, 191

[44] Matteo Campanelli, Mario Carrillo, Ignacio Cascudo, Dario Fiore, Danilo Francati, and Rosario Gennaro. On the power of polynomial preprocessing: Proving computations in sublinear time, and more. Cryptology ePrint Archive, Paper 2025/238, 2025. URL https://eprint.iacr.org/2025/238. 196

[45] Alessandro Chiesa, Yuncong Hu, Mary Maller, Pratyush Mishra, Psi Vesely, and Nicholas P. Ward. Marlin: Preprocessing zkSNARKs with universal and updatable SRS. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part I*, volume 12105 of *LNCS*, pages 738–768. Springer, Cham, May 2020. doi: 10.1007/978-3-030-45721-1_26. 15, 19, 86, 151

[46] Alessandro Chiesa, Ryan Lehmkuhl, Pratyush Mishra, and Yinuo Zhang. Eos: Efficient private delegation of zkSNARK provers. In Joseph A. Calandrino and Carmela Troncoso, editors, *USENIX Security 2023*, pages 6453–6469. USENIX Association, August 2023. 195

[47] Henry Corrigan-Gibbs and Dan Boneh. Prio: Private, robust, and scalable computation of aggregate statistics. In *NSDI 2017*, pages 259–282. USENIX Association, 2017. 86

[48] Ronald Cramer, Ivan Bjerre Damgård, and Jesper Buus Nielsen. *Secure Multiparty Computation and Secret Sharing*. Cambridge University Press, 2015. doi: 10.1017/CBO9781107337756. 116

[49] Ivan Damgård and Jesper Buus Nielsen. Scalable and unconditionally secure multiparty computation. In *Advances in Cryptology - CRYPTO*, pages 572–590, 2007. 84, 85

[50] Ivan Damgård, Marcel Keller, Enrique Larraia, Valerio Pastro, Peter Scholl, and Nigel P. Smart. Practical covertly secure MPC for dishonest majority - or: Breaking the SPDZ limits. In Jason Crampton, Sushil Jajodia, and Keith Mayes, editors, *ESORICS 2013*, volume 8134 of *LNCS*, pages 1–18. Springer, Berlin, Heidelberg, September 2013. doi: 10.1007/978-3-642-40203-6_1. 84, 85

[51] Pankaj Dayama, Arpita Patra, Protik Paul, Nitin Singh, and Dhinakaran Vinayagamurthy. How to prove any NP statement jointly? efficient distributed-prover zero-knowledge protocols. *Proc. Priv. Enhancing Technol.*, 2022(2):517–556, 2022. 5, 86, 118

[52] Vanesa Daza, Carla Ràfols, and Alexandros Zacharakis. Updateable inner product argument with logarithmic verifier and applications. In Aggelos Kiayias, Markulf Kohlweiss, Petros Wallden, and Vassilis Zikas, editors, *PKC 2020, Part I*, volume 12110 of *LNCS*, pages 527–557. Springer, Cham, May 2020. doi: 10.1007/978-3-030-45374-9_18. xi, 17, 18, 19, 20, 21, 22, 26, 27, 52, 53

[53] Moumita Dutta, Chaya Ganesh, and Neha Jawalkar. Succinct verification of compressed sigma protocols in the updatable SRS setting. In Qiang Tang and Vanessa Teague, editors, *PKC 2024, Part II*, volume 14602 of *LNCS*, pages 305–336. Springer, Cham, April 2024. doi: 10.1007/978-3-031-57722-2_10. iv, 8, 15, 194, 195

[54] Moumita Dutta, Chaya Ganesh, Sikhar Patranabis, Shubh Prakash, and Nitin Singh. Batching-efficient RAM using updatable lookup arguments. In Bo Luo, Xiaojing Liao, Jun Xu, Engin Kirda, and David Lie, editors, *ACM CCS 2024*, pages 4077–4091. ACM Press, October 2024. doi: 10.1145/3658644.3670356. iv, 8, 140

[55] Moumita Dutta, Chaya Ganesh, Sikhar Patranabis, and Nitin Singh. Compute, but verify: Efficient multiparty computation over authenticated inputs. In Kai-Min Chung

and Yu Sasaki, editors, *ASIACRYPT 2024, Part VI*, volume 15489 of *LNCS*, pages 133–166. Springer, Singapore, December 2024. doi: 10.1007/978-981-96-0938-3_5. iv, 8, 76, 195

