UNIVERSITY COLLEGE LONDON

Efficient Zero-Knowledge Proofs and their Applications

Author:
Andrea Cerulli

Supervisor:
Prof. Jens Groth

A thesis submitted in partial fulfilment
of the requirements for the degree of

Doctor of Philosophy
of the
University College London

Department of Computer Science
University College London

April 25, 2019

Declaration of Authorship

I, Andrea Cerulli, declare that this thesis titled, “Efficient Zero-Knowledge Proofs and their Applications” and the work presented in it are my own. Where information has been derived from other sources, I confirm that this has been indicated in the work.

Abstract

A zero-knowledge proof is a fundamental cryptographic primitive that enables the verification of statements without revealing unnecessary information. Zero-knowledge proofs are a key component of many cryptographic protocols and, often, one of their main efficiency bottlenecks. In recent years there have been great advances in improving the efficiency of zero-knowledge proofs, bring them closer to wide deployability.

In this thesis we make another step towards the construction of computationally-efficient zero-knowledge proofs. Specifically, we construct efficient zero-knowledge proofs for the satisfiability of arithmetic circuits for which the computational cost of the prover is only a constant factor more expensive than direct evaluation of the circuit. We also construct efficient zero-knowledge proofs to check the correct execution of (Tiny)RAM programs. In this case the computational cost for the prover is a super-constant factor larger than executing the program directly. Our proofs also support efficient verification and small proof sizes. For security, they rely on symmetric primitives and could potentially withstand attacks from quantum computers.

On a different research direction, we look at group signatures, a fundamental primitive which relies on zero-knowledge proofs. A group signature enables users to sign anonymously on behalf of a group of users. In case of dispute a Manager can identify the author of a signature and potentially banish the user from the group. In this thesis we address the fundamental question of defining the security of fully dynamic group signatures, for which the users can join and leave at any time. Differently from other restricted settings, this case has been largely overlooked in the past. Our security model is general, does not implicitly assume existing design paradigms and captures the security of existing models for more restricted settings.

Impact Statement

Zero-knowledge proofs are cryptographic protocols that enable the verification of some information without compromising the privacy of the data. The potential applications of this primitive are numerous both inside and outside academia. In cryptography, zero-knowledge proofs are used in the construction of other important primitives, such as digital signature schemes and secure public key encryption schemes, and they are used within protocols to enforce honest behaviour of the participants. Zero-knowledge proofs are a fundamental tool for the preservation of privacy, enabling medical applications that use healthcare data, the verification of cloud computation, and various distributed ledger technologies.

The main obstacle to the widespread adoption of zero-knowledge proofs is related to their efficiency. The performance is usually measured in terms of the computational costs to generate and verify the proofs, as well as the amount of space required to store them. Nowadays, the main bottleneck is in the cost of computing the proofs.

The results presented in this work make important steps towards the development of computationally-optimal zero-knowledge proofs. The relevance of our results is mainly theoretical, and more effort is required to improve the practical costs of our solutions. On the other hand, this work opens interesting possibilities that are likely to lead to further improvements in the future.

In this work we also investigate the security of group signatures, which is an important cryptographic application of zero-knowledge proofs. Group signatures enable a member in a group to sign on behalf of the group without revealing its identity. Group signatures can be used in several applications, including anonymous credential schemes, e-cash and for remote attestation in Intel SGX. Our work contributes to the study of group signatures by introducing a strong security model which helps capturing security in a more realistic setting than previous existing models. For example, our model captures security in the presence of highly adversarial authorities and allows for maximal flexibility for the group members.

Acknowledgements

What I learned here at UCL goes well beyond this thesis and what can be summarised in this brief note. I feel very lucky I had the chance to embark on this journey and I will always carry the memories and the experiences of these years with me.

The first person that made all of this possible is Jens Groth, who I thank immensely for guiding me and for all the lessons he thought me. He always had the time to answer my (polynomially) many queries. Having now reached the limit of permitted queries, it is now time to hand in my $\mathsf{Thesis} \leftarrow \mathcal{A}^{\mathcal{J}}$.

Over the years, I had the pleasure to work closely with Pyrros, Christophe, Jonathan, Essam, Mary, Sune, Mohammad, Vasilis, Emiliano, Claudio, Raphael and Sebastian, from which I learned much more that I would have ever imagined. There are many more persons with whom I had the pleasure to share parts of this journey. A special thanks to Enrico who bared with me over countless many coffee breaks and who has taught me how to play table football.

I owe my family a huge debt of gratitude for the support of all these years, for always believing in me and to always make me feel close to home. Most of all I wish to thank Arianna, for always being there when I need her. None of this would have happened without you. I love you.

Thank you all,

Andrea

Contents

List of Figures

List of Tables

To Arianna

Chapter 1

Introduction

The "Knowledge of London", or simply "the Knowledge", is an examination system for taxi drivers in London, United Kingdom. Every aspiring taxi driver must go through an extensive period of training to memorise all the landmarks and streets of the British capital. As a part of the examination process, a candidate must undergo an interview in which they are requested to situate the position of two arbitrary points of interest of the city and to find the best route between them, without the aid of a map or any technology. This interactive interviewing process is repeated few times over the years to ensure the aspirant taxi driver has the necessary Knowledge to fulfil the position.

Alice thinks she can beat the Knowledge and that she can find better routes than Bob, the Knowledge examiner. Alice aims to prove Bob she knows better than him, but at the same time she is concerned about revealing him the fruits of her hard work. For his part, Bob does not believe her until she can prove it to him.

Cryptography offers solutions to overcome the above impasse. A zero-knowledge proof is a cryptographic protocol that allows Alice to convince Bob she has the necessary Knowledge, without disclosing any additional information about it. This is a powerful primitive that can be used in many security applications, whenever it is necessary to strike a balance between verifiability of information and privacy. Examples of these include medical applications using healthcare data, cloud computation, and distributed ledger technology.

Zero-knowledge proofs are ubiquitous in cryptography. For instance, they are used in constructions of public-key encryption schemes, digital signatures, voting

systems, auction systems, e-cash, secure multiparty computation, and verifiable outsourced computation. Unfortunately, zero-knowledge proofs are an expensive component of all of these, and it is therefore important for them to be as efficient as possible.

In this thesis we investigate the efficiency of zero-knowledge proofs and we make important steps towards the achievement of optimal performances. Among the numerous applications of zero-knowledge proofs we consider group signatures and discuss their security formalisation. In this work we propose a new model which offers stringent properties and it captures the security of more realistic scenarios than done by existing models.

1.1 Efficient Zero-Knowledge Proofs

A zero-knowledge proof is a protocol between two parties, called prover and verifier, which consents the prover to convince the verifier about the validity of a statement. There are three main properties a zero-knowledge proof needs to satisfy. *Completeness* states that if prover and verifier follow the protocol, the verifier accepts the validity of a true statement. *Soundness* guarantees the prover cannot deceive the verifier into accepting the validity of a false statement, even if the prover deviates from the specifications of the protocol. *Zero knowledge* prevents the verifier to learn anything from the protocol apart from the validity of the statement.

The performance of zero-knowledge proofs can be measured by a number of parameters. These include the prover's running time, the verifier's running time, the size of the exchanged messages and the number of rounds the prover and verifier interact. Current state of the art zero-knowledge proofs achieve very good performance on verification time, communication and round complexity, which makes the prover's running time the crucial bottleneck. The main research challenge we focus in this work is to construct prover-efficient zero-knowledge proofs.

The statements we are interested in proving are about checking the correctness of some computation performed on some, possibly private, input. The instances of computation we use in this work are either arithmetic circuits or TinyRAM programs [BCG+13b], a RAM machine specifically designed to construct efficient proofs of program execution. The main question that we pose in this thesis is whether it is

possible to construct zero-knowledge proofs, for which the computational cost of the prover is only a constant factor more expensive than directly executing the computation in the instance.

We give a positive answer to the above question and construct zero-knowledge proofs that are highly efficient asymptotically. In the case of arithmetic circuits consisting of N additions and multiplications gates over a finite field we construct proofs with the following computational efficiency.

- Prover time is $\mathcal{O}(N)$ field additions and multiplications.
- Verifier time is $\mathcal{O}(N)$ field *additions*.

In the case of TinyRAM programs terminating in T steps we construct proofs with the following computational efficiency.

- Prover time is $\mathcal{O}(\alpha T)$ TinyRAM operations.
- Verifier time is $\mathrm{poly}(\lambda)\sqrt{T}$ TinyRAM operations.

Where λ is the security parameter and $\alpha = \omega(1)$ is an arbitrarily small superconstant function. Our proofs for TinyRAM program execution fell short of our goal of achieving constant computational overhead for the prover. Nonetheless, they reduce considerably the overhead incurred with respect to the state of the art, bringing us one step closer to the hoped result.

Our zero-knowledge proofs have perfect completeness, i.e., when the prover knows a satisfactory witness she is always able to convince the verifier. Our constructions are proofs of knowledge, that is, not only does the prover demonstrate the statement is true but also that she knows a witness. The proofs have special honest-verifier zero knowledge, which means that honest verifier strategies are simulateable without knowing a witness. The flavour of knowledge soundness and special honest-verifier zero-knowledge depends on the underlying commitment scheme we use. When instantiated with statistically binding commitment schemes, we obtain proofs (statistically knowledge sound) with computational zero-knowledge. When we use statistically hiding commitments we obtain arguments (computationally knowledge sound) with statistical special honest verifier zero-knowledge. The communication complexity of our proofs with unconditional soundness is only $\mathcal{O}(N)$ and $\mathcal{O}(T)$ field elements,

respectively. Our arguments with computational soundness have sub-linear communication of $\text{poly}(\lambda)\sqrt{N}$ and $\text{poly}(\lambda)\sqrt{T}$ field elements, respectively, when the commitments are compact. Round complexity is also low, when we optimize for computational efficiency for prover and verifier we only use $\mathcal{O}(\log\log N)$ and $\mathcal{O}(\log\log T)$ rounds.

Our constructions are modular and consist of three steps. First, we construct proofs in a communication model we call the Ideal Linear Commitment (ILC) channel. In the ILC model, the prover can commit vectors of secret field elements to the channel. The verifier may later query openings to linear combinations of the committed vectors, which the channel will answer directly. We show that idealising the techniques by Groth et al. [Gro09] gives us efficient proofs in the ideal linear commitment model.

Next, we compile proofs in the ILC model into proof and argument systems using non-interactive commitment schemes. However, unlike previous works we do not commit directly to the vectors. Instead, we encode the vectors as randomized codewords using a linear error-correcting code. We now consider the codewords as rows of a matrix and commit to the columns of that matrix.

Finally, we instantiate the scheme with concrete error-correcting codes and non-interactive commitment schemes. To achieve the best asymptotic efficiency we use linear-time computable error correcting codes [DI14] and linear-time computable commitment schemes [IKO+09; AHI+17b].

1.2 Group Signatures

A group signature scheme [CvH91] allows a member of a group to anonymously sign messages on behalf of the group. Group membership is administered by a designated group manager. In the case of a dispute, the group manager or a designated opening authority have the ability to identify the signer and attribute the signature to her.

In static group signatures, the group members are fixed once and for all during the setup phase. Partially dynamic group signatures allow the enrolment of members in the group at any time but members cannot leave or be removed from the group once they have joined. In many real-life applications, on the other hand, it is desirable to let

the users join and leave at any time. Group signatures offering this level of flexibility are referred to as fully dynamic.

Static and partially dynamic group signatures have been rigorously studied and their security properties are nowadays well-established [BMW03; BSZ05; KY06]. Unfortunately, the same cannot be said about the fully dynamic variant, despite being the more relevant one for applications. Existing constructions of fully dynamic signatures follow different design paradigms and the existing security models tend to implicitly assume the different approaches. The resulting security definitions do not necessarily capture the security of different approaches and in most cases are not properly formalised.

In this work we address the question of defining a rigorous security model for fully dynamic group signatures. We consider both the case of a single group manager, and the case where the role of the group manager who oversees the group is separated by the role of the opening authority who can identify a signer. Our security definitions are general and applicable across different design paradigms. Most importantly, our model offers stringent security requirements and considers, when possible, security against compromised authorities with adversarially generated keys.

Alongside our general definitions for the fully dynamic case, we present relaxations of our model to the case of partially dynamic and static group signatures. We show that the obtained notions cover the existing formal security models for these two settings. Lastly, we show a generic construction which is secure with respect to the strongest variant of our definitions.

1.3 Structure and Content

In Chapter 2 we cover the related work, identify current gaps in the literature and position our contributions. Next, in Chapter 3 we introduce the notation, we present the definitions of zero-knowledge proofs of knowledge together with the ILC model and recall the definitions of linear error-correcting codes and commitment schemes. In Chapter 4 and 5 we present our proofs in the ILC model for the satisfiability of arithmetic circuits and for the correct execution of TinyRAM programs, respectively. In Chapter 6 we discuss how to compile proofs in the ILC model into standard proofs

and arguments. In the same chapter we discuss the compiled version of the proofs presented in Chapters 4 and 5 and their instantiations using error correcting codes and commitment schemes. In Chapter 7 we diverge from the topic of the previous ones and we present our security model for fully dynamic group signatures. Chapter 8 concludes this thesis with some future research directions left open by this work.

The research presented in Chapters 4 and 6 was joint work with Jonathan Bootle, Essam Ghadafi, Jens Groth, Mohammad Hajiabadi and Sune Jakobsen, and it was published in [BCG+17]. The author contributed to design of some of the ILC building-block sub-proofs used in the construction and to the instantiation using error correcting codes and commitments. The author did not actively contributed to the design of the known permutation proof system and its full specifications have not been included in this thesis. In Chapter 6 we introduce a novel primitive called Exposure-Resilient Encoding, which has not appeared before.

The research presented in Chapter 5 was joint work with Jonathan Bootle, Jens Groth, Sune Jakobsen and Mary Maller, and it was published in [BCG+18]. The author contributed to the arithmetization of the TinyRAM relation, its deconstruction into small relations, the novel bit-decomposition as well as the design of some of the proofs over the ILC. The author did not actively contributed to the design of the unknown permutation and lookup proof systems and their full specifications have not been included in this thesis.

The research presented in Chapter 7 significantly updates the work published in [BCC+16a] jointly with Jonathan Bootle, Pyrros Chaidos, Essam Ghadafi and Jens Groth. The security model introduced in Chapter 7 refines, generalises and addresses few shortcomings of the previously published model. The author contributed substantially to the redesign of the model and its generalisation, with the collaboration of Essam Ghadafi and Jens Groth.

All the research leading to the results included in this thesis has received funding from the European Research Council under the European Union's Seventh Framework Programme (FP/2007-2013) / ERC Grant Agreement n. 307937.

Chapter 2

Literature Review

Since their introduction zero-knowledge proofs have played a central role in cryptography and have been used in a vast number of applications such as digital signatures, secure public key encryption schemes, multiparty computation, electronic cash, and internet voting, just to name a few. The impact of zero-knowledge proofs, however, is well beyond their applicability as their conception represented a shift of paradigm in the way we think and define security in general. In this literature review we examine the research done around zero-knowledge proofs, with particular focus on the efficiency of zero-knowledge proofs for general statements.

Among the numerous applications of zero-knowledge proofs we take a look at group signatures, another important primitive that has received considerable attention. Rather than on efficiency, the focus here is on different security models and different design paradigms of group signatures.

2.1 Zero-Knowledge Proofs and Arguments

Zero-knowledge proofs were invented by Goldwasser, Micali and Rackoff [GMR89] and since then a large amount of research has been devoted to their study. A zero-knowledge proof is a protocol between two parties, called the prover and the verifier, which allow the prover to convince the verifier about the validity of a statement. The type of statements we are interested in proving are of the form $u \in \mathcal{L}$, where $\mathcal{L}$ is a language in NP. There are three main properties a zero-knowledge proof needs to satisfy: completeness, which states that if prover and verifier follow the protocol, the verifier accepts the validity of a true statement; soundness, which guarantees the prover cannot deceive the verifier into accepting the validity of a false statement, even if the

prover deviates from the specifications of the protocol; zero knowledge, which prevents the verifier to learn anything from the protocol apart from the validity of the statement.

Much of the early work in the area was devoted to explore the classes of languages admitting zero-knowledge proofs and to the different flavours of their properties. Goldreich, Micali and Wigderson [GMW91] showed that all languages in NP admit computational zero-knowledge proofs under the assumption that uniform one-way functions exist. This was then generalised by Impagliazzo and Yung [IY87] and Ben-Or et al. [BGG+88] to all languages admitting an interactive proof system. Shamir [Sha92] then showed that this complexity class coincides with PSPACE in the acclaimed result IP=PSPACE.

The first zero-knowledge proofs only achieved zero-knowledge in the presence of a computationally bounded verifier but Brassard, Chaum and Crepeau [BCC88] showed proofs can also have statistical zero-knowledge, for which zero-knowledge holds against an unbounded verifier. However unless the polynomial time hierarchy collapses, NP-complete languages cannot have proof systems that are both sound against unbounded provers and zero-knowledge against unbounded verifiers [For89; AH91]. Therefore, if one aims to construct proofs for all languages in NP, one must choose between proof systems with statistical soundness and proof systems with computational soundness. The latter are usually called argument systems as opposed to proof systems, which indicate the former. For more restricted classes of languages, on the other hand, it is possible to construct proofs with both statistical zero-knowledge and statistical soundness [SV00]. Naor et al. [NOV+92], showed that assuming the existence of one-way permutations, all languages in NP admit perfect zero-knowledge arguments, i.e. for which unbounded verifiers have zero probability of learning any additional information. Later, Nguyen, Ong and Vadhan [NOV06] showed that one-way functions suffice to construct statistical zero-knowledge arguments for all languages in NP.

Another key notion introduced by [GMR89] is the one of proof of knowledge, which strengthen the soundness property of a proof system and requires that if the prover convinces the verifier, then she must know a witness w certifying the validity of the statement, i.e. for $u \in \mathcal{L}$. The first formalisations of this property were done by

Feige, Fiat and Shamir [FFS88], Tompa and Woll [TW87], and Feige and Shamir [FS89]. The last cited work also showed that all languages in NP admitted both perfect zero-knowledge arguments of knowledge based on the hardness of discrete logarithm assumption, as well as computational arguments of knowledge based on the existence of one-way functions. The definition of proof of knowledge was then refined by Bellare and Goldreich [BG92] who addressed few shortcomings in the earlier formalizations. Later, Lindell [Lin03] introduced a related notion called witness-extended emulation, which Groth [Gro04] then showed to imply the usual definition of argument of knowledge in the public parameter model.

Aside from the flavours of their different properties, zero-knowledge proofs can be compared with respect to their efficiency. The main metrics used to measure their performance are interaction, communication and computational complexity, which are discussed next.

2.1.1 Interaction

The interaction of a proof system is measured by the number of messages prover and verifier send to each other. A move consists of a single message from one party to the other, while a pair of consecutive moves makes a round.

The first constant-round perfect zero-knowledge argument for all languages in NP was constructed by Brassard, Crépeau and Yung [BCY91], which was based on the hardness of the discrete logarithm problem and counted a total of 6 moves. Feige and Shamir [FS89] then reduced the number of rounds, giving both a 4-moves perfect zero-knowledge argument of knowledge based on the hardness of discrete logarithm and a 4-moves computational zero-knowledge argument of knowledge based on one-way permutations. Later, another 4-moves computational zero-knowledge argument for all NP languages was given by Bellare, Jakobsson and Yung [BJY97] which only relied on the existence of one-way functions. These arguments are optimal with respect to the round complexity, as Goldreich and Krawczyk [GK96b] showed that 3 moves proofs and arguments cannot be constructed for non-trivial languages. In the same work the authors showed the impossibility of constant round public-coin proof systems, i.e. proof systems for which the verifier is restricted to send the outcome of his random coin tosses to the prover. Public-coin proof systems were introduced by

Babai [Bab85] under the name of Arthur-Merlin games and they were later shown to be equivalent to the usual notion of (private-coin) interactive proof systems [GS89].

With respect to computational zero-knowledge proofs with statistical soundness, the first constant-round construction for all languages in NP was given by Goldreich and Kahan [GK96a], based on the existence of claw-free functions and which achieved 5 moves of interaction. Much later, Lindell showed how to construct 5-moves computational zero-knowledge proofs of knowledge for all NP. These protocols are optimal with respect to their round complexity as Katz [Kat12] showed that proofs with unbounded soundness require at least 5 moves.

The above lower bounds on the minimal amount of interaction can be circumvented by assuming the existence of an honestly generated common reference string, which is shared between prover and verifier. Blum, Feldman and Micali [BFM88] showed that, in this setting, it is possible to construct non-interactive computational zero-knowledge proofs for all languages in NP, where the prover only sends a single message to the verifier. The question on whether non-interactive statistical (and perfect) zero-knowledge arguments existed for all NP languages remained unanswered for a long time. This was finally settled affirmatively by Groth, Ostrovsky and Sahai [GOS12] using groups equipped with a bilinear map.

An alternative approach to minimise interaction between prover and verifier is to apply a general transform to an interactive proof system and turn it into a non-interactive one. This methodology was first illustrated by Fiat and Shamir [FS86] who used hash functions to construct signature schemes from interactive identification protocols. A similar transform was introduced by Fischlin [Fis05] to achieve straight-line extractability, i.e., the security proof of (knowledge) soundness does not rely on rewinding techniques. The security of these transforms is proven in the random oracle model [BR93], in which the hash function is modelled as a truly random function. The use of this model is controversial due to the existence of signatures and encryption schemes that are provably secure in the random oracle model, but for which every instantiation of the random oracle leads to an insecure scheme [GK03; CGH04; ABG+13]. More recently, similar transforms were suggested in [Lin15; CPS+16]. These are less efficient than the Fiat-Shamir transform, but they can be proven secure in the non-programmable random oracle model, and thus they weaken the assumptions made

on the hash function.

2.1.2 Communication

The communication complexity of a proof system consists in the overall size of the messages exchanged between prover and verifier. This is usually measured with respect to either the size of the instance, e.g. the circuit size, or the size of the witness.

Using probabilistic checkable proofs (PCPs [AS92]), Kalai and Raz [KR08] constructed interactive zero-knowledge proofs for boolean circuits of constant depth, with a communication complexity that is polynomial in the size of the witness. In the same settings and for the same languages, Ishai et al. [IKO+09] used an innovative approach based on multiparty computation (MPC) to construct interactive zero-knowledge proofs of quasi-linear size in the length of the witness. Goldwasser, Kalai and Rothblum [GKR15] also showed zero-knowledge proofs with quasi-linear communication in the witness size, but extended these to boolean circuits with polylogarithmic depth. For arbitrary boolean circuits, [IKO+09] showed a different construction with linear communication complexity in the size of the circuit. In the non-interactive settings, Gentry et al. [GGI+15] used fully homomorphic encryption [Gen09] to construct zero-knowledge proofs with proof size that is linear in the witness size plus an additive polynomial factor in the security parameter.

It is unlikely [GH98; GVW02] that sublinear communication can be achieved in proof systems with statistical soundness. For arguments, on the other hand, the situation is different as Kilian [Kil92] constructed a constant round zero-knowledge argument system with polylogarithmic communication complexity. In Kilian's argument the prover constructs a PCP and then hashes it using a Merkle tree to produce a short proof for the verifier. Until recently, these techniques did not produce practical arguments due to the cost involved in the computation of the PCP. The introduction of interactive oracle proofs (IOPs) [BCS16] made this type of proof system a realistic option, improving the efficiency of the prover. Most notably, the recent work by Ben-Sasson et al. [BBH+18] presented a new IOP-based argument system, known as STARKs, which also has polylogarithmic communication, and it is optimised for better practicality. Another PCP-based approach was introduced by Ishai et al. [IKO07]

to give arguments with a laconic prover, such that the communication from the prover to the verifier is minimal and it consists of a constant number of ciphertexts only.

In the non-interactive settings, the first sublinear size argument was given by Micali [Mic94] using the Fiat-Shamir transform. Working in the common reference string model and using non-falsifiable assumptions ([Nao03]), Groth [Gro10] gave a pairing-based non-interactive zero-knowledge argument consisting of a constant number of group elements. Follow-up works on succinct non-interactive arguments of knowledge (SNARKs) have shown that it is possible to have both a modest size common reference string and proofs as small as 3 group elements [BCC+12; GGP+13; PHG+16; BCC+13; Gro16; GM17; GKM+18]. In [Gro16] it was shown that SNARKs need at least 2 group elements, although it is currently not known whether arguments of this size exist.

2.1.3 Verifier Computation

In general, the verifier has to read the entire instance u since even a single deviating bit may render the statement false. However, in many cases an instance can be represented more compactly than the witness and the instance may be small compared to the computational effort it takes to verify a witness. In these cases it is possible to get sublinear verification time compared to the time it takes to check the relation defining the language $\mathcal{L}$. This is for instance the case for the SNARKs mentioned above, where the verification time only depends on the size of the instance but not on the complexity of the relation. Having low verifier complexity can be important even if we are not interested in zero-knowledge and even for languages in P, as for instance in the case of verifiable computation. Verifiable computation has taken two distinct flavours in the literature: in the first a prover wishes to prove that they have correctly computed a public function on a private input, e.g. private database searches; in the second a verifier wishes to offload a large computation on a public input to a more powerful prover. In the latter case, the influential work by Goldwasser, Kalai, and Rothblum [GKR15] used efficient short probabilistically checkable proofs to get sublinear verifier computation. This was later made non-interactive in [GGP10]. Further improvements were made by Cormode, Mitzenmacher and Thaler [CMT12; Tha13]

yielding sub-linear verifcation time and quasi-linear prover time. These proof systems are very efficient for the verifier and have been implemented [VSB+13] but they only work for languages in P.

2.1.4 Prover Computation

In the previous sections we discussed the incredible advances in improving the efficiency of zero-knowledge proof systems with respect to interaction, communication and verification time. Nowadays, the main challenge is to reduce the computational cost of the prover. In this section we focus on the efficiency of proof systems for circuit satisfiability and to check the correctness of the execution of RAM programs.

Boolean and Arithmetic Circuits. Many classic zero-knowedge proofs rely on cyclic groups starting from the work by Schnorr [Sch91]. These techniques can be generalised to NP-complete languages such as boolean and arithmetic circuit satisfiability [CDS94; CD98; Gro09; BCC+16b]. In the last cited work, Bootle et al. gave discrete-logarithm based arguments for arithmetic circuit satisfiability with logarithmic communication and round complexity. The computational cost for the prover amounts to $\mathcal{O}(N)$ group exponentiations, where N is the number of gates in the circuit. For the discrete logarithm assumption to hold, the group must have superpolynomial size in the security parameter λ, so exponentiations incur a significant overhead compared to direct evaluation of the witness in the circuit: each group exponentiation costs $\omega(\log \lambda)$ field operations for a finite field of size $|\mathbb{F}| = \lambda^{\omega(1)}$. The SNARKs mentioned earlier achieve very compact arguments and efficient verification time, while their main bottleneck is in the prover computational cost. These arguments also rely on cyclic groups and, similarly to the ones above, they require the prover to compute $\mathcal{O}(N)$ exponentiations and incur a similar overhead.

Recently, Baum et al. [BBC+18] adapted techniques from [Gro09; BCC+16b] to the lattice setting and constructed an interactive zero-knowledge argument for the satisfiability of arithmetic circuits with sublinear communication, i.e. $\mathcal{O}(\sqrt{N \log(N)})$ field elements. In their argument the computational overhead for the prover is $\omega(\log(N))$ field operations, with respect to cost of evaluating the circuit.

An alternative approach to these techniques is to use the “MPC in the head" paradigm introduced in the aforementioned work by Ishai et al. [IKO+09]. The main idea behind their general approach is for the prover to first execute in *her head* an MPC protocol computing the verification circuit of some relation $\mathcal{R}$, and then to commit to the views of all the imaginary participants. Next, the verifier asks the prover to open a subset of the committed views and checks the correctness of these as well as their consistency with each other. Soundness and zero-knowledge follow from robustness and privacy properties of the MPC protocol, respectively. Applying this framework to efficient MPCs gives asymptotically efficient zero-knowledge proofs. For example, the perfectly secure MPC of [DI06] is used in [IKO+09] to obtain zero-knowledge proofs for the satisfiability of Boolean circuits with communication linear in the circuit size, $\mathcal{O}(N)$, and a computational cost of $\Omega(\lambda N)$, for circuits of size N and security parameter λ. Damgård, Ishai and Krøigaard [DIK10] used the MPC framework to construct zero-knowledge proofs for the satisfiability of arithmetic circuits. Their construction, which reduces the computational overhead of the prover at expense of increasing its communication, achieves $\mathrm{polylog}(\lambda)N$ complexity for both computation and communication. Instead of focusing on theoretical performance, ZKBoo [GMO16] and its subsequent optimisation ZKB++ [CDG+17] are practical implementations of a “3PC in the head" style zero-knowledge proof for boolean circuit satisfiability. Communication grows linearly in the circuit size in both proofs, and a superlogarithmic number of repetitions is required to make the soundness error negligible, but the speed of the symmetric key primitives makes practical performance good. Following the same paradigm, Ligero [AHI+17a] used an optimised version of the MPC of [DI06] to construct arguments with $\mathcal{O}(\lambda\sqrt{N})$ communication for both boolean and arithmetic circuits. Despite the good practical efficiency, the asymptotic computational cost for the prover remains quasi-linear in the size of the circuit.

Jawurek, Kerschbaum and Orlandi [JKO13] used a very different approach to building zero-knowledge proofs based on garbled circuits. Their approach proved [FNO15; CGM16] to be very efficient in practice for constructing proofs for languages represented as boolean circuits. These techniques are appealing for proving small statements, as they require only a constant number of symmetric-key operations per gate

in the circuit, while the main bottleneck is in their communication complexity. Asymptotically, this approach yields computational and communication complexity of $\mathcal{O}(\lambda N)$ bit operations and bits, respectively, when λ is the cost of a single symmetric-key operation.

Ben-Sasson et al. [BCG+16a] constructed a 3-round public-coin IOP (with soundness error $1/2$) for Boolean circuit satisfiability with linear proof length and quasi-linear running time for both the prover and the verifier. Moreover, the constructed IOP has constant query complexity (the number of opening queries requested by the verifier), while prior PCP constructions require sub-linear query complexity. Another follow-up work by Ben-Sasson et al. [BCG+16b] gave 2-round zero-knowledge IOPs (duplex PCPs) for any language in NTIME$(T(n))$ with quasi-linear prover computation in $n + T(n)$. Aurora [BCR+18] is another IOP-based zero knowledge argument for rank-1 constraint satisfaction problems (R1CS), which are a generalisation of arithmetic circuits. This system achieves proofs of size $\mathcal{O}(\log^2 N)$ fields element; quasi-linear computation for the prover, i.e. $\mathcal{O}(N \log N)$ field operations; and linear verification, i.e. $\mathcal{O}(N)$ field operations.

Verification of outsourced computation [GKR15; CMT12; Tha13; WHG+16] mostly focuses on verifying deterministic computation and does not offer zero-knowledge. However, recent works augmented these with cryptographic techniques to also achieve zero-knowledge [ZGK+17; WJB+17; WTS+18]. Hyrax [WTS+18] offers an implementation with good concrete performance. This system has sublinear communication and verification, while the prover computation is dominated by $\mathcal{O}(dN + S \log S)$ field operations for a depth d and width S circuit when the witness is small compared to the circuit size. If, in addition, the circuit can be parallelised into many identical sub-computations, the prover cost can be further reduced to $O(dN)$ field operations.

To summarise, in all the above proof systems the prover incurs in a super-linear cost in the size of the instance. The recent work of [BCG+17], which is also presented in this thesis, shows that it is possible to construct efficient proofs and arguments where the prover only pays a constant computational overhead with respect to the time needed to directly check the instance given the witness, e.g. the cost of evaluating the circuit. We construct both proofs and arguments for the satisfiability of arithmetic circuits over a large field. For a circuit of size N, the prover computational costs

consists of $\mathcal{O}(N)$ multiplications, while the verification cost only requires $\mathcal{O}(N)$ field *additions*. The communication complexity of our proofs with unconditional soundness is only $\mathcal{O}(N)$ field elements, while our arguments with computational soundness have sub-linear communication of $\text{poly}(\lambda)\sqrt{N}$ field elements when the commitments are compact. Round complexity is also low. When we optimize for computational efficiency the argument needs $\mathcal{O}(\log \log N)$ rounds.

Our work extends techniques from [Gro09] without requiring homomorphic properties of the underlying commitment scheme. To achieve this, we first replace the homomorphic commitments with a combination of error correcting codes and standard commitments, and we then introduce a new efficient technique to prove knowledge of the openings of committed values. Since the commitment scheme does not need to be homomorphic, we can construct it from either one-way functions or collision-resistant hash functions. To get optimal computational efficiency we consider linear-time computable instantiations of these primitives, such as from [IKO+08] and [AHI+17b], and we obtain linear-time computable commitments.

From a technical point of view, our work shares some similarities with the one of Cramer et al. [CDP12], which introduces techniques for verifying multiplicative relations of committed values. When applied to zero-knowledge proofs for the satisfiability of Boolean circuits, the asymptotic communication and computation complexities of [CDP12] are close to [IKO+09], although with smaller constants. Unlike [CDP12], we do not require any homomorphic property from the commitment scheme, and instead of relying on linear secret sharing schemes with product reconstruction, we only require linear error-correcting codes.

From an high-level perspective, Ligero [AHI+17a] shares some similarities with our arguments, even though the two results stemmed, independently, from two different research directions. Their construction does not rely on homomorphic commitments, similarly to ours, but it uses instead Reed-Solomon codes and Merkle trees. Compared to Ligero, we achieve better asymptotic performance for the prover. This is partly due to the choice of error correcting codes we use in our instantiation, which are computable in linear time [DI14]. If we were to replace them with Reed-Solomon codes, we would get efficiency comparable with Ligero with respect to all metrics. The converse, on the other hand, does not hold since the error correcting code is only one

of the ingredients used to obtain linear time computation for the prover.

Our techniques do not fare as well in the case of Boolean circuits, as the soundness of our arguments requires the field to be large and representing bits as field elements would introduce a superconstant multiplicative overhead for the prover. We therefore stress that the significant performance improvement of our proofs and arguments over the state of the art only relates to arithmetic circuits. Table 2.1 summarises the efficiency of the most prover-efficient proofs and arguments.

Work	Prover	Verifier	Communication	Rounds	Soundness
[DIK10]	$\text{polylog}(\lambda)N\ \mathbb{F}^{\times}$	$\mathcal{O}(N)\ \mathbb{F}^{\times}$	$\mathcal{O}(N)$	$\mathcal{O}(1)$	Statistical
Ligero [AHI+17a]	$\mathcal{O}(N \log N)\ \mathbb{F}^{\times}$	$\mathcal{O}(N)\ \mathbb{F}^{\times}$	$\mathcal{O}(\lambda\sqrt{N})$	$\mathcal{O}(1)$	Computational
Aurora[BCR+18]	$\mathcal{O}(N \log N)\ \mathbb{F}^{\times}$	$\mathcal{O}(N)\ \mathbb{F}^{\times}$	$\mathcal{O}(\log^2 N)$	$\mathcal{O}(\log N)$	Computational
This Thesis	$\mathcal{O}(N)\ \mathbb{F}^{\times}$	$\mathcal{O}(N)\ \mathbb{F}^{+}$	$\mathcal{O}(N)$	$\mathcal{O}(\text{loglog}N)$	Statistical
This Thesis	$\mathcal{O}(N)\ \mathbb{F}^{\times}$	$\mathcal{O}(N)\ \mathbb{F}^{+}$	$\mathcal{O}(\lambda\sqrt{N})$	$\mathcal{O}(\text{loglog}N)$	Computational

TABLE 2.1: Efficiency comparison of the most efficient proofs and arguments for the satisfiability of arithmetic circuits with respect to prover computation. $\mathbb{F}^{\times}$ and $\mathbb{F}^{+}$ stand for the cost of field multiplications and additions, respectively. Communication is measured in field elements of size $\log |\mathbb{F}|$ bits.

Correct Program Execution. In practice, most computation does not resemble circuit evaluation but is instead done by computer programs processing one instruction at a time. There has been a sustained effort to construct efficient zero-knowledge proofs that support real-life computation, i.e., proving statements of the form "when executing program P on public input x and private input y we get the output z." In the context of SNARKs there are already several systems to prove the correct execution of programs written in C [PHG+16; BFR+13; BCG+13a; WSR+15]. These systems generally involve a *front-end* which compiles the program into an arithmetic circuit which is then fed into a cryptographic *back-end*. Much work has been dedicated to improve both sides, resulting in systems achieving different trade-offs between efficiency and expressiveness of the computation.

When we want to reason theoretically about zero-knowledge proofs for correct program execution, it is useful to abstract program execution as a random-access machine that can address in each instruction an arbitrary location in memory and perform integer operations on it. For closer resemblance to real-life computation, we can bound the integers to a specific word size and specify a more general set of operations the random-access machine can execute. TinyRAM [BCG+13b; BCG+13a] is a

prominent example of a computational model bridging the gap between theory and real-word computation. It comes with a compiler from C to TinyRAM code and underpins several implementations of zero-knowledge proofs for correct program execution [BCG+13a; BCT+14; BCT+17; CTV15; BRS17; BBH+18], where the prover time is $\Omega(T \log^2 \lambda)$ for a program execution that takes time T. Similar efficiency is also achieved, asymptotically, by other proof systems that can compile (restricted) C programs and prove correct execution such as Pinocchio [PHG+16], Pantry [BFR+13] and Buffet [WSR+15]. There are three main sources of this superlogarithmic overhead. The first one is due to the cost of verifying Boolean operations over words of $\mathcal{O}(\log \lambda)$ length. Each of these is checked by using an arithmetic circuit of size $\mathcal{O}(\log \lambda)$ that takes the individual bits of each word as input. The second source of overhead is due to the way memory consistency is checked. This is done by using a permutation network, which can be implemented by an arithmetic circuit of quasi-linear size in T. The last source of overhead is due to the cryptographic back-end used to prove that all arithmetic circuits involved are satisfied. This usually relies on cyclic groups, such as SNARKs, for which the prover incurs a computational superlogarithmic overhead, as discussed in the previous section. A notable exception is STARK [BBH+18] that only relies on the existence of collision-resistant hash functions. However, this scheme is still susceptible to the first two sources of overhead.

Techniques from the aforementioned work of [JKO13] also found applications in zero-knowledge proofs to check the execution of RAM programs [HMR15; MRS17]. These proofs achieve amortized sublinear communication complexity in the size of the memory used by the program, however the computational overhead of the prover is still large. For instances that can be represented as RAM programs terminating in T steps and using memory of size M, a stand-alone execution of these proofs yields communication and computation with $\mathrm{polylog}(M)$ overhead compared to the running time T of the RAM program.

In all the above proof systems to check the correct execution of programs, the computational overhead for the prover is superlogarithmic with respect to the execution time of the program. The recent work of [BCG+18], which is presented in this thesis, constructed the first arguments for the correct execution of a TinyRAM program achieving a sublogarithmic overhead. More precisely, our work reduces the prover's

overhead from $\Omega(\log^2 \lambda)$ to an arbitrarily small superconstant $\alpha = \omega(1)$. In our proofs and arguments, the verification time is sublinear in the execution time of the program.

In order to reduce the three sources of overhead discussed above, we diverge from the previous approaches. Firstly, we introduce a different type of decomposition to efficiently check boolean operations using field arithmetic. This is combined with an efficient proof for lookup relations, which is used to check that committed values are consistent with a committed lookup table. Secondly, instead of using permutation network to check memory consistency, we use a technique based on the invariance of roots in polynomials as first suggested by Neff [Nef01] in the context of verifiable shuffles. This technique was recently used for a similar purpose in vSQL [ZGK+17], a system for verifying database queries. Lastly, instead of relying on cyclic groups, we adopt the same techniques we previously introduced to check the satisfiability of arithmetic circuits. The efficiency of our proofs and arguments for the execution of TinyRAM programs is given in Table 2.2, along with the efficiency of the most efficient zero-knowledge arguments. The costs reported in the table are in the case prover and verifier are implemented as TinyRAM programs.

Work	Prover	Verifier	Communication	Rounds	Soundness
[BCT+14]	$\mathcal{O}(\alpha T \log^2 T)$	$\omega(L + \lvert v\rvert)$	$\omega(1)$	1	Comp.
[BBH+18]	$\Omega(T \log^2 T)$	$\mathrm{poly}(\lambda)(\log T + L + \lvert v\rvert)$	$\omega(\lambda \log T)$	$\mathcal{O}(\log T)$	Comp.
This Thesis	$\mathcal{O}(\alpha T)$	$\mathrm{poly}(\lambda)(\sqrt{T} + L + \lvert v\rvert)$	$\mathrm{poly}(\lambda)(\sqrt{T} + L + \lvert v\rvert)$	$\mathcal{O}(\log \log T)$	Comp.
This Thesis	$\mathcal{O}(\alpha T)$	$\mathrm{poly}(\lambda)(\sqrt{T} + L + \lvert v\rvert)$	$\omega(T)$	$\mathcal{O}(\log \log T)$	Statistical

TABLE 2.2: Efficiency comparisons between our proofs and arguments with the most efficient zero-knowledge arguments for the correct execution of TinyRAM programs, at security level $2^{-\omega(\log \lambda)}$. Computation is measured in TinyRAM steps and communication in words of length $W = \Theta(\log \lambda)$ with λ the security parameter and $\alpha = \omega(1)$ an arbitrarily small superconstant function. L is the length of the TinyRAM program, $|v|$ the size of the public inputs to the program, and T its running time.