[56] Liam Eagen, Dario Fiore, and Ariel Gabizon. cq: Cached quotients for fast lookups. Cryptology ePrint Archive, Report 2022/1763, 2022. URL https://eprint.iacr.org/2022/1763. xi, 7, 143, 144, 146, 148, 149, 155, 164, 188, 191, 192

[57] Daniel Escudero. An introduction to secret-sharing-based secure multiparty computation. Cryptology ePrint Archive, Report 2022/062, 2022. URL https://eprint.iacr.org/2022/062. 119

[58] Dankrad Feist and Dmitry Khovratovich. Fast amortized KZG proofs. Cryptology ePrint Archive, Report 2023/033, 2023. URL https://eprint.iacr.org/2023/033. 146, 157, 163, 174

[59] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *CRYPTO'86*, volume 263 of *LNCS*, pages 186–194. Springer, Berlin, Heidelberg, August 1987. doi: 10.1007/3-540-47721-7_12. 4, 5, 11, 12, 17, 25, 80, 95, 97, 100, 103, 121, 124, 142, 151, 193

[60] Dario Fiore and Anca Nitulescu. On the insecurity of snarks in the presence of oracles. In *Proceedings, Part I, of the 14th International Conference on Theory of Cryptography - Volume 9985*, page 108–138, Berlin, Heidelberg, 2016. Springer-Verlag. ISBN 9783662536407. doi: 10.1007/978-3-662-53641-4_5. URL https://doi.org/10.1007/978-3-662-53641-4_5. 88

[61] Georg Fuchsbauer, Eike Kiltz, and Julian Loss. The algebraic group model and its applications. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part II*, volume 10992 of *LNCS*, pages 33–62. Springer, Cham, August 2018. doi: 10.1007/978-3-319-96881-0_2. 11, 145, 152

[62] Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru. PLONK: Permutations over Lagrange-bases for oecumenical noninteractive arguments of knowledge. Cryptology ePrint Archive, Report 2019/953, 2019. URL https://eprint.iacr.org/2019/953. 86, 189

[63] Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru. Plonk: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. Cryptology ePrint Archive, Report 2019/953, 2019. https://ia.cr/2019/953. 15, 19

[64] Ariel Gabizon, Zachary J. Williamson, and Oana-Madalina Ciobotaru. Plonk: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. *IACR Cryptol. ePrint Arch.*, 2019:953, 2019. URL https://api.semanticscholar.org/CorpusID:201685538. 145, 151, 152, 176

[65] Chaya Ganesh, Hamidreza Khoshakhlagh, Markulf Kohlweiss, Anca Nitulescu, and Michal Zajac. What makes fiat–shamir zksnarks (updatable srs) simulation extractable? Cryptology ePrint Archive, Paper 2021/511, 2021. URL https://eprint.iacr.org/2021/511. https://eprint.iacr.org/2021/511. 88, 128

[66] Chaya Ganesh, Claudio Orlandi, Mahak Pancholi, Akira Takahashi, and Daniel Tschudi. Fiat-shamir bulletproofs are non-malleable (in the random oracle model). Cryptology ePrint Archive, Paper 2023/147, 2023. URL https://eprint.iacr.org/2023/147. https://eprint.iacr.org/2023/147. 88, 128

[67] Chaya Ganesh, Sikhar Patranabis, and Nitin Singh. Samaritan: Linear-time prover SNARK from new multilinear polynomial commitments. Cryptology ePrint Archive, Paper 2025/419, 2025. URL https://eprint.iacr.org/2025/419. 151

[68] Sanjam Garg, Aarushi Goel, Abhishek Jain, Guru-Vamsi Policharla, and Sruthi Sekar. zkSaaS: Zero-Knowledge SNARKs as a service. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 4427–4444, Anaheim, CA, August 2023. USENIX Association. ISBN 978-1-939133-37-3. URL https://www.usenix.org/conference/usenixsecurity23/presentation/garg. 86, 87

[69] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct NIZKs without PCPs. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 626–645. Springer, Berlin, Heidelberg, May 2013. doi: 10.1007/978-3-642-38348-9_37. 3

[70] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In Alfred Aho, editor, *19th ACM STOC*, pages 218–229. ACM Press, May 1987. doi: 10.1145/28395.28420. 76