Concurrently to [BCG+18], Zhang et al. [ZGK+18] developed and implemented a scheme for verifying RAM computations. Like us and [ZGK+17], they avoid the use of permutation networks by using permutation proofs based on polynomial invariance by Neff [Nef01]. The idea underlying their technique to prove the correct fetch of an operation is also related to the idea underpinning our lookup proofs. However, there are significant differences between the techniques used in our works. For example, they rely on techniques from [CMT12] to instantiate their proofs while we use techniques based on ideal linear commitments [BCG+18]. The proofs in [ZGK+18] are not

zero knowledge, since they leak the number of times each type of instruction is executed, while our proofs are zero knowledge. In terms of prover efficiency, [ZGK+18] focuses on concrete efficiency and yields impressive concrete performance. Asymptotically speaking, however, we are a polylogarithmic factor more efficient. This may require some explanations because they claim linear complexity for the prover. The reason is that they treat the prover as a TinyRAM machine with logarithmic word size in their performance measurement. Looking under the hood, we see that they use bit-decomposition to handle logical operations, which gives a constant overhead when you fix a particular word size, (e.g. 32 bits) but asymptotically the cost of this is logarithmic since it is linear in the word size. Also, their commitments are based on cyclic groups and the use of exponentiations introduces another superlogarithmic overhead for the prover when implemented in TinyRAM. Lastly, our arguments are based on assumptions that are currently believed to be post-quantum and thus they may offer some security against quantum adversaries.

2.2 Group Signatures

Group signatures, put forward by Chaum and van Heyst [CvH91], are a fundamental cryptographic primitive allowing a member of a group to anonymously sign messages on behalf of it. Group membership is administered by a designated group manager. In the case of a dispute, the group manager or a designated opening authority has the ability to identify the signer and attribute the signature to her.

Group Signatures without Revocation. After their introduction, a very prolific line of research has emerged on group signatures. The first efficient construction of group signature was given by Ateniese et al. [ACJ+00] based on both the Strong-RSA assumption and the DDH assumption in the random oracle model [BR93]. At that time, however, the security of group signatures was not very well understood and early constructions were proven secure via informal arguments using various interpretations of their requirements.

To rectify this situation, Bellare, Micciancio and Warinschi [BMW03] formalised the security definitions for static groups, in which the group members are fixed once

and for all during the setup phase. In their model, the group manager is also granted the authority of opening signatures and she is assumed to be fully trusted, except she may leak information to the adversary. Later on, Bellare, Shi and Zhang [BSZ05] and Kiayias and Yung [KY06] independently provided formal security definitions for partially dynamic groups, in which new group members can join the group at any time but cannot leave. The former model, following the suggestion of [CM98], separates the opening authority from the group manager, while the latter considers both roles overseen by the group manager. Aside from this, the main difference between the two models is that in [KY06] the group manager is trusted to give correct openings without providing proofs of attributions. More recently, Sakai et al. [SSE+12] extended the security model of partially dynamic groups by including the notion of *opening soundness*, which ensures that a valid signature only traces to one user.

Numerous efficient constructions secure in the above models have been suggested both in the random oracle model [ACJ+00; CL04; NS04; FI06; FY04; KY05; DP06; BCN+10; LLN+16; LLM+16; DS18] and in the standard one [ACH+05; Gro06; BW06; Gro07; BW07; AFG+16].

Group Signatures with Revocation. Since revocation is an essential feature of group signatures, different approaches have been proposed to remove members from the group. Bresson and Stern [BS01] realise revocation by requiring that the signer proves at the time of signing that her group membership certificate is not among those contained in a public *revocation list*. Along this line, Nakanishi et al. [NFH+10] gave an efficient scheme in the random oracle model based on revocation lists, offering constant size as well as signing and verification time. In the standard model, Libert, Peters and Yung [LPY12b; LPY12a] gave a number of efficient constructions of group signatures which use the subset cover framework [NNL01], originally used in the context of broadcast encryption, to form revocation lists.

Camenisch and Lysyanskaya [CL02] introduced *dynamic* accumulators, extending ideas of [BdM93] and [BP97], to handle efficient revocation of users. Accumulators allow the manager to give a compact representation of the set of current group members, who can then efficiently prove their membership in the accumulated set. Since then,

many other constructions adopted a similar approach to perform revocation [TX03; CG04; DKN+04; Ngu05].

Boneh, Boyen and Shacham [BBS04] showed how to achieve revocation by including information into a revocation list which allow unrevoked members to locally derive a new group public key and a new signing key. A similar approach was used both in [Son01] and [CG04] to extend [ACJ+00] scheme in order to support revocation and, in the case of [Son01], to additionally provide forward security.

Brickell [Bri04] considered a different approach to revocation known as *Verifier Local Revocation* (VLR). This was subsequently formalized by Boyen and Shacham [BS04] and further used in several constructions such as [NF07; LV09; LNR+18]. In VLR, the revocation information (i.e. revocation lists) is only sent to the verifiers (as opposed to both verifiers and signers), who can check whether a particular signature was generated by a revoked member. A similar approach is also used in Direct Anonymous Attestation (DAA) protocols [BCC04]. *Traceable Signatures* [KTY04] extend this idea, with a group manager that can release a trapdoor for each member, enabling their signatures to be traced back to the individual user.

Fully Dynamic Group Signatures from Ring Signatures. A generic approach to construct fully dynamic group signatures from accountable ring signatures was suggested in [BCC+15] and described in [BCC+16a]. Introduced by Rivest, Shamir and Tauman [RST01], ring signatures differ from group signatures by not having appointed authorities: signers decide upon which users to include into their anonymity set. Accountable ring signatures [XY04] additionally provide a way to identify the author of a signature, but differently from group signature the opening entity is chosen by the users at signing time. Bootle et al. [BCC+15] also gave an efficient instantiation of accountable ring signatures in the random oracle model based on the DDH assumption. By following [BCC+16a], this and other schemes of accountable ring signatures [LZC+16; KP17; DLO+18] can be used to obtain fully dynamic group signature schemes with efficiency similar to their ring counterpart.

Shortcomings in Existing Models. While static and partially dynamic group signatures have been properly formalised [BMW03; BSZ05; KY06; SSE+12] and their security definitions are now broadly accepted, the same cannot be said about their fully dynamic groups counterpart. Different revocation paradigms assume different, often informal, models which do not necessarily generalise to other approaches.

A first step to address this situation was done by Bootle et al. [BCC+16a], who proposed a general model for fully dynamic group signatures. Their definitions capture strong security requirements and include, when possible, security against malicious key generation for the authorities. This model was recently extended by Backes et al. [BHS18] to include two additional security properties called join and leave privacy, guaranteeing the privacy of users over their group membership status. El Kaafarani et al. [EKS18] also extended this security model in order to formalise anonymous reputation systems, which allows a set of user review products anonymously.

Despite not assuming a specific design paradigm, the model of [BCC+16a] makes several implicit assumptions regarding the functioning of group signatures, which limit its generality and applicability. For example, their model considers a fairly strong notion of traceability which implicitly restricts the way the manager can perform revocation of users, precluding the natural use of revocation lists.

In Chapter 7 we make a step further in modelling fully dynamic group signatures by considerably updating and generalising [BCC+16a]. Our model does not preclude current design approaches, it offers stringent security properties, and it exhibits the following refinements over existing models:

- We offer security definitions both in the case of a single group manager and in the case where her role is separated from the opening authority and consider, when possible, malicious key generation for the authorities.
- We abstract the requirements the model imposes on the register. Specifically, we do not assume the existence of a PKI or restrict the way the register can be instantiated.
- Our model captures a stronger correctness property, which is guaranteed even in case the system is populated with malicious users and a malicious opening authority.

- We consider a more general notion of traceability, covering both the one defined in [BCC+16a] and the one achieved by revocation list approaches.
- Our model allows concurrent joining sessions between the users and the group manager. Existing models restrict the state of the manager to be compartmentalised, thus making each joining session independent. By relaxing this, our model captures stronger notions of correctness and traceability.
- We formalise two additional security properties called *opening binding* and *opening soundness* capturing that signatures cannot be attributed, respectively, to multiple users, nor to anybody other than the legitimate signer.

Group Signatures from Post-Quantum Assumptions. Driven by the advances on quantum computing, post-quantum cryptography has received increasing attention in the recent years. This trend includes research on group signatures. Starting from the work of Gordon et al. [GKV10], progressively more efficient lattice based constructions have been proposed [CNR12; LLL+13; LNR+18; NZZ15; LNW15; LMN16; LLM+16; BCN18; LNW+18]. Among these, the only schemes supporting revocation of users are the VLR scheme of [LNR+18] and the fully dynamic group signature of [LNW+18], which is proven secure in [BCC+16a] model.

Aside from lattice assumptions, another typical approach to achieve post-quantum security is to rely only on symmetric-key primitives. In [AW04] and [CG04] it was shown, however, that fully-anonymous group signatures imply IND-CCA secure public key encryption, thus separating group signatures from symmetric-key primitives. Nonetheless, [CG04] showed that by relaxing the anonymity property it is possible to construct group signatures from one-way functions and non-interactive zero-knowledge proofs, which can be both instantiated using symmetric-key primitives. The weaker notion of anonymity achieved by this generic construction does not protect against key-exposure attacks. Following a similar approach to [CG04] and building on recent developments of zero-knowledge proofs, Boneh et al. [BEF19] proposed an efficient construction of group signatures based on symmetric-key primitives and weaker anonymity guarantees.

Chapter 3

Preliminaries and Definitions

3.1 Notation

Let S be a set, we write $s \leftarrow S$ for sampling an element s from the set S, where the sampling is assumed to be uniformly at random, unless otherwise specified. We write $y \leftarrow A(x)$ for an algorithm A that runs on input x and outputs y. We write $A^{B(z,\cdot)}(x)$ to indicate that A can execute B on a fixed input z, oblivious to A, and on an arbitrary input a to receive the output of $B(z, a)$. When an algorithm is invoked several times and keeps state between invocations, we may explicitly refer to the state st_A as an additional input and write $x \leftarrow A(a; \mathsf{st}_A)$. A simple example is a single invocation of a probabilistic algorithm that may run in two steps: it first samples randomness r, stored in the state, and then runs the rest of the algorithm $x \leftarrow A(a; r)$ deterministically. For algorithms A and B, $(x; y) \leftarrow \langle A(a); B(b)\rangle$ denotes the the joint execution of A, with input a, and B, with input b, where at the end A outputs x and B outputs y.

We use a security parameter $\lambda \in \mathbb{N}$ to indicate the desired level of security, with the intention that the higher it is, the more secure the scheme would be. In general, when we refer to polynomial time algorithms we mean that they run in polynomial time in the security parameter, and therefore we give, often implicitly, the security parameter written in unary 1^λ as input to the algorithms. We abbreviate deterministic polynomial time as DPT and probabilistic polynomial time as PPT.

Definition 3.1. *We say that a function $f : \mathbb{N} \to [0, 1]$ is* negligible *if for any constant $c > 0$ there exists λ_0 such that for all $\lambda > \lambda_0$*

$$f(\lambda) < \lambda^{-c},$$

or equivalently if $f(\lambda) = \lambda^{-\omega(1)}$. *We say that a function* $g : \mathbb{N} \to [0,1]$ *is* overwhelming *if* $1 - g$ *is negligible. Given two functions* f, g *we write* $f(\lambda) \approx g(\lambda)$ *if* $|f(\lambda) - g(\lambda)|$ *is negligible.*

For a positive integer n, $[n]$ denotes the set $\{1, \ldots, n\}$ and $[0, n]$ denotes $\{0, 1, \ldots, n\}$. We write $\Sigma_{[n]}$ for the symmetric group on the set $[n]$. We use bold letters such as $\boldsymbol{v}$ for row vectors and let $\mathbb{F}$ be a finite field. For $\boldsymbol{v} \in \mathbb{F}^n$ and a set $J = \{j_1, \ldots, j_k\} \subset [n]$ with $j_1 < \cdots < j_k$ we define the vector $\boldsymbol{v}|_J$ to be $(\boldsymbol{v}_{j_1}, \ldots, \boldsymbol{v}_{j_k})$. Similarly, for a matrix $V \in \mathbb{F}^{m \times n}$ we let $V|_J \in \mathbb{F}^{m \times k}$ be the submatrix of V restricted to the columns indicated in J.

In the next chapters we will make extensive use of the well-known Schwartz-Zippel Lemma [Sch80; Zip79] which we recall next.

Lemma 3.1 (Schwartz-Zippel)**.** *Let* $q \in \mathbb{F}[X_1, \ldots, X_n]$ *be a non-zero multivariate polynomial of total degree* d*, then*

$$\Pr\left[(x_1, \ldots, x_n) \leftarrow S^n \colon q(x_1, x_2, \ldots, x_n) = 0\right] \leq \frac{d}{|S|}$$

The typical application of this lemma is to test the identity of two polynomials. Given $q_1(X_1, \ldots, X_n), q_2(X_1, \ldots, X_n)$ it is sufficient to pick $(x_1, \ldots, x_n) \leftarrow \mathbb{F}^n$ and check if $q_1(x_1, \ldots, x_n) - q_2(x_1, \ldots, x_n) = 0$. As we are interested in arguing that pairs of polynomials are identical with overwhelming probability, we set $S = \mathbb{F}$ for a finite field of superpolynomial size, i.e. $|\mathbb{F}| = \lambda^{\omega(1)}$, and restrict our attention to polynomials with total degree polynomial in the security parameter, i.e. $d = \mathrm{poly}(\lambda)$.

3.2 Models of Computations

In the next chapters we will give proofs of knowledge for statements of the form "I know some x such that y is equal to $f(x)$", where the instance f represents some computation on a private input x. A natural way to express any computation is by using circuits. However, in practice one may be interested in statements that are not immediately expressed in this form, e.g. the execution of a computer program. Alongside circuits we thus consider RAM programs, which model computation and memory access more closely to how computers works. Concretely, in Chapters 4 and 5 we will

give proofs of knowledge for the satisfiability of arithmetic circuits and the correct execution of TinyRAM programs. When stating the efficiency of prover and verifier, we will consider them to be implemented as either arithmetic circuits or TinyRAM programs, with the aim of measuring the overhead incurred by prover and verifier with respect to directly evaluate the circuit or execute the program.

3.2.1 Arithmetic Circuits

An arithmetic circuit over a field $\mathbb{F}$ is a directed acyclic graph whose vertices are called gates. Gates of in-degree 0 are called input gates, which are associated with variables x_i over $\mathbb{F}$ or with constant field elements. All other gates in the circuit are either multiplication gates or addition gates, which are labelled as $\otimes$ and $\oplus$, respectively. Here we only consider circuits consisting of fan-in 2 gates, i.e. all multiplication and addition gates have in-degree equal to 2, but we allow for arbitrary fan-out.

Arithmetic circuits naturally compute polynomials over a field $\mathbb{F}$. In the example circuit in Figure 3.1, the output y_1 corresponds to the polynomial $7x_1(x_2 + 3)$.

The *size* of a circuit refers to the total number of addition and multiplication gates, while the *depth* is the longest path between inputs and outputs. The width of a circuit usually refers to the maximum number of gates in the same layer of the circuit. The example circuit in Figure 3.1 has size equal to 7, depth equal to 5 and width equal to 3. The size of the circuit corresponds to the total number of operations required to evaluate it, while the depth is the minimal number of sequential operations required to evaluate the circuit by using as many parallel processes as the width of the circuit.

3.2.2 TinyRAM

TinyRAM is a random-access machine architecture operating on W-bit words and using K registers. Here we describe the key features of TinyRAM and we refer the reader to [BCG+13b] for the full specifications. A state of the TinyRAM machine consists of the following:

- A program P consisting of a list of L instructions.
- A program counter pc storing, in a W-bit word, the line number of the program instruction to be executed.

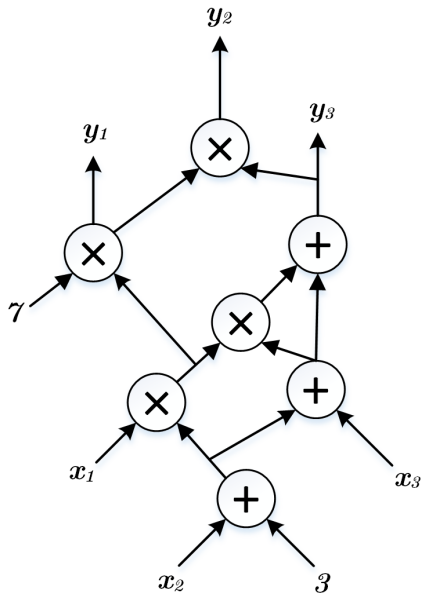

FIGURE 3.1: Example of an arithmetic circuit.

- The content of the K registers $\mathsf{reg}_0, \ldots, \mathsf{reg}_{K-1}$, storing a W-bit word each.
- A condition flag flag of size one bit.
- M words of memory each of size W-bit, and with addresses $0, \ldots, M-1$.

The TinyRAM specification includes two read-only tapes to retrieve its inputs but with little loss of efficiency we may assume the program starts by reading the tapes into memory. The specification [BCG+13b] calls a program *proper* if it starts with a preamble that reads all inputs into memory. As shown in [BCG+13b] this can be done by an 8-line TinyRAM program in $\sim 5v$ steps, for an input tape of length v. We will therefore skip the reading phase and assume the memory is initialised with the inputs (and 0 for the remaining words). Also, we will assume on initialisation that pc, and the registers are set equal to 0^W and flag equal to 0.

The TinyRAM instruction set consists of 27 instructions that include bit-wise logical operations, arithmetic operations, shifts, comparisons, jumps, and storing and loading data in memory. We recall the list of allowed instructions in Table 3.1, from which we removed the **read** instruction. Note that with respect to the original instruction set of [BCG+13b] the flag of the multiplication operations **mull**, **umulh**, **smulh** is

flipped: the flag is set equal to 0 to denote overflow or underflow[1].

Instruction	Operands	Effect	Flag
and	reg_i reg_j A	compute r_i as bitwise AND of r_j and A	result is 0^W
or	reg_i reg_j A	compute r_i as bitwise OR of r_j and A	result is 0^W
xor	reg_i reg_j A	compute r_i as bitwise XOR of r_j and A	result is 0^W
not	reg_i A	compute r_i as bitwise NOT of A	result is 0^W
add	reg_i reg_j A	compute $\text{r}_i = \text{r}_j + \text{A} \bmod 2^W$	overflow: $\text{r}_j + \text{A} \geq 2^W$
sub	reg_i reg_j A	compute $\text{r}_i = \text{r}_j - \text{A} \bmod 2^W$	borrow: $\text{r}_j < \text{A}$
mull	reg_i reg_j A	compute $\text{r}_i = \text{r}_j \times \text{A} \bmod 2^W$	$\neg$ overflow: $\text{r}_j \times \text{A} < 2^W$
umulh	reg_i reg_j A	compute r_i as upper W bits of $\text{r}_j \times \text{A}$	$\neg$ overflow: $\text{r}_i = 0$
smulh	reg_i reg_j A	compute r_i as upper W bits of the signed $2W$-bit $\text{r}_j \times_s \text{A}$ (**mull** gives lower word)	$\neg$ over/underflow: $\text{r}_i = 0$
udiv	reg_i reg_j A	compute r_i as quotient of r_j/A	division by zero: $\text{A} = 0$
umod	reg_i reg_j A	compute r_i as remainder of r_j/A	division by zero: $\text{A} = 0$
shl	reg_i reg_j A	compute r_i as r_j shifted left by A bits	MSB of r_j
shr	reg_i reg_j A	compute r_i as r_j shifted right by A bits	LSB of r_j
cmpe	reg_i A	compare if equal	equal: $\text{r}_i = \text{A}$
cmpa	reg_i A	compare if above	above: $\text{r}_i > \text{A}$
cmpae	reg_i A	compare if above or equal	above/equal: $\text{r}_i \geq \text{A}$
cmpg	reg_i A	signed compare if greater	greater: $\text{r}_i >_s \text{A}$
cmpge	reg_i A	signed compare if greater or equal	greater/equal: $\text{r}_i \geq_s \text{A}$
mov	reg_i A	set $\text{r}_i = \text{A}$	flag unchanged
cmov	reg_i A	if flag $= 1$ set $\text{r}_i = \text{A}$	flag unchanged
jmp	A	set pc $= A$	flag unchanged
cjmp	A	if flag $= 1$ set pc $=$ A	flag unchanged
cnjmp	A	if flag $= 0$ set pc $=$ A	flag unchanged
store	A reg_i	store in memory address A the word r_i	flag unchanged
load	reg_i A	set r_i to the word stored at address A	flag unchanged
answer	A	stall or halt returning the word A	flag unchanged

TABLE 3.1: TinyRAM instruction set, excluding the **read** command. The flag is set equal to 1 if the condition is met and 0 otherwise. If the pc exceeds the program length, i.e., pc $\geq L$, or we address a non-existing part of memory, i.e., in a **store** or **load** instruction A $\geq M$, the TinyRAM machine halts with answer 1.

A program consists of a sequence of L instructions, each taking up to three operands, e.g.

$$\textbf{add}\ \text{reg}_i\ \text{reg}_j\ A$$

The first operand, reg_i, usually points to the register storing the result of the operation, **add**, computed on the words specified by the next two operands, reg_j, A. The last operand A indicates an immediate value that could be either used directly in the operation or to point to the content of another register. The program terminates by using a special command **answer** that returns a word. We consider the program to have succeeded if it answers 0, otherwise we consider the answer to be a failure code.

[1]This simplifies the consistency check of the flag in Chapter 5. This modification can be avoided by introducing few additional constraints to check the consistency of the flag with the result of the instruction.

We write reg_i and r_i when referring to register i and to its content, respectively. We write A to refer to either a register or an immediate value specified in a program instruction and write A for the value stored therein. Depending on the instruction a word $\mathtt{a}$ may be interpreted as an unsigned value in $\{0, \ldots, 2^W - 1\}$ or as a signed value in $\{-2^{W-1}, \ldots, 2^{W-1} - 1\}$, which is represented in two's complement. Given a word $(a_{W-1}, \ldots, a_0) \in \{0,1\}^W$, the unsigned value is

$$\mathtt{a} = \sum_{i=0}^{W-1} a_i 2^i$$

while the signed value corresponding to the same word is

$$\mathtt{a}_\sigma = -a_{W-1}2^{W-1} + \sum_{i=0}^{W-2} a_i 2^i = -a_{W-1}2^W + \mathtt{a}$$

and a_{W-1} is used to encode the sign of the value. We distinguish operations over signed values by using subscript s, e.g. $\mathtt{a} \times_s \mathtt{b}$ and $\mathtt{a} \geq_s \mathtt{b}$ are used to denote product and comparison over the signed values, respectively.

Implementing Field Arithmetic in TinyRAM. In our proof systems prover and verifier perform computation in a large finite field $\mathbb{F}$ and it will be convenient express their efficiency in terms of TinyRAM operations. Typically, we have that $|\log(\mathbb{F})| > W$ and thus field elements require $e = \left\lceil \frac{\log(|\mathbb{F}|)}{W} \right\rceil$ words to be stored. Field addition can then be computed with $\mathcal{O}(e)$ TinyRAM operations. The cost of field multiplication depends on the implemented algorithm: schoolbook multiplication costs $\mathcal{O}(e^2)$, while using FFT it can be reduced up to $\mathcal{O}(e \log(e) \log\log(e))$ TinyRAM operations.

3.3 Proof of Knowledge and the ILC Channel

A *proof system* is defined by a triple of stateful PPT algorithms $(\mathcal{G}, \mathcal{P}, \mathcal{V})$, which we call the setup *generator*, the *prover* and *verifier*, respectively. The setup generator $\mathcal{G}$ creates public parameters pp that will be used by the prover and the verifier. We think of pp as being honestly generated, however, in the proofs we construct in the next chapters pp consists of parts that are either publicly verifiable or could be generated by the verifier.

The reason we place our definitions in the public parameter model is to simplify the exposition and to improve the efficiency of our proofs, not for security.

3.3.1 Relations and Languages

Let $\mathcal{R}$ be a polynomial time decidable ternary relation consisting of tuples (pp, u, w). Please note the special case where $\mathcal{R}$ ignores pp, in which case we have the standard definition of an NP-relation. We define the language corresponding to the relation $\mathcal{R}$ to be

$$\mathcal{L}_{\mathcal{R}} = \left\{ (pp, u) \colon \exists\, w,\ (pp, u, w) \in \mathcal{R} \right\}$$

We refer to u as the *instance* and w as the *witness* for the membership of the instance in the language, i.e. $(pp, u) \in \mathcal{L}_{\mathcal{R}}$. The *public parameter* pp will specify the security parameter λ, perhaps implicitly through its length, and may also contain other parameters used for specifying the relation, e.g. a description of a field. Typically, pp will also contain parameters that do not influence membership of $\mathcal{R}$ but may aid the prover and verifier, such as the description of an encoding function that they will use.

3.3.2 Communication Channels

In a proof system, $(\mathcal{P}, \mathcal{V})$ is a pair of interactive algorithms communicating with each other through a *communication channel* $\overset{\text{chan}}{\longleftrightarrow}$. We write $\langle \mathcal{P}(pp, u, w) \overset{\text{chan}}{\longleftrightarrow} \mathcal{V}(pp) \rangle$ to denote their interaction mediated by the channel. We let $\mathsf{view}_{\mathcal{V}} \leftarrow \langle \mathcal{P}(pp, u, w) \overset{\text{chan}}{\longleftrightarrow} \mathcal{V}(pp) \rangle$ be the view of the verifier in the execution, i.e., all his inputs and outputs, including his random coins and messages exchanged with the channel. Similarly, we let $\mathsf{tran}_{\mathcal{P}} \leftarrow \langle \mathcal{P}(pp, u, w) \overset{\text{chan}}{\longleftrightarrow} \mathcal{V}(pp) \rangle$ indicate the transcript of the communication between prover and channel. At the end of the interaction the verifier returns a bit b to denote he accepts ($b = 1$) or rejects ($b = 0$). Without loss in generality, we let the prover return $\mathsf{tran}_{\mathcal{P}}$, and write $(\mathsf{tran}_{\mathcal{P}}; b) \leftarrow \langle \mathcal{P}(pp, u, w) \overset{\text{chan}}{\longleftrightarrow} \mathcal{V}(pp) \rangle$ for the entire execution of the protocol.

In the *standard channel* $\longleftrightarrow$, all messages are forwarded between prover and verifier. In this case we may drop $\longleftrightarrow$ from the above notation. We also consider an *ideal linear commitment* channel, $\overset{\mathsf{ILC}}{\longleftrightarrow}$, or simply ILC, described in Figure 3.2. The ILC channel is defined by a field $\mathbb{F}$ and an integer $k \in \mathbb{N}$, both specified in $pp_{\mathsf{ILC}} \leftarrow \mathcal{G}_{\mathsf{ILC}}(1^{\lambda})$.

When using the ILC channel, the prover can submit a **commit** command to the channel to commit to vectors $\boldsymbol{v} \in \mathbb{F}^k$. The vectors remain secretly stored in the channel, and will not be forwarded to the verifier. Instead, the verifier only learns how many vectors the prover has committed to. The verifier can submit a **send** command to the ILC to send field elements $x \in \mathbb{F}$ to the prover. In addition, the verifier can also submit **open** queries to the ILC to obtain the opening of any linear combinations of the vectors sent by the prover. We stress that the verifier can request several linear combinations within a single **open** query, as depicted in Figure 3.2.

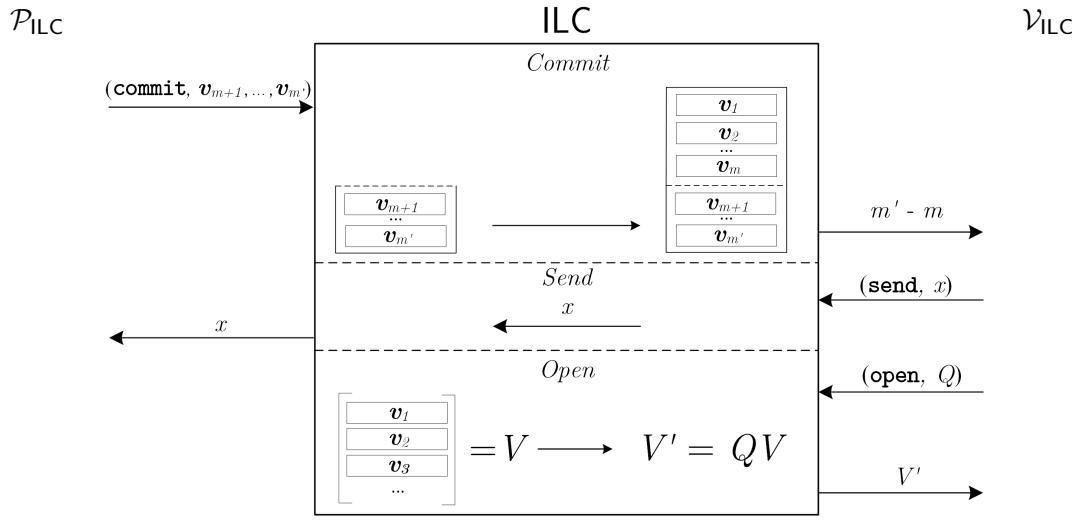

FIGURE 3.2: Description of the ILC channel.

LPCP and LIP. Proof systems over the ILC share similarities with other information-theoretic models, such as linear probabilistic checkable proofs (LPCP), introduced by [IKO07], and linear interactive proofs (LIP), introduced by [BCI+13]. In an LPCP the verifier sends queries to a PCP oracle consisting of vectors of field elements. The oracle responds to each query with a single field element obtained by applying the *same* linear function to the queried vector. A LIP is an interactive proof system where both the prover and verifier send vectors of field elements, but the prover can only send linear (or affine) transformations of the verifier's previously sent vectors. However, for our constructions it is important that the prover can compute on field elements sent by the verifier and, for instance, evaluate polynomials.

3.3.3 Proof of Knowledge

We say a proof system is *public coin* if the verifier's messages to the communication channel are chosen uniformly at random and independently of the actions of the prover, i.e., the verifier's messages to the prover correspond to the verifier's randomness ρ. All our proof systems will be public coin. In a proof system over the ILC channel, sequences of **commit**, **send** and **open** queries can alternate arbitrarily. However, since our proof systems are public coin we can without loss of generality assume the verifier does one large **open** query at the end of the protocol and then decides whether to accept or reject. We sometimes refer to a proof system over the ILC for which the verifier only makes a single query at the end of the proof as *non-adaptive*.

Definition 3.2 (Proof of Knowledge)**.** *A proof system $(\mathcal{G}, \mathcal{P}, \mathcal{V})$ for a relation $\mathcal{R}$ is called a* proof (argument) of knowledge *over a communication channel $\overset{\text{chan}}{\longleftrightarrow}$ if it has perfect completeness and statistical (computational) knowledge soundness as defined below.*

Completeness. Given an instance in the language, completeness guarantees that if both prover and verifier follow the specifications of the proof system, then the verifier should accept the proof.

Definition 3.3 (Perfect Completeness)**.** *A proof system $(\mathcal{G}, \mathcal{P}, \mathcal{V})$ for a relation $\mathcal{R}$ is* perfectly complete *if for all PPT adversaries $\mathcal{A}$*

$$\Pr\left[\begin{array}{c} pp \leftarrow \mathcal{G}(1^\lambda); (u, w) \leftarrow \mathcal{A}(pp); (\mathsf{tran}_{\mathcal{P}}; b) \leftarrow \langle \mathcal{P}(pp, u, w) \overset{\text{chan}}{\longleftrightarrow} \mathcal{V}(pp, u)\rangle\textbf{:} \\ (pp, u, w) \notin \mathcal{R} \ \vee \ b = 1 \end{array}\right] = 1$$

Knowledge Soundness. Soundness guarantees that a cheating prover should not be able to convince the verifier about the validity of a false statement. Knowledge soundness in addition guarantees that if a prover can convince the verifier about the validity of a statement, then she should know a valid witness for it.

Definition 3.4 ((Strong) Knowledge Soundness)**.** *A proof system $(\mathcal{G}, \mathcal{P}, \mathcal{V})$ for a relation $\mathcal{R}$ has* computational (strong) knowledge soundness *if for all DPT $\mathcal{P}^*$ there exists an*

expected PPT extractor $\mathcal{E}$ such that for all PPT adversaries $\mathcal{A}$

$$\Pr\left[\begin{array}{c} pp \leftarrow \mathcal{G}(1^\lambda); (u,s) \leftarrow \mathcal{A}(pp); w \leftarrow \mathcal{E}^{\langle \mathcal{P}^*(pp,u;s) \overset{\text{chan}}{\longleftrightarrow} \mathcal{V}(pp,u)\rangle}(pp,u): \\ (pp,u,w) \notin \mathcal{R} \ \wedge \ b = 1 \end{array}\right] \approx 0$$

Here the oracle $\langle \mathcal{P}^*(pp,u;s) \overset{\text{chan}}{\longleftrightarrow} \mathcal{V}(pp,u)\rangle$ runs a full protocol execution and if the proof is successful it returns the transcript $\mathsf{tran}_{\mathcal{P}^*}$ of the prover's communication with the channel. The extractor $\mathcal{E}$ can ask the oracle to rewind the proof to any point in a previous transcript and execute the proof again from this point on with fresh public-coin challenges from the verifier. We let $b \in \{0,1\}$ be the verifier's output in the first oracle execution, i.e., whether it accepts or not, and we think of s as the state of the prover, including its randomness.

If the definition holds also for unbounded $\mathcal{P}^*$ and $\mathcal{A}$ we say the proof has (strong) statistical knowledge soundness.

If the definition holds for a non-rewinding extractor, i.e., $\mathcal{E}$ only requires a single transcript of the prover's communication with the channel, we say the proof system has knowledge soundness with straight-line extraction.

The definition can be paraphrased as saying that if the prover in state s makes a convincing proof, then $\mathcal{E}$ can extract a witness for the instance u. Therefore, knowledge soundness implies soundness as if $\mathcal{E}$ extracts a witness for u, then the instance is clearly in the language.

We note that in the definition of knowledge soundness in the ILC model, the output of the transcript oracle includes the messages the prover sent to the ILC. Since the commitment channel behaves as an ideal functionality, it is guaranteed that the prover knows the content of the committed messages. In this setting, it is sufficient for the extractor to show that if the prover does not know a witness, then the committed messages will lead to an invalid transcript with overwhelming probability. Looking ahead to the following chapters, we will show how to compile a proof of knowledge over the ILC into one over the standard channel by instantiating the ideal commitment with a concrete commitment scheme. In this way we can separate the statistical properties of the proof of knowledge constructed over the ILC from the computational properties introduced by the different instantiations of the commitment scheme.

Our definition above is stronger than the usual definition of knowledge soundness in the public parameter model [DF02]. The difference is that, in standard definition, the knowledge extractor can interact freely with the cheating prover, while in our definition we restrict $\mathcal{E}$ to observe accepting transcripts obtained from the interaction of the cheating prover with the honest verifier. We recall the standard definition of knowledge soundness following the C-style formulation of [Unr12]. It is straightforward to see that the definition above implies the following.

Definition 3.5 (Knowledge soundness)**.** *A proof system $(\mathcal{G}, \mathcal{P}, \mathcal{V})$ for a relation $\mathcal{R}$ has* computational knowledge soundness *with knowledge error $\kappa(\lambda)$ if for all DPT $\mathcal{P}^*$ there exists an expected PPT extractor $\mathcal{E}$ and such that for all PPT adversaries $\mathcal{A}$*

$$\Pr\left[\begin{array}{c} pp \leftarrow \mathcal{G}(1^\lambda); (u,s) \leftarrow \mathcal{A}(pp); w \leftarrow \mathcal{E}^{\mathcal{P}^*(pp,u;s)}(pp,u): \\ (pp,u,w) \in \mathcal{R} \end{array}\right] + \kappa(\lambda) \;\geq$$

$$\Pr\left[\begin{array}{c} pp \leftarrow \mathcal{G}(1^\lambda); (u,s) \leftarrow \mathcal{A}(pp); (\mathsf{tran}_{\mathcal{P}^*}; b) \leftarrow \langle \mathcal{P}^*(pp,u;s) \overset{\text{chan}}{\longleftrightarrow} \mathcal{V}(pp,u)\rangle: \\ b = 1 \end{array}\right]$$

If the definition holds also for unbounded $\mathcal{P}^*$ and $\mathcal{A}$ we say the proof has statistical knowledge soundness*.*

Special Honest Verifier Zero-Knowledge. A proof of knowledge is *zero-knowledge* if it does not reveal any information about the witness apart from what can be inferred from the validity of the statement. In the next chapters we will show public-coin proofs of knowledge that have *special honest-verifier zero-knowledge*. This means that if the verifier's challenges are known, or even adversarially chosen, then it is possible to simulate the verifier's view without the witness. In other words, the simulator works for verifiers who may use non-uniform random coins in choosing their challenges but that follow the specification of the protocol as an honest verifier would.

Definition 3.6 (Special Honest-Verifier Zero-Knowledge)**.** *A public-coin proof of knowledge $(\mathcal{G}, \mathcal{P}, \mathcal{V})$ for a relation $\mathcal{R}$ is* computationally special honest-verifier zero-knowledge

(SHVZK) *if there exists a PPT simulator $\mathcal{S}$ such that for all stateful interactive PPT adversaries $\mathcal{A}$ that output randomness ρ for the verifier,*

$$\Pr\left[\begin{array}{c} pp \leftarrow \mathcal{G}(1^\lambda); (u, w, \rho) \leftarrow \mathcal{A}(pp); \mathsf{view}_{\mathcal{V}} \leftarrow \mathcal{S}(pp, u, \rho): \\ (pp, u, w) \in R \ \wedge \ \mathcal{A}(\mathsf{view}_{\mathcal{V}}) = 1 \end{array}\right] \approx$$

$$\Pr\left[\begin{array}{c} pp \leftarrow \mathcal{G}(1^\lambda); (u, w, \rho) \leftarrow \mathcal{A}(pp); \mathsf{view}_{\mathcal{V}} \leftarrow \langle \mathcal{P}(pp, u, w) \overset{\text{chan}}{\longleftrightarrow} \mathcal{V}(pp, u; \rho)\rangle: \\ (pp, u, w) \in R \ \wedge \ \mathcal{A}(\mathsf{view}_{\mathcal{V}}) = 1 \end{array}\right]$$

We say the proof is statistically SHVZK *if the definition holds also against unbounded $\mathcal{A}$, and we say the proof is* perfect SHVZK *if the probabilities are exactly equal.*

From Special Honest-Verifier to General Zero-Knowledge. Special honest-verifier zero-knowledge only guarantees the simulator works for verifiers following the proof system specifications. Generally, SHVZK may not give a sufficient level of privacy and it may be preferable to consider general zero-knowledge, i.e. where the simulator works for arbitrary malicious verifier strategies. Nonetheless, constructing efficient SHVZK proofs is considered an important step towards the realisation of general zero-knowledge proofs as there exists several transformations that allow to convert one to the other with little impact on their efficiency. Here we recall few of these transformations intended for generic public coin proof systems.

In the Fiat-Shamir transform [FS86] the verifier's challenges are computed using a cryptographic hash function applied to the transcript up to the challenge. The Fiat-Shamir transform is more generally used to turn a public-coin proof into a non-interactive one. Since interaction with the verifier is no longer needed, general zero-knowledge is immediately achieved. If the hash function can be computed in linear time in the input size, then the Fiat-Shamir transform only incurs an additive linear overhead in computation for the prover and verifier. The drawback of the Fiat-Shamir transform is that security is usually proved in the random oracle model [BR93] where the hash function is modelled as an ideal random function.

In the common reference string model and using a trapdoor commitment scheme, Damgård [Dam00] gives a transformation yielding concurrently secure protocols for Σ-Protocols, i.e. preserving zero-knowledge even in the case of several concurrent

executions of the same proof system. The idea is for the prover to commit to its first message using a trapdoor commitment scheme, which is then opened in the last move of the protocol. The transformation was optimized by Groth [Gro04] using the idea that for each public-coin challenge x, the prover first commits to a value x', then the verifier sends a value x'', after which the prover opens the commitment and uses the challenge $x = x' + x''$. The coin-flipping can be interleaved with the rest of the proof, which means the transformation preserves the number of rounds and only incurs a very small efficiency cost to do the coin-flipping for the challenges.

If one does not wish to rely on a common reference string for security, one can use a private-coin transformation where the verifier does not reveal the random coins used to generate the challenges sent to the prover (hence the final protocol is no longer public coin). One example is the Micciancio and Petrank [MP03] transformation (yielding concurrently secure protocols) while incurring a small overhead of $\omega(\log \lambda)$ with respect to the number of rounds as well as the computational and communication cost in each round. The transformation preserves the soundness and completeness errors of the original protocol; however, it does not preserve statistical zero-knowledge as the obtained protocol only has computational zero-knowledge.

There are other public-coin transformations to general zero-knowledge e.g. Goldreich et al. [GSV98]. The transformation relies on a random-selection protocol between the prover and verifier to specify a set of messages and it restricts the verifier to choose challenges from this set. However, the transformed proof only achieves inverse polynomial soundness error, and thus requires $\omega(1)$ sequential repetitions to reduce the soundness error to a negligible function.

3.3.4 Efficiency Measures

There are several metrics to consider when assessing the efficiency of proofs of knowledge. The main four parameters we will consider are round complexity, communication complexity, prover's and verifier's computation.

The round complexity μ measures the number of messages between prover and verifier. A *move* is a single message going from one party to the other and a *round* consists of a pair of moves, one from the prover to the verifier and one from the verifier to the prover. The last round of the proof system may consist of a single move from

the prover to the verifier. In the ILC channel a prover's move consists of sending a **commit** command to the ILC, which forwards the number of committed vectors to the verifier, and a verifier's move consists of sending a **send** command to the the ILC, which forwards the message to the prover.

The communication complexity consists of the overall size of the messages sent during the protocol and we measure it in either the number of field elements of size $\log|\mathbb{F}|$ or the number of W-bits words. In the ILC channel we separate it into *prover's communication*, counting the number of vectors sent by the prover to the ILC channel, and verifier's communication, counting the number of **send** messages the verifier forwards to the prover. In addition to these we consider the *query complexity* qc counting the number of **open** queries made by the verifier, i.e. the number of linear combinations the ILC is being asked to open.

The prover's computation $T_{\mathcal{P}}$ and verifier's computation $T_{\mathcal{V}}$ correspond to the number of operations performed by the two parties, respectively. We measure these in either the number of field operations or TinyRAM steps.

3.4 Linear-Time Linear Error-Correcting Codes

A *code* over an alphabet Σ is a subset $\mathcal{C} \subseteq \Sigma^n$. A code $\mathcal{C}$ is associated with an encoding function $E_{\mathcal{C}} : \Sigma^k \to \Sigma^n$ mapping messages of length k into *codewords* of length n. We assume there is a setup algorithm $\mathsf{Gen}_{E_{\mathcal{C}}}$ which takes as input the description of an alphabet Σ and the parameter $k \in \mathbb{N}$, and it outputs an encoding function $E_{\mathcal{C}}$, which is represented as either an arithmetic circuit or as a TinyRAM program.

In what follows, we restrict our attention to $\mathbb{F}$-*linear codes* for which the alphabet is a finite field $\mathbb{F}$, the code $\mathcal{C}$ is a k-dimensional linear subspace of $\mathbb{F}^n$, and $E_{\mathcal{C}}$ is an $\mathbb{F}$-linear map. The *rate* of the code is defined to be $\frac{k}{n}$. The *Hamming distance* between two vectors $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{F}^n$ is denoted by $\mathsf{hd}(\boldsymbol{x}, \boldsymbol{y})$ and corresponds to the number of coordinates in which $\boldsymbol{x}, \boldsymbol{y}$ differ. The *(minimum) distance* of a code is defined to be the minimum Hamming distance $\mathsf{hd}_{\min}$ between distinct codewords in $\mathcal{C}$. We denote by $[n, k, \mathsf{hd}_{\min}]_{\mathbb{F}}$ a linear code over $\mathbb{F}$ with length n, dimension k and minimum distance $\mathsf{hd}_{\min}$. The *Hamming weight* of a vector $\boldsymbol{x}$ is $\mathsf{wt}(\boldsymbol{x}) = |\{i \in [n] : \boldsymbol{x}_i \neq 0\}|$.

In the next sections, we will use families of linear codes achieving asymptotically good parameters. More precisely, we require codes with linear length, $n = \Theta(k)$, and linear distance, $\mathsf{hd}_{\min} = \Theta(k)$, in the *dimension* k of the code. We recall that random linear codes achieve with high probability the best trade-off between distance and rate. However, in this work we are particularly concerned with computational efficiency of the encoding procedure and random codes are not known to be optimal in terms of efficiency.

To obtain zero-knowledge proofs and arguments with linear cost for the prover, we need to use codes that can be encoded in linear time. Starting from the seminal work of Spielman [Spi96], there has been a rich stream of research [GI01; GI02; GI03; GI05; DI14; CDD+16] regarding linear codes with linear-time encoding. In our proofs, we can employ one of the families of codes presented by Druk and Ishai [DI14]. These are defined over a generic finite field $\mathbb{F}$ and meet all the above linearity requirements.

Theorem 3.1 ([DI14])**.** *There exist constants $c_1 > 1$ and $c_2 > 0$ such that for every finite field $\mathbb{F}$ there exists a family of $[\lceil c_1 k \rceil, k, \lfloor c_2 k \rfloor]_{\mathbb{F}}$ linear codes. These can be encoded by a uniform family of linear-size arithmetic circuits, consisting of $\mathcal{O}(k)$ addition gates.*

The above codes, as the all the other families of linear-time encodable codes, use bipartite expander graphs which can be instantiated using the explicit construction of [CRV+02]. By representing the encoding function as an arithmetic circuit, the size of the circuit is of about $6Dk$ addition gates, where $D = \mathcal{O}(1)$ is the left-regularity of the expander graph. If we represent the encoding function as a TinyRAM program, encoding requires about $24Dke$ steps and $4(D+1)ke$ words of memory, where $e = \frac{\log(|\mathbb{F}|)}{W}$ is the number of words required to store a field element.

In [DI14], Druk and Ishai suggested another family of codes which achieves better parameters, meeting the Gilbert-Varshamov bound. This family can be still encoded in linear time and would make our proofs and arguments more efficient, but without having an impact on their asymptotic complexity. Differently from the one recalled above, this construction is probabilistic. We opt for the explicit construction of Theorem 3.1 which enables to verify the correct generation of the code.

3.5 Commitment Schemes

A non-interactive commitment scheme allows a sender to commit to a secret message and to later reveal the message in a verifiable way. Here we are interested in commitment schemes that take an arbitrary length message as input, so the message space is $\{0,1\}^*$. A commitment scheme is defined by a pair of PPT algorithms $(\mathsf{CSetup}, \mathsf{CCommit})$.

$\mathsf{CSetup}(1^\lambda) \to ck$**:** it receives the security parameter as input and returns a commitment key ck.

$\mathsf{CCommit}_{ck}(m) \to c$**:** given a message $m \in \{0,1\}^*$, it picks randomness $r \leftarrow \{0,1\}^{\mathrm{poly}(\lambda)}$ and computes a commitment $c = \mathsf{CCommit}_{ck}(m;r)$.

A commitment scheme must satisfy *binding*, in the sense that it should be infeasible to open a commitment to two distinct messages. A commitment scheme must be *hiding*, meaning that the commitment does not reveal anything about the committed message.

Definition 3.7 (Binding)**.** *A commitment scheme is* computationally binding *if for all PPT adversaries* $\mathcal{A}$

$$\Pr\left[\begin{array}{c} ck \leftarrow \mathsf{CSetup}(1^\lambda);\ (m_0, r_0, m_1, r_1) \leftarrow \mathcal{A}(ck) : \\ m_0 \neq m_1 \ \wedge\ \mathsf{CCommit}_{ck}(m_0; r_0) = \mathsf{CCommit}_{ck}(m_1; r_1) \end{array}\right] \approx 0.$$

If this holds also for unbounded adversaries, we say the commitment scheme is statistically binding.

Definition 3.8 (Hiding)**.** *A commitment scheme is* computationally hiding *if for all PPT stateful adversaries* $\mathcal{A}$

$$\Pr\left[\begin{array}{c} ck \leftarrow \mathsf{CSetup}(1^\lambda);\ (m_0, m_1) \leftarrow \mathcal{A}(ck);\ b \leftarrow \{0,1\};\ c \leftarrow \mathsf{CCommit}_{ck}(m_b) : \\ \mathcal{A}(c) = b \end{array}\right] \approx \frac{1}{2},$$

where $\mathcal{A}$ *outputs messages of equal length* $|m_0| = |m_1|$*. If the definition holds also for unbounded adversaries, we say the commitment scheme is* statistically hiding.

We will be interested in using highly efficient commitment schemes. We say a commitment scheme is *linear-time* if the time to compute $\mathsf{CCommit}_{ck}(m)$ is $\mathrm{poly}(\lambda) + \mathcal{O}(|m|)$

bit operations, which we assume corresponds to $\mathrm{poly}(\lambda) + \mathcal{O}(\frac{|m|}{W})$ TinyRAM operations. We will also be interested in having small size commitments. We say a commitment scheme is *compact* if there is a polynomial $p(\lambda)$ such that commitments have size at most $p(\lambda)$ regardless of how long the message is. We say a commitment scheme is *public coin* if there is a polynomial $\ell(\lambda)$ such that $\mathsf{CSetup}(1^\lambda)$ picks the commitment key uniformly at random as $ck \leftarrow \{0,1\}^{\ell(\lambda)}$. We will now discuss some candidate linear-time commitment schemes.

Statistically Hiding Commitments. Applebaum et al. [AHI+17b] constructed low-complexity families of collision-resistant hash functions, where it is possible to evaluate the hash function in linear time in the message size. Their construction is based on the binary shortest vector problem (bSVP) assumption, which is related to finding non-trivial low-weight vectors in the null space of a matrix over $\mathbb{F}_2$. Collision resistant hash functions can be coupled with universal hash functions [CW77] to obtain compact statistically hiding and computationally binding commitment schemes, as shown by Halevi and Micali [HM96]. Their transformation is very efficient and it only requires one application of the universal hash function on a short input[2] and two evaluations of the collision resistance hash function. Thus, if we instantiate it with the collision resistant hash function with [AHI+17b], we obtain a compact linear-time public-coin statistically hiding commitment scheme.

Statistically Binding Commitments. Ishai et al. [IKO+08] proposed linear-time computable pseudorandom generators (PRGs) with arbitrary polynomial stretch based on the PRGs with linear stretch in NC^0 of Applebaum et al. [AIK08], which rely on the intractability assumptions of decoding sparsely generated linear codes [Ale11]. Moreover, [IKO+08] suggested how to use these PRG to obtain linear-time statistically binding commitment schemes. Given a (not linear-time) statistically binding commitment scheme $\mathsf{CCommit}$, to commit to a long message m one first picks a short seed s for the pseudorandom generator, stretches it to $t = \mathrm{PRG}(s)$ such that $|t| = |m|$ and computes

$$\mathsf{CCommit}'(m) := (\mathsf{CCommit}_{ck}(s), t \oplus m)$$

[2]More precisely, the construction requires to pick a random universal hash function mapping a random string to the digest of the message. The cost of this step is usually comparable to one evaluation of the hash function, as in the constructions suggested in [CW77; HM96]

Assuming that the base commitment scheme $\mathsf{CCommit}$ is statistically binding and PRG computable in linear time, the resulting commitment $\mathsf{CCommit}'$ is a linear-time statistically binding commitment scheme for messages of arbitrary length. It can also easily be seen that commitments have the same length as the message, plus an additive polynomial overhead that depends only on the security parameter. The construction also preserves the public-coin property of the internal commitment scheme.

Chapter 4

Proofs for the Satisfiability of Arithmetic Circuits in the ILC Model

In this chapter we give efficient zero-knowledge proofs of knowledge for the satisfiability of arithmetic circuits in the ILC model. We recall that in this model, which we introduced in Section 3.3, prover and verifier interact using an Ideal Linear Commitment communication channel, or ILC. The ILC allows the prover to **commit** to vectors of length k over a finite field $\mathbb{F}$ by sending them to the channel. The ILC channel stores the received vectors and communicates the number of vectors it received to the verifier. The verifier can **send** messages in $\mathbb{F}$ to the prover via the channel and can query the ILC to **open** arbitrary linear combinations of the committed vectors sent by the prover. The field $\mathbb{F}$ and the vector length k are determined by the public parameters $pp_{\mathsf{ILC}} \leftarrow \mathcal{G}_{\mathsf{ILC}}(1^\lambda)$.

The efficiency of our proof system is affected by the parameter k. For an arithmetic circuit with N addition and multiplication gates the best efficiency is achieved when $k \approx \sqrt{N}$. In our proofs both prover and verifier computational costs are linear in the size of the circuit. More precisely, the prover has to compute $\mathcal{O}(N)$ field multiplications thus she achieves constant overhead with respect to the direct evaluation of the circuit. On the other hand, the computational cost of the verifier is only $\mathcal{O}(N)$ *additions* in the field. In Chapter 6 we will show how to compile proofs over the ILC into proofs over the standard channel. Moreover, we will show efficient instantiations which preserve the computational complexity prover and verifier have in the ILC model.

Our proofs over the ILC achieve perfect completeness, perfect special honest verifier zero knowledge, and statistical strong knowledge soundness. The soundness of our proofs crucially relies on the use of the Schwartz-Zippel Lemma and, therefore, it is affected by the size of the underlying field. In order to get negligible soundness error without incurring in repetitions of the proofs, we consider the size of the field to be superpolynomial size, i.e. $|\mathbb{F}| \approx \lambda^{\omega(1)}$.

In [Gro09], Groth gave several efficient zero-knowledge arguments for a wide class of statements related to linear algebra. The main tool used in the arguments is a homomorphic commitment scheme to vectors, which is instantiated with Pedersen commitments. The proofs in this chapter can be seen as an abstraction of the arguments of [Gro09] in an idealised vector commitment setting. By constructing proofs in this model, we can extract the information theoretic properties of these arguments, decoupling them from the computational cryptographic assumptions used. In Chapter 6 we will then introduce a generic transformation, which compiles proofs over the ILC into proofs and arguments over the standard channel using different cryptographic assumptions. At high level, this process enables to exploit techniques that are typically used in conjunction with homomorphic commitments into a setting were the commitment scheme does not offer such a property.

Similarly to [Gro09], we follow a modular approach to construct our proofs. We describe proofs to efficiently check that committed vectors contain publicly known values, for the sum of committed vectors, and for the Hadamard (entry-wise) product of committed vectors. We also outline the construction of a proof for a known permutation relation, and refer to [BCG+17] for the full specifications. Lastly, we show how to combine these building blocks to give a proof for the satisfiability of an arithmetic circuit.

4.1 Relation for the Satisfiability of an Arithmetic Circuit

Consider an arithmetic circuit with a total of N fan-in 2 gates, which can be either addition gates or multiplication gates over a finite field $\mathbb{F}$. Each gate has two inputs, which we refer to as *left* and *right*, and one output wire. We allow for arbitrary fan-out, therefore the output wire of each gate can be attached to the input wires of several

other gates. In total, we have $3N$ input and output wires feeding in and out of the gates.

An instance consists of the description of an arithmetic circuit and comprises a set of gates, the connection of wires between gates, and known values assigned to some of the inputs and outputs. A circuit is said to be *satisfiable* if there exists an assignment complying with all the gates, the wiring, and the known values specified in the instance. More precisely, a witness for a given instance consists of assignments to the input and output wires of each gate such that

(1) The known values specified in the instance match the corresponding wire assignments in the witness.

(2) The output of every addition gate corresponds to the sum of the input wires of the same gate.

(3) The output of every multiplication gate corresponds to the product of the input wires of the same gate.

(4) The value of an output wire (or an input *gate*) matches the values of all input wires connected to it.

In our proof for the satisfiability of arithmetic circuits, the prover commits to all the wire assignments using the ILC channel and then shows that these satisfy the above conditions. To facilitate these proofs, we format the instance and the witness, similarly to [Gro09], so that it can be easily parsed by the prover and verifier when using the ILC channel.

Formatting the Witness. The witness consists of the wire assignments to all input and output wires, and we consider these to be arranged into row vectors $\boldsymbol{v}_i \in \mathbb{F}^k$, where k is the vector length of the ILC. Without loss of generality we assume both the number of addition gates and the number of multiplication gates to be divisible by k, which can always be satisfied by adding few dummy gates to the circuit, e.g. $0 \oplus 0 = 0$ or $0 \otimes 0 = 0$. We can then number addition gates from $(1, 1)$ to (m_A, k) and multiplication gates $(m_A + 1, 1)$ to $(m_A + m_M, k)$, where $m_A \cdot k$ is the total number of addition gates and $m_M \cdot k$ is the total number of multiplication gates. We insert

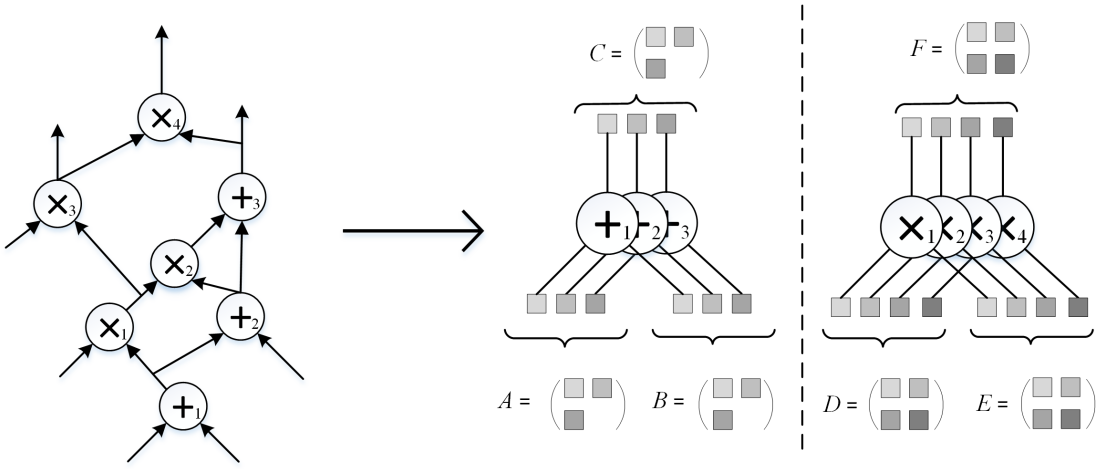

FIGURE 4.1: Representation of an arithmetic circuit and arrangements of the wires into 6 matrices.

assignments to left inputs, right inputs and outputs of addition gates into entries of three matrices $A, B, C \in \mathbb{F}^{m_A \times k}$, respectively. We sort entries to the matrices so that wires attached to the same gate correspond to the same entry of the three matrices, as shown in Figure 4.1. A valid assignment to the wires then satisfies $A + B = C$. We proceed in a similar way for the $m_M \cdot k$ multiplication gates to obtain three matrices $D, E, F \in \mathbb{F}^{m_M \times k}$ such that $D \circ E = F$, where $\circ$ denotes the Hadamard (i.e. entry-wise) product of matrices. All the committed wires then constitute a matrix

$$V = \begin{pmatrix} A \\ B \\ C \\ D \\ E \\ F \end{pmatrix} \in \mathbb{F}^{(3m_A+3m_M)\times k}$$

Formatting the Instance. We consider instances of the form $u = (m_A, m_M, \pi, \{\boldsymbol{u}_i\}_{i \in S})$. As above, m_A and m_M determine the number of addition and multiplication gates in the circuit. The vectors $\boldsymbol{u}_i$ contain the publicly known assignments to wires. Without loss of generality, we assume the gates to be sorted so that the wire values specified in the instance correspond to *full* rows in V and are indexed by S. Again, this is without loss of generality because we can always add a few dummy gates to the

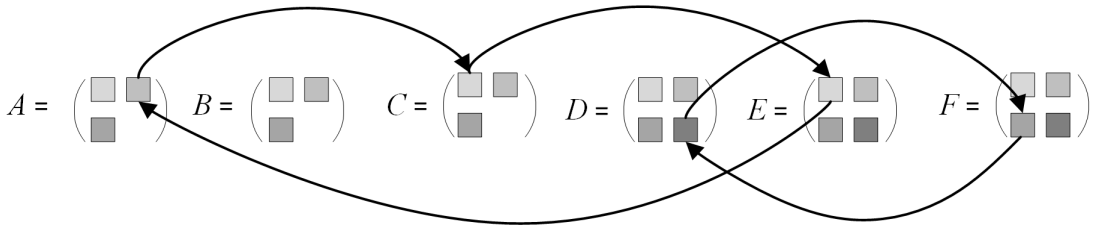

FIGURE 4.2: Representation of the wiring of a circuit: cycles $((1,2),(5,1),(9,1))$ and $((8,2),(12,1))$.

circuit and to the instance to complete a row. Lastly, π is a permutation encoding the wiring of the arithmetic circuit. For each wire, we can write a cycle $((i_1,j_1),\ldots,(i_t,j_t))$ that lists the coordinates of entries in V corresponding to the same wire. Then we let $\pi \in \Sigma_{[3m_A+3m_M]\times[k]}$ be the product of all these cycles, which unambiguously defines the wiring of the circuit. To give an example using the circuit in Figure 4.1, the output wire of the first addition gate, $V_{5,1}$, also appears as input in the first multiplication gate, $V_{9,1}$, and the second addition gate, $V_{1,2}$. Therefore, if they appear as entries $(1,2),(5,1),(9,1)$ in the matrix V defined by the rows $\boldsymbol{v}_i$, then we would have the cycle $((1,2),(5,1),(9,1))$ indicating entries that must be identical. The output of the third multiplication gate, $V_{12,1}$, feeds into the left input of the fourth multiplication gate, $V_{8,2}$, so this may give us a cycle $((8,2),(12,1))$ of entries that should have the same value. An illustration of these two cycles is given in Figure 4.2. The permutation π is the product of all these cycles that define which entries have the same value.

While keeping the above formatting in mind, we now write the relation $\mathcal{R}_{\mathrm{AC}}$ for the satisfiability of an arithmetic circuit as follows.

$$\mathcal{R}_{\mathrm{AC}} = \left\{ \begin{array}{ll} (pp,u,w) := \Big((\mathbb{F},k,*)\ ,\ (m_A,m_M,\pi,\{\boldsymbol{v}_i\}_{i\in S})\ ,\ \big(\{\boldsymbol{v}_i\}_{i\in\bar{S}}\big)\Big) : & \\ \quad m = 3m_A + 3m_M & \wedge\ \pi \in \Sigma_{[m]\times[k]} \\ \wedge\ S \subseteq [m] & \wedge\ \bar{S} = [m] \setminus S \\ \wedge\ A = (\boldsymbol{v}_i)_{i=1}^{m_A} & \wedge\ D = (\boldsymbol{v}_i)_{i=3m_A+1}^{3m_A+m_M} \\ \wedge\ B = (\boldsymbol{v}_i)_{i=m_A+1}^{2m_A} & \wedge\ E = (\boldsymbol{v}_i)_{i=3m_A+m_M+1}^{3m_A+2m_M} \\ \wedge\ C = (\boldsymbol{v}_i)_{i=2m_A+1}^{3m_A} & \wedge\ F = (\boldsymbol{v}_i)_{i=3m_A+2m_M+1}^{3m_A+3m_M} \\ \wedge\ A + B = C & \wedge\ D \circ E = F \\ \wedge\ V = (\boldsymbol{v}_i)_{i=1}^{m} & \wedge\ V_{i,j} = V_{\pi(i,j)}\ \forall\ (i,j) \in [m]\times[k] \end{array} \right\}$$

4.2 ILC Proofs for Simple Relations

In our proof for the satisfiability of arithmetic circuits, the prover starts by committing to all the wires values and then she shows that these satisfy the conditions specified in $\mathcal{R}_{\text{AC}}$. To do this we follow a modular approach. We start by giving proofs over the ILC for simple relations and, in the next section, we combine these to give a full proof for $\mathcal{R}_{\text{AC}}$. In this section we present ILC proofs for equality, sum and product relations. We also give a decomposition of the proof for a known permutation relation, referring to [BCG+17] for its complete specifications. All these relations involve vectors that have been previously committed to by the prover. To indicate that a vector $\boldsymbol{v}$ in the witness has been committed, we sometimes include $[\boldsymbol{v}]$ in the instance. To keep the notation as light as possible, we only use this when there may be some confusion about which vectors the statements refer to; for example, when we give the statement as input to the verifier.

4.2.1 ILC Proof for the Correct Opening of Committed Vectors

We start with a proof to check the equality of vectors included in the instance with vectors committed by the prover in the ILC, which corresponds to the following equality relation.

$$\mathcal{R}_{\text{eq}} = \left\{ \begin{array}{c} (pp_{\mathsf{ILC}}, u, w) = ((\mathbb{F}, k, *),\ (\{\boldsymbol{u}_i\}_{i\in S}),\ (\{\boldsymbol{v}_i\}_{i\in S})) : \\ \forall\, i \in S,\ \boldsymbol{u}_i = \boldsymbol{v}_i \end{array} \right\}$$

In this proof the prover $\mathcal{P}_{\text{eq}}$ has to simply commit to vectors by sending a command $(\textbf{commit}, \boldsymbol{v}_1, \ldots, \boldsymbol{v}_s)$ to the ILC. As soon as the prover commits, the ILC informs the verifier he received $s = |S|$ vectors. As we will use this proof as a building block for more complex proofs, the vectors may have already been committed, and the verifier notified. In this case no further action is required by the prover. The verifier picks a random challenge $x \leftarrow \mathbb{F}$ and queries the channel to open the linear combination $X = (x, x^2, \ldots, x^s)$. The channel replies with the opening $\boldsymbol{v} = \sum_{i=1}^{s} x^i \boldsymbol{v}_i$. The verifier then computes the same linear combination on his vectors and checks it against the opening received by the channel. The description of the proof systems $(\mathcal{G}_{\mathsf{ILC}}, \mathcal{P}_{\text{eq}}, \mathcal{V}_{\text{eq}})$

is given in Figure 4.3, where we use the symbols $/^*$ and $^*/$ to delimit the lines that may have already been executed in other proofs.

$\mathcal{P}_{\text{eq}}((\mathbb{F},k),(\{\boldsymbol{u}_i\}_{i\in S}),(\{\boldsymbol{v}_i\}_{i\in S}))$	$\mathcal{V}_{\text{eq}}((\mathbb{F},k),(\{\boldsymbol{u}_i\}_{i\in S},\{[\boldsymbol{v}_i]\}_{i\in S}))$
$/^*$ **Round 0:**	$/^*$ **Round 0:**
ILC $\leftarrow \bullet$ Send $(\textbf{commit}, \boldsymbol{v}_1, \ldots, \boldsymbol{v}_s)$ to the ILC $^*/$	ILC $\rightarrow \circ$ Get message s from the ILC $^*/$
	Query:
	• $x \leftarrow \mathbb{F}$
	• $X := (x^1, \ldots, x^s)$
	ILC $\leftarrow \bullet$ Send $(\textbf{open}, X)$ to the ILC
	ILC $\rightarrow \circ$ Get response $\boldsymbol{v}$ from the ILC
	• If $\boldsymbol{v} = \sum_{i=1}^{s} x^i \boldsymbol{u}_i$ return 1, else return 0

FIGURE 4.3: Proof of Knowledge for the $\mathcal{R}_{\text{eq}}$ relation. Steps marked with ILC $\rightarrow \circ$ and ILC $\leftarrow \bullet$ denote incoming and outgoing messages to the ILC, respectively.

Theorem 4.1. $(\mathcal{G}_{\mathsf{ILC}}, \mathcal{P}_{\text{eq}}, \mathcal{V}_{\text{eq}})$ *is a proof of knowledge for the relation* $\mathcal{R}_{\text{eq}}$ *in the* ILC *model with perfect completeness, statistical strong knowledge soundness with straight-line extraction and perfect special honest verifier zero-knowledge.*

Proof. The completeness of the proof is trivial. If the $\boldsymbol{u}_i$ are equal to the $\boldsymbol{v}_i$, then the verifier accepts the proof with probability 1.

The proof system has statistical strong knowledge soundness with straight-line extraction. This is because the knowledge extractor has already access to the transcript of the communication between the prover and the channel (see Definition 3.4), and thus it sees the vectors $\boldsymbol{v}_i$. By the Schwartz-Zippel Lemma, if the committed vectors are not equal to the $\boldsymbol{u}_j$, then they pass the consistency check with probability at most $\frac{s}{|\mathbb{F}|}$, which is negligible.

The proof is perfect SHVZK since the verifier knows the values in the instance. Given these, it is trivial to simulate the verifier's view. □

Efficiency. As we only use this proof as a building block, we assess its efficiency by ignoring the first step of prover and verifier (marked with $/^*$ and $^*/$). Since assume that the vectors have been previously committed by the prover, and since the verifier does not send challenges to the prover, there is no additional communication between prover and verifier and thus the round complexity is $\mu = 0$. Otherwise the round

complexity would be equal $\mu = 1$. The verifier makes a single query to the ILC channel, so the query complexity is $\mathsf{qc} = 1$. The prover does not perform any computation. The verifier's computational cost is $\mathcal{O}(sk)$ multiplications in $\mathbb{F}$.

Improved Verification. Although the above efficiency may look optimal, there is, perhaps surprisingly, a more efficient way to perform the above verification that only requires a linear number ($\mathcal{O}(sk)$) of field *additions*. This can be done by first encoding both the response $\boldsymbol{v}$ and the vectors $\boldsymbol{u}_j$ using linear-time linear error correcting codes with linear minimum distance, as the ones recalled in Theorem 3.1. Instead of checking that the entire encoded response $E_{\mathcal{C}}(\boldsymbol{v})$ is consistent with the encoded vectors $E_{\mathcal{C}}(\boldsymbol{u}_i)$, the verifier can spot check these. More precisely he can pick $\mathcal{O}(\lambda)$ columns at random, compute the linear combination on these selected columns and check them against the encoded response. If $\boldsymbol{v}$ is not equal to $\sum_{i=1}^{s} x^i \boldsymbol{u}_i$, then they differ in at least one entry. Since the error correcting code has linear minimum distance, then the verifier has constant probability of catching an error. Therefore, by picking uniformly and independently at random $\mathcal{O}(\lambda)$ columns, the probability of missing an error can be made negligible. Encoding all the vectors only requires $\mathcal{O}(sk)$ of additions, while the cost of evaluating the linear combinations in the selected columns amount to $\mathcal{O}(s\lambda)$ multiplications. For large enough $k \gg \lambda$, the total number of multiplication is a sublinear number of multiplications, i.e. $o(sk)$. Note that in our main proof we will set k to be approximately equal to $\sqrt{N}$, where N is a (large) polynomial in λ. The above strategy is similar to the one used by Damgård and Zakarias in [DZ13] to check the zero product of matrices, while in our case it is used it to test the identity of vectors.

Table 4.1 reports the efficiency of the proof for the correct opening of committed vectors with respect to the improved verification strategy.

Prover computation	$T_{\mathcal{P}_{\mathsf{ILC}}}$	$= //$
Verifier computation	$T_{\mathcal{V}_{\mathsf{ILC}}}$	$= \mathcal{O}(sk)\ \mathbb{F}^+$
Prover communication	t	$= //$
Verifier communication	C_{ILC}	$= 0$
Query complexity	qc	$= 1$
Round complexity	μ	$= 0$

TABLE 4.1: Efficiency of the proof of knowledge for $\mathcal{R}_{\text{eq}}$. $\mathbb{F}^+$ stands for the cost of a single field addition.

$\mathcal{P}_{\text{sum}}((\mathbb{F},k),\ s,\ (A,B,C))$	$\mathcal{V}_{\text{sum}}((\mathbb{F},k),\ (s,[A],[B],[C]))$
/ * **Round 0:**	/ * **Round 0:**
ILC $\leftarrow \bullet$ Send $(\textbf{commit}, A, B, C)$ to the ILC */	ILC $\rightarrow \circ$ Get message $3s$ from the ILC. */
	Query:
	• $x \leftarrow \mathbb{F}$
	• $X := (x^1, \ldots, x^s, x^1, \ldots, x^s, -x^1, \ldots, -x^s)$
	ILC $\leftarrow \bullet$ Send $(\textbf{open}, X)$ to ILC
	ILC $\rightarrow \circ$ Get response $\boldsymbol{v}$ from the ILC
	• If $\boldsymbol{v} = \mathbf{0}$ return 1, else return 0

FIGURE 4.4: Proof of Knowledge for the $\mathcal{R}_{\text{sum}}$ relation.

4.2.2 ILC Proof for the Sum of Committed Matrices

Next, we show a proof of knowledge of committed matrices $A, B, C \in \mathbb{F}^{s\times k}$ such that $A + B = C$, as specified in the following relation

$$\mathcal{R}_{\text{sum}} = \left\{ \begin{array}{c} (pp_{\mathsf{ILC}}, u) = ((\mathbb{F}, k, *),\ s,\ (A, B, C)) : \\ A, B, C \in \mathbb{F}^{s\times k} \quad \wedge \quad A + B = C \end{array} \right\}$$

As for the previous proof, the prover starts by committing to all the row vectors of the matrices A, B, C. We assume the prover commits to the rows of the matrices in order, starting with all the rows of A, B and then C. If the matrices have already been committed to by the prover, she may remain inactive. The verifier receives the message $3s$ from the ILC, notifying the reception of the committed vectors. Next, the verifier picks a challenge $x \leftarrow \mathbb{F}$, constructs the query

$$X := (x^1, x^2, \ldots, x^s, x^1, x^2, \ldots, x^s, -x^1, -x^2, \ldots, -x^s)$$

and then sends $(\textbf{open}, X)$ to the ILC. The idea is that the i-th rows in the matrices A, B, C are associated with the same power of x in the query, but the ones in C have opposite sign. Therefore, we expect them to sum up to zero if $A + B - C = 0$ holds. The description of the proof of knowledge for $\mathcal{R}_{\text{sum}}$ is given in Figure 4.4 where steps marked with ILC $\rightarrow \circ$ and ILC $\leftarrow \bullet$ denote incoming and outgoing messages to the ILC, respectively.

Theorem 4.2. *$(\mathcal{G}_{\mathsf{ILC}}, \mathcal{P}_{\text{sum}}, \mathcal{V}_{\text{sum}})$ is a proof of knowledge for the relation $\mathcal{R}_{\text{sum}}$ in the* ILC *model with perfect completeness, statistical strong knowledge soundness with straight-line*

extraction, and perfect special honest verifier zero-knowledge.

Proof. The completeness of the proof is trivial as if $A + B - C$ is equal to the zero matrix, then the ILC answers with $\boldsymbol{v} = \mathbf{0}$ with probability 1.

The proof has statistical strong knowledge soundness with straight-line extraction. This is because the knowledge extractor has access to the communication between the prover and the channel, and thus sees the row vectors of matrices A, B and C. If $A + B \neq C$ then at least one column in $A + B - C$ is not the zero vector. By the Schwartz-Zippel Lemma, the probability of the ILC returning $\mathbf{0}$ is at most $\frac{s}{|\mathbb{F}|}$, which is negligible.

The proof is perfect zero-knowledge as the verifier only sees the zero vector, which can be trivially simulated.

□

Efficiency. As before, we assess the efficiency by ignoring the first step of prover and verifier (marked with $/^*$ and $^*/$). In this case the round complexity is equal to 0 as there is no additional communication between prover and verifier, as the verifier only communicates with the ILC. The prover does not perform any computation. The verifier makes one query to the ILC and its computational cost is dominated by computing $s - 1$ multiplications in $\mathbb{F}$ to construct the query. Table 4.2 reports the costs of the the proof of knowledge for $\mathcal{R}_{\text{sum}}$.

Prover computation	$T_{\mathcal{P}_{\mathsf{ILC}}}$	$= //$
Verifier computation	$T_{\mathcal{V}_{\mathsf{ILC}}}$	$= (s-1)\,\mathbb{F}^{\times}$
Prover communication	t	$= //$
Verifier communication	C_{ILC}	$= 0$
Query complexity	qc	$= 1$
Round complexity	μ	$= 0$

TABLE 4.2: Efficiency of the proof of knowledge for $\mathcal{R}_{\text{sum}}$. $\mathbb{F}^{\times}$ stands for the cost of a single field multiplication.

4.2.3 ILC proof for the Hadamard Product of Committed Matrices

We now describe a proof of knowledge for the Hadamard (entry-wise) product of matrices $A, B, C \in \mathbb{F}^{\mathsf{mn}\times k}$ such that $A \circ B = C$, as specified in the following relation

$$\mathcal{R}_{\text{prod}} = \left\{ \begin{array}{c} (pp_{\mathsf{ILC}}, u) = ((\mathbb{F}, k),\ (\mathsf{mn}),\ (A, B, C)) : \\ A, B \in \mathbb{F}^{\mathsf{mn}\times k} \quad \wedge \quad A \circ B = C \end{array} \right\}$$

Main Idea. Firstly, we take a look to the case $\mathsf{m} = 1$ such that matrices A, B, C consist of n rows each. If we parse each matrix as the set of its row vectors, we have that $A \circ B = C$ can be written as $\boldsymbol{a}_j \circ \boldsymbol{b}_j = \boldsymbol{c}_j$ for $1 \leq j \leq \mathsf{n}$. Unless they have already been committed, the prover starts by committing to the vectors $\boldsymbol{a}_j, \boldsymbol{b}_j, \boldsymbol{c}_j$ using the ILC. Then the verifier picks a challenge $x \leftarrow \mathbb{F}$ and sends it to the prover by forwarding the command $(\textbf{send}, x)$ to the ILC. After receiving x the prover computes the following polynomials in the indeterminate Z with coefficient in $\mathbb{F}^k$

$$\boldsymbol{a}'(Z) = \sum_{j=1}^{\mathsf{n}} \boldsymbol{a}_j x^j Z^j \qquad\qquad \boldsymbol{b}'(Z) = \sum_{j=1}^{\mathsf{n}} \boldsymbol{b}_j Z^{-j}$$

and compute their product in $\mathbb{F}^k[Z]$

$$\boldsymbol{a}'(Z) \circ \boldsymbol{b}'(Z) = \sum_{j=1}^{\mathsf{n}} \boldsymbol{a}_j \circ \boldsymbol{b}_j x^j + \sum_{r=1-\mathsf{n}, r\neq 0}^{\mathsf{n}-1} \boldsymbol{e}_r Z^r$$

where $\circ$ denotes the the standard convolution product of polynomials with the Hadamard product applied to the coefficients. Note that the the products $\boldsymbol{a}_j \circ \boldsymbol{b}_j$ are all positioned in the constant term of the polynomial $\boldsymbol{a}'(Z) \circ \boldsymbol{b}'(Z)$, while the products $\boldsymbol{a}_i \circ \boldsymbol{b}_j$ for $i \neq j$ are inside the *cancellation terms* $\boldsymbol{e}_r$. The prover sends the command $(\textbf{commit}, \{\boldsymbol{e}_r\}_{r=1-\mathsf{n}, r\neq 0}^{\mathsf{n}-1})$ to commit to vectors $\boldsymbol{e}_r$. These correspond to the coefficient of $\boldsymbol{a}'(Z) \circ \boldsymbol{b}'(Z)$ excluding its constant term. At this point the verifier picks a challenge $z \leftarrow \mathbb{F}$ and queries the ILC to give the following linear combinations of the vectors $\boldsymbol{a}_j, \boldsymbol{b}_j, \boldsymbol{c}_j, \boldsymbol{e}_r$, which are all stored in the ILC.

$$\boldsymbol{a}'(z) = \sum_{j=1}^{\mathsf{n}} \boldsymbol{a}_j x^j z^j \qquad \boldsymbol{b}'(z) = \sum_{j=1}^{\mathsf{n}} \boldsymbol{b}_j z^{-j} \qquad \boldsymbol{c}'(z) = \sum_{j=1}^{\mathsf{n}} \boldsymbol{c}_j x^j + \sum_{r=1-\mathsf{n}, r\neq 0}^{\mathsf{n}-1} \boldsymbol{e}_r z^r$$

If the statement is true, i.e. $\boldsymbol{a}_j \circ \boldsymbol{b}_j = \boldsymbol{c}_j$, then the constant term in $\boldsymbol{a}'(Z) \circ \boldsymbol{b}'(Z)$ is equal $\sum_{j=1}^{\mathsf{n}} \boldsymbol{c}_j x^j$. The verifier finally checks whether $\boldsymbol{a}'(z) \circ \boldsymbol{b}'(z) = \boldsymbol{c}'(z)$, in which case she returns 1.

Observation 1. The above approach is clearly not zero knowledge, since the openings returned by the ILC only depend on the challenges and the values in the witness. To make the above solution zero-knowledge, in her first move the prover commits to some *blinding factors* $\boldsymbol{a}'_0, \boldsymbol{b}'_0 \leftarrow \mathbb{F}^k$ and $\boldsymbol{c}'_0 := \boldsymbol{a}'_0 \circ \boldsymbol{b}'_0$. Then, it is sufficient to extend the linear combinations queried to the ILC by the (honest) verifier to include the blinding factors, i.e.

$$\boldsymbol{a}'(z) + \boldsymbol{a}'_0 \qquad \boldsymbol{b}'(z) + \boldsymbol{b}'_0 \qquad \boldsymbol{c}'(z) + \boldsymbol{c}'_0$$

Note that the polynomials defined in the protocol need to be changed to take into account the blinders. We defer this to the full specification of the proof, which can be found below.