[71] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on computing*, 18(1):186–208, 1989. 1

[72] Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. Attribute-based encryption for fine-grained access control of encrypted data. In Ari Juels, Rebecca N. Wright, and Sabrina De Capitani di Vimercati, editors, *ACM CCS 2006*, pages 89–98. ACM Press, October / November 2006. doi: 10.1145/1180405.1180418. Available as Cryptology ePrint Archive Report 2006/309. 57

[73] Jens Groth. Short pairing-based non-interactive zero-knowledge arguments. In Masayuki Abe, editor, *ASIACRYPT 2010*, volume 6477 of *LNCS*, pages 321–340. Springer, Berlin, Heidelberg, December 2010. doi: 10.1007/978-3-642-17373-8_19. 3

[74] Jens Groth. On the size of pairing-based non-interactive arguments. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 305–326. Springer, Berlin, Heidelberg, May 2016. doi: 10.1007/978-3-662-49896-5_11. 3, 86

[75] Jens Groth, Markulf Kohlweiss, Mary Maller, Sarah Meiklejohn, and Ian Miers. Updatable and universal common reference strings with applications to zk-SNARKs. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part III*, volume 10993 of *LNCS*, pages 698–728. Springer, Cham, August 2018. doi: 10.1007/978-3-319-96878-0_24. 11, 15, 26, 152

[76] Carmit Hazay, Muthuramakrishnan Venkitasubramaniam, and Mor Weiss. Your reputation's safe with me: Framing-free distributed zero-knowledge proofs. Cryptology ePrint Archive, Paper 2022/1523, 2022. URL https://eprint.iacr.org/2022/1523. https://eprint.iacr.org/2022/1523. 78, 84, 87

[77] Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. Constant-size commitments to polynomials and their applications. In Masayuki Abe, editor, *ASIACRYPT 2010*, volume 6477 of *LNCS*, pages 177–194. Springer, Berlin, Heidelberg, December 2010. doi: 10.1007/978-3-642-17373-8_11. 13, 145, 152, 154, 195

[78] Jonathan Katz, Alex J. Malozemoff, and Xiao Wang. Efficiently enforcing input validity in secure two-party computation. Cryptology ePrint Archive, Report 2016/184, 2016. https://ia.cr/2016/184. 84

[79] Joe Kilian. Founding cryptography on oblivious transfer. In *20th ACM STOC*, pages 20–31. ACM Press, May 1988. doi: 10.1145/62212.62215. 76

[80] Joe Kilian. A note on efficient zero-knowledge proofs and arguments. In *Proceedings of the twenty-fourth annual ACM symposium on Theory of computing*, pages 723–732, 1992. 3

[81] Russell W. F. Lai, Giulio Malavolta, and Viktoria Ronge. Succinct arguments for bilinear group arithmetic: Practical structure-preserving cryptography. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 2057–2074. ACM Press, November 2019. doi: 10.1145/3319535.3354262. 18, 19, 21, 23, 58, 59, 61

[82] Hyeonbum Lee, Seunghun Paik, Hyunjung Son, and Jae Hong Seo. Cougar: Cubic root verifier inner product argument under discrete logarithm assumption. Cryptology ePrint Archive, Paper 2024/616, 2024. URL https://eprint.iacr.org/2024/616. 195

[83] Jonathan Lee. Dory: Efficient, transparent arguments for generalised inner products and polynomial commitments. In Kobbi Nissim and Brent Waters, editors, *Theory of Cryptography*, pages 1–34, Cham, 2021. Springer International Publishing. ISBN 978-3-030-90453-1. 17, 18, 19, 20

[84] Helger Lipmaa. Progression-free sets and sublinear pairing-based non-interactive zero-knowledge arguments. In Ronald Cramer, editor, *TCC 2012*, volume 7194 of *LNCS*, pages 169–189. Springer, Berlin, Heidelberg, March 2012. doi: 10.1007/978-3-642-28914-9_10. 3

[85] Helger Lipmaa. Succinct non-interactive zero knowledge arguments from span programs and linear error-correcting codes. In Kazue Sako and Palash Sarkar, editors, *ASIACRYPT 2013, Part I*, volume 8269 of *LNCS*, pages 41–60. Springer, Berlin, Heidelberg, December 2013. doi: 10.1007/978-3-642-42033-7_3. 3