Observation 2. Our final goal is to construct efficient proofs of knowledge for which the prover only incurs a constant overhead in computation. The approach described above however does not suffice to achieve this level of efficiency. The bottleneck for the prover is in the computation of the product $\boldsymbol{a}'(Z) \circ \boldsymbol{b}'(Z)$, which has degree n and coefficients in $\mathbb{F}^k$. Using FFT multiplication algorithms, this product requires about $\mathcal{O}(k\mathsf{n}\log(\mathsf{n}))$ field multiplications. Therefore the prover incurs in a $\mathcal{O}(\log(\mathsf{n}))$ computational overhead with respect to compute the Hadamard product of matrices, which costs $\mathcal{O}(k\mathsf{n})$ field multiplications.

Reducing the Prover's Overhead. To overcome this issue we follow a technique introduced in [Gro09], which allows to reduce computation for the prover at the cost of increasing the interaction with the verifier. The main idea behind this technique is to first group the rows into sets of $\mathsf{m} \approx \log(\mathsf{n})$ rows, and then to use a compression step which reduces the size of the groups by a half. After $\log\log(\mathsf{n})$ iterations, the number of row vectors can be reduced by a factor $\frac{1}{\log(\mathsf{n})}$, such that it is possible to follow the approach described above without incurring in a superconstant overhead. Moreover,

the compression steps only require a linear number of operations, keeping the overall prover running time equal to $\mathcal{O}(\mathsf{mn}k)$ field multiplications.

Consider matrices A, B, C with mn row vectors each: $\boldsymbol{a}_{i,j}, \boldsymbol{b}_{i,j}, \boldsymbol{c}_{i,j} \in \mathbb{F}^k$ for $0 \leq i \leq \mathsf{m}-1$ and $0 \leq j \leq \mathsf{n}-1$. Without loss of generality we assume that $\mathsf{m} = 2^\mu$ for some integer $\mu \in \mathbb{N}$. We will compress $2\mathsf{mn}$ vectors $\boldsymbol{a}_{i,j}, \boldsymbol{b}_{i,j}$ of length k into $2\mathsf{n}$ vectors $\boldsymbol{a}'_j, \boldsymbol{b}'_j$ of the same length. The compressed vectors are computed by first inserting vectors $\boldsymbol{a}_{i,j}$ and $\boldsymbol{b}_{i,j}$ into distinct coefficients of $2\mathsf{n}$ multivariate polynomials in $Y_0, \ldots, Y_{\mu-1}$, and then by progressively evaluating these at μ challenges $y_0, \ldots, y_{\mu-1}$ sent by the verifier. More precisely, vector $\boldsymbol{a}_{i,j}x^i$ is positioned in the j-th polynomial $\boldsymbol{a}'_j(Y_0, \ldots, Y_{\mu-1})$ as coefficient of $Y_0^{i_0} \cdots Y_{\mu-1}^{i_{\mu-1}}$, where x is the first challenge sent by the verifier, and $(i_{\mu-1}, i_{\mu-2}, \ldots, i_0) \in \{0,1\}^\mu$ is the binary expansion of i. Computing these coefficients only requires $\mathcal{O}(\mathsf{mn}k)$ field multiplications. In a similar fashion the vector $\boldsymbol{b}_{i,j}$ is embedded into the coefficient of $Y_0^{-i_0} \cdots Y_{\mu-1}^{-i_{\mu-1}}$ in the polynomial $\boldsymbol{b}'_j(Y_0, \ldots, Y_{\mu-1})$, as shown below.

$$\boldsymbol{a}'_j(Y_0, \ldots, Y_{\mu-1}) = \sum_{i=0}^{\mathsf{m}-1} \boldsymbol{a}_{i,j} x^i Y_0^{i_0} Y_1^{i_1} \ldots Y_{\mu-1}^{i_{\mu-1}} \qquad \text{For } 0 \leq j \leq \mathsf{n}-1$$

$$\boldsymbol{b}'_j(Y_0, \ldots, Y_{\mu-1}) = \sum_{i=0}^{\mathsf{m}-1} \boldsymbol{b}_{i,j} Y_0^{-i_0} Y_1^{-i_1} \ldots Y_{\mu-1}^{-i_{\mu-1}} \qquad \text{For } 0 \leq j \leq \mathsf{n}-1$$

By evaluating all the $2\mathsf{n}$ polynomials in $Y_0 := y_0$ we reduce the number of coefficients in each of the polynomials by a factor 2, as shown in the following.

$$\boldsymbol{a}'_j(y_0, Y_1, \ldots, Y_{\mu-1}) = \sum_{i=0}^{\frac{\mathsf{m}}{2}-1} \underbrace{\left(\boldsymbol{a}_{2i,j} x^{2i} + \boldsymbol{a}_{2i+1,j} x^{2i+1} y_0\right)}_{\boldsymbol{\alpha}_{1,i,j}} Y_1^{i_1} \ldots Y_{\mu-1}^{i_{\mu-1}} \quad \text{For } 0 \leq j \leq \mathsf{n}-1$$

$$\boldsymbol{b}'_j(y_0, Y_1, \ldots, Y_{\mu-1}) = \sum_{i=0}^{\frac{\mathsf{m}}{2}-1} \underbrace{\left(\boldsymbol{b}_{2i,j} + \boldsymbol{b}_{2i+1,j} y_0^{-1}\right)}_{\boldsymbol{\beta}_{1,i,j}} Y_1^{-i_1} \ldots Y_{\mu-1}^{-i_{\mu-1}} \qquad \text{For } 0 \leq j \leq \mathsf{n}-1$$

The coefficients $\boldsymbol{\alpha}_{1,i,j}, \boldsymbol{\beta}_{1,i,j}$ of each of the above $2\mathsf{n}$ polynomials can be computed with $\frac{\mathsf{m}k}{2}$ multiplications and $\frac{\mathsf{m}k}{2}$ additions. By first evaluating these on $Y_1 := y_1$, and then

the other challenges $y_2, \ldots, y_{\mu-1}$, the number of coefficients is halved at each evaluation, until we obtain the following $2\mathsf{n}$ vectors.

$$\begin{aligned}
\boldsymbol{a}'_j := \boldsymbol{a}'_j(y_0, \ldots, y_{\mu-1}) &= \sum_{i=0}^{\mathsf{m}-1} \boldsymbol{a}_{i,j} x^i y_0^{i_0} y_1^{i_1} \ldots y_{\mu-1}^{i_{\mu-1}} \\
&= \sum_{i=0}^{\frac{\mathsf{m}}{2}-1} \boldsymbol{\alpha}_{1,i,j} y_1^{i_1} \ldots y_{\mu-1}^{i_{\mu-1}} \\
&= \cdots \\
&= \boldsymbol{\alpha}_{\mu-1,0,j} + \boldsymbol{\alpha}_{\mu-1,1,j} y_{\mu-1} \qquad \text{For } 0 \leq j \leq \mathsf{n}-1
\end{aligned}$$

$$\begin{aligned}
\boldsymbol{b}'_j := \boldsymbol{b}'_j(y_0, \ldots, y_{\mu-1}) &= \sum_{i=0}^{\mathsf{m}-1} \boldsymbol{b}_{i,j} y_0^{-i_0} y_1^{-i_1} \ldots y_{\mu-1}^{-i_{\mu-1}} \\
&= \sum_{i=0}^{\frac{\mathsf{m}}{2}-1} \boldsymbol{\beta}_{1,i,j} y_1^{-i_1} \ldots y_{\mu-1}^{-i_{\mu-1}} \\
&= \cdots \\
&= \boldsymbol{\beta}_{\mu-1,0,j} + \boldsymbol{\beta}_{\mu-1,1,j} y_{\mu-1}^{-1} \qquad \text{For } 0 \leq j \leq \mathsf{n}-1
\end{aligned}$$

As the cost of computing these evaluations is halved at each iteration, the above vectors require in total $\mathcal{O}(\mathsf{m}\mathsf{n}k)$ field additions and multiplications to compute. At this point we can proceed as in the case $\mathsf{m} = 1$ and embed the above $2\mathsf{n}$ vectors into the coefficients of two polynomials in Z of degree n.

$$\begin{aligned}
\boldsymbol{a}'(Z) &= \sum_{j=0}^{\mathsf{n}-1} \boldsymbol{a}'_j(y_0, \ldots, y_{\mu-1}) x^{j\mathsf{m}+1} Z^j \\
\boldsymbol{b}'(Z) &= \sum_{j=0}^{\mathsf{n}-1} \boldsymbol{b}'_j(y_0, \ldots, y_{\mu-1}) Z^{-j}
\end{aligned}$$

If we take the Hadamard product of the two polynomials above, the products $\boldsymbol{a}'_j \circ \boldsymbol{b}'_j$ end up in the constant coefficient, i.e., Z^0. Similarly, all other cancellation products $\boldsymbol{a}'_j \circ \boldsymbol{b}'_{j'}$, for $j \neq j'$, end up in the coefficients $\boldsymbol{e}_r$ of other powers of Z. If we expand the products of $\boldsymbol{a}'_j \circ \boldsymbol{b}'_j$ we can also observe that the products of $\boldsymbol{a}_{i,j} \circ \boldsymbol{b}_{i,j}$ are placed in the constant term of the Y_t, while all other products $\boldsymbol{a}_{i,j} \circ \boldsymbol{b}_{i',j}$ for $i \neq i'$ are contained in

the cancellation terms $\boldsymbol{d}_i^+, \boldsymbol{d}_i^-$.

$$\begin{aligned}\boldsymbol{a}'(Z) \circ \boldsymbol{b}'(Z) &= \sum_{j=0}^{\mathsf{n}-1} \boldsymbol{a}'_j \circ \boldsymbol{b}'_j x^{j\mathsf{m}+1} + \sum_{r=2-\mathsf{n}, r\neq 0}^{\mathsf{n}-2} \boldsymbol{e}_r Z^r \\ &= \sum_{i=0,j=0}^{\mathsf{m}-1,\mathsf{n}-1} \boldsymbol{a}_{i,j} \circ \boldsymbol{b}_{i,j} x^{i+j\mathsf{m}+1} + \sum_{t=0}^{\mu-1} \left(\boldsymbol{d}_t^+ y_t + \boldsymbol{d}_t^- y_t^{-1}\right) + \sum_{r=2-\mathsf{n}, r\neq 0}^{\mathsf{n}-2} \boldsymbol{e}_r Z^r\end{aligned}$$

Computing this product requires $\mathcal{O}(k\mathsf{n}\log(\mathsf{n}))$ field multiplications, which is linear in $\mathsf{mn}k$ if we set $\mathsf{m} \approx \log(\mathsf{n})$. The cancellation terms $\boldsymbol{d}_t^+, \boldsymbol{d}_t^-$ have the following expressions

$$\boldsymbol{d}_t^+ = \sum_{j=0}^{\mathsf{n}-1} \sum_{i=0}^{\frac{\mathsf{m}}{2^{t+1}}-1} \boldsymbol{\alpha}_{t,2i+1,j} \circ \boldsymbol{\beta}_{t,2i,j} \qquad \boldsymbol{d}_t^- = \sum_{j=0}^{\mathsf{n}-1} \sum_{i=0}^{\frac{\mathsf{m}}{2^{t+1}}-1} \boldsymbol{\alpha}_{t,2i,j} \circ \boldsymbol{\beta}_{t,2i+1,j}$$

where $\boldsymbol{\alpha}_{0,i,j} := \boldsymbol{a}_{i,j} x^i$ and $\boldsymbol{\beta}_{0,i,j} := \boldsymbol{b}_{i,j}$. Similarly to the computation of the $\boldsymbol{\alpha}_{t,i,j}, \boldsymbol{\beta}_{t,i,j}$, the computation of $\boldsymbol{d}_t^+, \boldsymbol{d}_t^-$ requires $\frac{\mathsf{mn}k}{2^{t+1}}$ multiplications to compute. Summing over t we obtain a total of $\mathcal{O}(\mathsf{mn}k)$ multiplications. Therefore, the total computational cost for both compressing the vectors and computing the above polynomial expression is $\mathcal{O}(\mathsf{mn}k)$ field multiplications, i.e. a linear number of operations in the size of the matrices.

To summarise, we first outlined a way to check Hadamard products of vectors by evaluating a polynomial expression at a random challenge point. Then, we showed a compression technique which enables to keep the prover's computational overhead constant by increasing the interaction with the verifier. In each of the additional rounds of interaction, the prover commits to a pair of cancellation vectors $\boldsymbol{d}_t^+, \boldsymbol{d}_t^-$ and the verifier replies with the next challenge y_t. The full specifications of the prover $\mathcal{P}_{\text{prod}}$ and verifier $\mathcal{V}_{\text{prod}}$ of the zero-knowledge proof of knowledge over the ILC for the relation $\mathcal{R}_{\text{prod}}$ are given in Figure 4.5 and 4.6. As for the previous proofs of knowledge, we use the convention that steps marked with ILC $\rightarrow \circ$ and ILC $\leftarrow \bullet$ denote incoming and outgoing messages to the ILC, respectively.

Theorem 4.3. *$(\mathcal{G}_{\mathsf{ILC}}, \mathcal{P}_{\text{prod}}, \mathcal{V}_{\text{prod}})$ is a proof of knowledge for the relation $\mathcal{R}_{\text{prod}}$ in the* ILC *model with perfect completeness, statistical strong knowledge soundness with straight-line extraction and perfect special honest verifier zero-knowledge.*

$\mathcal{P}_{\text{prod}}((\mathbb{F}, k, *), (\mathsf{mn}),\ (A, B, C))$

Round 1:

- Parse $A = \{\boldsymbol{a}_{i,j}\}, B = \{\boldsymbol{b}_{i,j}\}, C = \{\boldsymbol{c}_{i,j}\}$ for $(i, j) \in [0, \mathsf{m} - 1] \times [0, \mathsf{n} - 1]$
- Set $\mu := \lfloor \log(\mathsf{m}) \rfloor$
- $\boldsymbol{a}'_{-1}, \boldsymbol{b}'_{-1} \leftarrow \mathbb{F}^k$
- $\boldsymbol{c}'_{-1} := \boldsymbol{a}'_{-1} \circ \boldsymbol{b}'_{-1}$

/* ILC ← • Send (**commit**, $\{\boldsymbol{a}_{i,j}\}, \{\boldsymbol{b}_{i,j}\}, \{\boldsymbol{c}_{i,j}\}$) to the ILC */

ILC ← • Send (**commit**, $\boldsymbol{a}'_{-1}, \boldsymbol{b}'_{-1}, \boldsymbol{c}'_{-1}$) to the ILC

Round 2:

ILC → ∘ Get x from ILC

- For $(i, j) \in [0, \mathsf{m} - 1] \times [0, \mathsf{n} - 1]$:
  - $\boldsymbol{\alpha}_{0,i,j} := \boldsymbol{a}_{i,j} x^i$
  - $\boldsymbol{\beta}_{0,i,j} := \boldsymbol{b}_{i,j}$
- $\boldsymbol{d}_0^+ := \sum_{j=0}^{\mathsf{n}-1} \sum_{i=0}^{\frac{\mathsf{m}}{2}-1} \boldsymbol{\alpha}_{0,2i+1,j} \circ \boldsymbol{\beta}_{0,2i,j}$, $\boldsymbol{d}_0^- := \sum_{j=0}^{\mathsf{n}-1} \sum_{i=0}^{\frac{\mathsf{m}}{2}-1} \boldsymbol{\alpha}_{0,2i,j} \circ \boldsymbol{\beta}_{0,2i+1,j}$

ILC ← • Send (**commit**, $\boldsymbol{d}_0^+, \boldsymbol{d}_0^-$) to the ILC

(For $t = 1$ to $\mu - 1$)
Round $t + 2$:

ILC → ∘ Get y_{t-1} from ILC

- $\mathsf{m}' := \frac{\mathsf{m}}{2^t}$
- For $(i, j) \in [0, \mathsf{m}' - 1] \times [0, \mathsf{n} - 1]$:
  - $\boldsymbol{\alpha}_{t,i,j} := \boldsymbol{\alpha}_{t-1,2i,j} + \boldsymbol{\alpha}_{t-1,2i+1,j} y_{t-1}$
  - $\boldsymbol{\beta}_{t,i,j} := \boldsymbol{\beta}_{t-1,2i,j} + \boldsymbol{\beta}_{t-1,2i+1,j} y_{t-1}^{-1}$
- $\boldsymbol{d}_t^+ := \sum_{j=0}^{\mathsf{n}-1} \sum_{i=0}^{\frac{\mathsf{m}'}{2}-1} \boldsymbol{\alpha}_{t,2i+1,j} \circ \boldsymbol{\beta}_{t,2i,j}$, $\boldsymbol{d}_t^- := \sum_{j=0}^{\mathsf{n}-1} \sum_{i=0}^{\frac{\mathsf{m}'}{2}-1} \boldsymbol{\alpha}_{t,2i,j} \circ \boldsymbol{\beta}_{t,2i+1,j}$

ILC ← • Send (**commit**, $\boldsymbol{d}_t^+, \boldsymbol{d}_t^-$) to the ILC

Round $\mu + 2$:

ILC → ∘ Get $y_{\mu-1}$ from ILC

- For $j \in [0, \mathsf{n} - 1]$:
  - $\boldsymbol{a}'_j := \boldsymbol{\alpha}_{\mu-1,0,j} + \boldsymbol{\alpha}_{\mu-1,1,j} y_{\mu-1}$
  - $\boldsymbol{b}'_j := \boldsymbol{\beta}_{\mu-1,0,j} + \boldsymbol{\beta}_{\mu-1,1,j} y_{\mu-1}^{-1}$
- $\boldsymbol{a}'(Z) := \boldsymbol{a}'_{-1} + \sum_{j=0}^{\mathsf{n}-1} \boldsymbol{a}'_j x^{j\mathsf{m}+1} Z^j$
- $\boldsymbol{b}'(Z) := \boldsymbol{b}'_{-1} + \sum_{j=0}^{\mathsf{n}-1} \boldsymbol{b}'_j Z^{-j}$
- $\boldsymbol{c}' := \boldsymbol{c}'_{-1} + \sum_{i=0}^{\mathsf{m}-1} \sum_{j=0}^{\mathsf{n}-1} \boldsymbol{c}_{i,j} x^{i+j\mathsf{m}+1}$
- $\boldsymbol{d}' := \sum_{t=0}^{\mu-1} (\boldsymbol{d}_t^+ y_t + \boldsymbol{d}_t^- y_t^{-1})$
- $\boldsymbol{e}(Z) := \sum_{r=1-\mathsf{n}, r\neq 0}^{\mathsf{n}-1} \boldsymbol{e}_r Z^r := \boldsymbol{a}'(Z) \circ \boldsymbol{b}'(Z) - \boldsymbol{a}'_{-1} \circ \boldsymbol{b}'_{-1} - \sum_{j=0}^{\mathsf{n}-1} \boldsymbol{a}'_j \circ \boldsymbol{b}'_j x^{j\mathsf{m}+1}$

ILC ← • Send (**commit**, $\{\boldsymbol{e}_r\}_{r=1-\mathsf{n}, r\neq 0}^{\mathsf{n}-1}$) to the ILC

FIGURE 4.5: Prover $\mathcal{P}_{\text{prod}}$ for the proof of knowledge for $\mathcal{R}_{\text{prod}}$.

$\mathcal{V}_{\text{prod}}((\mathbb{F}, k, *), (\mathsf{mn}, [A], [B], [C]))$

Round 1:

/ * ILC $\rightarrow \circ$ Get message $3\mathsf{mn}$ from the ILC */

ILC $\rightarrow \circ$ Get message 3 from the ILC

- $x \leftarrow \mathbb{F}$

ILC $\leftarrow \bullet$ Send $(\textbf{send}, x)$ to the ILC

(For $t = 0$ to $\mu - 1$)
Round $t + 2$:

ILC $\rightarrow \circ$ Get message 2 from the ILC

- $y_t \leftarrow \mathbb{F}$

ILC $\leftarrow \bullet$ Send $(\textbf{send}, y_t)$ to the ILC

Round $\mu + 2$:

ILC $\rightarrow \circ$ Get message $2\mathsf{n} - 1$ from the ILC

Query:

- $z \leftarrow \mathbb{F}$

ILC $\leftarrow \bullet$ Send $(\textbf{open}, \boldsymbol{a}'(z), \boldsymbol{b}'(z), \boldsymbol{c}' + \boldsymbol{d}' + \boldsymbol{e}(z))$ to the ILC

ILC $\rightarrow \circ$ Get openings $(\boldsymbol{v}_1, \boldsymbol{v}_2, \boldsymbol{v}_3)$ from the ILC

- If $\boldsymbol{v}_1 \circ \boldsymbol{v}_2 = \boldsymbol{v}_3$ return 1, else return 0

FIGURE 4.6: Verifier $\mathcal{V}_{\text{prod}}$ of the proof of knowledge for $\mathcal{R}_{\text{prod}}$.

Proof. We start by showing completeness. If the statement is true then $\boldsymbol{a}_{i,j} \circ \boldsymbol{b}_{i,j} = \boldsymbol{c}_{i,j}$ and

$$\begin{aligned}
\boldsymbol{a}'(z) \circ \boldsymbol{b}'(z) &= \boldsymbol{a}'_{-1} \circ \boldsymbol{b}'_{-1} + \sum_{j=0}^{\mathsf{n}-1} \boldsymbol{a}'_j \circ \boldsymbol{b}'_j x^{j\mathsf{m}+1} + \sum_{r=1-\mathsf{n}, r\neq 0}^{\mathsf{n}-1} \boldsymbol{e}_r z^r \\
&= \boldsymbol{a}'_{-1} \circ \boldsymbol{b}'_{-1} + \sum_{i=0,j=0}^{\mathsf{m}-1,\mathsf{n}-1} \boldsymbol{a}_{i,j} \circ \boldsymbol{b}_{i,j} x^{i+j\mathsf{m}+1} + \sum_{t=0}^{\mu-1} \left(\boldsymbol{d}_t^+ y_t + \boldsymbol{d}_t^- y_t^{-1}\right) + \sum_{r=1-\mathsf{n}, r\neq 0}^{\mathsf{n}-1} \boldsymbol{e}_r Z^r \\
&= \boldsymbol{c}'_{-1} + \sum_{i=0,j=0}^{\mathsf{m}-1,\mathsf{n}-1} \boldsymbol{c} x^{i+j\mathsf{m}+1} + \sum_{t=0}^{\mu-1} \left(\boldsymbol{d}_t^+ y_t + \boldsymbol{d}_t^- y_t^{-1}\right) + \sum_{r=1-\mathsf{n}, r\neq 0}^{\mathsf{n}-1} \boldsymbol{e}_r z^r \\
&= \boldsymbol{c}' + \boldsymbol{d}' + \boldsymbol{e}(z)
\end{aligned}$$

where the first and the last terms correspond to the verification equation $\mathcal{V}_{\text{prod}}$ performs on the queries $\boldsymbol{a}(z), \boldsymbol{b}(z)$ and $\boldsymbol{c}' + \boldsymbol{d}' + \boldsymbol{e}(z)$. Therefore the verifier accepts with probability 1.

For honest-verifier zero-knowledge, we describe how to simulate the verifier's view efficiently. Note that in a real transcript, the response to the first two opening queries is uniformly distributed as $\boldsymbol{a}'_{-1}, \boldsymbol{b}'_{-1}$ are picked uniformly at random. The simulator picks $\boldsymbol{v}_1, \boldsymbol{v}_2 \leftarrow \mathbb{F}^k$ and sets $\boldsymbol{v}_3 := \boldsymbol{v}_1 \circ \boldsymbol{v}_2$ and sets the response to the query to be $\boldsymbol{v}_1, \boldsymbol{v}_2, \boldsymbol{v}_3$. The simulated response passes the verification check. Moreover, it is distributed uniformly at random conditioned on passing the verification, and thus it is identically distributed to a real transcript. Hence, the proof system has perfect special honest verifier zero knowledge.

Next, we show that the proof has statistical strong knowledge soundness with straight-line extraction. The knowledge extractor is straight-line because it has access to the communication transcript between the prover and the channel, and therefore it sees the row vectors of matrices A, B, C composing the witness. It remains to show that for any deterministic malicious prover $\mathcal{P}^*_{\text{prod}}$, if the committed vectors are not a valid witness for $\mathcal{R}_{\text{prod}}$, then the verifier has negligible probability of accepting the statement. We prove this by induction on the values of μ. We begin with the base case

$\mu = 0$, for which $\mathsf{m} = 1$. Consider the following polynomials in X, Z

$$\boldsymbol{a}'(X,Z) = \boldsymbol{a}'_{-1} + \sum_{j=0}^{\mathsf{n}-1} \boldsymbol{a}_{0,j} X^{j+1} Z^j$$
$$\boldsymbol{b}'(X,Z) = \boldsymbol{b}'_{-1} + \sum_{j=0}^{\mathsf{n}-1} \boldsymbol{b}_{0,j} Z^{-j}$$
$$\boldsymbol{c}'(X) = \boldsymbol{c}'_{-1} + \sum_{j=0}^{\mathsf{n}-1} \boldsymbol{c}_{0,j} X^{j+1}$$
$$\boldsymbol{e}(X,Z) = \sum_{r=1-\mathsf{n}, r\neq 0}^{\mathsf{n}-1} \boldsymbol{e}_r(X) Z^r$$

The verifier's check is equivalent to test the following polynomial identity at random challenges x, z.

$$\boldsymbol{a}'(X,Z) \circ \boldsymbol{b}'(X,Z) - \boldsymbol{c}'(X) = \boldsymbol{e}(X,Z)$$

The polynomial on the right-hand side has the coefficient of Z^0 equal to $\mathbf{0}$, while the coefficient of Z^0 on the left-hand side is equal to

$$\boldsymbol{a}'_{-1} \circ \boldsymbol{b}'_{-1} - \boldsymbol{c}'_{-1} + \sum_{j=0}^{\mathsf{n}-1} (\boldsymbol{a}_{0,j} \circ \boldsymbol{b}_{0,j} - \boldsymbol{c}_{0,j}) X^{j+1} \tag{4.1}$$

If the statement is false, there exists at least one index j for which $\boldsymbol{a}_{0,j} \circ \boldsymbol{b}_{0,j} - \boldsymbol{c}_{0,j} \neq 0$ and thus the polynomial (4.1) is non-zero. By the Schwartz-Zippel Lemma, the probability of this polynomial evaluating to $\mathbf{0}$ at a random point x is at most $\frac{\mathsf{n}+1}{|\mathbb{F}|}$. By applying the Schwartz-Zippel Lemma again, if the polynomials $\boldsymbol{a}'(x,Z) \circ \boldsymbol{b}'(x,Z) - \boldsymbol{c}'(x)$ and $\boldsymbol{e}(x,Z)$ are not equal, then the probability that $\boldsymbol{a}'(x,z) \circ \boldsymbol{b}'(x,z) - \boldsymbol{c}'(x) = \boldsymbol{e}(z)$ over the random choices of z is at most $\frac{2\mathsf{n}-2}{|\mathbb{F}|}$. Overall, the probability of the verifier returning 1 on a false statement is bounded by $\frac{3\mathsf{n}-1}{|\mathbb{F}|}$, which is negligible. Assume now that for μ' challenges $y_0, \ldots, y_{\mu'-1}$, the verifier has negligible probability of accepting a false statement, then we show the same holds in the case of using $\mu' + 1$ challenges.

Let $\mathsf{m}' = 2^{\mu'}$ and consider the following bivariate polynomials in $Y_{\mu'}, Z$

$$\begin{aligned}
\boldsymbol{a}'(Y_{\mu'}, Z) &= \boldsymbol{a}'_{-1} + \sum_{i=0,j=0}^{2\mathsf{m}'-1,\mathsf{n}-1} \boldsymbol{a}_{i,j} x^{i+2\mathsf{m}'j+1} y_0^{i_0} \cdots y_{\mu'-1} Y_{\mu'}^{i_{\mu'}} Z^j \\
\boldsymbol{b}'(Y_{\mu'}, Z) &= \boldsymbol{b}'_{-1} + \sum_{i=0,j=0}^{2\mathsf{m}'-1,\mathsf{n}-1} \boldsymbol{b}_{i,j} y_0^{-i_0} \cdots y_{\mu'-1} Y_{\mu'}^{-i_{\mu'}} Z^{-j} \\
\boldsymbol{c}' &= \boldsymbol{c}'_{-1} + \sum_{i=0,j=0}^{2\mathsf{m}'-1,\mathsf{n}-1} \boldsymbol{c}_{i,j} x^{i+2\mathsf{m}'j+1} \\
\boldsymbol{d}'(Y_{\mu'}) &= \sum_{t=0}^{\mu'-1} \left(\boldsymbol{d}_t^+ y_t + \boldsymbol{d}_t^- y_t^{-1}\right) + \left(\boldsymbol{d}_{\mu'}^+ Y_{\mu'} + \boldsymbol{d}_{\mu'}^- Y_{\mu'}^{-1}\right) \\
\boldsymbol{e}(Y_{\mu'}, Z) &= \sum_{r=-\mathsf{n},r\neq 0}^{\mathsf{n}} \boldsymbol{e}_r(Y_{\mu'}) Z^r
\end{aligned}$$

The verifier's check is equivalent to testing the following polynomial identity at random challenges $y_{\mu'}, z$.

$$\boldsymbol{a}'(Y_{\mu'}, Z) \circ \boldsymbol{b}'(Y_{\mu'}, Z) - \boldsymbol{c}' - \boldsymbol{d}'(Y_{\mu'}) = \boldsymbol{e}(Z)$$

The coefficient of Z^0 on the left-hand side is equal to

$$(\boldsymbol{a}'_{-1} \circ \boldsymbol{b}'_{-1} - \boldsymbol{c}'_{-1}) + \sum_{i=0,j=0}^{\mathsf{m}-1,\mathsf{n}-1} \boldsymbol{a}_{i,j} \circ \boldsymbol{b}_{i,j} x^{i+j\mathsf{m}+1} + \sum_{t=0}^{\mu'-1} \left(\boldsymbol{d}_t^+ y_t + \boldsymbol{d}_t^- y_t^{-1}\right) + \left(\boldsymbol{d}_t^+ Y_{\mu'} + \boldsymbol{d}_t^- Y_{\mu'}^{-1}\right) \tag{4.2}$$

By assumption, for a false statement the probability that

$$(\boldsymbol{a}'_{-1} \circ \boldsymbol{b}'_{-1} - \boldsymbol{c}'_{-1}) + \sum_{i=0,j=0}^{\mathsf{m}-1,\mathsf{n}-1} \boldsymbol{a}_{i,j} \circ \boldsymbol{b}_{i,j} x^{i+j\mathsf{m}+1} + \sum_{t=0}^{\mu'-1} \left(\boldsymbol{d}_t^+ y_t + \boldsymbol{d}_t^- y_t^{-1}\right) = \mathbf{0}$$

over the random choices of $x, y_0, \ldots, y_{\mu-1}$ is negligible, and therefore the polynomial (4.2) is non-zero with overwhelming probability. By the Schwartz-Zippel Lemma, the probability that the evaluation of (4.2) at a random challenge $y_{\mu'}$ gives $\mathbf{0}$ is negligible. The rest of the proof follows as in the base case. □

Efficiency. If we set $\mu \approx \log(\mathsf{m})$, the round complexity of the proof system is $\log(\mathsf{m}) + 2$. The verifier sends $\log(\mathsf{m}) + 1$ field elements to the prover through the ILC channel and has a computational cost dominated by mn field multiplications to construct the

opening queries and $\mathcal{O}(k)$ field multiplications to check it. The query complexity is $\mathsf{qc} = 3$ and the prover's communication consists of commitments to $2(\mu + \mathsf{n} + 1)$ vectors in $\mathbb{F}^k$, excluding the commitments to the rows of the matrices A, B, C. The prover has to compute coefficients $\boldsymbol{\alpha}_{t,i,j}, \boldsymbol{\beta}_{t,i,j}$ and cancellation terms $\boldsymbol{d}_t^+, \boldsymbol{d}_t^-$, which overall require $\mathcal{O}(\mathsf{mn}k)$ field multiplications. To compute the other cancellation factors $\boldsymbol{e}_r$, the prover first computes the product $\boldsymbol{a}'(Z) \circ \boldsymbol{b}'(Z)$ which requires $\mathcal{O}(k\mathsf{m}\log(\mathsf{m}))$ and then subtracts the coefficients of $\boldsymbol{c}'$ and $\boldsymbol{d}'$. The efficiency of the proof system is summarised in Table 4.3.

Prover computation	$T_{\mathcal{P}_{\mathsf{ILC}}}$	$= \mathcal{O}(k\mathsf{n}\log\mathsf{n} + k\mathsf{mn})\ \mathbb{F}^\times$
Verifier computation	$T_{\mathcal{V}_{\mathsf{ILC}}}$	$= \mathcal{O}(\mathsf{mn} + k)\ \mathbb{F}^\times$
Prover communication	t	$= 2(\log(\mathsf{m}) + \mathsf{n} + 1)\ \log\lvert\mathbb{F}^k\rvert$
Verifier communication	C_{ILC}	$= (\log(\mathsf{m}) + 2)\ \log\lvert\mathbb{F}\rvert$
Query complexity	qc	$= 3$
Round complexity	μ	$= \log(\mathsf{m}) + 2$

TABLE 4.3: Efficiency of the proof system for the Hadamard product relation $\mathcal{R}_{\text{prod}}$. $\mathbb{F}^\times$ stands for the cost of a single multiplication and $\log|\mathbb{F}|$ for the size of a field element.

4.2.4 ILC Proof for a Known Permutation Relation

We will now outline a proof of knowledge for the known permutation of the entries in two matrices $A, B \in \mathbb{F}^{\mathsf{mn}\times k}$. The corresponding relation is

$$\mathcal{R}_{\text{kperm}} = \left\{ \begin{array}{c} (pp_{\mathsf{ILC}}, u) = ((\mathbb{F}, k)\ ,\ (\mathsf{mn}, \pi), (A, B)) : \\ A, B \in \mathbb{F}^{\mathsf{mn}\times k}\ \wedge\ \pi \in \Sigma_{[\mathsf{mn}]\times[k]}\ \wedge \quad A_{\pi(i,j)} = B_{i,j}\ \forall (i,j)\ \in [\mathsf{mn}] \times [k] \end{array} \right\}$$

In [Gro09], Groth gave a known permutation argument where the prover only has to compute a linear number of field multiplications[1]. The idea behind the argument is for the verifier to pick a random Vandermonde matrix $V \in \mathbb{F}^{\mathsf{mn}\times k}$ and challenge the prover to show that

$$\sum_{i=1,j=1}^{\mathsf{mn},k} A_{i,j}V_{\pi(i,j)} = \sum_{i=1,j=1}^{\mathsf{mn},k} B_{i,j}V_{i,j}$$

[1]This is the computational cost for the prover if we ignore the cost of committing to vectors. The argument relies on homomorphic commitments and is instantiated with Pedersen commitments [Ped91] in groups of superpolynomial size. Therefore committing costs a linear number of exponentiations, which dominates the computational cost of the prover.

If we were to follow this approach, we would similarly get a proof for which the computational cost of the prover is a linear number of multiplications. The verifier would also incur a linear number of multiplications to compute the Vandermonde matrix challenge. Here we outline how this can be improved by reducing the computational cost of the verifier to achieve a linear number of *additions*.

To reduce the computational cost for the verifier, the main idea is to let the prover help construct the challenge matrix used in the permutation proof, and then let the verifier check that the matrix was correctly formed. Since our proof to check $\mathcal{R}_{\text{eq}}$ relations only costs a linear number of additions to verify, we can reduce the computational cost for the verifier to a linear number of additions and sublinear number of multiplications.

We represents elements $(i, j) \in [\mathsf{mn}] \times [k]$ as integers $(i-1)k+j \in [\mathsf{mn}k]$ and think of the permutation π to be either in $\Sigma_{[\mathsf{mn}]\times[k]}$ or in $\Sigma_{[\mathsf{mn}k]}$ depending on the representation used. We assume that integers in $[\mathsf{mn}k]$ can be mapped injectively into elements of the finite field $\mathbb{F}$, and consider the auxiliary matrices $V, V', J \in \mathbb{F}^{\mathsf{mn}\times k}$ such that $V_{i,j} = (i-1)k + j$, $V'_{i,j} = \pi((i-1)k + j)$, and $J_{i,j} = 1$ i.e.,

$$V = \begin{pmatrix} 1 & 2 & \cdots & k \\ k+1 & k+2 & \cdots & 2k \\ & & \ddots & \\ & & \cdots & \mathsf{mn}k \end{pmatrix} \qquad V' = \begin{pmatrix} \pi(1) & \pi(2) & \cdots & \pi(k) \\ \pi(k+1) & \pi(k+2) & \cdots & \pi(2k) \\ & & \ddots & \\ & & \cdots & \pi(\mathsf{mn}k) \end{pmatrix}$$

$$J = \begin{pmatrix} 1 & & 1 \\ & \ddots & \\ 1 & & 1 \end{pmatrix}$$

Suppose the prover has committed to the rows of the two matrices $A, B \in \mathbb{F}^{m\times k}$. The idea behind the construction is to let the the verifier pick random challenges x, y and let the prover commit to matrices $U = yV - xJ$ and $U' = yV' - xJ$. The verifier can efficiently compute the entries of U and U' with $2\mathsf{mn}k - 1$ additions. Then, he can efficiently check that the prover has committed to the correct matrices by using a proof for an equality relation $\mathcal{R}_{\text{eq}}$, which can be verified with $\mathcal{O}(\mathsf{mn}k)$ additions. Notice that $V_{\pi(i,j)} = V'_{i,j}$ and therefore if B contains the permuted entries of A, then also $B + yV'$

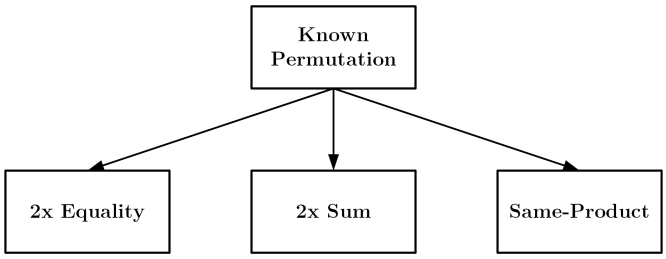

FIGURE 4.7: Decomposition of the Known Permutation Proof Over the ILC into proofs for simpler relations.

contains a permutation of the entries in $A + yV$. On the other hand, if the statement is false, then with overwhelming probability over the choices of y there will be entries in $B + yV'$ that do not appear in $A + yV$. To show that these two matrices contain the permuted entries of each another, the prover uses a (same-)product proof to show that the product of all the entries in the matrix $A + U$ is equal to the product of all the entries in $B + U'$, i.e.,

$$\prod_{i,j}^{\mathsf{mn},k} (A_{i,j} + yV_{i,j} - x) = \prod_{i,j}^{\mathsf{mn},k} (B_{i,j} + y\pi(V_{i,j}) - x) \tag{4.3}$$

By the Schwartz-Zippel Lemma this is unlikely to hold over the random choice of x unless indeed $B + yV'$ contains a permutation of the entries in $A + yV$.

The above idea relies on the invariance of polynomials under permutation of their roots. This was already used by Neff [Nef01] in the context of verifiable shuffles and later by Groth [Gro09] in proof systems for the unknown permutation relation.

At high-level our proof system for $\mathcal{R}_{\text{kperm}}$ decomposes into five sub-proofs as illustrated in Figure 4.7.

Same-Product Relation. Before moving to the description of the proof system for $\mathcal{R}_{\text{kperm}}$ we spell out the same-product relation $\mathcal{R}_{\text{same-prod}}$ which is used to check that (4.3) holds. More precisely, the relation $\mathcal{R}_{\text{same-prod}}$ checks that the product of all entries in a matrix A is the same as the product of all entries of a matrix B, i.e.,

$$\mathcal{R}_{\text{same-prod}} = \left\{ \begin{array}{c} (pp_{\mathsf{ILC}}, u) = ((\mathbb{F}, k), (\mathsf{mn}), (A, B)) : \\ A, B \in \mathbb{F}^{\mathsf{mn} \times k} \quad \wedge \quad \prod_{i,j}^{\mathsf{mn},k} A_{i,j} = \prod_{i,j}^{\mathsf{mn},k} B_{i,j} \end{array} \right\}$$

We omit the description of the proof system $(\mathcal{G}_{\mathsf{ILC}}, \mathcal{P}_{\text{same-prod}}, \mathcal{V}_{\text{same-prod}})$ over the ILC for the above relation, which can be found in [BCG+17], and recall the following result without proof.

Theorem 4.4 ([BCG+17]). *$(\mathcal{G}_{\mathsf{ILC}}, \mathcal{P}_{\text{same-prod}}, \mathcal{V}_{\text{same-prod}})$ is a proof system for the relation $\mathcal{R}_{\text{same-prod}}$ in the ILC model with perfect completeness, statistical strong knowledge soundness with straight-line extraction, and perfect special honest verifier zero-knowledge.*

This proof system uses as a building block the Hadamard product proof of Section 4.2.3 and computationally achieves similar asymptotics. The costs of the proof system are listed in Table 4.4.

Prover computation	$T_{\mathcal{P}_{\mathsf{ILC}}}$	$= \mathcal{O}(k\mathsf{n}\log\mathsf{n} + k\mathsf{mn})\ \mathbb{F}^{\times}$
Verifier computation	$T_{\mathcal{V}_{\mathsf{ILC}}}$	$= \mathcal{O}(\mathsf{mn} + k)\ \mathbb{F}^{\times}$
Prover communication	t	$= \mathcal{O}(\mathsf{mn})\ \log\lvert\mathbb{F}^k\rvert$
Verifier communication	C_{ILC}	$= (\log(\mathsf{m}) + 2)\ \log\lvert\mathbb{F}\rvert$
Query complexity	qc	$= 11$
Round complexity	μ	$= \log(\mathsf{m}) + 2$

TABLE 4.4: Efficiency of the proof system for the same-product relation $\mathcal{R}_{\text{same-prod}}$. $\mathbb{F}^{\times}$ stands for the cost of a single field multiplication and $\log|\mathbb{F}|$ for the size of a field element.

Proof System for the Known Permutation Relation. The description of the proof system for the known permutation relation $\mathcal{R}_{\text{kperm}}$ is given in Figure 4.8. We assume that the prover has already committed to the rows of matrices A, B and delimit this step with $/^*$ and $^*/$. Notice that in case the matrices A and B are equal, the entries in A lying on the same cycle in the decomposition of π are guaranteed to be the same. This special case is the one we will use in the next section within the proof for the satisfiability of an arithmetic circuit, and specifically to check the consistency of the wiring of the circuit.

Theorem 4.5. *$(\mathcal{G}_{\mathsf{ILC}}, \mathcal{P}_{\text{perm}}, \mathcal{V}_{\text{perm}})$ is a proof system for the relation $\mathcal{R}_{\text{kperm}}$ in the ILC model with perfect completeness, statistical strong knowledge soundness with straight-line extraction, and perfect special honest verifier zero-knowledge.*

Proof. We start by showing completeness. If the statement is true, then the matrices A' and B' are such that $A'_{\pi(i,j)} = B'_{i,j}$. Therefore each entry of A' appears somewhere in matrix B'. Perfect completeness follows by the perfect completeness of the sub-proofs.

$\mathcal{P}_{\text{kperm}}(pp_{\mathsf{ILC}}, (\mathsf{mn}, \pi), (A, B))$	$\mathcal{V}_{\text{kperm}}(pp_{\mathsf{ILC}}, (\mathsf{mn}, \pi, [A], [B]))$
Round 1:	**Round 1**
/ *ILC $\leftarrow \bullet$ Send $(\textbf{commit}, A, B)$ to the ILC */	/ *ILC $\rightarrow \circ$ Get message $2\mathsf{mn}$ from the ILC */
	• For $(i, j) \in [\mathsf{mn}] \times [k]$
	– $V_{i,j} := (i-1)k + j$
	– $V'_{i,j} := \pi((i-1)k + j)$
	• $x, y \leftarrow \mathbb{F}$
	• $U := yV - xJ$
	• $U' := yV' - xJ$
	ILC $\leftarrow \bullet$ Send $(\textbf{send}, x, y)$ to the ILC
Round 2 to $\log \mathbf{m} + 2$:	**Round 2 to $\log \mathbf{m} + 2$:**
ILC $\rightarrow \circ$ Get message (x, y) from ILC	ILC $\rightarrow \circ$ Get message $4\mathsf{mn}$ from the ILC
• For $(i, j) \in [\mathsf{mn}] \times [k]$	• Run $\mathcal{V}_{\text{eq}}(pp_{\mathsf{ILC}}, (U, [U]))$
– $V_{i,j} := (i-1)k + j$	• Run $\mathcal{V}_{\text{eq}}(pp_{\mathsf{ILC}}, (U', [U'])$
– $V'_{i,j} := \pi((i-1)k + j)$	• Run $\mathcal{V}_{\text{sum}}(pp_{\mathsf{ILC}}, (\mathsf{mn}, [A], [U], [A']))$
• $U := yV - xJ$	• Run $\mathcal{V}_{\text{sum}}(pp_{\mathsf{ILC}}, (\mathsf{mn}, [B], [U'], [B']))$
• $U' := yV' - xJ$	• Run $\mathcal{V}_{\text{same-prod}}(pp_{\mathsf{ILC}}, (\mathsf{mn}, [A'], [B']))$
• $A' := A + U$	• If all the sub-proofs accept return 1, else return 0
• $B' := B + U'$	
ILC $\leftarrow \bullet$ Send $(\textbf{commit}, U, U', A', B')$ to the ILC	
• Run $\mathcal{P}_{\text{eq}}(pp_{\mathsf{ILC}}, U, U)$	
• Run $\mathcal{P}_{\text{eq}}(pp_{\mathsf{ILC}}, U', U')$	
• Run $\mathcal{P}_{\text{sum}}(pp_{\mathsf{ILC}}, (\mathsf{mn}), (A, U, A'))$	
• Run $\mathcal{P}_{\text{sum}}(pp_{\mathsf{ILC}}, (\mathsf{mn}), (B, U', B'))$	
• Run $\mathcal{P}_{\text{same-prod}}(pp_{\mathsf{ILC}}, (\mathsf{mn}), (A', B'))$	

FIGURE 4.8: Proof of knowledge for the relation $\mathcal{R}_{\text{kperm}}$.

Next we show statistical strong knowledge soundness. As usual, the knowledge extractor sees the vectors sent from the prover to the ILC, and therefore it has straight-line extraction. The knowledge soundness of the equality and of the sum sub-proofs guarantees that, apart with negligible probability, matrices U, U' are correctly formed and that committed matrices A', B' are the sums of $A + U$, and $B + U'$, respectively. Moreover, from the knowledge soundness of the same-product proof, we get $\prod_{i,j}(A_{i,j} + yV_{i,j} - x) = \prod_{i,j}(B_{i,j} + y\pi(V_{i,j}) - x)$. If the statement is false, then A, B have different entries and the Schwartz-Zippel Lemma tells us that also $A + yV$ and $B + yV'$ have different entries with overwhelming probability, over the choices of y. By applying

the Schwartz-Zippel Lemma again, the probability over the random choice of $x \leftarrow \mathbb{F}$ of the above equality to hold is at most $\frac{\mathsf{mn}k}{|\mathbb{F}|}$, which is negligible.

Finally, the proof system has perfect SHVZK. Matrices U and U' can be computed given the statement and the random challenges sent from the verifier. These are sufficient to simulate the view of the equality sub-proofs. The sub-proofs for the sum relation can be trivially simulated. The simulator then is only required to execute the simulator of the same-product sub-proof. □

Efficiency. The round complexity of the proof is of one round more than the same-product proof, and thus $\log(\mathsf{m})+3$. The number of queries done by the verifier is equal to the sum of the queries of the sub-proofs, i.e. $\mathsf{qc} = 15$. The verifier communication is $\mathcal{O}(\log(\mathsf{m}))$ challenges and his computational complexity is dominated by $\mathcal{O}(\mathsf{mn}k)$ field additions. The communication complexity for the prover is $\mathcal{O}(\mathsf{mn})$ vectors of length k, which are sent within the same-product sub-proof. The computational complexity of the prover is dominated by the cost of the same-product proof, which is in turn dominated by the cost of the product proof of Section 4.2.3, and therefore is equal to $\mathcal{O}(\mathsf{mn}k + k\mathsf{n}\log(\mathsf{n}))$. The efficiency of the proof system is summarised in Table 4.9.

Prover computation	$T_{\mathcal{P}_{\mathsf{ILC}}}$	$= \mathcal{O}(k\mathsf{n}\log\mathsf{n} + k\mathsf{mn})\ \mathbb{F}^{\times}$
Verifier computation	$T_{\mathcal{V}_{\mathsf{ILC}}}$	$= \mathcal{O}(k\mathsf{mn})\ \mathbb{F}^{+}$
Prover communication	t	$= \mathcal{O}(\mathsf{mn})\ \log\lvert\mathbb{F}^k\rvert$
Verifier communication	C_{ILC}	$= (\log\mathsf{m}+3)\ \log\lvert\mathbb{F}\rvert$
Query complexity	qc	$= 15$
Round complexity	μ	$= \log\mathsf{m}+3$

FIGURE 4.9: Efficiency of the proof of knowledge for the $\mathcal{R}_{\mathrm{kperm}}$ relation. $\mathbb{F}^{\times}$ stands for the cost of a single field multiplication, $\mathbb{F}^{+}$ stands for the cost of a single field addition, and $\log|\mathbb{F}|$ for the size of a field element.

4.3 ILC Proofs for the Satisfiability of an Arithmetic Circuit

In this section we give our proof of knowledge for the satisfiablity of an arithmetic circuit over the ILC. Given the relation specified in the previous section, we can rewrite

the relation $\mathcal{R}_{\text{AC}}$ for the satisfiability of an arithmetic circuit as follows.

$$\mathcal{R}_{\text{AC}} = \left\{ \begin{array}{ll} (pp, u, w) := \Big((\mathbb{F}, k, *) \ , \ (m_A, m_M, \pi, \{\boldsymbol{u}_i\}_{i\in S}) \ , \ \big(\{\boldsymbol{v}_i\}_{i\in[m]}\big) \Big) : & \\ \quad m = 3m_A + 3m_M & \wedge\, S \subseteq [m] \\ \wedge\, A = (\boldsymbol{v}_i)_{i=1}^{m_A} & \wedge\, D = (\boldsymbol{v}_i)_{i=3m_A+1}^{3m_A+m_M} \\ \wedge\, B = (\boldsymbol{v}_i)_{i=m_A+1}^{2m_A} & \wedge\, E = (\boldsymbol{v}_i)_{i=3m_A+m_M+1}^{3m_A+2m_M} \\ \wedge\, C = (\boldsymbol{v}_i)_{i=2m_A+1}^{3m_A} & \wedge\, F = (\boldsymbol{v}_i)_{i=3m_A+2m_M+1}^{3m_A+3m_M} \\ & \wedge\, V = (\boldsymbol{v}_i)_{i=1}^{m} \\ \wedge\, (pp, \{\boldsymbol{u}_i\}_{i\in S}, \{\boldsymbol{v}_i\}_{i\in S}) \in \mathcal{R}_{\text{eq}} & \wedge\, (pp, m_A, (A, B, C)) \in \mathcal{R}_{\text{sum}} \\ \wedge\, (pp, m_M, (D, E, F)) \in \mathcal{R}_{\text{prod}} & \wedge\, (pp, (m, \pi), (V, V)) \in \mathcal{R}_{\text{perm}} \end{array} \right\}$$

In the proof for arithmetic circuit satisfiability, the prover starts by committing to all values $\{\boldsymbol{v}_i\}_{i=1}^{m}$ using the conventions we introduced in Section 4.1. She will then execute the following sub-proofs in parallel

- An equality proof for showing that consistency of the wire values $\boldsymbol{u}_{i\in S}$ in the instance with the committed vectors $\boldsymbol{v}_{i\in S}$.
- A proof for the sum of committed matrices A, B, C to show the consistency of wire values with the addition gates in the circuit, i.e. $A + B = C$.
- A proof for the Hadamard product of committed matrices D, E, F to show the consistency of wire values with the multiplication gates, i.e. $D \circ E = F$.
- A known permutation proof to show the consistency of the wire assignments with the wiring of the circuit: this is done by showing that the matrix V storing the assignments to the wire values remains unchanged when applying the permutation π encoding the wiring of the circuit, i.e. $V_{i,j} = V_{\pi(i,j)}$.

The description of our proof of knowledge for the relation $\mathcal{R}_{\text{AC}}$ is given in Figure 4.10.

Theorem 4.6. *$(\mathcal{G}_{\mathsf{ILC}}, \mathcal{P}_{\text{AC}}, \mathcal{V}_{\text{AC}})$ is a proof system for $\mathcal{R}_{\text{AC}}$ in the ILC model with perfect completeness, statistical strong knowledge soundness with straight-line extraction, and perfect special honest-verifier zero-knowledge.*

Proof. Perfect completeness follows from the perfect completeness of the sub-proofs.

$\mathcal{P}_{\mathsf{AC}}(pp_{\mathsf{ILC}}, (m_A, m_M, \pi, \{\boldsymbol{u}_i\}_{i\in S}), (\{\boldsymbol{v}_i\}_{i\in[m]}))$	$\mathcal{V}_{\mathsf{AC}}(pp_{\mathsf{ILC}}, (m_A, m_M, \pi, \{\boldsymbol{u}_i\}_{i\in S}, \{[\boldsymbol{v}_i]\}_{i\in[m]}))$
• $A := (\boldsymbol{v}_i)_{i=1}^{m_A}$	
• $B := (\boldsymbol{v}_i)_{i=m_A+1}^{2m_A}$	
• $C := (\boldsymbol{v}_i)_{i=2m_A+1}^{3m_A}$	
• $D := (\boldsymbol{v}_i)_{i=3m_A+1}^{3m_A+m_M}$	
• $E := (\boldsymbol{v}_i)_{i=3m_A+m_M+1}^{3m_A+2m_M}$	
• $F := (\boldsymbol{v}_i)_{i=3m_A+2m_M+1}^{3m_A+3m_M}$	
• $U := (\boldsymbol{v}_i)_{i\in S}$	
• $V := (\boldsymbol{v}_i)_{i=1}^{3(m_A+m_M)}$	
ILC $\leftarrow$ • Send $(\textbf{commit}, V)$ to the ILC	ILC $\rightarrow$ ∘ Get message $3(m_A + m_M)$ from the ILC
• Run $\mathcal{P}_{\text{eq}}(pp_{\mathsf{ILC}}, U, U)$	• $m := 3(m_A + m_M)$
• Run $\mathcal{P}_{\text{sum}}(pp_{\mathsf{ILC}}, (m_A), (A, B, C))$	• $U := (\boldsymbol{u}_i)_{i\in S}$
• Run $\mathcal{P}_{\text{prod}}(pp_{\mathsf{ILC}}, (m_M), (D, E, F))$	• Run $\mathcal{V}_{\text{eq}}(pp_{\mathsf{ILC}}, U, [U])$
• Run $\mathcal{P}_{\text{perm}}(pp_{\mathsf{ILC}}, (3(m_A+m_M), \pi), (V, V))$	• Run $\mathcal{V}_{\text{sum}}(pp_{\mathsf{ILC}}, (m_A, [A], [B], [C]))$
	• Run $\mathcal{V}_{\text{prod}}(pp_{\mathsf{ILC}}, (m_M, [D], [E], [F]))$
	• Run $\mathcal{V}_{\text{perm}}(pp_{\mathsf{ILC}}, (m, \pi, [V], [V]))$
	• If all the sub-proofs accept return 1, Else return 0

FIGURE 4.10: Proof of knowledge for the relation $\mathcal{R}_{\text{AC}}$ over the ILC model.

Statistical strong knowledge soundness follows from the strong knowledge soundness of the sub-proofs. The statistical strong knowledge soundness of the equality sub-proof guarantees that commitments to values included in the instance indeed contain the publicly known values. The correctness of the addition and the multiplication gates follows from the statistical knowledge soundness of the respective sub-proofs. Finally, as we have argued above, the permutation sub-proof guarantees that the committed values respect the wiring of the circuit. Since all sub-proofs have knowledge soundness with straight line extraction, so does the main proof.

Perfect SHVZK follows from the perfect SHVZK of the sub-proofs. A simulated transcript is obtained by combining the outputs of the simulators of all the sub-proofs.

□

Efficiency. The efficiency of our arithmetic circuit satisfiability proof in the ILC model is given in Table 4.5. The computational cost is dominated by the permutation proof, where the matrices have higher dimensions, so we choose m, n such that $m = 3m_A + 3m_M = \mathsf{mn}$. The total number of gates is $N = \frac{\mathsf{mn}k}{3}$. The asymptotic results displayed

below are obtained when $\mathsf{m} = \mathcal{O}(\log N)$ and the vector length k specified by pp_{ILC} is approximately $\sqrt{N}$. While k does not play a significant role in the efficiency over the ILC, the compiled argument from Chapter 6 will achieve optimal communication complexity over the standard channel for $k \approx \sqrt{N}$. The query complexity qc is the number of linear combinations the verifier queries from the ILC channel in the opening query. The verifier communication C_{ILC} is the number of messages sent from the verifier to the prover via the ILC channel and in our proof system it is proportional to the number of rounds. Let μ be the number of rounds in the ILC proof and $t_1, \ldots, t_\mu$ be the numbers of vectors that the prover sends to the ILC channel in each round, and let $t = \sum_{i=1}^{\mu} t_i$.

Prover computation	$T_{\mathcal{P}_{\mathsf{ILC}}}$	$= \mathcal{O}(N)\ \mathbb{F}^{\times}$
Verifier computation	$T_{\mathcal{V}_{\mathsf{ILC}}}$	$= \mathcal{O}(N)\ \mathbb{F}^{+}$
Prover Communication	t	$= \mathcal{O}\left(\sqrt{N}\right) \log \lvert\mathbb{F}^k\rvert$
Verifier communication	C_{ILC}	$= \mathcal{O}(\log\log(N))\ \log \lvert\mathbb{F}\rvert$
Query complexity	qc	$= 20$
Round complexity	μ	$= \mathcal{O}(\log\log(N))$

TABLE 4.5: Efficiency of our proof of knowledge for the relation $\mathcal{R}_{\mathrm{AC}}$ in the ILC model. $\mathbb{F}^{\times}$ stands for the cost of a single field multiplication, $\mathbb{F}^{+}$ stands for the cost of a single field addition, and $\log |\mathbb{F}|$ for the size of a field element.

Chapter 5

Proofs for the Execution of TinyRAM Programs in the ILC Model

In the previous chapter we constructed proofs for the evaluation of arithmetic circuits. In practice however, computation is usually not expressed in form of circuits, but it is more conveniently expressed as computer programs. In this chapter we move one step closer towards practicality and construct proofs for the correct execution of RAM programs. More precisely we give proofs for the correct execution of TinyRAM programs in the ILC model. TinyRAM is a particularly convenient RAM machine to use when constructing zero-knowledge proofs for correct program execution: it has an expressive architecture that allows to efficiently compile high-level languages into machine code (see for example [BCG+13a]); the architecture contains a minimal instruction set which can be efficiently reduced to a set of algebraic operations. While the proofs presented in this chapter are mostly of theoretical interest, we see them as an important step towards reducing the concrete computational overhead of proofs for correct execution in practice.

Our proofs achieve perfect completeness, statistical strong knowledge soundness and perfect special honest verifier zero knowledge in the ILC model. As in the previous chapter, the knowledge soundness of our proofs relies on applications of the Schwartz-Zippel Lemma and thus we set the ILC to operate on a large field, i.e. $|\mathbb{F}| \approx \lambda^{\omega(1)}$.

Our proof system is highly efficient for computationally intensive programs for which the execution time dominates the other parameters, i.e. memory and program

length. For a TinyRAM program terminating in T steps, our prover computational cost amounts to $\mathcal{O}(T)$ field multiplications, while the verifier runs in $\mathcal{O}(\sqrt{T})$ field multiplications. We also measure the performance of prover and verifier in TinyRAM operations to get a more concrete figure of their computational overhead. We consider TinyRAM to operate on words of length $W = \Theta(\log(\lambda))$ so that the program can address and use a polynomial amount of memory $M = \text{poly}(\lambda)$. Therefore, the ratio $e = \frac{\log(|\mathbb{F}|)}{W} = \omega(1)$ is superconstant in the security parameter, and the cost of a field multiplication implemented in TinyRAM is $\alpha = \mathcal{O}(e^2)$. This means that if we had to implement our prover as a TinyRAM program using the same word size as the program in the instance, her running time would be $\mathcal{O}(\alpha T)$ steps, for an arbitrarily small superconstant function $\alpha(\lambda) = \omega(1)$. In Chapter 6 we will show how to compile these ILC proofs into standard proofs and arguments over the standard channel. We also suggest instantiations that preserve the computational efficiency achieved over the ILC.

5.1 Overview

Typical applications of zero-knowledge proofs are concerned with assuring that a participant in a protocol is behaving honestly. This can be formulated as checking that a participant, supposedly running program P, on public input x and private input w provides the correct output z. Without loss of generality, we can formulate the verification as an extended program that takes public input $v = (x, z)$ and answers 0 if and only if z is the output of the computation. We therefore formulate correct program execution as the program just answering 0.

In our proof system instances are of the form $u = (P, v, T, M)$, where P is a TinyRAM program, v is a list of words given as input to the program, T is a time bound, and M is the size of the memory. A witness w is another list of words and can be seen as the private inputs to the program. We assume without loss of generality that the witness is appended by 0's, such that $|v| + |w| = M$ and that the program starts with the memory initialised to these words.

The statement we want to prove is that the program P terminates with the instruction **answer** 0 within T steps, using M words of memory on the public input v, and

private input w. We let the public parameter define the word size, the number of registers, and upper bounds on the program size, time and memory. Correct program execution is given by the relation

$$\mathcal{R}_{\text{TinyRAM}} = \left\{ \begin{array}{c} (pp, u, w) := ((W, K, *), (P, v, T, M), w) : \\ P \text{ is a TinyRAM program with } W\text{-bit words, } K \text{ registers,} \\ \text{and } M \text{ words of addressable memory, which on inputs } v \text{ and } w \\ \text{terminates in } T \text{ steps with the instruction } \textbf{answer } 0. \end{array} \right\}$$

Our main interest is to prove the correct execution of programs that require "heavy computation", i.e. for which the number of executed steps outweigh the other parameters. We will assume throughout the chapter that $T \gg L + M$, where L is the number of instructions in the program.

In practice, the typical approach ([BCG+13a; BCT+14]) to check the above relation is to first reduce the statement to a set of arithmetic constraints or to an arithmetic circuit, and then use an efficient proof system to verify the consistency of these. In all existing systems to verify the correct program execution, the prover incurs in a polylogarithmic ($\omega(\log \lambda)$) overhead in computation. The source of this overhead in these systems is both at the front-end level, i.e. in the arithmetization of the program, and at the cryptographic back-end level, i.e. the choice of the proof system used for handling the reduced statement. For the back-end, the choice of the proof system is usually SNARK-based, in which the prover computes a linear number (in the circuit size) of *exponentiations* in a cyclic group of superpolynomial size, each requiring a superlogarithmic number of field operations. The source of the overhead at the front-end has to do with the circuit produced by the reduction which is of size $\Omega(T \log^2(T))$. In this chapter we focus on reducing the overheads at the back-end level and in Chapter 6 we discuss instantiations of our ILC proofs which preserve their computational efficiency, i.e. the compilation only introduces a constant computational overhead.

Improved Arithmetization. At an high level, [BCG+13a; BCT+14] check the execution of TinyRAM by committing to the *execution trace* of the program, which stores the state of the computation at each step, embedded it into field elements. The correct transition from one step in the execution trace to the next one is checked by an

arithmetic circuit taking as inputs the two states of the computation. To check that the memory is correctly accessed during the execution, the prover commits to another copy of the execution trace, but this time sorted first by memory access and then by execution time, rather than execution time only. The consistency of two consecutive accesses to the same memory location can be checked by another arithmetic circuit taking as inputs two consecutive steps in the memory-sorted trace. The only thing left to check is the consistency of the two committed execution traces, i.e. that they are a permutation of each other.

One way to check the consistency of the two sorted traces is by using another arithmetic circuit which embeds a permutation network, such as a Beneš network [Ben65]. For T nodes in the network, these require $\mathcal{O}(\log(T))$ layers of switches of constant size, and can be evaluated by an arithmetic circuit of size $\mathcal{O}(T(\log T)^2)$. To reduce this overhead, instead of evaluating a permutation by using an arithmetic circuit, we simply check the existence of a permutation that maps one memory access to the next one at the same address. This can be done much more efficiently using techniques similar to the ones we outlined in Section 4.2.4, which were first suggested by Neff [Nef01] in the context of verifiable shuffles. In this way, we avoid committing to the memory-sorted execution trace and we check the memory directly on the time-sorted trace.

We recall that the instruction set of TinyRAM (Table 3.1) contains both arithmetic operations such as addition and multiplication of words, and logical operations such as bit-wise XOR, AND and OR. To verify the execution of a logical instruction using arithmetic operations one can decompose words into single bits that are handled individually. Bit-decomposition makes it easy to implement the logical operations using arithmetic circuits, but introduces an overhead when embedding bits into full size field elements. More specifically, if a program uses a polynomial amount of memory $M = \mathrm{poly}(\lambda)$, then we need to allow the word size used by the TinyRAM machine to be at least $W = \Omega(\log(\lambda))$ to be able to address all the memory used. Hence, if we use bit-decomposition, the size of the arithmetic circuit used to check the transition of states is proportional to W. Checking all T transitions in the execution trace is equivalent to checking a circuit of size $\mathcal{O}(T \log(\lambda))$. To remove the overhead of bit-decomposition we introduce a less costly decomposition. While additions and multiplications are

manageable using a natural embedding of words into field elements, such a representation is not well suited to logical operations. However, instead of decomposing words into individual bits, we decompose them into interleaved odd-position bits and even-position bits. A tuple (a_3, a_2, a_1, a_0) can for instance be decomposed into $(a_3, 0, a_1, 0) + (0, a_2, 0, a_0)$. The key point of this idea is that adding two interleaved even bit nibbles yields $(0, a_2, 0, a_0) + (0, b_2, 0, b_0) = (a_2 \wedge b_2, a_2 \oplus b_2, a_0 \wedge b_0, a_0 \oplus b_0)$. Using another decomposition into odd-position and even-position bits we can now extract the XORs and the ANDs. Using this core idea, it is possible to represent all logical operations using field additions together with decomposition into odd and even-position bits. This reduces the verification of logical operations to verify correct decomposition into odd and even bits.

To enable decomposition proofs into odd and even-position bits, we develop a new lookup proof that makes it possible to check that a field element belongs to a table of permitted values. By creating a lookup table of all words with even-position bits, we make it possible to verify such decompositions. Odd and even-bits decomposition and lookup proofs also enable to verify that a field element represents represents a valid word within the range $[0, \ldots, 2^W - 1]$.

Chapter Outline. In the next section we describe how to embed the execution trace of a TinyRAM program into field elements, the relation of correct program execution induced by this encoding and describe the idea behind the even/odd-bit decomposition. In Section 5.3 we then look at the decomposition of the arithmetized TinyRAM relation in terms of few building-block relations. In Section 5.4 we give an overview of the proof systems for the building blocks. We refer to [BCG+18] for the full specifications of the unknown permutation and lookup proofs. Lastly, in Section 5.5 we combine them to give a proof for the correct execution of a TinyRAM program.

5.2 Arithmetization of TinyRAM

As a first step towards the realisation of proofs for the correct execution of TinyRAM programs we translate $\mathcal{R}_{\mathrm{TinyRAM}}$ into a more amenable relation involving elements in a finite field. Given a TinyRAM machine with word-size W and a finite field $\mathbb{F}$,

we can in a natural way embed words into field elements by encoding a word $a \in \{0, \ldots, 2^W - 1\}$ as the field element $a \cdot 1_{\mathbb{F}} = 1_{\mathbb{F}} + \cdots + 1_{\mathbb{F}}$ (a times). We will use finite fields of characteristic $p > 2^{2W} - 2^{W-1}$ because then sums and products of words are less than p and we avoid overflow when applying field operations to the embedded words. We encode the program P, memory and states of a TinyRAM program as tuples of field elements. We then introduce a new relation $\mathcal{R}^{\mathsf{field}}_{\mathrm{TinyRAM}}$ consisting of a set of arithmetic constraints these encodings should satisfy to guarantee the correct program execution. The relation will take instances $u = (P, v, T, M)$, and witnesses w consisting of the encodings of program, memory, the execution trace as well as a set of auxiliary field elements. It will be the case that the encoding of the witness can be done alongside an execution of the program in $\mathcal{O}(L + M + T)$ field operations.

5.2.1 Formatting the Witness

Given a correct program execution we encode program, memory and states of the TinyRAM machine as field elements and arrange them in a number of tables as pictured in Figure 5.1. The execution table Exe, contains the field elements encoding the states of the TinyRAM machine. It consists of T rows, where row t describes the state at the beginning of step t. A row includes field elements that encode the time t, the program counter pc_t, the instruction $\mathsf{inst}_{\mathsf{pc}_t}$ corresponding to the program line pc_t, an immediate value A_t, the values $\mathsf{r}_{0,t}, \ldots, \mathsf{r}_{K-1,t}$ contained in the registers $\mathsf{reg}_0, \ldots, \mathsf{reg}_{K-1}$ at time t, and the flag flag_t. The next row contains the resulting state of the TinyRAM machine at time $t + 1$. Each row also includes a memory address addr_t, and the value $\mathsf{v}_{\mathsf{addr}_t}$ stored at this address after the execution of the step, as well as a constant number of auxiliary field elements to be specified later that will be used to check correctness of program execution.

The next table is the program table Prog, which contains the field elements encoding of the TinyRAM program P. Each row contains the description of one line of the program, consisting of one instruction, at most three operands, and possibly an immediate value, i.e. a constant value specified by the instruction. Note that the instruction and the operands are encoded together using a single single field element. Furthermore, we introduce a constant number of auxiliary field elements in each row. These entries can be efficiently computed given the program line stored in the same

Time	pc	Instruction	Immediate	reg_0	...	reg_{K-1}	Flag	Address	Value	$\boldsymbol{aux}_{\mathsf{Exe}}$
1	0	inst_0	A_0	0	...	0	0	0	0	...
					⋮					
t	pc_t	$\mathsf{inst}_{\mathsf{pc}_t}$	A_t	$\mathsf{r}_{0,t}$	...	$\mathsf{r}_{K-1,t}$	flag_t	addr_t	$\mathsf{v}_{\mathsf{addr}_t}$	...
$t+1$	pc_{t+1}	$\mathsf{inst}_{\mathsf{pc}_{t+1}}$	A_{t+1}	$\mathsf{r}_{0,t+1}$	...	$\mathsf{r}_{K-1,t+1}$	flag_{t+1}	addr_{t+1}	$\mathsf{v}_{\mathsf{addr}_{t+1}}$	...
					⋮					
T	pc_T	**answer**	0	$\mathsf{r}_{0,T}$	...	$\mathsf{r}_{K-1,T}$	flag_T	addr_T	$\mathsf{v}_{\mathsf{addr}_T}$	...

(A) The execution table Exe.

pc	Instruction	Immediate	$\boldsymbol{aux}_{\mathsf{Prog}}$
0	inst_0	A_0	...
	⋮		
$L-1$	inst_{L-1}	A_{L-1}	...

(B) The program table Prog.

Address	Initial value	usd
0	0	0
1	v_1	0
	⋮	
$M-1$	v_{M-1}	0
0	0	1
1	v_1	1
	⋮	
$M-1$	v_{M-1}	1

(C) The memory table Mem.

Values
0
1
4
5
⋮
$\sum_{i=0}^{\frac{W}{2}-1} 2^{2i}$

(D) The table EvenBits.

Values	Powers
0	1
1	2
2	4
3	8
⋮	⋮
$W-1$	2^{W-1}
W	0

(E) Table Pow.

FIGURE 5.1: The execution table Exe, the program table Prog, the memory table Mem, the table EvenBits and the table Pow.

row and will help verifying its execution, e.g. we encode the position of input and output registers as auxiliary field elements.

The memory table Mem has rows that list the possible memory addresses, their initial values, and an auxiliary field element $\mathsf{usd} \in \{0, 1\}$. For every pair of address and corresponding initial value, the memory table Mem contains a row in which $\mathsf{usd} = 0$ and another row in which $\mathsf{usd} = 1$. Recall that the memory is initialised with input words listed in v, w, i.e., the input words contributing to the instance and witness of the relation $\mathcal{R}_{\mathrm{TinyRAM}}$.

In addition to these, we also consider two auxiliary lookup tables EvenBits, Pow.

The former contains the encoding of words of length W whose binary expansion has 0 in all odd positions. This table contains $2^{\frac{W}{2}}$ field elements and will be used to check that certain field elements encode a word of length W. The latter table stores in the first column the encoding of integers in $i \in [0, W]$, and in the second column the powers $2^i \mod 2^W$. This table contains $2(W+1)$ field elements and will be used to efficiently check shift instructions.

5.2.2 Arithmetized TinyRAM Relation

Let $(\mathsf{Exe}, \mathsf{Prog}, \mathsf{Mem}, \mathsf{EvenBits}, \mathsf{Pow})$ be the tables of field elements encoding the program execution and the auxiliary values. We can now reformulate the correct execution of a TinyRAM program defined by $\mathcal{R}_{\mathrm{TinyRAM}}$ as a relation that imposes a number of constraints to the entries in the tables:

$$\mathcal{R}^{\mathsf{field}}_{\mathrm{TinyRAM}} = \left\{ \begin{array}{c} (pp, u, w) := ((W, K, \mathbb{F}, *), (P, v, T, M), \mathsf{w}) : \\ \mathsf{w} := (\mathsf{Exe}, \mathsf{Prog}, \mathsf{Mem}, \mathsf{EvenBits}, \mathsf{Pow}, *) \\ (pp, (P, v, T, M), \mathsf{w}) \in \mathcal{R}_{\mathrm{check}} \\ (pp, (T, M), \mathsf{w}) \in \mathcal{R}_{\mathrm{mem}} \\ (pp, \bot, \mathsf{w}) \in \mathcal{R}_{\mathrm{step}} \end{array} \right\}$$

The relations $\mathcal{R}_{\mathrm{check}}, \mathcal{R}_{\mathrm{mem}}, \mathcal{R}_{\mathrm{step}}$ jointly guarantee that the witness w consists of field elements encoding a correct TinyRAM execution that answers 0 in T steps using M words of memory, public input v, and additional private inputs. More precisely,

- $\mathcal{R}_{\mathrm{check}}$: it checks the initial values v of the memory are correctly included into Mem, the program P is correctly encoded in Prog, the tables $\mathsf{EvenBits}$ and Pow contain the correct encodings of the auxiliary lookup tables, the initial state of the TinyRAM machine is correct and that it terminates with **answer** 0 in step T.
- $\mathcal{R}_{\mathrm{mem}}$: it checks that memory usage is consistent throughout the execution of the program. That is, if a memory value is loaded at time t then it should match the last stored value at the same address.
- $\mathcal{R}_{\mathrm{step}}$: it checks that each step of the execution has been performed correctly.

In Section 5.3 we describe $\mathcal{R}_{\text{check}}$, $\mathcal{R}_{\text{mem}}$ and $\mathcal{R}_{\text{step}}$, gradually decomposing them into smaller and simpler relations. Ultimately, we specify each of these subrelations in terms of some building blocks: equality, lookup, unknown permutation and range relations.

5.2.3 Building-Block Relations

Next, we describe the building-block relations used in the decomposition of $\mathcal{R}^{\mathsf{field}}_{\text{TinyRAM}}$. As for the previous chapter the following relations refer to tables in the witness that have been previously committed by the prover. In the instance, we sometimes denote commitments to tables as $[\mathsf{Tab}]$ to clarify which statement we refer to. However, we avoid doing so in the description of the relations to reduce the complexity of the notation.

Equality Relations. The equality relation $\mathcal{R}_{\text{eq}}$ checks that rows Tab_i of a table Tab in the witness encode tuples $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_m$ of given W-bit words

$$\mathcal{R}_{\text{eq}} = \left\{ \begin{array}{c} (pp, u, w) := ((W, K, \mathbb{F}, *), (\boldsymbol{v}_1, \ldots, \boldsymbol{v}_m), \mathsf{Tab}) \ : \\ \mathsf{Tab} := \{\mathsf{Tab}_i\}_i \wedge \mathsf{Tab}_i = \boldsymbol{v}_i \cdot 1_{\mathbb{F}} \ \forall \ i \in [m] \end{array} \right\}$$

This is equivalent to the equality relation we introduced in Section 4.2.1. The only difference here is that public entries in the instance may be represented as words, instead of field elements. Therefore the equality relation above includes checking that a word has been correctly encoded in a field element.

Lookup Relations. A lookup relation checks the membership of a tuple of field elements $\boldsymbol{w}$ in the set of rows of a table Tab. This differs from the previous relation as both $\boldsymbol{w}$ and Tab are in the witness.

$$\mathcal{R}_{\text{lookup}} = \left\{ \begin{array}{c} (pp, u, w) := ((W, K, \mathbb{F}, *), \perp, (\boldsymbol{w}, \mathsf{Tab})) \ : \\ \mathsf{Tab} := \{\mathsf{Tab}_i\}_i \wedge \exists \ i : \mathsf{Tab}_i = \boldsymbol{w} \end{array} \right\}$$

We extend this relation in the natural way to check the membership of multiple tuples $\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots$ in a table.

Unknown Permutation Relations. An unknown permutation relation can be used to check that two ordered sets of the same size are permutation of each other. Differently from the known permutation relation we introduced in Section 4.2.4, the permutation may not be publicly known.

$$\mathcal{R}_{\text{perm}} = \left\{ \begin{array}{c} (pp, u, w) := ((W, K, \mathbb{F}, *), m, (\{\mathtt{a}_i, \mathtt{b}_i\}_{i=1}^{m}, \pi)) : \\ \pi \in \Sigma_m \wedge \mathtt{a}_{\pi(i)} = \mathtt{b}_i \end{array} \right\}$$

Range Relations. We use a range relation to check that a field element $\mathtt{a}$ can be written as a W-bit word, i.e., $\mathtt{a}$ is in the range $[0, 2^W - 1]$.

$$\mathcal{R}_{\text{range}} = \left\{ \begin{array}{c} (pp, u, w) := ((W, K, \mathbb{F}, *), W, \mathtt{a}) : \\ 0 \leq \mathtt{a} \leq 2^W - 1 \end{array} \right\}$$

As we will see next, by including the table EvenBits in the witness we can reduce a range relation to a lookup relation and a set of linear consistency checks.

5.2.4 Efficient Bit Decomposition for Range and Logical Relations

Embedding W-bit words into elements of a field of size $p > 2^{2W} - 2^{W-1}$ enables efficient verification of arithmetic TinyRAM instructions since the sum and the product of two W-bits word does not cause a modular reduction in the field. However, as we embed words into a large field one has to additionally check that inputs and output of an instruction can be represented as W-bits words. Another inconvenience is that logical operations cannot be checked directly using field operations on the embedded words. Both issues can be addressed by storing for each word $\mathtt{a}$ additional W field elements embedding the binary decomposition of $\mathtt{a}$. Looking ahead to the next sections, this decomposition would make the prover incur a multiplicative overhead proportional to $W = \Omega(\log \lambda)$. As our main goal is to reduce this overhead, we devise a different approach. In this section we introduce a new decomposition technique to enable efficient verification of range and logical relations.

Let $\mathtt{a}$ be a field element that can be written as a W-bit word. For $\mathtt{a} = \sum_{i=0}^{W-1} \mathtt{a}_i 2^i$ such that $\mathtt{a}_i \in \{0_{\mathbb{F}}, 1_{\mathbb{F}}\}$ we write $\mathtt{a} \leftrightarrow (\mathtt{a}_{W-1}, \ldots, \mathtt{a}_2, \mathtt{a}_1, \mathtt{a}_0)$. While proving range relations and logical operations, we find it helpful to split $\mathtt{a}$ into a pair of field elements

$\mathtt{a}_\mathtt{o}, \mathtt{a}_\mathtt{e}$ corresponding to the even-position bits and the odd-positions bits (shifted into even-position) of the word stored in $\mathtt{a}$, i.e.

$$\mathtt{a}_\mathtt{e} \leftrightarrow (0, \mathtt{a}_{W-2}, \ldots, 0, \mathtt{a}_2, 0, \mathtt{a}_0) \qquad \mathtt{a}_\mathtt{o} \leftrightarrow (0, \mathtt{a}_{W-1}, \ldots, 0, \mathtt{a}_3, 0, \mathtt{a}_1)$$

Given this decomposition one can recompute $\mathtt{a}$ as follows.

$$\mathtt{a} = 2\mathtt{a}_\mathtt{o} + \mathtt{a}_\mathtt{e}$$

Range Relations. To check that a field element $\mathtt{a}$ is in the range $[0, 2^W - 1]$, one could use a lookup table of size 2^W storing all values in the range and check that $\mathtt{a}$ is one of the entries in the table. However, this would give a table of size 2^W which is too large, as 2^W may even exceed the running time T of the program in the instance. Instead, we can use a shorter table $\mathsf{EvenBits}$ of size $2^{\frac{W}{2}}$, storing all the words with odd-position bits equal to zero. To check that $\mathtt{a}$ is in the range $[0, 2^W - 1]$ it is sufficient to check that $\mathtt{a} = 2\mathtt{a}_\mathtt{o} + \mathtt{a}_\mathtt{e}$ for $\mathtt{a}_\mathtt{o}, \mathtt{a}_\mathtt{e} \in \mathsf{EvenBits}$. This is summarised in the following relation

$$\mathcal{R}_{\text{range}} = \left\{ \begin{array}{c} (pp, u, w) := ((W, K, \mathbb{F}, *), \bot, (\mathtt{a}, (\mathtt{a}_\mathtt{o}, \mathtt{a}_\mathtt{e}), \mathsf{EvenBits})) : \\ (pp, \bot, ((\mathtt{a}_\mathtt{o}, \mathtt{a}_\mathtt{e}), \mathsf{EvenBits})) \in \mathcal{R}_{\text{lookup}} \ \wedge \ \mathtt{a} = 2\mathtt{a}_\mathtt{o} + \mathtt{a}_\mathtt{e} \end{array} \right\}$$

This relation can be extended to a decomposition using $\kappa = \mathcal{O}(1)$ words of length $\frac{W}{\kappa}$, reducing the size of the lookup table to $|\mathsf{EvenBits}| = 2^{\frac{W}{\kappa}}$. Each range check then requires κ lookups and a linear check over κ terms. To get good efficiency, it is important that $2^{\frac{W}{\kappa}} \ll T$. Here we assume for simplicity $2^{\frac{W}{2}} \ll T$, which allows us to use $\kappa = 2$ but our proof system can be modified to handle any $T = \text{poly}(\lambda)$ with an appropriate choice of κ.