[86] Helger Lipmaa, Janno Siim, and Michał Zając. Counting vampires: From univariate sumcheck to updatable zk-snark. In *Advances in Cryptology – ASIACRYPT 2022: 28th International Conference on the Theory and Application of Cryptology and Information Security, Taipei, Taiwan, December 5–9, 2022, Proceedings, Part II*, page 249–278, Berlin, Heidelberg, 2023. Springer-Verlag. ISBN 978-3-031-22965-7. doi: 10.1007/978-3-031-22966-4_9. URL https://doi.org/10.1007/978-3-031-22966-4_9. 19

[87] Tobias Looker, Vasilis Kalos, Andrew Whitehead, and Mike Lodder. The bbs signature scheme. Internet Engineering Task Force, 2022. https://identity.foundation/bbs-signature/draft-irtf-cfrg-bbs-signatures.html. 80, 95, 196

[88] Mary Maller, Sean Bowe, Markulf Kohlweiss, and Sarah Meiklejohn. Sonic: Zero-knowledge SNARKs from linear-size universal and updatable structured reference strings. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 2111–2128. ACM Press, November 2019. doi: 10.1145/3319535.3339817. 15, 19

[89] Ralph C. Merkle. A digital signature based on a conventional encryption function. In Carl Pomerance, editor, *CRYPTO'87*, volume 293 of *LNCS*, pages 369–378. Springer, Berlin, Heidelberg, August 1988. doi: 10.1007/3-540-48184-2_32. 141

[90] Silvio Micali. Cs proofs. In *Proceedings 35th Annual Symposium on Foundations of Computer Science*, pages 436–453. IEEE, 1994. 3

[91] Alex Ozdemir and Dan Boneh. Experimenting with collaborative zk-SNARKs: Zero-knowledge proofs for distributed secrets. In Kevin R. B. Butler and Kurt Thomas, editors, *USENIX Security 2022*, pages 4291–4308. USENIX Association, August 2022. 5, 86

[92] Alex Ozdemir, Riad S. Wahby, Barry Whitehat, and Dan Boneh. Scaling verifiable computation using efficient set accumulators. In Srdjan Capkun and Franziska Roesner, editors, *USENIX Security 2020*, pages 2075–2092. USENIX Association, August 2020. 8, 141, 142

[93] Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. Pinocchio: Nearly practical verifiable computation. In *2013 IEEE Symposium on Security and Privacy*, pages 238–252. IEEE Computer Society Press, May 2013. doi: 10.1109/SP.2013.47. 3

[94] Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In Joan Feigenbaum, editor, *CRYPTO'91*, volume 576 of *LNCS*, pages 129–140. Springer, Berlin, Heidelberg, August 1992. doi: 10.1007/3-540-46766-1_9. 78

[95] Torben Pryds Pedersen. Distributed provers with applications to undeniable signatures. In Donald W. Davies, editor, *EUROCRYPT'91*, volume 547 of *LNCS*, pages 221–242. Springer, Berlin, Heidelberg, April 1991. doi: 10.1007/3-540-46416-6_20. 5, 84, 86

[96] Chris Peikert. On error correction in the exponent. In Shai Halevi and Tal Rabin, editors, *TCC 2006*, volume 3876 of *LNCS*, pages 167–183. Springer, Berlin, Heidelberg, March 2006. doi: 10.1007/11681878_9. 104

[97] David Pointcheval and Olivier Sanders. Short randomizable signatures. In Kazue Sako, editor, *CT-RSA 2016*, volume 9610 of *LNCS*, pages 111–126. Springer, Cham, February / March 2016. doi: 10.1007/978-3-319-29485-8_7. 2, 6, 76, 77, 79, 80, 82, 84, 85, 110, 128, 130, 132, 194, 195

[98] Jim Posen and Assimakis A. Kattis. Caulk+: Table-independent lookup arguments. Cryptology ePrint Archive, Report 2022/957, 2022. URL https://eprint.iacr.org/2022/957. 7, 143, 144, 148, 155, 161, 162, 164, 191

[99] Amit Sahai and Brent R. Waters. Fuzzy identity-based encryption. In Ronald Cramer, editor, *EUROCRYPT 2005*, volume 3494 of *LNCS*, pages 457–473. Springer, Berlin, Heidelberg, May 2005. doi: 10.1007/11426639_27. 57