Bitwise AND and XOR. It turns out that the above decomposition can be also exploited to check the correctness of logical operations. Here we describe the high-level idea and omit for example how to check the consistency of the flag. A full description on how the check all the TinyRAM instructions is given in Section 5.3.3. Let $\mathtt{a}, \mathtt{b}$ be the inputs of the bit-wise AND or bit-wise XOR operation, and let $\mathtt{c}$ be the output. To verify the correctness of the operation, e.g. $\mathtt{a} \wedge \mathtt{b} = \mathtt{c}$, consider the decompositions of the

inputs into their odd and even-position bits, namely $\mathtt{a} = 2\mathtt{a}_\mathtt{o} + \mathtt{a}_\mathtt{e}$ and $\mathtt{b} = 2\mathtt{b}_\mathtt{o} + \mathtt{b}_\mathtt{e}$. If we take the sum of the integers storing the even-positions $\mathtt{a}_\mathtt{e}$ and $\mathtt{b}_\mathtt{e}$ we get

$$\begin{aligned}\mathtt{a}_\mathtt{e} + \mathtt{b}_\mathtt{e} &\leftrightarrow (0, \mathtt{a}_{W-2}, \ldots, 0, \mathtt{a}_0) + (0, \mathtt{b}_{W-2}, \ldots, 0, \mathtt{b}_0) \\ &= (\mathtt{a}_{W-2} \wedge \mathtt{b}_{W-2}, \mathtt{a}_{W-2} \oplus \mathtt{b}_{W-2}, \ldots, \mathtt{a}_0 \wedge \mathtt{b}_0, \mathtt{a}_0 \oplus \mathtt{b}_0)\end{aligned}$$

The above contains the bit-wise AND of the even bits of $\mathtt{a}$ and $\mathtt{b}$ placed in odd position and the bit-wise XOR of the even bits of $\mathtt{a}$ and $\mathtt{b}$ in even position. We can consider taking again the decomposition of $\mathtt{a}_\mathtt{e} + \mathtt{b}_\mathtt{e}$ into its odd and even-position bits, i.e.

$$\begin{aligned}\mathtt{e}_\mathtt{o} &\leftrightarrow (0, \mathtt{a}_{W-2} \wedge \mathtt{b}_{W-2}, 0, \ldots, 0, \mathtt{a}_0 \wedge \mathtt{b}_0) \\ \mathtt{e}_\mathtt{e} &\leftrightarrow (0, \mathtt{a}_{W-2} \oplus \mathtt{b}_{W-2}, 0, \ldots, 0, \mathtt{a}_0 \oplus \mathtt{b}_0)\end{aligned}$$

such that $\mathtt{a}_\mathtt{e} + \mathtt{b}_\mathtt{e} = 2\mathtt{e}_\mathtt{o} + \mathtt{e}_\mathtt{e}$. Then half of the bits of $\mathtt{a} \wedge \mathtt{b}$ are stored in $\mathtt{e}_\mathtt{o}$ and half of the bits of $\mathtt{a} \oplus \mathtt{b}$ are stored in $\mathtt{e}_\mathtt{e}$. We can repeat the above procedure starting from the odd-position bits of $\mathtt{a}$ and $\mathtt{b}$ to get the following

$$\begin{aligned}\mathtt{a}_\mathtt{o} + \mathtt{b}_\mathtt{o} \quad &\leftrightarrow (0, \mathtt{a}_{W-1}, \ldots, 0, \mathtt{a}_1) + (0, \mathtt{b}_{W-1}, \ldots, 0, \mathtt{b}_1) \\ &= (\mathtt{a}_{W-1} \wedge \mathtt{b}_{W-1}, \mathtt{a}_{W-1} \oplus \mathtt{b}_{W-1}, \ldots, \mathtt{a}_1 \wedge \mathtt{b}_1, \mathtt{a}_1 \oplus \mathtt{b}_1) \quad \leftrightarrow 2\mathtt{o}_\mathtt{o} + \mathtt{o}_\mathtt{e}\end{aligned}$$

where $\mathtt{o}_\mathtt{o}$ stores half of the bits of $\mathtt{a} \wedge \mathtt{b}$ and $\mathtt{o}_\mathtt{e}$ stores and half of the bits of $\mathtt{a} \oplus \mathtt{b}$. Putting everything together, given the decompositions $\mathtt{a}_\mathtt{o}, \mathtt{a}_\mathtt{e}, \mathtt{b}_\mathtt{o}, \mathtt{b}_\mathtt{e}, \mathtt{o}_\mathtt{o}, \mathtt{o}_\mathtt{e}, \mathtt{e}_\mathtt{o}, \mathtt{e}_\mathtt{e} \in \mathsf{EvenBits}$ such that the following hold

$$\mathtt{a} = 2\mathtt{a}_\mathtt{o} + \mathtt{a}_\mathtt{e} \qquad \mathtt{b} = 2\mathtt{b}_\mathtt{o} + \mathtt{b}_\mathtt{e} \qquad \mathtt{a}_\mathtt{o} + \mathtt{b}_\mathtt{o} = 2\mathtt{o}_\mathtt{o} + \mathtt{o}_\mathtt{e} \qquad \mathtt{a}_\mathtt{e} + \mathtt{b}_\mathtt{e} = 2\mathtt{e}_\mathtt{o} + \mathtt{e}_\mathtt{e}$$

then the bit-wise AND and XOR of $\mathtt{a}$ and $\mathtt{b}$ are given by the followings

$$\mathtt{a} \wedge \mathtt{b} = 2\mathtt{o}_\mathtt{o} + \mathtt{e}_\mathtt{o} \qquad \mathtt{a} \oplus \mathtt{b} = 2\mathtt{o}_\mathtt{e} + \mathtt{e}_\mathtt{e}$$

To check that $\mathtt{a} \wedge \mathtt{b} = \mathtt{c}$ is then sufficient to commit to the above decompositions and check their consistency with $\mathtt{a}$ and $\mathtt{b}$ as well as check that $\mathtt{c} = 2\mathtt{o}_\mathtt{o} + \mathtt{e}_\mathtt{o}$. This can

be summarised in the following relation $\mathcal{R}_{\mathsf{AND}}$

$$\mathcal{R}_{\mathsf{AND}} = \left\{ \begin{array}{c} (pp, u, w) := ((W, K, \mathbb{F}, *), \bot, (\mathtt{a}, \mathtt{b}, \mathtt{c}, (\mathtt{a_o}, \mathtt{a_e}, \mathtt{b_o}, \mathtt{b_e}, \mathtt{o_o}, \mathtt{o_e}, \mathtt{e_o}, \mathtt{e_e}), \mathsf{EvenBits})) : \\ (pp, \bot, (\mathtt{a}, (\mathtt{a_o}, \mathtt{a_e}), \mathsf{EvenBits})) \in \mathcal{R}_{\text{range}} \;\; (pp, \bot, (\mathtt{a_o} + \mathtt{b_o}, (\mathtt{o_o}, \mathtt{o_e}), \mathsf{EvenBits})) \in \mathcal{R}_{\text{range}} \\ (pp, \bot, (\mathtt{b}, (\mathtt{b_o}, \mathtt{b_e}), \mathsf{EvenBits})) \in \mathcal{R}_{\text{range}} \;\; (pp, \bot, (\mathtt{a_e} + \mathtt{b_e}, (\mathtt{e_o}, \mathtt{e_e}), \mathsf{EvenBits})) \in \mathcal{R}_{\text{range}} \\ \mathtt{c} = 2\mathtt{o_o} + \mathtt{e_o} \end{array} \right\}$$

A similar relation can be given for the bit-wise XOR and the other logical operations.

5.3 Decomposition of TinyRAM Relation

Here we gradually decompose the relation $\mathcal{R}^{\mathsf{field}}_{\text{TinyRAM}}$ into the building-block relations we introduced earlier, as well as a set of linear and quadratic constraints the entries in w have to satisfy. Figure 5.2 illustrates the entire decomposition of $\mathcal{R}^{\mathsf{field}}_{\text{TinyRAM}}$ into progressively simpler relations.

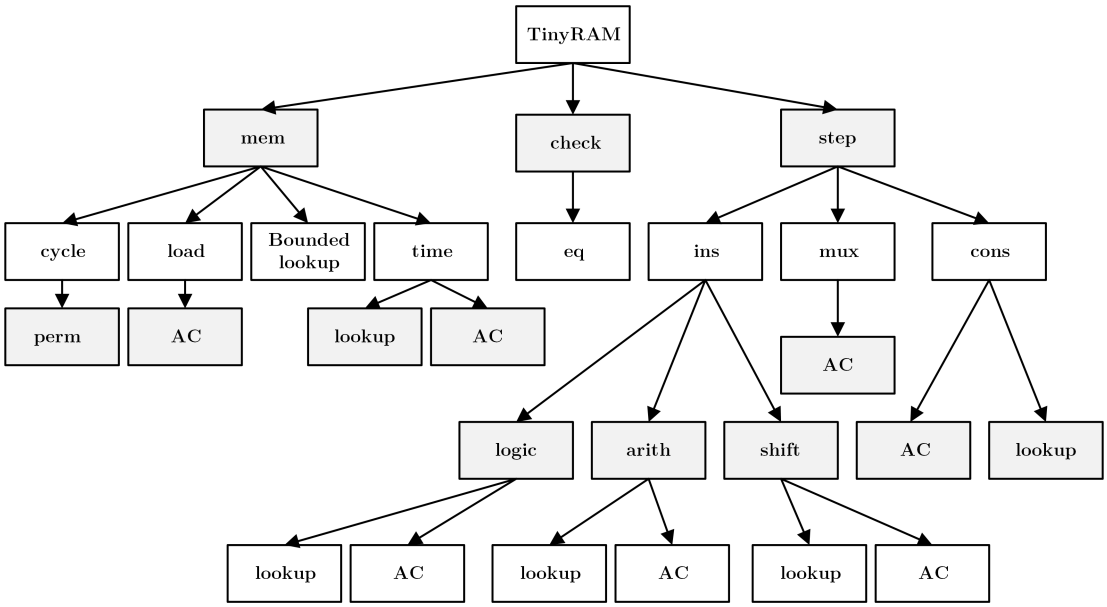

FIGURE 5.2: Diagram of the decompositon of TinyRAM into equality, lookup, permutation, range relations and arithmetic constraints.

5.3.1 Checking the Correctness of Values

The role of $\mathcal{R}_{\text{check}}$ is to check that w consists of the correct number of field elements that can be partitioned into the appropriate tables and also to check that specific entries in these tables are correct. In more details, the relation $\mathcal{R}_{\text{check}}$ is specified by the following conditions

- The first row Exe_1 of the execution table Exe contains the following values: time is set equal to 1, the program counter pc_1 is equal to 0, the instruction $\mathsf{inst}_{\mathsf{pc}_1}$ is equal to the first instruction of the program, the immediate value A_0 is the first immediate value of the program, and the contents of the registers $\mathsf{r}_{i,1}$, the memory address addr_1 and its content value $\mathsf{v}_{\mathsf{addr}_1}$ are all set equal to 0.

- The last row Exe_T contains the following values: the time is set equal to T, the instruction $\mathsf{inst}_{\mathsf{pc}_T}$ is **answer**, and the immediate value is 0. Without loss of generality we assume that the instruction **answer** 0 is positioned in the last line $(L-1)$ of the Prog table.

- The auxiliary lookup table EvenBits contains the embeddings of all W-bit words with 0 in all odd positions, i.e.

$$\mathsf{EvenBits} = \left(0, 1, 4, 5 \ldots, \sum_{i=0}^{\frac{W}{2}-1} 2^{2i}\right)$$

- The auxiliary lookup table Pow contains the embeddings of all integers $i \in [0, W]$ and their powers $2^i \mod 2^W$.

- The program table Prog contains the correct field element embedding of the program P as well as the correct auxiliary entries. In the relation $\mathcal{R}_{\text{eq}}$ checking table Prog we omitted the auxiliary entries which we have not yet specified. It will later become clear that these entries can be efficiently computed given the program P and checked within the above relation.

- The memory table Mem contains the correct embedding of the input words listed in v and of the auxiliary entry usd. This only refers to the first $|v|$ entries of the memory table, as the rest is initialised with the private inputs w.

The above conditions are included in the relation $\mathcal{R}_{\text{check}}$.

$$\mathcal{R}_{\text{check}} = \left\{ \begin{array}{l} (pp, u, w) := ((W, K, \mathbb{F}, *), (P, v, T, M), \mathsf{w}) : \\ \mathsf{w} := (\mathsf{Exe}, \mathsf{Prog}, \mathsf{Mem}, \mathsf{EvenBits}, *), \\ \mathsf{Exe} := \{\mathsf{Exe}_t\}_{t=1}^{T}, \quad \mathsf{Prog} := \{\mathsf{Prog}_i\}_{i=0}^{L-1} \\ \mathsf{Prog}_0 := (0, \mathsf{inst}_0, \mathsf{A}_0, \ldots) \\ (pp, (1, 0, \mathsf{inst}_0, \mathsf{A}_0, 0, \ldots, 0, \ldots), \mathsf{Exe}_1) \in \mathcal{R}_{\text{eq}} \\ (pp, (T, L-1, \mathbf{answer}, 0, \ldots), \mathsf{Exe}_T) \in \mathcal{R}_{\text{eq}} \\ \left(pp, \left(0, 1, 4, 5, \ldots, \sum_{i=0}^{\frac{W}{2}-1} 2^{2i}\right), \mathsf{EvenBits}\right) \in \mathcal{R}_{\text{eq}} \\ \left(pp, \left\{(i, 2^i \bmod 2^W)\right\}_{i=0}^{W}, \mathsf{Pow}\right) \in \mathcal{R}_{\text{eq}} \\ (pp, P, \mathsf{Prog}) \in \mathcal{R}_{\text{eq}} \quad (pp, v, \mathsf{Mem}) \in \mathcal{R}_{\text{eq}} \end{array} \right\}$$

5.3.2 Checking Memory Consistency

The relation $\mathcal{R}_{\text{mem}}$ checks that the memory is used consistently across different steps in the execution. For instance, if at step t a value is loaded from memory, then it should be equal to the last value stored at the same address. If it is the first time a memory address is accessed, we need to ensure consistency with its initial value. If two consecutive memory accesses to the same address are placed into two adjacent rows of Exe it is easy to check their consistency. However, this is generally not the case since the Exe table is sorted by execution time rather than memory access. To avoid committing to the memory-sorted execution trace we devise a way to check the consistency of memory accesses located in any position of Exe. Overall the memory consistency relation $\mathcal{R}_{\text{mem}}$ decomposes as follows

$$\mathcal{R}_{\text{mem}} = \left\{ \begin{array}{ll} (pp, u, w) := ((W, K, \mathbb{F}, *), (T, M), \mathsf{w}) : & \\ \mathsf{w} := (\mathsf{Exe}, \mathsf{Prog}, \mathsf{Mem}, \mathsf{EvenBits}, \pi, *), & \\ \mathsf{Exe} := \{\mathsf{Exe}_t\}_{t=1}^{T} & \mathsf{Mem} := \{\mathsf{Mem}_j\}_{j=0}^{2M-2} \\ (pp, T, (\mathsf{Exe}, \pi)) \in \mathcal{R}_{\text{cycle}}, & (pp, T, (\mathsf{Exe}, \mathsf{EvenBits})) \in \mathcal{R}_{\text{time}} \\ (pp, (T, M), (\mathsf{Exe}, \mathsf{Mem})) \in \mathcal{R}_{\text{blookup}}, & (pp, T, \mathsf{Exe}) \in \mathcal{R}_{\text{load}} \end{array} \right\}$$

To help with checking the memory consistency, we include in each row of the execution table the following auxiliary entries

$$\boldsymbol{aux}_{\mathsf{Exe}} = \begin{array}{|c|c|c|c|c|c|c|} \hline \tau_{\text{link}} & \mathsf{v}_{\text{link}} & \mathsf{v}_{\text{init}} & \mathsf{usd} & \mathsf{S} & \mathsf{L} & \cdots \\ \hline \end{array}$$

where τ_{link} contains the previous time-step at which the current address was accessed. In case this row stores the first time a location is accessed, we use τ_{link} to store the last time-step the location is accessed. This will be convenient to show that sequential accesses to the same memory location can be arranged in cycles. Similarly, v_{link} stores the value contained in the address after time τ_{link}, unless this is the first time that location is accessed, in which case it stores the last value stored in that location. The value v_{init} is a copy of the initial value assigned to that memory location, which is also stored in the memory table Mem. The value usd is a flag which is set equal to 0 if this is the first time we access the current memory address, and 1 otherwise. The values S, L are flags set equal to 1 in case the current instruction is a **store** or **load** operation, respectively, and 0 otherwise. The values S, L are also stored in the auxiliary entries of the program table and thus their consistency is checked by $\mathcal{R}_{\text{check}}$ together with the program.

$$\boldsymbol{aux}_{\mathsf{Prog}} = \boxed{\mathsf{S}}\boxed{\mathsf{L}}\boxed{\cdots}$$

Memory Accesses form Cycles. We check memory consistency by specifying cycles of memory accesses, so that consecutive terms in a cycle correspond to the time of two consecutive accesses to the same memory location. We let the memory access pattern in the rows of Exe being in correspondence with a permutation $\pi \in \Sigma_{[T]}$ defined by such cycles. By using the above auxiliary entries, the relation $\mathcal{R}_{\text{cycle}}$ checks that all memory accesses (i.e. for which $\mathsf{S} + \mathsf{L} = 1$) relative to the same address addr are connected into cycles, and that rows not involving memory operations ($\mathsf{S} + \mathsf{L} = 0$) are not included in these cycles. Moreover, if two accesses are connected by a cycle the relation checks that τ_{link} stores the time of the previous access and v_{link} the content of the memory at that time.

$$\mathcal{R}_{\text{cycle}} = \left\{ \begin{array}{c} (pp, u, w) := ((W, K, \mathbb{F}, *), T, (\mathsf{Exe}, \pi)) : \\ \mathsf{Exe}_t := (t, \ldots, \mathsf{addr}_t, \mathsf{v}_{\text{link}t}, \tau_{\text{link}t}, \ldots, \mathsf{S}_t, \mathsf{L}_t, \ldots) \text{ for } t \in [T] \\ \boldsymbol{a}_t := (\tau_{\text{link}t}, \mathsf{addr}_t, \mathsf{v}_{\text{link}t}, \mathsf{S}_t + \mathsf{L}_t) \text{ for } t \in [T] \\ \boldsymbol{b}_t := (t, \mathsf{addr}_t, \mathsf{v}_{\mathsf{addr}_t}, \mathsf{S}_t + \mathsf{L}_t) \text{ for } t \in [T] \\ ((W, K, \mathbb{F}, *), T, (\{\boldsymbol{a}_i, \boldsymbol{b}_i\}_{i=1}^{T}, \pi)) \in \mathcal{R}_{\text{perm}} \end{array} \right\}$$

We recall that the unknown permutation relation over tuples $\boldsymbol{a}_i, \boldsymbol{b}_j$ checks that exists a permutation $\pi \in \Sigma_{[T]}$ such that $\boldsymbol{b}_i = \boldsymbol{a}_{\pi(i)}$ for $i \in [T]$, i.e.

$$t = \tau_{\mathrm{link}\pi(t)} \qquad \mathsf{addr}_t = \mathsf{addr}_{\pi(t)} \qquad \mathsf{v}_{\mathsf{addr}_t} = \mathsf{v}_{\mathrm{link}\pi(t)} \qquad \mathsf{S}_t + \mathsf{L}_t = \mathsf{S}_{\pi(t)} + \mathsf{L}_{\pi(t)}$$

To construct zero-knowledge proofs for the correct program execution of TinyRAM programs, it is important not to leak the memory access pattern, as this may depend on the secret inputs w of the program. Therefore, the permutation cannot be publicly known.

Memory Accesses are in the Correct Order. Consecutive terms in a cycle should correspond to the consecutive time-steps in which the memory is accessed. To check that the memory cycles are time-ordered we can simply verify that[1] $t > \tau_{\mathrm{link}t}$ for any given time-step $t \in [T]$. Since memory accesses are connected into cycles, the first time we access a new memory location the τ_{link} entry stores the last point in time that location is accessed by the program. In this case we verify that $t \leq \tau_{\mathrm{link}t}$. To perform these checks we use the auxiliary entry usd, which we set $\mathsf{usd} = 0$ for the first time a memory location is accessed, and $\mathsf{usd} = 1$ otherwise.

The above inequalities can be checked by showing that either $t - \tau_{\mathrm{link}t} - 1 \geq 0$ or $\tau_{\mathrm{link}t} - t \geq 0$, respectively. Moreover, by using the auxiliary value usd, both checks can be combined

$$(1 - \mathsf{usd})(\tau_{\mathrm{link}t} - t) + \mathsf{usd}(t - \tau_{\mathrm{link}t} - 1) \geq 0$$

Since we have a range relation in place we can then add a pair of field elements $\mathsf{t}_\mathsf{o}, \mathsf{t}_\mathsf{e} \in \mathsf{EvenBits}$ in the auxiliary entries of the execution table

$$\boldsymbol{aux}_{\mathsf{Exe}} = \begin{array}{|c|c|c|c|} \hline \cdots & \mathsf{t}_\mathsf{o} & \mathsf{t}_\mathsf{e} & \cdots \\ \hline \end{array}$$

[1] For this to be sufficient we also need the time-steps stored in the execution table to be correct. This is ensured by the $\mathcal{R}_{\mathrm{check}}$ and $\mathcal{R}_{\mathrm{cons}}$ (which appears later) relations.

and check that $(1-\mathsf{usd})(\tau_{\mathrm{link}t}-t)+\mathsf{usd}(t-\tau_{\mathrm{link}t}-1)\in[0,2^W]$. The relation $\mathcal{R}_{\mathrm{time}}$ incorporates the above conditions

$$\mathcal{R}_{\mathrm{time}}=\left\{\begin{array}{c}(pp,u,w):=((W,K,\mathbb{F},*),T,(\mathsf{Exe},\mathsf{EvenBits}))\,:\\ \mathsf{Exe}_t:=(t,\ldots,\tau_{\mathrm{link}t},\ldots,\mathsf{usd}_t,\ldots,\mathsf{t}_\circ,\mathsf{t}_\mathsf{e},\ldots)\text{ for }t\in[T]\\ \forall\, t\in[T]:\mathsf{t}:=(1-\mathsf{usd})(\tau_{\mathrm{link}t}-t)+\mathsf{usd}(t-\tau_{\mathrm{link}t}-1)\wedge\\ (pp,\bot,(\mathsf{t},(\mathsf{t}_\circ,\mathsf{t}_\mathsf{e}),\mathsf{EvenBits}))\in\mathcal{R}_{\mathrm{range}}\end{array}\right\}$$

Memory Locations are in no more than one Cycle. To ensure that the cycles correspond to sequences of memory addresses we also need that all the rows touching the same memory address are included in the *same* cycle. Since the cycles are time-ordered, they need to include one time-step for which $\mathsf{usd}=0$ to close a cycle. Thus, we can ensure each memory location to be part of at most on cycle by letting usd to be set equal to 0 at most once for each memory address. We introduce a *bounded* lookup relation $\mathcal{R}_{\mathrm{blookup}}$ to address this requirement. The relation checks that for any row in Exe, the tuple $(\mathsf{addr}_t,\mathsf{v}_{\mathrm{init}t},\mathsf{usd})$ is contained in one row of the table Mem and that each row $(j,\mathsf{v}_j,0)$ of Mem is accessed at most once by the program. At the same time this relation checks that the values $\mathsf{v}_{\mathrm{init}}$ stored in the execution table are consistent with the initialisation of the memory.

$$\mathcal{R}_{\mathrm{blookup}}=\left\{\begin{array}{c}(pp,u,w):=((W,K,\mathbb{F},*),(T,M),(\mathsf{Exe},\mathsf{Mem}))\,:\\ \mathsf{Exe}_t:=(t,\ldots,\mathsf{addr}_t,\ldots,\mathsf{v}_{\mathrm{init}t},\mathsf{usd},\ldots)\text{ for }t\in[T]\\ \forall\, t\in[T]:\ (pp,\bot,((\mathsf{addr}_t,\mathsf{v}_{\mathrm{init}t},\mathsf{usd}),\mathsf{Mem}))\in\mathcal{R}_{\mathrm{lookup}}\wedge\\ \forall\ (j,\mathsf{v}_j,0)\in\mathsf{Mem}:(\ldots,j,\ldots,\mathsf{v}_j,0,\ldots)\text{ occurs at most once in }\mathsf{Exe}\end{array}\right\}$$

Load Instructions are Consistent. Finally, we are only left to check that if the program executes a **load** instruction the value $\mathsf{v}_{\mathsf{addr}_t}$ loaded from memory is consistent with the value stored at the same address at the previous access. Similarly, if **load** is executed on a new memory location, then the value loaded should match with the initial value $\mathsf{v}_{\mathrm{init}t}$. No additional checks are required for **store** instructions. These checks

are incorporated in the relation $\mathcal{R}_{\text{load}}$.

$$\mathcal{R}_{\text{load}} := \left\{ \begin{array}{c} (pp, u, w) = ((W, K, \mathbb{F}, *), T, \mathsf{Exe}) : \\ \mathsf{Exe}_t := (t, \ldots, \mathsf{addr}_t, \mathsf{v}_{\mathsf{addr}_t}, \tau_{\text{link}t}, \mathsf{v}_{\text{link}t}, \mathsf{v}_{\text{init}t}, \mathsf{usd}_t, \ldots) \text{ for } t \in [T] \\ \forall\, t \in [T] : \mathsf{L}_t(\mathsf{v}_{\mathsf{addr}_t} - \mathsf{v}_{\text{init}t} + \mathsf{usd}_t(\mathsf{v}_{\text{init}t} - \mathsf{v}_{\text{link}t})) = 0 \end{array} \right\}$$

Later, when we will give proofs for all the arithmetic constraints included in the decomposition of $\mathcal{R}^{\mathsf{field}}_{\text{TinyRAM}}$, it will be convenient to express them as either quadratic or linear constraints. The one included in the relation $\mathcal{R}_{\text{time}}$ can be expressed as a pair of quadratic constraints by introducing another auxiliary entry LU_t in the execution table and check that both the followings hold

$$\mathsf{LU}_t = \mathsf{usd}_t(\mathsf{v}_{\text{init}t} - \mathsf{v}_{\text{link}t}) \qquad \mathsf{L}_t(\mathsf{v}_{\mathsf{addr}_t} - \mathsf{v}_{\text{init}t} + \mathsf{LU}_t) = 0$$

5.3.3 Checking Correct Execution of Instructions

The relation $\mathcal{R}_{\text{step}}$ guarantees that each step of the execution has been performed correctly. This involves checking for each row Exe_t of the execution table that the stored words are in the range $[0, 2^W - 1]$, the flag_t is a bit, the program counter pc_t matches the instruction as well as the immediate value A_t in the program, and that inst_t is correctly executed. An instruction takes some inputs, e.g., values indicated by the operands reg_j, A (or the flag) and as a result may change the program counter, a register value, a value stored at a memory address, and the flag. Since we have already checked memory correctness, if the operation is a load or store we may assume the memory value $\mathsf{v}_{\mathsf{addr}_t}$ is correct. Overall the relation $\mathcal{R}_{\text{step}}$ decomposes into three relations $\mathcal{R}_{\text{mux}}, \mathcal{R}_{\text{cons}}$ and $\mathcal{R}_{\text{inst}}$ as follows.

$$\mathcal{R}_{\text{step}} = \left\{ \begin{array}{l} (pp, u, w) := ((W, K, \mathbb{F}, *), \bot, \mathsf{w}) : \\ \mathsf{w} := (\mathsf{Exe}, \mathsf{Prog}, \mathsf{Mem}, \mathsf{EvenBits}, *) \ \wedge \ \mathsf{Exe} := \{\mathsf{Exe}_t\}_{t=1}^{T} \\ \forall t \in [1, \ldots, T-1] : \\ \qquad (pp, \bot, (\mathsf{Exe}_t, \mathsf{Exe}_{t+1})) \in \mathcal{R}_{\text{mux}} \\ \qquad (pp, \bot, (\mathsf{Exe}_i, \mathsf{Exe}_{i+1}, \mathsf{Prog})) \in \mathcal{R}_{\text{cons}} \\ \qquad (pp, \bot, (\mathsf{Exe}_i, \mathsf{Exe}_{i+1}, \mathsf{EvenBits})) \in \mathcal{R}_{\text{inst}} \end{array} \right\}$$

To help checking the correctness of the instructions, the rows of the execution and program tables include the following auxiliary entries

$$\boldsymbol{aux}_{\mathsf{Exe}} = \begin{array}{|c|c|c|c|c|c|c|c|c|c|c|c|} \hline \ldots & \mathtt{a} & \mathtt{b} & \mathtt{c} & \mathtt{d} & \boldsymbol{out} & \boldsymbol{s}_{\mathtt{a}} & \boldsymbol{s}_{\mathtt{b}} & \boldsymbol{s}_{\mathtt{c}} & \boldsymbol{s}_{\mathtt{d}} & \boldsymbol{s}_{\mathsf{out}} & \boldsymbol{s}_{\mathsf{ch}} & \ldots \\ \hline \end{array}$$

$$\boldsymbol{aux}_{\mathsf{Prog}} = \begin{array}{|c|c|c|c|c|c|c|} \hline \ldots & \boldsymbol{s}_{\mathtt{a}} & \boldsymbol{s}_{\mathtt{b}} & \boldsymbol{s}_{\mathtt{c}} & \boldsymbol{s}_{\mathtt{d}} & \boldsymbol{s}_{\mathsf{out}} & \boldsymbol{s}_{\mathsf{ch}} \\ \hline \end{array}$$

These consist of some *temporary variables* $\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}$, an output vector $\boldsymbol{out}$, and some *selection vectors* $\boldsymbol{s}_{\mathtt{a}}, \ldots, \boldsymbol{s}_{\mathsf{ch}}$ which are also listed in the program table. The temporary variables are used to store a copy of the inputs and outputs of an instruction. For example, if we have to check an addition operation $\mathbf{add}\ \mathsf{reg}_i\ \mathsf{reg}_j\ A$, we let $\mathtt{c} = \mathsf{r}_{i,t+1}, \mathtt{a} = \mathsf{r}_{j,t}, \mathtt{b} = \mathsf{A}_t$ and check $\mathtt{c} = \mathtt{a} + \mathtt{b}$. The advantage of using the temporary variables is that for each addition operation we check, we will always have the inputs and output in $\mathtt{a}, \mathtt{b}$ and $\mathtt{c}$, instead of handling multiple registers holding inputs and output in arbitrary order.

The Execution Table and the Program Table are Consistent. The consistency relation $\mathcal{R}_{\text{cons}}$ checks that the time is correctly incremented and that the program counter is in the correct range, i.e. $\mathsf{pc}_{t+1} \in \{0, \ldots, L-1\}$ and is incremented unless a jump-instruction is executed. It also checks that the instruction, the immediate value and the selection vectors stored in the execution table are consistent with the program line indexed by pc. Furthermore, it checks that the content of the registers does not change, unless specified by the instruction, e.g. the register storing the result of the computation. For this we use a selection vector $\boldsymbol{s}_{\mathsf{ch}}$ of length $K+2$. Let $\widetilde{\mathsf{Exe}}_t = (\mathsf{pc}_t, \mathsf{r}_{0,t}, \ldots, \mathsf{r}_{K-1,t}, \mathsf{flag}_t)$ be the restriction of the row Exe_t to the entries concerning the program counter, the register values and the flag. The selection vector $\boldsymbol{s}_{\mathsf{ch}}$ has entries equal to 0 in correspondence of entries of $\widetilde{\mathsf{Exe}}_t$ changing during the execution, and equal to 1 for entries that do not change in the execution. The consistency relation

$\mathcal{R}_{\text{cons}}$ is defined as follows

$$\mathcal{R}_{\text{cons}} = \left\{ \begin{array}{c} (pp, u, w) := ((W, K, \mathbb{F}, *), \bot, (\mathsf{Exe}_t, \mathsf{Exe}_{t+1}, \mathsf{Prog})) : \\ \mathsf{Exe}_t := (t, \mathsf{pc}_t, \mathsf{inst}_t, \mathsf{A}_t, \ldots, \mathsf{r}_{0,t}, \ldots, \mathsf{r}_{K-1,t}, \ldots, \mathsf{S}_t, \mathsf{L}_t, \ldots, \boldsymbol{s}_{\mathtt{a}}, \boldsymbol{s}_{\mathtt{b}}, \boldsymbol{s}_{\mathtt{c}}, \boldsymbol{s}_{\mathtt{d}}, \boldsymbol{s}_{\text{out}}, \boldsymbol{s}_{\text{ch}}) \wedge \\ \mathsf{Exe}_{t+1} := (t', \mathsf{pc}_{t+1}, \ldots, \mathsf{r}_{0,t+1}, \ldots, \mathsf{r}_{K-1,t+1}, \ldots) \\ t' = t + 1 \ \wedge \ \mathsf{pc}_{t+1} \in \{0, \ldots, L-1\} \wedge \\ \boldsymbol{s}_{\text{ch}} \circ (\widetilde{\mathsf{Exe}}_{t+1} - \widetilde{\mathsf{Exe}}_t - (1, 0, \ldots, 0)) = (0, 0, \ldots, 0) \wedge \\ (pp, \bot, ((\mathsf{pc}_t, \mathsf{inst}_t, \mathsf{A}_t, \mathsf{S}_t, \mathsf{L}_t, \boldsymbol{s}_{\mathtt{a}}, \boldsymbol{s}_{\mathtt{b}}, \boldsymbol{s}_{\mathtt{c}}, \boldsymbol{s}_{\mathtt{d}}, \boldsymbol{s}_{\text{out}}, \boldsymbol{s}_{\text{ch}}), \mathsf{Prog})) \in \mathcal{R}_{\text{lookup}} \end{array} \right\}$$

Notice that the entries of $\boldsymbol{s}_{\text{ch}}$ are determined by the program line and can be easily computed from it. The above relation, together with $\mathcal{R}_{\text{check}}$, guarantees that $\boldsymbol{s}_{\text{ch}}$ is consistent with the program line is related to and that the copy of $\boldsymbol{s}_{\text{ch}}$ in the execution table matches the one stored in the program table. This also applies to the rest of the selection vectors.

Ensuring Temporary Values are Correct. A multiplexing relation $\mathcal{R}_{\text{mux}}$ is used to check that values $\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}$ are consistent with operands contained in inst_t. To check operations on temporary values $\mathtt{a}, \mathtt{b}, \mathtt{c}$ and $\mathtt{d}$ it requires to multiplex the corresponding register, immediate, and memory values in and out of the temporary values. This is done with the aid of selection vectors $\boldsymbol{s}_{\mathtt{a}}, \boldsymbol{s}_{\mathtt{b}}, \boldsymbol{s}_{\mathtt{c}}, \boldsymbol{s}_{\mathtt{d}}$, which are bit-vectors encoding the operands of an instruction. Each row of the execution table includes multiple variables that may be selected as an operand, e.g., $\mathsf{pc}_t, \mathsf{A}_t, \mathsf{r}_{0,t}, \ldots$ and variables in the next row of the execution table $\mathsf{pc}_{t+1}, \mathsf{A}_{t+1}, \mathsf{r}_{0,t+1}, \ldots$ may also be selected. A selection vector will have a bit for each of these variables indicating whether it is picked or not. For instance, if we let $\boldsymbol{s}_{\mathtt{a}} = (0, 0, 1, 0, \ldots, 0)$ this corresponds to pick $\mathtt{a}$ as $\mathsf{r}_{0,t}$.

Let $\overline{\mathsf{Exe}_t} = (\mathsf{pc}_t, \mathsf{A}_t, \mathsf{r}_{0,t}, \ldots, \mathsf{r}_{K-1,t}, \mathsf{flag}_t, \mathsf{addr}_t, \mathsf{v}_{\mathsf{addr}_t})$ be the tuple of selectable entries of row Exe_t and let $\boldsymbol{s}_{\mathtt{a}}, \boldsymbol{s}_{\mathtt{b}}, \boldsymbol{s}_{\mathtt{c}}, \boldsymbol{s}_{\mathtt{d}}$ be binary vectors of length $2|\overline{\mathsf{Exe}_t}|$. We can then express the multiplexing relation $\mathcal{R}_{\text{mux}}$ in terms of inner product relations as follows

$$\mathcal{R}_{\text{mux}} = \left\{ \begin{array}{c} (pp, u, w) := ((W, K, \mathbb{F}, *), \bot, (\mathsf{Exe}_t, \mathsf{Exe}_{t+1})) : \\ \mathsf{Exe}_t := (t, \ldots, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}, \boldsymbol{out}, \boldsymbol{s}_{\mathtt{a}}, \boldsymbol{s}_{\mathtt{b}}, \boldsymbol{s}_{\mathtt{c}}, \boldsymbol{s}_{\mathtt{d}}, \ldots) \\ \mathtt{a} = \boldsymbol{s}_{\mathtt{a}} \cdot (\overline{\mathsf{Exe}_t} || \overline{\mathsf{Exe}}_{t+1}) \quad \mathtt{b} = \boldsymbol{s}_{\mathtt{b}} \cdot (\overline{\mathsf{Exe}_t} || \overline{\mathsf{Exe}}_{t+1}) \\ \mathtt{c} = \boldsymbol{s}_{\mathtt{c}} \cdot (\overline{\mathsf{Exe}_t} || \overline{\mathsf{Exe}}_{t+1}) \quad \mathtt{d} = \boldsymbol{s}_{\mathtt{d}} \cdot (\overline{\mathsf{Exe}_t} || \overline{\mathsf{Exe}}_{t+1}) \end{array} \right\}$$

5.3.4 Instruction Checker Relation

Multiplexing the operands into temporary variables leaves us with the task of checking that a given instruction is correctly executed on $\mathtt{a}, \mathtt{b}, \mathtt{c}$ and $\mathtt{d}$. Since we want our proof system to be zero knowledge, we cannot reveal which operation we execute in a given step. This means, in particular, that our instruction checker relation $\mathcal{R}_{\text{inst}}$ has to include checks for all the 26 TinyRAM instructions. However, we can still obtain significant savings compared to using 26 independent instruction checkers. We make the key observation that many operations are closely related. For instance checking a subtraction operation **sub** reg_i reg_j A corresponds to check $\mathtt{c} = \mathtt{a} + \mathtt{b}$ with $\mathtt{c} = \mathsf{r}_{j,t}, \mathtt{a} = \mathsf{r}_{i,t+1}, \mathtt{b} = \mathsf{A}_t$, which is of the same form as an addition operation. Using clever multiplexing we reduce the checking of the 26 possible instructions to check the correctness of nine easily computable values $\mathsf{AND}, \mathsf{XOR}, \mathsf{OR}, \mathsf{SUM}, \mathsf{SSUM}, \mathsf{PROD}, \mathsf{SPROD}, \mathsf{MOD}$, SHIFT and four additional values $\mathsf{FLAG}_1, \mathsf{FLAG}_2, \mathsf{FLAG}_3, \mathsf{FLAG}_4$ for the consistency of the flag. We include all these values into the vector $\boldsymbol{out}$, i.e.

$$\boldsymbol{out} = (\mathsf{AND}, \mathsf{XOR}, \mathsf{OR}, \mathsf{SUM}, \mathsf{SSUM}, \mathsf{PROD}, \mathsf{SPROD}, \mathsf{MOD}, \mathsf{SHIFT}, \mathsf{FLAG}_1, \mathsf{FLAG}_2, \mathsf{FLAG}_3, \mathsf{FLAG}_4)$$

Each instruction can be verified by checking that an appropriate subset of the values are 0. For instance, we will check[2] that $\mathsf{SUM} = \mathtt{a} + \mathtt{b} - \mathtt{c}$, and if the operation is an addition, we will also check that $\mathsf{SUM} = 0$. Similarly, we will define all the entries in $\boldsymbol{out}$ in terms of $\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}$. For each operation only a subset of the entries in $\boldsymbol{out}$ will be relevant. As for the selection of the operands, we use a binary selection vector $\boldsymbol{s}_{\text{out}}$ to select which entries of $\boldsymbol{out}$ are relevant for each operation and check that $\boldsymbol{s}_{\text{out}} \circ \boldsymbol{out} = \mathbf{0}$, where $\circ$ is the entry-wise product.

The instruction checker relation $\mathcal{R}_{\text{inst}}$ checks that entries $\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}$ are in the range $[0, \ldots, 2^W - 1]$, the vector $\boldsymbol{out}$ is consistent with $\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}$, and that the relevant entries in $\boldsymbol{out}$ are all equal to zero. We divide the entries of $\boldsymbol{out}$ into four groups: logical $(\mathsf{AND}, \mathsf{XOR}, \mathsf{OR})$, arithmetic $(\mathsf{SUM}, \mathsf{PROD}, \mathsf{SSUM}, \mathsf{SPROD}, \mathsf{MOD})$, shift (SHIFT), and flag $(\mathsf{FLAG}_1, \mathsf{FLAG}_2, \mathsf{FLAG}_3, \mathsf{FLAG}_4)$. By specifying constraints to all these entries, we can directly verify all the logical, arithmetic, and shifts operations after which the

[2] The actual definition of SUM will also incorporate the correctness of the flag.

variables are named. The $\mathcal{R}_{\text{inst}}$ can be decomposed as follows.

$$\mathcal{R}_{\text{inst}} = \left\{ \begin{array}{c} (pp, u, w) := ((W, K, \mathbb{F}, *), \bot, (\mathsf{Exe}_t, \mathsf{Exe}_{t+1}, \mathsf{EvenBits})) : \\ \mathsf{Exe}_t := (t, \ldots, \boldsymbol{out}, \ldots, \boldsymbol{s}_{\text{out}}, \ldots) \ \wedge \ \boldsymbol{s}_{\text{out}} \circ \boldsymbol{out} = \boldsymbol{0} \\ (pp, \bot, (\mathsf{Exe}_t, \mathsf{Exe}_{t+1}, \mathsf{EvenBits})) \in \mathcal{R}_{\text{logic}} \\ (pp, \bot, (\mathsf{Exe}_t, \mathsf{Exe}_{t+1}, \mathsf{EvenBits})) \in \mathcal{R}_{\text{arith}} \\ (pp, \bot, (\mathsf{Exe}_t, \mathsf{Exe}_{t+1}, \mathsf{EvenBits})) \in \mathcal{R}_{\text{shift}} \end{array} \right\}$$

In what follows we describe the relations $\mathcal{R}_{\text{logic}}, \mathcal{R}_{\text{arith}}, \mathcal{R}_{\text{shift}}$, specifying what constraints the entries of $\boldsymbol{out}$ need to satisfy and the appropriate choices of selection vectors $\boldsymbol{s}_{\mathtt{a}}, \boldsymbol{s}_{\mathtt{b}}, \boldsymbol{s}_{\mathtt{c}}, \boldsymbol{s}_{\mathtt{d}}, \boldsymbol{s}_{\text{out}}$ for each TinyRAM operation. A summary of these selection vectors for all the operations is given in Table 5.1.

Before moving to describe how to verify each group of operations, we recall that each line in the program consists of a TinyRAM instruction (Table 3.1) and up to three operands, e.g. **add** reg_i reg_j A. The first operand (reg_i) usually points at the register storing the result of the operation (**add**) computed on the words specified by the next two operands (reg_j, A). The last operand A indicates an immediate value that could be either used directly in the operation or to point to the content of another register. We refer to the value to be used in the operation generically as A, stressing that the selection between either the immediate value or a register value can be handled by using the appropriate selection vector.

Logical Operations. Logical operations can be verified using the odd/even-bits decomposition introduced in Section 5.2.4. We recall that for bit-wise AND and XOR if we let $\mathtt{a}, \mathtt{b}$ store the inputs and consider their decomposition into the following

$$\mathtt{a} = 2\mathtt{a}_{\mathtt{o}} + \mathtt{a}_{\mathtt{e}} \qquad \mathtt{b} = 2\mathtt{b}_{\mathtt{o}} + \mathtt{b}_{\mathtt{e}} \qquad \mathtt{a}_{\mathtt{o}} + \mathtt{b}_{\mathtt{o}} = 2\mathtt{o}_{\mathtt{o}} + \mathtt{o}_{\mathtt{e}} \qquad \mathtt{a}_{\mathtt{e}} + \mathtt{b}_{\mathtt{e}} = 2\mathtt{e}_{\mathtt{o}} + \mathtt{e}_{\mathtt{e}}$$

then the result of the operations can be computed as follows

$$\mathtt{a} \wedge \mathtt{b} = 2\mathtt{o}_{\mathtt{o}} + \mathtt{e}_{\mathtt{o}} \qquad \mathtt{a} \oplus \mathtt{b} = 2\mathtt{o}_{\mathtt{e}} + \mathtt{e}_{\mathtt{e}}$$

Let $\mathsf{AND} = 2\mathtt{o}_{\mathtt{o}} + \mathtt{e}_{\mathtt{o}} - \mathtt{c}$ and $\mathsf{XOR} = 2\mathtt{o}_{\mathtt{e}} + \mathtt{e}_{\mathtt{e}} - \mathtt{c}$. We can verify the execution of an instruction **and** reg_i reg_j A by setting the selection vectors so that $\mathtt{a} = \mathsf{A}_t, \mathtt{b} = \mathsf{r}_{j,t}, \mathtt{c} =$

Operation	$\boldsymbol{s}_{\mathtt{a}}$	$\boldsymbol{s}_{\mathtt{b}}$	$\boldsymbol{s}_{\mathtt{c}}$	$\boldsymbol{s}_{\mathtt{d}}$	$\boldsymbol{s}_{\text{out}}$	$\boldsymbol{s}_{\text{ch}}$
and	A_t	$\mathsf{r}_{j,t}$	$\mathsf{r}_{i,t+1}$	/	$\mathsf{AND}, \mathsf{FLAG}_1, \mathsf{FLAG}_2$	$\mathsf{r}_i, \mathsf{flag}$
or	A_t	$\mathsf{r}_{j,t}$	$\mathsf{r}_{i,t+1}$	/	$\mathsf{OR}, \mathsf{FLAG}_1, \mathsf{FLAG}_2$	$\mathsf{r}_i, \mathsf{flag}$
xor	A_t	$\mathsf{r}_{j,t}$	$\mathsf{r}_{i,t+1}$	/	$\mathsf{XOR}, \mathsf{FLAG}_1, \mathsf{FLAG}_2$	$\mathsf{r}_i, \mathsf{flag}$
not	A_t	$2^W - 1$	$\mathsf{r}_{i,t+1}$	/	$\mathsf{XOR}, \mathsf{FLAG}_1, \mathsf{FLAG}_2$	$\mathsf{r}_i, \mathsf{flag}$
add	A_t	$\mathsf{r}_{j,t}$	$\mathsf{r}_{i,t+1}$	0	SUM	$\mathsf{r}_i, \mathsf{flag}$
sub	A_t	$\mathsf{r}_{i,t+1}$	$\mathsf{r}_{j,t}$	0	SUM	$\mathsf{r}_i, \mathsf{flag}$
mull	A_t	$\mathsf{r}_{j,t}$	$\mathtt{c}$	$\mathsf{r}_{i,t+1}$	$\mathsf{PROD}, \mathsf{FLAG}_1, \mathsf{FLAG}_2$	$\mathsf{r}_i, \mathsf{flag}$
umulh	A_t	$\mathsf{r}_{j,t}$	$\mathsf{r}_{i,t+1}$	$\mathtt{d}$	$\mathsf{PROD}, \mathsf{FLAG}_1, \mathsf{FLAG}_2$	$\mathsf{r}_i, \mathsf{flag}$
smulh	A_t	$\mathsf{r}_{j,t}$	$\mathsf{r}_{i,t+1}$	$\mathtt{d}$	$\mathsf{SPROD}, \mathsf{FLAG}_1, \mathsf{FLAG}_2$	$\mathsf{r}_i, \mathsf{flag}$
umod	$\mathsf{r}_{i,t+1}$	$\mathtt{b}$	A_t	$\mathsf{r}_{j,t}$	$\mathsf{MOD}, \mathsf{FLAG}_1, \mathsf{FLAG}_2, \mathsf{FLAG}_3$	$\mathsf{r}_i, \mathsf{flag}$
udiv	$\mathtt{a}$	$\mathsf{r}_{i,t+1}$	A_t	$\mathsf{r}_{j,t}$	$\mathsf{MOD}, \mathsf{FLAG}_1, \mathsf{FLAG}_2, \mathsf{FLAG}_3$	$\mathsf{r}_i, \mathsf{flag}$
shl	A_t	$\mathsf{r}_{j,t}$	$\mathtt{c}$	$\mathsf{r}_{i,t+1}$	$\mathsf{SHIFT}, \mathsf{FLAG}_4$	$\mathsf{r}_i, \mathsf{flag}$
shr	A_t	$\mathsf{r}_{j,t}$	$\mathsf{r}_{i,t+1}$	$\mathtt{d}$	$\mathsf{SHIFT}, \mathsf{FLAG}_4$	$\mathsf{r}_i, \mathsf{flag}$
cmpe	A_t	$\mathsf{r}_{i,t}$	$\mathtt{c}$	/	$\mathsf{XOR}, \mathsf{FLAG}_1, \mathsf{FLAG}_2$	flag
cmpa	$\mathsf{r}_{i,t}$	$\mathtt{b}$	A_t	0	SUM	flag
cmpae	$\mathsf{r}_{i,t}$	$\mathtt{b}$	A_t	1	SUM	flag
cmpg	$\mathsf{r}_{i,t}$	$\mathtt{b}$	A_t	0	SSUM	flag
cmpge	$\mathsf{r}_{i,t}$	$\mathtt{b}$	A_t	1	SSUM	flag
mov	A_t	pc_{t+1}	0	/	XOR	r_i
jmp	A_t	$\mathsf{r}_{i,t+1}$	0	/	XOR	pc
cmov	$\mathsf{r}_{i,t+1}$	A_t	0	$\mathsf{r}_{j,t}$	MOD	r_i
cjmp	pc_{t+1}	A_t	0	$\mathsf{pc}_t + 1$	MOD	pc
cnjmp	pc_{t+1}	$\mathsf{pc}_t + 1$	0	A_t	MOD	pc
store	$\mathsf{v}_{\mathsf{addr}_t}$	$\mathsf{r}_{i,t}$	0	0	XOR	/
load	$\mathsf{v}_{\mathsf{addr}_t}$	$\mathsf{r}_{i,t+1}$	0	0	XOR	r_i

TABLE 5.1: Choices of selection vectors to ensure that $\mathcal{R}_{\text{inst}}$ is satisfied. The entries specified in the table correspond to the entries of $\boldsymbol{s}_{\mathtt{a}}, \boldsymbol{s}_{\mathtt{b}}, \boldsymbol{s}_{\mathtt{c}}, \boldsymbol{s}_{\mathtt{d}}, \boldsymbol{s}_{\text{out}}$ which are set equal to 1, while the rest are set equal to 0. The entries specified in the table correspond to the entries of $\boldsymbol{s}_{\text{ch}}$ which are set equal to 0. Where the selection vector is the zero vector we write /. We assume that constant entries $0, 1, 2^W - 1$ are stored in the execution table and that they can be selected by $\boldsymbol{s}_{\mathtt{a}}, \boldsymbol{s}_{\mathtt{b}}, \boldsymbol{s}_{\mathtt{c}}, \boldsymbol{s}_{\mathtt{d}}$.

$\mathsf{r}_{i,t+1}$ and check that $\mathsf{AND} = 0$. Similarly, a bit-wise XOR operation can be checked by setting the same selection vectors as above and check that $\mathsf{XOR} = 0$.

Bit-wise OR and NOT. Given $\mathtt{a} \wedge \mathtt{b}$ and $\mathtt{a} \oplus \mathtt{b}$ we can compute $\mathtt{a} \vee \mathtt{b}$ in the following way

$$\mathtt{a} \vee \mathtt{b} = (\mathtt{a} \wedge \mathtt{b}) + (\mathtt{a} \oplus \mathtt{b})$$

Let $\mathsf{OR} = \mathsf{XOR} + \mathsf{AND} + \mathtt{c}$. To verify the execution of **or** reg_i reg_j A it is sufficient to set the selection vectors such that $\mathtt{a} = \mathsf{A}_t, \mathtt{b} = \mathsf{r}_{j,t}, \mathtt{c} = \mathsf{r}_{i,t+1}$ and check that $\mathsf{OR} = 0$, which happens if and only if $\mathtt{c} = \mathtt{a} \vee \mathtt{b}$.

The bit-wise NOT can be handled by computing the bit-wise XOR of A with the word $2^W - 1$. We can use an additional auxiliary entry in the execution table storing the word $2^W - 1$ and use the selector vector to route $\mathtt{b}$ to it.

Flag. The execution of the above logical operations can also affect the flag. Specifically, the flag is set equal to 1 exactly when the output is equal to the 0 word. This can be verified by letting

$$\mathsf{FLAG}_1 = \mathsf{flag}_{t+1} \cdot \mathtt{c} \qquad \mathsf{FLAG}_2 = (\mathsf{flag}_{t+1} + \mathtt{c}) \cdot \mathtt{a}_{\mathsf{flag}} - 1$$

and checking both of them to be equal to 0. The first condition guarantees that at least one among $\mathtt{c}$ and flag_{t+1} is zero, while the second guarantees that not both of them are equal to zero. In fact, FLAG_2 can be made equal to 0 by choosing $\mathtt{a}_{\mathsf{flag}}$ as the inverse of $\mathsf{flag}_{t+1} + \mathtt{c}$ unless their sum is 0.

We append the decompositions of $\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}$ as well as $\mathtt{o}_\mathtt{o}, \mathtt{o}_\mathtt{e}, \mathtt{e}_\mathtt{o}, \mathtt{e}_\mathtt{e}$ $\mathtt{a}_{\mathsf{flag}}$ to the auxiliary entries of the execution table, i.e.

$\boldsymbol{aux}_{\mathsf{Exe}} =$

...	$\mathtt{a}_\mathtt{o}$	$\mathtt{a}_\mathtt{e}$	$\mathtt{b}_\mathtt{o}$	$\mathtt{b}_\mathtt{e}$	$\mathtt{c}_\mathtt{o}$	$\mathtt{c}_\mathtt{e}$	$\mathtt{d}_\mathtt{o}$	$\mathtt{d}_\mathtt{e}$	$\mathtt{o}_\mathtt{o}$	$\mathtt{o}_\mathtt{e}$	$\mathtt{e}_\mathtt{o}$	$\mathtt{e}_\mathtt{e}$	$\mathtt{a}_{\mathsf{flag}}$	...

We now give the $\mathcal{R}_{\text{logic}}$ relation which includes all the above checks, as well as the range checks on $\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}$.

$$\mathcal{R}_{\text{logic}} = \left\{ \begin{array}{ll} (pp, u, w) := ((W, K, \mathbb{F}, *), \bot, (\mathsf{Exe}_t, \mathsf{Exe}_{t+1}, \mathsf{EvenBits})) : & \\ \\ \mathsf{Exe}_t := (t, \ldots, \ldots, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}, \boldsymbol{out}, \ldots, \mathtt{a}_\mathtt{o}, \mathtt{a}_\mathtt{e}, \mathtt{b}_\mathtt{o}, \mathtt{b}_\mathtt{e}, \mathtt{c}_\mathtt{o}, \mathtt{c}_\mathtt{e}, \mathtt{d}_\mathtt{o}, \mathtt{d}_\mathtt{e}, \mathtt{e}_\mathtt{o}, \mathtt{e}_\mathtt{e}, \mathtt{o}_\mathtt{o}, \mathtt{o}_\mathtt{e}, \mathtt{a}_{\mathsf{flag}}, \ldots) & \\ \boldsymbol{out} := (\mathsf{AND}, \mathsf{XOR}, \mathsf{OR}, \ldots, \mathsf{FLAG}_1, \mathsf{FLAG}_2, \ldots) & \\ \mathsf{Exe}_{t+1} := (t+1, \ldots, \mathsf{flag}_{t+1}, \ldots) & \\ (pp, \bot, (\mathtt{a}, (\mathtt{a}_\mathtt{o}, \mathtt{a}_\mathtt{e}), \mathsf{EvenBits}) \in \mathcal{R}_{\text{range}} & (pp, \bot, (\mathtt{b}, (\mathtt{b}_\mathtt{o}, \mathtt{b}_\mathtt{e}), \mathsf{EvenBits}) \in \mathcal{R}_{\text{range}} \\ (pp, \bot, (\mathtt{c}, (\mathtt{c}_\mathtt{o}, \mathtt{c}_\mathtt{e}), \mathsf{EvenBits}) \in \mathcal{R}_{\text{range}} & (pp, \bot, (\mathtt{d}, (\mathtt{d}_\mathtt{o}, \mathtt{d}_\mathtt{e}), \mathsf{EvenBits}) \in \mathcal{R}_{\text{range}} \\ (pp, \bot, (\mathtt{a}_\mathtt{o} + \mathtt{b}_\mathtt{o}, (\mathtt{o}_\mathtt{o}, \mathtt{o}_\mathtt{e}), \mathsf{EvenBits}) \in \mathcal{R}_{\text{range}} & (pp, \bot, (\mathtt{a}_\mathtt{e} + \mathtt{b}_\mathtt{e}, (\mathtt{e}_\mathtt{o}, \mathtt{e}_\mathtt{e}), \mathsf{EvenBits}) \in \mathcal{R}_{\text{range}} \\ \mathsf{XOR} = 2\mathtt{o}_\mathtt{e} + \mathtt{e}_\mathtt{e} - \mathtt{c} \quad \mathsf{AND} = 2\mathtt{o}_\mathtt{o} + \mathtt{e}_\mathtt{o} - \mathtt{c} & \mathsf{OR} = \mathsf{XOR} + \mathsf{AND} + \mathtt{c} \\ \mathsf{FLAG}_1 = \mathsf{flag}_{t+1} \cdot \mathtt{c} & \mathsf{FLAG}_2 = (\mathsf{flag}_{t+1} + \mathtt{c}) \cdot \mathtt{a}_{\mathsf{flag}} - 1 \end{array} \right\}$$

The choices for the selection vectors are given in Table 5.1. For example, an AND operation is checked by using selection vectors such that $\mathtt{a} = \mathsf{A}_t, \mathtt{b} = \mathsf{r}_{j,t}, \mathtt{c} = \mathsf{r}_{i,t+1}$. The entries of $\boldsymbol{out}$ that are equal to 1 are $\mathsf{AND}, \mathsf{FLAG}_1, \mathsf{FLAG}_2$. The entries of $\boldsymbol{s}_{\text{ch}}$ that are equal to 0 r_i and flag, which are the entries that can be affected by the execution of a bit-wise AND.

Integer Operations. Embedding W-bit words into elements of a field of size $p > 2^{2W} - 2^{W-1}$ enables efficient verification of arithmetic TinyRAM instructions since the sum and the product of two W-bits word does not cause a modular reduction in the field.

Addition and Subtraction. The execution of an addition operation **add** reg_i reg_j A can be verified by picking selection vectors such that $\mathtt{a} = \mathsf{A}_t, \mathtt{b} = \mathsf{r}_{j,t}, \mathtt{c} = \mathsf{r}_{i,t+1}$ and then checking that the following holds

$$\mathtt{a} + \mathtt{b} - \mathtt{c} - 2^W \mathsf{flag}_{t+1} = 0$$

Note that this is equal to 0 if and only if $\mathtt{c}$ contains the result of $\mathtt{a} + \mathtt{b}$ with the flag flag_{t+1} indicating overflow.

The same check can be used to verify a subtraction operation **sub** reg_i reg_j A by swapping the role of the selection vector $\boldsymbol{s}_\mathtt{b}, \boldsymbol{s}_\mathtt{c}$ and letting $\mathtt{b} = \mathsf{r}_{i,t+1}$ and $\mathtt{c} = \mathsf{r}_{j,t}$. The

above equation is identically 0 if and only if $\mathtt{b}$ is equal to the difference of $\mathtt{c}$ and $\mathtt{a}$ where the flag flag_{t+1} denotes borrow.

Let $\mathsf{SUM} = \mathtt{a}+\mathtt{b}-\mathtt{c}-2^W\mathsf{flag}_{t+1}+\mathtt{d}$. We can check both additions and subtractions by letting $\mathtt{d} = 0$ and checking that $\mathsf{SUM} = 0$. While the temporary variable $\mathtt{d}$ is not required for the verification of the above operations, we will see later that including this variable simplifies the verification of other operations.

Multiplications. TinyRAM instruction set includes three multiplication instructions: **mull**, for computing the lower word of the product of two unsigned integers; **umull**, for computing the upper word of the product of two unsigned integers; **umulh**, for computing the upper word of the product of two signed integers.

Let $\mathsf{PROD} = \mathtt{a}\cdot\mathtt{b}-\mathtt{d}-2^W\mathtt{c}$. We can then verify the correct execution of **mull** reg_i reg_j A by setting the temporary variables such that $\mathtt{a} = \mathsf{A}_t, \mathtt{b} = \mathsf{r}_{j,t}, \mathtt{d} = \mathsf{r}_{i,t+1}$, $\mathtt{c}$ to contain some non-deterministic advice and check that $\mathsf{PROD} = 0$. Note that the latter is equal to 0 if and only if $\mathtt{d}$ stores the lower word of the product $\mathtt{a}\cdot\mathtt{b}$ and the upper word is stored in $\mathtt{c}$.

For the execution of **umulh** reg_i reg_j A we can simply change the role of the selector vectors $\boldsymbol{s}_{\mathtt{c}}, \boldsymbol{s}_{\mathtt{d}}$, letting $\mathtt{c} = \mathsf{r}_{i,t+1}$ and $\mathtt{d}$ be some non-deterministic advice, and checking that $\mathsf{PROD} = 0$.

Signed Integers. Signed W-bit words use the two's complement representation to store an integer. We write $\sigma_{\mathtt{a}}$ for the most significant bit of a word $\mathtt{a}$. We recall that according to the two's complement representation, a negative word $\mathtt{a}$ has $\sigma_{\mathtt{a}} = 1$. We define the corresponding field element $\mathtt{a}_\sigma = -\sigma_{\mathtt{a}}2^W + \mathtt{a} \in \{-2^{W-1}, \ldots, 2^{W-1}-1\}$. Note that in case $\mathtt{a}$ is negative, the most significant bit of $\mathtt{a}$, and thus of $\mathtt{a}_\circ$, is equal to 1. Given the decomposition of $\mathtt{a}$ it is easy to check the sign is correct by testing $\mathtt{a}_\circ+(1-2\sigma_{\mathtt{a}})2^{W-2} \in \mathsf{EvenBits}$, which is correct if and only if the most significant bit of $\mathtt{a}_\circ$ is equal to $\sigma_{\mathtt{a}}$. Thus, in order to ensure that $\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}$ are in the correct range, and the matching signs and signed values are correct we include the following entries in the auxiliary inputs of the execution table

$$\boldsymbol{aux}_{\mathsf{Exe}} = \begin{array}{|c|c|c|c|c|c|c|c|}\hline \ldots & \mathtt{a}_\sigma & \sigma_{\mathtt{a}} & \mathtt{b}_\sigma & \sigma_{\mathtt{b}} & \mathtt{c}_\sigma & \sigma_{\mathtt{c}} & \ldots \\ \hline \end{array}$$

To verify the execution of signed multiplication operations **smulh** reg_i reg_j A we proceed similarly to **umullh**. Let $\mathsf{SPROD} = \mathtt{a}_\sigma \cdot \mathtt{b}_\sigma - \mathtt{d} - 2^W \mathtt{c}_\sigma$. By selecting $\mathtt{a} = \mathsf{A}_t, \mathtt{b} = \mathsf{r}_{j,t}, \mathtt{c} = \mathsf{r}_{i,t+1}$, and letting $\mathtt{d}$ be some non-deterministic advice, we can verify the execution of signed multiplication by checking that $\mathsf{SPROD} = 0$.

The consistency of the flag of the signed and unsigned multiplication operations can be verified by checking $\mathsf{FLAG}_1 = 0, \mathsf{FLAG}_2 = 0$.

We notice that the two's complement representation allows to perform additions and subtractions by reusing the unsigned operation. However, in order to verify comparison operations for signed integers it will be helpful to define a signed counterpart of SUM. Thus, we define

$$\mathsf{SSUM} = \mathtt{a}_\sigma + \mathtt{b} - \mathtt{c}_\sigma - 2^W \mathsf{flag}_{t+1} + \mathtt{d}$$

Modular Reduction. To check the execution of modular reduction **umod** reg_i reg_j A, let $\mathtt{d} = \mathsf{r}_{j,t}$, $\mathtt{c}$ be the modulus A_t, $\mathtt{b}$ be the quotient of the division $\frac{\mathtt{d}}{\mathtt{c}}$, and $\mathtt{a}$ be the remainder $\mathsf{r}_{i,t+1}$. We can then check that $\mathtt{d} - \mathtt{b} \cdot \mathtt{c} - \mathtt{a} = 0$. This check however is not sufficient to guarantee the correctness of the the operation **umod**. For example, in case the modulus is set equal to 0, the operation should return 0 and set the flag equal to 1. We can get around this by first checking that $\mathsf{FLAG}_1 = 0$ and $\mathsf{FLAG}_2 = 0$, which ensures that $\mathsf{flag}_{t+1} = 1$ if and only if $\mathtt{c} = 0$. Then, by checking the following

$$-\mathsf{flag} \cdot \mathtt{d} + \mathtt{d} - \mathtt{b} \cdot \mathtt{c} - \mathtt{a} = 0 \tag{5.1}$$

In case $\mathtt{c} = 0$, we have that $\mathsf{flag}_{t+1} = 1$ and then $\mathtt{a} = 0$. Otherwise if $\mathtt{c} \neq 0$, then $\mathsf{flag}_{t+1} = 0$ and the check corresponds to the previous one.

The last thing that we need to check in order to guarantee the correctness of the computation is that $\mathtt{a} < \mathtt{c}$, in case $\mathtt{c} \neq 0$. We can do this by doing a range check on the value $\mathtt{c} - \mathtt{a} - 1$, which involves computing its odd/even-bits decomposition $\mathsf{r}_\mathsf{o}, \mathsf{r}_\mathsf{e}$ and checking that $\mathtt{c} - \mathtt{a} - 1 = 2\mathsf{r}_\mathsf{o} + \mathsf{r}_\mathsf{e}$. We can include the decomposition $\mathsf{r}_\mathsf{o}, \mathsf{r}_\mathsf{e}$ into the auxiliary entries of the execution table

$$\boldsymbol{aux}_{\mathsf{Exe}} = \begin{array}{|c|c|c|c|} \hline \cdots & \mathsf{r}_\mathsf{o} & \mathsf{r}_\mathsf{e} & \cdots \\ \hline \end{array}$$

and check both that $\mathsf{r}_\mathtt{o}, \mathsf{r}_\mathtt{e} \in \mathsf{EvenBits}$ and that

$$(1 - \mathsf{flag}_{t+1})(\mathtt{c} - \mathtt{a} - 1 - 2\mathsf{r}_\mathtt{o} - \mathsf{r}_\mathtt{e}) = 0 \tag{5.2}$$

Since the above checks also include the verification of the correctness of the quotient, we can reuse them to check the execution of **udiv** operation. It is sufficient to swap the selector vectors $\boldsymbol{s}_\mathtt{a}, \boldsymbol{s}_\mathtt{b}$ so that $\mathsf{r}_{i,t+1} = \mathtt{b}$ and check that Eq. (5.1). Again, this check is not sufficient in the case of division by $\mathtt{c} = 0$. In this case the operation is expected to return 0 and set the flag equal to 1. The above check does not suffice since it is merely checking that the remainder $\mathtt{a}$ is equal to 0, instead of the quotient $\mathtt{b}$. This can be addressed by additionally checking the following

$$\mathtt{b} \cdot \mathsf{flag}_{t+1} = 0 \tag{5.3}$$

In case $\mathtt{c} \neq 0$, we still need to ensure that $\mathtt{a} < \mathtt{c}$ to guarantee the correctness of the result. Note that we can combine Eq. (5.2) and (5.3) into a single equation

$$\mathsf{FLAG}_3 = \mathtt{b} \cdot \mathsf{flag}_{t+1} + (1 - \mathsf{flag}_{t+1})(\mathtt{c} - \mathtt{a} - 1 - 2\mathsf{r}_\mathtt{o} - \mathsf{r}_\mathtt{e})$$

This does not affect the **umod** operation since in case $\mathsf{flag}_{t+1} = 1$, the prover can simply set the non-deterministic advice $\mathtt{b}$ equal to 0.

Looking ahead, Equation 5.1 will be used to check other operations by appropriate choices of the selector vector. With this goal in mind, it will be useful to replace Equation 5.1 with the following

$$\mathsf{MOD} = \mathsf{flag}_{t+1}(\mathtt{b} - \mathtt{d}) + \mathtt{d} - \mathtt{b} \cdot \mathtt{c} - \mathtt{a}$$

Note that replacing Eq. (5.1) with $\mathsf{MOD} = 0$ does not affect the checks done for **umod**, **udiv**. In case $\mathsf{flag}_{t+1} = 0$, the two equations are equivalent. In case $\mathsf{flag}_{t+1} = 1$, we have that $\mathtt{c} = 0$ which means that $\mathtt{b} - \mathtt{a} = 0$. In this case we get that $\mathsf{FLAG}_3 = 0$ implies $\mathtt{b} = 0$, and hence $\mathtt{a} = 0$, as in the case of Eq. (5.1).

The relation $\mathcal{R}_{\mathrm{arith}}$ incorporates all the checks for the above arithmetic operations.

$$\mathcal{R}_{\text{arith}} = \left\{ \begin{array}{ll} (pp, u, w) := ((W, K, \mathbb{F}, *), \perp, (\mathsf{Exe}_t, \mathsf{Exe}_{t+1}, \mathsf{EvenBits})) : & \\ & \\ \mathsf{Exe}_t := (t, \ldots, \mathsf{flag}_t, \ldots, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}, \boldsymbol{out}, \ldots, \mathtt{a}_\mathtt{o}, \mathtt{b}_\mathtt{o}, \mathtt{c}_\mathtt{o}, \ldots \sigma_\mathtt{a}, \sigma_\mathtt{b}, \sigma_\mathtt{c}, \mathsf{r}_\mathtt{o}, \mathsf{r}_\mathtt{e} \ldots,) & \\ \boldsymbol{out} := (\ldots, \mathsf{SUM}, \mathsf{SSUM}, \mathsf{PROD}, \mathsf{SPROD}, \mathsf{MOD}, \ldots, \mathsf{FLAG}_3, \ldots) & \\ \mathsf{Exe}_{t+1} := (t+1, \ldots, \mathsf{flag}_{t+1}, \ldots) & \\ (pp, \perp, (\mathtt{a}_\mathtt{o} + (1 - 2\sigma_\mathtt{a})2^{W-2}, \mathsf{EvenBits}) \in \mathcal{R}_{\text{lookup}} & \\ (pp, \perp, \mathtt{c}_\mathtt{o} + (1 - 2\sigma_\mathtt{c})2^{W-2}, \mathsf{EvenBits}) \in \mathcal{R}_{\text{lookup}} & \\ (pp, \perp, \mathtt{b}_\mathtt{o} + (1 - 2\sigma_\mathtt{b})2^{W-2}, \mathsf{EvenBits}) \in \mathcal{R}_{\text{lookup}} & \\ & \\ \sigma_\mathtt{a}, \sigma_\mathtt{b}, \sigma_\mathtt{c}, \mathsf{flag}_t, \mathsf{flag}_{t+1} \in \{0,1\} & \mathtt{a}_\sigma = -\sigma_\mathtt{a} 2^W + \mathtt{a} \\ \mathtt{b}_\sigma = -\sigma_\mathtt{b} 2^W + \mathtt{b} & \mathtt{c}_\sigma = -\sigma_\mathtt{c} 2^W + \mathtt{c} \\ \mathsf{SUM} = \mathtt{a} + \mathtt{b} - \mathtt{c} - 2^W \mathsf{flag}_{t+1} + \mathtt{d} & \mathsf{PROD} = \mathtt{a} \cdot \mathtt{b} - \mathtt{d} - 2^W \mathtt{c} \\ \mathsf{MOD} = \mathsf{flag}_{t+1}(\mathtt{b} - \mathtt{d}) + \mathtt{d} - \mathtt{b} \cdot \mathtt{c} - \mathtt{a} & \\ \mathsf{SSUM} = \mathtt{a}_\sigma + \mathtt{b} - \mathtt{c}_\sigma - 2^W \mathsf{flag}_{t+1} + \mathtt{d} & \mathsf{SPROD} = \mathtt{a}_\sigma \cdot \mathtt{b}_\sigma - \mathtt{d} - 2^W \mathtt{c}_\sigma \\ \mathsf{FLAG}_3 = \mathtt{b} \cdot \mathsf{flag}_{t+1} + (1 - \mathsf{flag}_{t+1})(\mathtt{c} - \mathtt{a} - 1 - 2\mathsf{r}_\mathtt{o} - \mathsf{r}_\mathtt{e}) & \end{array} \right\}$$

Shift Operations. Operations **shl** reg_i $\mathsf{reg}_j A$ and **shr** reg_i $\mathsf{reg}_j A$ are used to shift the word $\mathsf{r}_{j,t}$ of A positions to the left, respectively to the right, filling the vacant positions with 0. The flag is set to the most significant bit and least significant bit of $\mathsf{r}_{j,t}$, respectively.

Following the observation that to shift a word by A positions is equivalent to multiply or divide $\mathsf{r}_{j,t}$ by 2^A, we can treat shifts similarly to the integers operations shown above. To efficiently check the correctness of shift operations we include into w the table Pow storing the pairs $(a, 2^a \mod 2^W)$ for $a \in [0, W]$

Values	Powers
0	1
1	2
2	4
3	8
⋮	⋮
$W-1$	2^{W-1}
W	0

TABLE 5.2: Table Pow.

In addition to Pow we store two additional values $\mathtt{a}_{\text{shift}}, \mathtt{a}_{\text{power}}$ in the auxiliary

entries of Exe. Say that $\mathtt{a}$ stores the value A_t, the offset of a shift operation. Then the entry $\mathtt{a}_{\text{shift}}$ can be used to check if $\mathsf{A}_t \in [W]$ by checking that

$$\mathtt{a}_{\text{shift}}(\mathtt{a}_{\text{shift}} - 1) = 0 \qquad \wedge \qquad (1 - \mathtt{a}_{\text{shift}})(W - \mathtt{a} - 2\mathsf{r}_{\mathtt{o}} - \mathsf{r}_{\mathtt{e}}) = 0 \tag{5.4}$$

where the first equation checks that $\mathtt{a}_{\text{shift}}$ is a bit and the second checks that if $\mathtt{a} > W$ then $\mathtt{a}_{\text{shift}} = 1$. The entry $\mathtt{a}_{\text{power}}$ can be used to store the element $2^{\mathtt{a}} \mod 2^W$ in case $\mathtt{a}_{\text{shift}} = 0$ and 0 otherwise. We can then check that $\mathtt{a}_{\text{power}}$ is consistent with $\mathtt{a}$ by checking that

$$(pp, (\mathtt{a} + \mathtt{a}_{\text{shift}}(W - \mathtt{a}), \mathtt{a}_{\text{power}}), \mathsf{Pow}) \in \mathcal{R}_{\text{lookup}} \tag{5.5}$$

A left shift operation **shl** can be checked by setting $\mathtt{b} = \mathsf{r}_{j,t}, \mathtt{d} = \mathsf{r}_{i,t+1}$, respectively, and checking that the following is equal to 0

$$\mathsf{SHIFT} = \mathtt{a}_{\text{power}} \cdot \mathtt{b} - \mathtt{d} - 2^W \mathtt{c} \tag{5.6}$$

where $\mathtt{c}$ is some non-deterministic advice. The consistency of the flag with the most significant bit of $\mathsf{r}_{j,t}$ is checked by the following

$$\mathsf{flag}_{t+1} - \sigma_{\mathtt{b}} = 0$$

Observe that if we had registers storing two words of W-bits each, shifting a W-bit word to the right of A positions would correspond to shifting the same word to the left by $W - \mathsf{A}$ positions and taking the resulting upper W bits. Since the size of the field $\mathbb{F}$ is big enough, we can use the above observations to check a right shift as a left shift. This allows us to reuse most of the above checks. It is sufficient to set $\mathtt{a} = W - \mathsf{A}_t, \mathtt{b} = \mathsf{r}_{j,t}, \mathtt{d} = \mathsf{r}_{i,t+1}$, let $\mathtt{c}$ be some non-deterministic advice and check conditions (5.4), (5.5) and (5.6) as well as the following

$$\mathsf{flag}_{t+1} - \rho_{\mathtt{b}} = 0$$

where $\rho_{\mathtt{b}}$ is the least significant bit of $\mathtt{b}$. We can introduce an additional auxiliary value $\mathtt{b}_{\mathsf{flag}}$ in the auxiliary information of the program and duplicated in the execution table which is set equal to 1 for left shifts and equal to 0 otherwise. We can then merge the

above flag checks in the following

$$\mathsf{FLAG}_4 = \mathsf{flag}_{t+1} - \mathtt{b}_{\mathsf{flag}}\sigma_{\mathtt{b}} - (1 - \mathtt{b}_{\mathsf{flag}})\rho_{\mathtt{b}}.$$

We can now give the relation $\mathcal{R}_{\text{shift}}$ for the correct execution of shifts operations.

$$\mathcal{R}_{\text{shift}} = \left\{ \begin{array}{c} (pp, u, w) := ((W, K, \mathbb{F}, *), \bot, (\mathsf{Exe}_t, \mathsf{Exe}_{t+1}, (\mathsf{EvenBits}, \mathsf{Pow}))) : \\ \\ \mathsf{Exe}_t := (t, \ldots, \mathsf{flag}_t, \ldots, \mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}, \boldsymbol{out}, \ldots, \mathtt{b}_{\mathtt{e}}, \ldots, \mathtt{b}_{\mathsf{flag}}, \sigma_{\mathtt{b}}, \ldots, \mathtt{a}_{\text{shift}}, \mathtt{a}_{\text{power}}) \\ \boldsymbol{out} := (\ldots, \mathsf{SHIFT}, \ldots, \mathsf{FLAG}_4) \\ \mathsf{Exe}_{t+1} := (t+1, \ldots, \mathsf{flag}_{t+1}, \ldots) \\ \mathtt{a}_{\text{shift}} \in \{0, 1\} \\ (pp, \bot, \mathtt{b}_{\mathtt{e}} + (1 - 2\rho_{\mathtt{b}}), \mathsf{EvenBits}) \in \mathcal{R}_{\text{lookup}} \\ (pp, (\mathtt{a} + \mathtt{a}_{\text{shift}}(W - \mathtt{a}), \mathtt{a}_{\text{power}}), \mathsf{Pow}) \in \mathcal{R}_{\text{lookup}} \\ \mathsf{SHIFT} = \mathtt{a}_{\text{power}} \cdot \mathtt{b} - \mathtt{d} - 2^W \mathtt{c} \\ \mathsf{FLAG}_4 = \mathsf{flag}_{t+1} - \mathtt{b}_{\mathsf{flag}}\sigma_{\mathtt{b}} - (1 - \mathtt{b}_{\mathsf{flag}})\rho_{\mathtt{b}} \end{array} \right\}$$

Remaining Operations. The instruction relation $\mathcal{R}_{\text{inst}}$ ensures that logical, arithmetic and shift operations are carried out correctly. The remaining TinyRAM operations that need to be checked are move and jump operations, memory operations, and a terminating answer operation. The correct execution of these can be reduced to check the correctness of the previous operations by choosing the selection vectors appropriately.

Comparison Operations. The compare-equal instruction **compe** reg_i A sets the flag equal to 1 if $\mathsf{r}_{i,t} = \mathsf{A}_t$, and 0 otherwise. To check the execution of this operation we can set $\mathtt{a} = \mathsf{A}_t, \mathtt{b} = \mathsf{r}_{i,t}$ and check that either their bit-wise XOR is equal to 0 or that $\mathsf{flag}_{t+1} = 0$. Therefore, we can reuse the checks specified for the bit-wise XOR to check the compare-equal instruction.

The compare-above instruction **compa** reg_i A sets the flag equal to 1 if and only if $\mathsf{r}_{i,t} > \mathsf{A}_t$ and to 0 otherwise. Let $\mathtt{a} = \mathsf{r}_{i,t}$, $\mathtt{c} = \mathsf{A}_t$, $\mathtt{d} = 0$, and $\mathtt{b}$ store some non-deterministic advice. By checking that $\mathsf{SUM} = 0$ we have that $(\mathtt{a} - \mathtt{c}) = (2^W \mathsf{flag}_{t+1} - \mathtt{b})$. Note that if $\mathsf{flag}_{t+1} = 1$, the right-hand side is greater than 0 since $\mathtt{b} \in [0, 2^W - 1]$ has been checked by the relation $\mathcal{R}_{\text{logic}}$, and therefore also the left-hand side is greater than

0. Thus, if the $\mathsf{flag}_{t+1} = 1$ then $\mathtt{a} > \mathtt{c}$. On the other hand, if $\mathsf{flag}_{t+1} = 0$, the right-hand side is less or equal than 0 and thus $\mathtt{a} \leq \mathtt{c}$. The operation **compg** is the equivalent of **compa** for the comparison of signed integers and can be checked in a similar way by checking that $\mathsf{SSUM} = \mathtt{a}_\sigma + \mathtt{b} - \mathtt{c}_\sigma - 2^W \mathsf{flag}_{t+1} + \mathtt{d}$ is equal to 0, where $\mathtt{d}$ is also set equal to 0 and the other temporary variables are set as above.