[100] Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In Gilles Brassard, editor, *CRYPTO'89*, volume 435 of *LNCS*, pages 239–252. Springer, New York, August 1990. doi: 10.1007/0-387-34805-0_22. 82

[101] Claus-Peter Schnorr. Efficient signature generation by smart cards. *Journal of Cryptology*, 4(3):161–174, January 1991. doi: 10.1007/BF00196725. 82

[102] Berry Schoenmakers, Meilof Veeningen, and Niels de Vreede. Trinocchio: Privacy-preserving outsourcing by distributed verifiable computation. In Mark Manulis, Ahmad-Reza Sadeghi, and Steve Schneider, editors, *ACNS 16International Conference on Applied Cryptography and Network Security*, volume 9696 of *LNCS*, pages 346–366. Springer, Cham, June 2016. doi: 10.1007/978-3-319-39555-5_19. 5, 86

[103] Srinath Setty, Justin Thaler, and Riad Wahby. Unlocking the lookup singularity with lasso. Cryptology ePrint Archive, Paper 2023/1216, 2023. URL https://eprint.iacr.org/2023/1216. https://eprint.iacr.org/2023/1216. 142, 144, 156

[104] Adi Shamir. How to share a secret. *Communications of the Association for Computing Machinery*, 22(11):612–613, November 1979. doi: 10.1145/359168.359176. 103, 116, 119, 124

[105] Riad S. Wahby, Srinath T. V. Setty, Zuocheng Ren, Andrew J. Blumberg, and Michael Walfish. Efficient RAM and control flow in verifiable outsourced computation. In *NDSS 2015*. The Internet Society, February 2015. doi: 10.14722/ndss.2015.23097. 142, 144

[106] Michael Walfish and Andrew J. Blumberg. Verifying computations without reexecuting them. *Commun. ACM*, page 74–84, 2015. 141

[107] Howard Wu, Wenting Zheng, Alessandro Chiesa, Raluca Ada Popa, and Ion Stoica. DIZK: A distributed zero knowledge proof system. In William Enck and Adrienne Porter Felt, editors, *USENIX Security 2018*, pages 675–692. USENIX Association, August 2018. 86

[108] Andrew Chi-Chih Yao. Protocols for secure computations (extended abstract). In *23rd FOCS*, pages 160–164. IEEE Computer Society Press, November 1982. doi: 10.1109/SFCS.1982.38. 76

[109] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *27th FOCS*, pages 162–167. IEEE Computer Society Press, October 1986. doi: 10.1109/SFCS.1986.25. 76

[110] Arantxa Zapico, Vitalik Buterin, Dmitry Khovratovich, Mary Maller, Anca Nitulescu, and Mark Simkin. Caulk: Lookup arguments in sublinear time. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *ACM CCS 2022*, pages 3121–3134. ACM Press, November 2022. doi: 10.1145/3548606.3560646. 7, 143, 144, 148, 155, 164

[111] Arantxa Zapico, Ariel Gabizon, Dmitry Khovratovich, Mary Maller, and Carla Ràfols. Baloo: Nearly optimal lookup arguments. Cryptology ePrint Archive, Report 2022/1565, 2022. URL https://eprint.iacr.org/2022/1565. 7, 143, 144, 148, 155, 164

[112] Yihua Zhang, Marina Blanton, and Fattaneh Bayatbabolghani. Enforcing input correctness via certification in garbled circuit evaluation. In Simon N. Foley, Dieter Gollmann, and Einar Snekkenes, editors, *ESORICS 2017, Part II*, volume 10493 of *LNCS*, pages 552–569. Springer, Cham, September 2017. doi: 10.1007/978-3-319-66399-9_30. 84

[113] Yuncong Zhang, Shifeng Sun, and Dawu Gu. Efficient kzg-based univariate sum-check and lookup argument. In *PKC 2024*, volume 14602, pages 400–425, 2024. 191

[114] Yupeng Zhang, Daniel Genkin, Jonathan Katz, Dimitrios Papadopoulos, and Charalampos Papamanthou. vRAM: Faster verifiable RAM with program-independent preprocess-

ing. In *2018 IEEE Symposium on Security and Privacy*, pages 908–925. IEEE Computer Society Press, May 2018. doi: 10.1109/SP.2018.00013. 142, 144