Similarly, the **compae** reg_i A sets the flag equal to 1 if and only if $\mathsf{r}_{i,t} \geq \mathsf{A}_t$ and to 0 otherwise. We can set the selection vectors as for the previous operation while setting $\mathtt{d} = 1$. Whenever $\mathsf{SUM} = 0$ we will have that $(\mathtt{a} - \mathtt{c}) = (2^W \mathsf{flag}_{t+1} - \mathtt{b} - 1)$. Now, $\mathsf{flag}_{t+1} = 1$ makes the right-hand side greater or equal than 0, and thus $\mathtt{a} \geq \mathtt{c}$. On the other hand, $\mathsf{flag}_{t+1} = 0$ makes the right-hand side strictly less than 0, giving us $\mathtt{a} < \mathtt{c}$. The operation **compge** is the equivalent of **compa** for the comparison of signed integers and can be checked in a similar way by checking that $\mathsf{SSUM} = \mathtt{a}_\sigma + \mathtt{b} - \mathtt{c}_\sigma - 2^W \mathsf{flag}_{t+1} + \mathtt{d}$ is equal to 0, where $\mathtt{d}$ is also set equal to 1 and the other temporary variables are set as above.

Move and Jump Operations. The instruction **mov** reg_i A stores the value A_t into reg_i and the instruction **jmp** A sets the program counter pc_{t+1} equal to A_t. We can check both these operations by storing input and output in $\mathtt{a}$ and $\mathtt{b}$, respectively, and check that their bit-wise XOR is equal to 0. We summarise the selection vectors for these operations in Table 5.1.

The operation **cmov** is the conditional operation executing a **mov** instruction only in the case the flag_t is set equal to 1. It can be verified by setting the selection vectors so that $\mathtt{a} = \mathsf{r}_{i,t+1}, \mathtt{b} = \mathsf{A}_t, \mathtt{c} = 0, \mathtt{d} = \mathsf{r}_{i,t}$ and check that $\mathsf{MOD} = 0$. Note that when $\mathtt{c} = 0$, the latter amounts to check

$$\mathsf{flag}_{t+1}(\mathtt{b} - \mathtt{d}) + \mathtt{d} - \mathtt{a} = 0$$

which checks that $\mathsf{A}_t = \mathsf{r}_{i,t}$ in case $\mathsf{flag}_t = 1$ and that reg_i remains the same in case $\mathsf{flag}_t = 0$, i.e. $\mathsf{r}_{i,t} = \mathsf{r}_{i,t}$. To conclude, by setting the entry of vector $\boldsymbol{s}_{\mathrm{ch}}$ relative to the flag equal to 1, we can check that $\mathsf{flag}_t = \mathsf{flag}_{t+1}$.

The conditional operation **cjmp** executes a jump operation only in case $\mathsf{flag}_t = 1$. This can be checked in the same way as above by setting $\mathtt{a} = \mathsf{pc}_{t+1}, \mathtt{b} = \mathsf{A}_t, \mathtt{c} = 0, \mathtt{d} = \mathsf{pc}_t + 1$ and check that $\mathsf{MOD} = 0$. The conditional operations **cnjmp** only performs

a jump instruction if $\mathsf{flag}_t = 0$. It is sufficient to swap the roles of $\mathtt{b}$ a $\mathtt{d}$ and check $\mathsf{MOD} = 0$.

Memory Operations. The consistency of memory operations across the execution has been checked by the memory check relation $\mathcal{R}_{\mathrm{mem}}$. Among other things, $\mathcal{R}_{\mathrm{mem}}$ verifies that entries addr_t and $\mathsf{v}_{\mathsf{addr}_t}$ are updated consistently. The last thing that remains to check is that when performing load and store operations, the registers are updated consistently to the value stored in $\mathsf{v}_{\mathsf{addr}_t}$. This involves checking equality between $\mathsf{v}_{\mathsf{addr}_t}$ and either the value stored in the input or output register, which can be done by checking that their bit-wise XOR is equal to 0.

Answer. A correct program execution terminates in step T with **answer** 0. With little loss of generality we can assume this is done by jumping to the last line of the program, which has instruction **answer** A specifying immediate value $\mathsf{A} = 0$, which we check in $\mathcal{R}_{\mathrm{check}}$. A correct program execution only executes **answer** once so we also need to ensure the program execution does not encounter an **answer** instruction prematurely. We ensure this by removing all **answer** instructions from the program table Prog such that no execution step checked by $\mathcal{R}_{\mathrm{inst}}$ for $t = 1, \ldots, T-1$ can be an answer instruction.

5.4 ILC proofs for Building Blocks

In the previous sections we described the arithmetization of TinyRAM and its full decomposition into few building-block relations: equality, unknown permutation, lookup, range relations and sets of quadratic constraints. Next, we give an outline of the proof systems over the ILC for these building blocks, referring to [BCG+18] for the complete specifications of the permutation and lookup proofs. The proof systems described in this section refer to matrices and vectors committed using the ILC. In the next section we will see how to parse the witness w into several matrices and how to combine these building blocks to give proofs for the correct execution of TinyRAM programs.

5.4.1 ILC proofs for Equality Relations

The equality relation $\mathcal{R}_{\text{eq}}$ is analogous to the one we introduced in the Section 4.2.1, were we also described a proof system in the ILC model. In Table 5.3 we restate the efficiency of the proof system to check the equality of s public vectors with committed vectors in the ILC.

Prover computation	$T_{\mathcal{P}_{\mathsf{ILC}}}$	$= //$
Verifier computation	$T_{\mathcal{V}_{\mathsf{ILC}}}$	$= \mathcal{O}(sk)\ \mathbb{F}^+$
Prover communication	t	$= //$
Verifier communication	C_{ILC}	$= 0$
Query complexity	qc	$= 1$
Round complexity	μ	$= 0$

TABLE 5.3: Efficiency of the proof of knowledge for $\mathcal{R}_{\text{eq}}$. $\mathbb{F}^+$ stands for the cost of a single field addition.

5.4.2 ILC Proof for Unknown Permutation Relations

We will now give the outline of a proof system over the ILC for the relation $\mathcal{R}_{\text{perm}}$. To express the statement in a format that is more compatible with the ILC channel we write the values to be permuted as entries in matrices. Therefore we reformulate the unknown permutation relation as follows.

$$\mathcal{R}_{\text{perm}} = \left\{ \begin{array}{c} (pp_{\mathsf{ILC}}, u) := ((W, K, \mathbb{F}, k)\ ,\ (\mathsf{mn}), (A, B, \pi)) : \\ A, B \in \mathbb{F}^{\mathsf{mn}\times k}\ \wedge\ \pi \in \Sigma_{[\mathsf{mn}]\times[k]}\ \wedge\ A_{\pi(i,j)} = B_{i,j}\ \forall (i,j)\ \in [\mathsf{mn}] \times [k] \end{array} \right\}$$

The proof system for an unknown permutation relation unfolds (Figure 5.3) very similarly to the case of a known permutation relation we outlined in Section 4.2.4. Define $J \in \mathbb{F}^{\mathsf{mn}\times k}$ to be the matrix that has 1 in all entries, i.e.,

$$J = \begin{pmatrix} 1 & & 1 \\ & \ddots & \\ 1 & & 1 \end{pmatrix}$$

Suppose the prover has committed to the rows of the two matrices $A, B \in \mathbb{F}^{m\times k}$. The idea behind the construction is to let the verifier pick a random challenge x and let the prover commit to $A - xJ$ and $B - xJ$. The prover will now convince the verifier

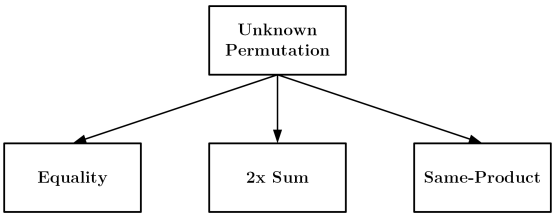

FIGURE 5.3: Decomposition of the unknown permutation proof over the ILC.

that the product of the entries in $A - xJ$ is equal to the product of the entries in $B - xJ$, i.e.

$$\prod_{i,j}^{\mathsf{mn},k} (A_{i,j} - x) = \prod_{i,j}^{\mathsf{mn},k} (B_{i,j} - x).$$

As in the case of a known permutation, this can be done by executing a proof for the same-product of matrices.

Permutation of Tuples. When checking the consistency of the memory, we use a permutation relation over tuples of entries in the execution table rather than single entries. This means that all entries in a tuple are subject to the same permutation. Over the ILC this can be phrased as checking that entries in matrices $\{A_i\}_i$ are permutation of entries in $\{B_i\}_i$, for the same permutation π. Our proof system can be extended to check an unknown permutation of collection of matrices by introducing an additional round of interaction between prover and verifier. It is sufficient to let the verifier pick a random challenge $z \leftarrow \mathbb{F}$ and let the prover commit to the matrices $A = zA_1 + z^2A_2 + z^3A_3 + \ldots$, and $B = zB_1 + z^2B_2 + z^3B_3 + \ldots$. The prover shows that the compressed matrices A and B are correctly formed using a proof for the correct sum of committed vectors (Section 4.2.2[3]) and that these two matrices are permutation of each other. In the specific case of the $\mathcal{R}_{\text{cycle}}$ relation, the last entry of the tuple consists of elements of the form $\mathsf{S} + \mathsf{L}$ which is used to distinguish between memory operations from the rest. In our final proof the entries of S and L will be committed into distinct matrices. These two can be merged into a single one in the compression step by weighting them with the same power of z, i.e. $z^4\mathsf{L} + z^4\mathsf{S} = z^4(\mathsf{S} + \mathsf{L})$.

[3]We only presented a proof for the sum of vectors rather than for their weighted sum. However, it is easy to see that the proof can be generalised by changing the verifier's opening query to include the weights of the linear combination. In the notation, we differentiate the case of a weighted sum by including the weights of the linear combination in the instance.

The description of our unknown permutation proof for $\mathcal{R}_{\text{perm}}$ on collection of matrices is given in Fig. 5.4. As in the previous chapter we delimit steps that are executed in other proofs with / * and * /, and include $[A]$ in the verifier's inputs to denote that the statement refers to the commitment of A in the ILC. Steps marked with ILC $\rightarrow \circ$ and ILC $\leftarrow \bullet$ denote incoming and outgoing messages to the ILC, respectively.

$\mathcal{P}_{\text{perm}}(pp_{\mathsf{ILC}}, (\mathsf{mn}, c), (\{A_i, B_i\}_{i=1}^{c}, \pi))$	$\mathcal{V}_{\text{perm}}(pp_{\mathsf{ILC}}, (\mathsf{mn}, c, (\{[A_i], [B_i]\}_{i=1}^{n})))$
Round 1:	**Round 1:**
/ *ILC $\leftarrow \bullet$ Send (**commit**, $\{A_i, B_i\}_{i=1}^{c}$) to the ILC * /	/ * ILC $\rightarrow \circ$ Get message $2\mathsf{mn}c$ from the ILC * /
	• $z \leftarrow \mathbb{F}$
	• $\mathbf{z} := (z, z^2, \ldots, z^n)$
	ILC $\leftarrow \bullet$ Send (**send**, z) to the ILC
Round 2:	**Round 2:**
ILC $\rightarrow \circ$ Get message z from the ILC	ILC $\rightarrow \circ$ Get message $2\mathsf{mn}$ from the ILC
• $A := \sum_{i=1}^{c} A_i z^i$	• $x \leftarrow \mathbb{F}$
• $B := \sum_{i=1}^{c} B_i z^i$	• For $(i, j) \in [\mathsf{mn}] \times [k]$:
• $\mathbf{z} := (z, z^2, \ldots, z^c)$	– $J'_{i,j} := x$
ILC $\leftarrow \bullet$ Send (**commit**, A, B) to the ILC	ILC $\leftarrow \bullet$ Send (**send**, x) to the ILC
Round 3 to log m + 4:	**Round 3 to log m + 4:**
ILC $\rightarrow \circ$ Get message x from the ILC	ILC $\rightarrow \circ$ Get message $3\mathsf{mn}$ from the ILC
• For $(i, j) \in [\mathsf{mn}] \times [k]$:	• Run $\mathcal{V}_{\mathsf{sum}}(pp_{\mathsf{ILC}}, (\mathsf{mn}, \mathbf{z}, [A_1], \ldots, [A_n], [A]))$
– $J'_{i,j} := x$	• Run $\mathcal{V}_{\mathsf{sum}}(pp_{\mathsf{ILC}}, (\mathsf{mn}, \mathbf{z}, [B_1], \ldots, [B_n], [B]))$
• $A' := A - J'$	• Run $\mathcal{V}_{\mathsf{eq}}(pp_{\mathsf{ILC}}, (\mathsf{mn}, J', [J']))$
• $B' := B - J'$.	• Run $\mathcal{V}_{\mathsf{sum}}(pp_{\mathsf{ILC}}, (\mathsf{mn}, [A'], [J'], [A])$
ILC $\leftarrow \bullet$ Send (**commit**, J', A', B') to the ILC	• Run $\mathcal{V}_{\mathsf{sum}}(pp_{\mathsf{ILC}}, (\mathsf{mn}, [B'], [J'], [B])$
• Run $\mathcal{P}_{\mathsf{sum}}\,(pp_{\mathsf{ILC}}, (\mathsf{mn}, \mathbf{z}), (A_1, \ldots, A_c, A))$	• Run $\mathcal{V}_{\mathsf{same\text{-}prod}}(pp_{\mathsf{ILC}}, (\mathsf{mn}, [A'], [B']))$
• Run $\mathcal{P}_{\mathsf{sum}}\,(pp_{\mathsf{ILC}}, (\mathsf{mn}, \mathbf{z}), (B_1, \ldots, B_c, B))$	• If all the sub-proofs accept return 1, else return 0
• Run $\mathcal{P}_{\mathsf{eq}}(pp_{\mathsf{ILC}}, J', J')$	
• Run $\mathcal{P}_{\mathsf{sum}}(pp_{\mathsf{ILC}}, (\mathsf{mn}), (A', J', A))$	
• Run $\mathcal{P}_{\mathsf{sum}}(pp_{\mathsf{ILC}}, (\mathsf{mn}), (B', J', B))$	
• Run $\mathcal{P}_{\mathsf{same\text{-}prod}}(pp_{\mathsf{ILC}}, (\mathsf{mn}), (A', B'))$	

FIGURE 5.4: Proof of knowledge for the relation $\mathcal{R}_{\text{perm}}$.

Theorem 5.1. *$(\mathcal{G}_{\mathsf{ILC}}, \mathcal{P}_{\text{perm}}, \mathcal{V}_{\text{perm}})$ is a proof system for the relation $\mathcal{R}_{\text{perm}}$ in the ILC model with perfect completeness, statistical strong knowledge soundness with straight-line extraction, and perfect special honest verifier zero-knowledge.*

Proof. We start by showing completeness. If the statement is true, then there exists a permutation π between the entries of matrices $\{A_i\}_i$ and the entries of $\{B_i\}_i$. This means that for the same permutation π we have $A'_{\pi(i,j)} = B'_{i,j}$. Therefore each entry of A' appears somewhere in matrix B'. Perfect completeness follows by the perfect completeness of the sub-proofs.

Next we show statistical strong knowledge soundness. As usual the knowledge extractor sees the vectors sent from the prover to the ILC, and therefore it has straight-line extraction. The knowledge soundness of the sum sub-proofs, guarantees that matrices A, B correspond to the weighted sums of A_i and B_i, respectively, apart with negligible probability. By the Schwartz-Zippel Lemma, if the statement is false then with overwhelming probability $(1 - \frac{c}{|\mathbb{F}|})$ over the choices of z there will be entries only contained either in A or B. The knowledge soundness of the equality and sum sub-proofs guarantees that, apart with negligible probability, matrix J' is correctly formed and that committed matrices A, B are the sums of $A' + J'$, and $B' + J'$, respectively. Moreover, from the knowledge soundness of the same-product proof, we get $\prod_{i,j}(A_{i,j} - x) = \prod_{i,j}(B_{i,j} - x)$. By applying the Schwartz-Zippel Lemma again, the probability over the random choice of $x \leftarrow \mathbb{F}$ of the above equality to hold is at most $\frac{\mathsf{m}\mathsf{n}k}{|\mathbb{F}|}$, which is negligible.

Finally, the proof system has perfect SHVZK. Matrix J' can be computed given the statement and the random challenges sent from the verifier. This is sufficient to simulate the view of the equality sub-proofs. The sub-proofs for the sum relation can be trivially simulated. The simulator then is only required to execute the simulator for the same-product sub-proof. □

Efficiency. The round complexity of the proof is of two rounds more than the same-product proof in the case of permutation of tuples, i.e. $\log(\mathsf{m}) + 4$ rounds. The number of verifier's queries is equal to the sum of the queries of the sub-proofs, i.e. $\mathsf{qc} = 16$ in the tuples case.

The verifier communication is $\mathcal{O}(\log(\mathsf{m}))$ challenges. The computational cost for the verifier is $\mathcal{O}(\mathsf{m}\mathsf{n} + k)$ field multiplications for the sum and same-product proofs. The equality sub-proof normally requires a linear number of additions. However, here the equality is used to check that matrix J' has entries all equal to x, and therefore

most of the computation can be reused. In this case the verifier can check that the response to the opening query is a vector with entries all equal to each other, and that this value is consistent with the linear combination of one column of J'. This only requires $\mathcal{O}(\mathsf{mn})$ field multiplications to compute. Therefore, the overall computational cost of the verifier is dominated by $\mathcal{O}(\mathsf{mn}+k)$ field multiplications, which is sublinear in the size of the matrices. It is worth noting that the verification cost of the known permutation proof of Section 4.2.4 is a linear number of additions.

The communication complexity for the prover is $\mathcal{O}(\mathsf{mn})$ vectors of length k, assuming that the tuples are of constant size ($c = \mathcal{O}(1)$), which is the only case we will use. The computational complexity of the prover is dominated by the cost of the same-product proof, which is in turn dominated by the cost of the product proof of Section 4.2.3 and therefore is equal to $\mathcal{O}(\mathsf{mn}k + k\mathsf{n}\log(\mathsf{n}))$. The efficiency of the proof system is summarised in Table 5.7.

Prover computation	$T_{\mathcal{P}_{\mathsf{ILC}}}$	$= \mathcal{O}(k\mathsf{n}\log\mathsf{n} + k\mathsf{mn})\ \mathbb{F}^{\times}$
Verifier computation	$T_{\mathcal{V}_{\mathsf{ILC}}}$	$= \mathcal{O}(k + \mathsf{mn})\ \mathbb{F}^{\times}$
Prover communication	t	$= \mathcal{O}(\mathsf{mn})\ \log\lvert\mathbb{F}^k\rvert$
Verifier communication	C_{ILC}	$= (\log\mathsf{m} + 4)\ \log\lvert\mathbb{F}\rvert$
Query complexity	qc	$= 16$
Round complexity	μ	$= \log\mathsf{m} + 4$

TABLE 5.4: Efficiency of the proof of knowledge for the $\mathcal{R}_{\text{perm}}$ relation. $\mathbb{F}^{\times}$ stands for the cost of a single field multiplication, $\mathbb{F}^{+}$ stands for the cost of a single field addition, and $\log|\mathbb{F}|$ for the size of a field element.

5.4.3 ILC Proofs for Lookup Relations

Here we give an idea behind the proof system over the ILC for lookup relations $\mathcal{R}_{\text{lookup}}$. As above, we reformulate the lookup relation $\mathcal{R}_{\text{lookup}}$ in terms of the entries in matrices for compatibility with the ILC channel. We think of the entries of a lookup table to be arranged in a matrix $B \in \mathbb{F}^{n_B \times k}$ and all the values we want to check in the lookup table to be arranged in a matrix $A \in \mathbb{F}^{n_A \times k}$. A lookup statement is that all entries $A_{i,j}$ in A are equal to some entry $B_{i',j'}$ in B. The corresponding relation is

$$\mathcal{R}_{\text{lookup}} = \left\{ \begin{array}{c} (pp_{\mathsf{ILC}}, u) := ((W, K, \mathbb{F}, k)\ ,\ (n_A, n_B),\ (A, B)) : \\ A \in \mathbb{F}^{n_A \times k} \quad \wedge \quad B \in \mathbb{F}^{n_B \times k} \quad \wedge \quad \{A_{i,j}\}_{i=1,j=1}^{n_A,k} \subseteq \{B_{i,j}\}_{i=1,j=1}^{n_B,k} \end{array} \right\}$$

In a completely identical way to the case of permutations of tuples, we can also extend the lookup relation to work on collection of matrices $\{A_i\}_i, \{B_i\}_i$.

The idea behind the construction of the proof is similar to the one used for checking permutations of entries in two matrices. Namely, we let the verifier pick a random challenge $x \leftarrow \mathbb{F}$ and have the prover show that

$$\prod_{i=1,j=1}^{n_A,k} (A_{i,j} - x) = \prod_{i=1,j=1}^{n_B,k} (B_{i,j} - x)^{e_{(i,j)}}$$

for some exponents $e_{i,j}$, denoting the number of times the entry $B_{i,j}$ in table B has be looked up from table A. For the left-hand side, the proof proceeds as in the permutation proof, where the prover commits to matrix $A - xJ$ for a matrix J with all entries equal to 1. For the right-hand side the prover has to commit to a matrix

$$E = \begin{pmatrix} e_{(1,1)_1} & \cdots & e_{(1,k)_1} \\ \vdots & & \vdots \\ e_{(1,1)_{\log(n_A k)}} & \cdots & e_{(1,k)_{\log(n_A k)}} \\ e_{(2,1)_1} & \cdots & e_{(2,k)_1} \\ \vdots & & \vdots \\ e_{(n_B,1)_{\log(n_A k)}} & \cdots & e_{(n_B,k)_{\log(n_A k)}} \end{pmatrix} \in \mathbb{F}^{n_B \log(n_A k) \times k}$$

where $e_{(i,j)_k}$ is the k-th bit in the binary decomposition of exponent $e_{(i,j)}$ associated with the lookup table entry $B_{i,j}$. The prover then shows the following holds

- The entries of E are all zeros and ones, i.e $(E \circ E = E)$.
- The sum of the exponents is equal to the number of entries in matrix A, i.e. $\sum_{i=1,j=1}^{n_B,k} e_{i,j} = n_A k$.
- Commit to a matrix $F \in \mathbb{F}^{n_B \log(n_A k) \times k}$ whose entries are equal to $(B_{i,j} - x)^{e_{(i,j)_k}}$. The correctness of this matrix is checked using few additional product and sum proofs.
- Use a same-product proof to show that product of entries in $A' = A - xJ$ is equal to the product of entries in F.

We refer to [BCG+18] for a description on how to perform each of the above checks and for the full specifications of the proof system $(\mathcal{G}_{\mathsf{ILC}}, \mathcal{P}_{\mathsf{lookup}}, \mathcal{V}_{\mathsf{lookup}})$. From there we also recall the next theorem which we state without proof.

Theorem 5.2 ([BCG+18]). $(\mathcal{G}_{\mathsf{ILC}}, \mathcal{P}_{\mathsf{lookup}}, \mathcal{V}_{\mathsf{lookup}})$ *is a proof system for the relation* $\mathcal{R}_{\mathrm{lookup}}$ *in the* ILC *model with perfect completeness, statistical strong knowledge soundness with straight-line extraction, and perfect special honest verifier zero-knowledge.*

This proof system uses as a building block the same-product proof which represents the dominant cost of the proof. The efficiency of the proof system is stated in Table 5.5, with respect to the case of lookup tables of constant size tuples. In the application of this proof, we typically have a much larger matrix A of elements that we lookup and a smaller lookup table B, so that n_A dominates $n_B \log(n_A k)$.

Prover computation	$T_{\mathcal{P}_{\mathsf{ILC}}}$	$= \mathcal{O}(n_A k + n_B k \log(n_A k))\ \mathbb{F}^{\times}$
Verifier computation	$T_{\mathcal{V}_{\mathsf{ILC}}}$	$= \mathcal{O}(k + n_A + n_B \log(n_A k))\ \mathbb{F}^{\times}$
Prover communication	t	$= \mathcal{O}(n_A + n_B \log(n_A k))\ \log \lvert\mathbb{F}^k\rvert$
Verifier communication	C_{ILC}	$= (\log(n_B) + 4)\ \log \lvert\mathbb{F}\rvert$
Query complexity	qc	$= 65$
Round complexity	μ	$= \log(n_B) + 4$

TABLE 5.5: Efficiency of the proof of knowledge for the $\mathcal{R}_{\mathrm{lookup}}$ relation. $\mathbb{F}^{\times}$ stands for the cost of a single field multiplication, $\mathbb{F}^{+}$ stands for the cost of a single field addition, and $\log |\mathbb{F}|$ for the size of a field element.

Bounded Lookups. In the process of checking the consistency of the memory, we introduced a bounded lookup relation to check that certain entries of the lookup table B could occur at most once in matrix A. It is easy to extend the previous lookup proof to incorporate the extra conditions required by a bounded lookup relation. The high level decomposition of the bounded lookup proof system is given in Figure 5.6

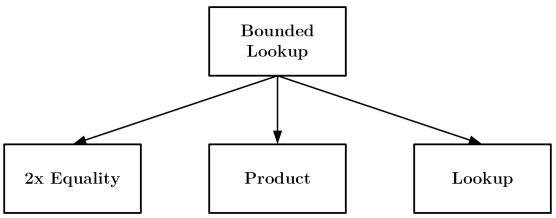

FIGURE 5.5: Decomposition of the bounded lookup proof over the ILC.

In the decomposition of the TinyRAM relation, the bounded lookup relation is used to check the consistency of tuples in rows of the execution table with rows of the

memory table Mem, which consist of three entries each (address, value, and usd). Let B_1, B_2, B_3 be the matrices composing the lookup table storing addresses, values, and usd flags respectively. Without loss of generality assume that the part of the lookup table that can be accessed at most once corresponds to the top half of the matrices $(B_1^\top, B_2^\top, B_3^\top)$ while the part that can be accessed multiple times corresponds to the bottom halves $(B_1^\perp, B_2^\perp, B_3^\perp)$. Similarly, consider matrices A_1, A_2, A_3 for the tuples of entries that we want to check in the table. As for the unknown permutation proof, we start compressing the above matrices into two matrices using a challenge $z \leftarrow \mathbb{F}$ from the verifier, i.e.

$$A = A_1 z + A_2 x^2 + A_3 z^3 \qquad B = B_1 z + B_2 x^2 + B_3 z^3$$

Once the matrices have been compressed, prover and verifier run a lookup proof on A and B. In the process, the prover has to commit to matrix $E \in \mathbb{F}^{n_B \log(n_A k) \times k}$ that encodes in binary form the number of times that each entry in B appears in A.

Let $B^\top$ be the top half of the matrix B containing the entries that can occur at most once in A, and $B^\perp$ be the bottom half of B. The prover commits to another matrix $G \in \mathbb{F}^{n_B \log(n_A k) \times k}$, the same size of E. This matrix contains zeros at entries relating to the positions in E that describe all but the least significant bits of the number of times $B^\top$ occurs in A. All the remaining entries of G are set equal to 1.

$$E = \begin{pmatrix} e_{(1,1)_1} & \cdots & e_{(1,k)_1} \\ e_{(1,1)_2} & \cdots & e_{(1,k)_2} \\ \vdots & & \vdots \\ e_{(1,1)_{\log(n_A k)}} & \cdots & e_{(1,k)_{\log(n_A k)}} \\ e_{(2,1)_1} & \cdots & e_{(2,k)_1} \\ \vdots & & \vdots \\ e_{(n_B,1)_{\log(n_A k)}} & \cdots & e_{(n_B,k)_{\log(n_A k)}} \end{pmatrix} \quad G = \begin{pmatrix} 0 & \cdots & 0 \\ 1 & \cdots & 1 \\ \vdots & & \vdots \\ 1 & \cdots & 1 \\ 0 & \cdots & 0 \\ \vdots & & \vdots \\ 0 & \cdots & 0 \end{pmatrix}$$

The correctness of the matrix G can be checked efficiently by using a proof system for equality relations. Lastly, the prover shows that $G \circ E$ is the zero matrix. This shows that E contains all zeroes for entries corresponding to $B^\top$, except possibly the least significant bit, which may be a zero or a one. The description of the proof system

over the ILC for the bounded lookup relation $\mathcal{R}_{\text{blookup}}$ is given in Figure 5.6.

$\mathcal{P}_{\mathsf{blookup}}(pp_{\mathsf{ILC}}, (n_A, n_B), (\{A_i\}_{i=1}^3, \{B_i\}_{i=1}^3))$	$\mathcal{V}_{\mathsf{blookup}}(pp_{\mathsf{ILC}}, (n_A, n_B, \{[A_i]\}_{i=1}^3, \{[B_i]\}_{i=1}^3))$
Round 1:	**Round 1:**
/ * ILC $\leftarrow \bullet$ Send (**commit**, $\{A_i, B_i\}_{i=1}^3$) to the ILC */	/ * ILC $\rightarrow \circ$ Get message $3(n_A + n_B)$ from the ILC */
	• $z \leftarrow \mathbb{F}$
	• $\mathbf{z} := (z, z^2, z^3)$
	ILC $\leftarrow \bullet$ Send (**send**, z) to the ILC
Round 2 to $\log n_B + 4$:	**Round 2 to $\log n_B + 4$:**
ILC $\rightarrow \circ$ Get message z from the ILC	ILC $\rightarrow \circ$ Get message $n_A + n_B + 2n_B \log(n_A k)$ from the ILC
• $\boldsymbol{z} := (z, z^2, z^3)$	• For $(i,j) \in [\frac{n_B \log(n_A k)}{2}] \times [k]$:
• $A := A_1 z + A_2 z^2 + A_3 z^3$	– If $i = 1 \pmod{\log(n_A k)}$: $G^{\top}_{i,j} := 0$
• $B := B_1 z + B_2 z^2 + B_3 z^3$	– Else: $G^{\top}_{i,j} := 1$
• For $(i,j) \in [\frac{n_B \log(n_A k)}{2}] \times [k]$:	– $G^{\bot}_{i,j} := 0$
– If $i = 1 \pmod{\log(n_A k)}$: $G^{\top}_{i,j} := 0$	– $O^{\top}_{i,j} := 0$
– Else: $G^{\top}_{i,j} := 1$	– $O^{\bot}_{i,j} := 0$
– $G^{\bot}_{i,j} := 0$	• Run $\mathcal{V}_{\text{sum}}(pp_{\mathsf{ILC}}, (n_A, \boldsymbol{z}, [A_1], [A_2], [A_3], [A]))$
– $O^{\top}_{i,j} := 0$	• Run $\mathcal{V}_{\text{sum}}(pp_{\mathsf{ILC}}, (n_B, \boldsymbol{z}, [B_1], [B_2], [B_3], [B]))$
– $O^{\bot}_{i,j} := 0$	• Run $\mathcal{V}_{\text{eq}}(pp_{\mathsf{ILC}}, (G, [G]))$
• $G := \begin{pmatrix} G^{\top} \\ G^{\bot} \end{pmatrix}, O := \begin{pmatrix} O^{\top} \\ O^{\bot} \end{pmatrix}$	• Run $\mathcal{V}_{\text{eq}}(pp_{\mathsf{ILC}}, (O, [O]))$
ILC $\leftarrow \bullet$ Send (**commit**, A, B, G, O) to the ILC	• Run $\mathcal{V}_{\text{lookup}}(pp_{\mathsf{ILC}}, (n_A, n_B, [A], [B]))$.
• Run $\mathcal{P}_{\text{sum}}(pp_{\mathsf{ILC}}, (n_A, \boldsymbol{z}), (A_1, A_2, A_3, A))$	• Run $\mathcal{V}_{\text{prod}}(pp_{\mathsf{ILC}}, (n_B \log(n_A k)), [G], [E], [O]))$
• Run $\mathcal{P}_{\text{sum}}(pp_{\mathsf{ILC}}, (n_B, \boldsymbol{z}), (B_1, B_2, B_3, B))$	• If all the sub-proofs accept return 1, else return 0
• Run $\mathcal{P}_{\text{eq}}(pp_{\mathsf{ILC}}, G, G)$	
• Run $\mathcal{P}_{\text{eq}}(pp_{\mathsf{ILC}}, O, O)$	
• Run $\mathcal{P}_{\text{lookup}}(pp_{\mathsf{ILC}}, (n_A, n_B), (A, B))$	
• Run $\mathcal{P}_{\text{prod}}(pp_{\mathsf{ILC}}, (n_B \log(n_A k)), (G, E, O))$	

FIGURE 5.6: Proof of knowledge for the relation $\mathcal{R}_{\text{blookup}}$. Matrix E is the exponent matrix used inside the proof for $\mathcal{R}_{\text{lookup}}$

Theorem 5.3. *$(\mathcal{G}_{\mathsf{ILC}}, \mathcal{P}_{\mathsf{blookup}}, \mathcal{V}_{\mathsf{blookup}})$ is a proof system for $\mathcal{R}_{\text{blookup}}$ in the ILC model with perfect completeness, statistical strong knowledge soundness with straight-line extraction, and perfect special honest verifier zero-knowledge.*

Proof. We start by showing completeness. If the statement is true, then all entries in (A_1, A_2, A_3) are also entries in (B_1, B_2, B_3). Moreover, entries in matrices $B_i^{\top}$ are only accessed once in the A_i. The same holds for the compressed matrices A, B. Matrices

G and O are constructed such that $G \circ E = O$. Perfect completeness then follows by the perfect completeness of all the sub-proofs.

Next we show statistical strong knowledge soundness. As usual the knowledge extractor sees the vectors sent from the prover to the ILC, and therefore it has straight-line extraction. The knowledge soundness of the sum sub-proofs, guarantees that matrices A, B correspond to the weighted sums of A_i and B_i, respectively, apart with negligible probability. By the Schwartz-Zippel Lemma, if the statement is false then with overwhelming probability $(1 - \frac{3}{|\mathbb{F}|})$ over the choices of z either there will be entries only contained either in A or B, or some entries of $B^\top$ are included more than once in A. The knowledge soundness of the equality sub-proofs guarantees that, apart with negligible probability, matrices G and O are correctly formed. From the knowledge soundness of the lookup proof, we get that all entries in A are contained in B, apart with negligible probability. This proof also guarantees that matrix E is correctly storing the number of times entries of B occur in A. Finally, the knowledge soundness of the product proof guarantees that if matrix $B^\top$ has entries occurring more than once in matrix A, then there is negligible probability that the verifier will accept that $G \circ E = O$.

Finally, the proof system has perfect SHVZK. Matrices G, O can be computed given the statement. This is sufficient to simulate the view of the equality sub-proofs. The sub-proofs for the sum relation can be trivially simulated. The transcript of a product subproof can be simulated by picking responses $\boldsymbol{v}_1, \boldsymbol{v}_2 \leftarrow \mathbb{F}^k$ and $\boldsymbol{v}_3 := \boldsymbol{v}_1 \circ \boldsymbol{v}_2$. The simulator then calls the simulators of the lookup proof and terminates. □

The efficiency of the bounded lookup proof (Table 5.6) is similar to the efficiency of the lookup proof. We stress that the verification cost of the equality sub-proofs can be reduced as in the case the unknown permutation proof, since the columns of the matrix are all the same.

5.4.4 ILC Proof for Range Relations

In Section 5.2.4 we showed that range relations can be reduced to lookup and sum relations. To perform range checks efficiently we batch them in a single proof. This

Prover computation	$T_{\mathcal{P}_{\mathsf{ILC}}}$	$= \mathcal{O}(n_A k + n_B k \log(n_A k))\ \mathbb{F}^{\times}$
Verifier computation	$T_{\mathcal{V}_{\mathsf{ILC}}}$	$= \mathcal{O}(k + n_A + n_B \log(n_A k))\ \mathbb{F}^{\times}$
Prover communication	t	$= \mathcal{O}(n_A + n_B \log(n_A k))\ \log\lvert\mathbb{F}^k\rvert$
Verifier communication	C_{ILC}	$= (\log(n_B) + 4)\ \log\lvert\mathbb{F}\rvert$
Query complexity	qc	$= 72$
Round complexity	μ	$= \log(n_B) + 4$

TABLE 5.6: Efficiency of the proof of knowledge for the $\mathcal{R}_{\text{blookup}}$ relation. $\mathbb{F}^{\times}$ stands for the cost of a single field multiplication, $\mathbb{F}^{+}$ stands for the cost of a single field addition, and $\log|\mathbb{F}|$ for the size of a field element.

is done by gathering all the values in $\mathtt{a} \in [0, 2^W - 1]$ into a single matrix A and insert their odd/even-bits decomposition $(\mathtt{a}_{\mathtt{o}}, \mathtt{a}_{\mathtt{e}})$ into two matrices $A_{\mathtt{o}}, A_{\mathtt{e}}$. Prover and verifier perform a lookup proof for matrices $A_{\mathtt{o}}, A_{\mathtt{e}}$ and matrix B storing the lookup table EvenBits. The only thing that remains to show is that the values in A are consistent with their odd/even-bits decomposition, i.e. $A = 2A_{\mathtt{o}} + A_{\mathtt{e}}$. The high-level decomposition of the range proof system over the ILC is given in Figure 5.7.

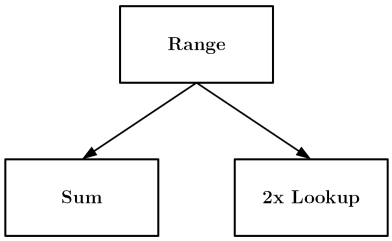

FIGURE 5.7: Decomposition of the range proof over the ILC.

We omit the formal description of the proof system for range relations as it consists of a straightforward combination of sum and lookup proofs. Furthermore, we have to check several other arithmetic constraints over the entries of two consecutive rows in the execution table. We can batch these together with the above sum relations in a single proof. Overall, range relations will be checked by lookup proofs and a proof for the consistency of arithmetic constraints which we describe next.

5.4.5 ILC Proof for Arithmetic Constraints

In the previous sections we decomposed the arithmetized TinyRAM relation into several simpler relations, some of which contained arithmetic constraints over the entries of one row of the execution table, $\mathcal{R}_{\text{time}}, \mathcal{R}_{\text{load}}$ or over pairs of consecutive rows in the execution table, $\mathcal{R}_{\text{mux}}, \mathcal{R}_{\text{cons}}, \mathcal{R}_{\text{inst}}, \mathcal{R}_{\text{logic}}, \mathcal{R}_{\text{arith}}, \mathcal{R}_{\text{shift}}$. One way to handle all these

constraints would be to gather all of them into an arithmetic circuit over the entries of Exe and then use the proof for the satisfiability of arithmetic circuits we gave in the Chapter 4. Since the set of constraints is the same for every pair of rows, the resulting arithmetic circuit would be a repetition of the same sub-circuit applied for every pair of consecutive rows. Since we are interested to prove evaluations of the same circuit many times on different inputs, there are batching techniques to perform this check more efficiently. Since all the constraints we used are either linear constraints or quadratic constraints over the entries of the execution table, we can use quadratic arithmetic programs (QAP), which were used by [GGP+13] in the context of succinct non-interactive argument of knowledge. Quadratic arithmetic programs are a means to batch together a set of quadratic equations into a single polynomial equation. In the interactive settings, similar batching techniques were used in [Bay14] and [BG18] to construct arguments for batch evaluation of polynomials.

A quadratic arithmetic program QAP over a field $\mathbb{F}$ consists of three sets of polynomials $\{u_i(X)\}_{i=0}^m$, $\{v_i(X)\}_{i=0}^m$, $\{w_i(X)\}_{i=0}^m$ and a target polynomial $t(X)$ such that

- All polynomials $u_i(X), v_i(X), w_i(X)$ and $t(X)$ are in $\mathbb{F}[X]$.
- The polynomials $u_i(X), v_i(X), w_i(X)$ have degree strictly lower than n, for $1 \leq i \leq m$.
- The degree of $t(X)$ is equal to n.

The *size* of the QAP is the size of the sets of polynomials, i.e. m, while the *degree* of the QAP is the degree of $t(X)$, i.e. n.

Let $\mathsf{QAP} = (\mathbb{F}, n, \{u_i(X), v_i(X), w_i(X)\}_{i=0}^m, t(X))$ be a quadratic arithmetic program. Elements $\{a_i\}_{i=1}^m \in \mathbb{F}$ are said to constitute a *valid assignment* to the QAP if the polynomial $t(X)$ divides the following polynomial

$$\left(u_0(X) + \sum_{i=1}^{m} a_i \cdot u_i(X)\right) \cdot \left(v_0(X) + \sum_{i=1}^{m} a_i \cdot v_i(X)\right) - \left(w_0(X) + \sum_{i=1}^{m} a_i \cdot w_i(X)\right)$$

Gennaro et al. [GGP+13] shows how to construct QAP for general arithmetic circuits, such that valid assignments to the wires of the circuit correspond to valid assignments of a QAP. Given a circuit with n inputs and s fan-in 2 multiplication gates they show how to construct a QAP of size $n + s$ and degree s ([GGP+13, Theorem 12]).

Overall, the relations $\mathcal{R}_{\text{time}}, \mathcal{R}_{\text{load}}, \mathcal{R}_{\text{mux}}, \mathcal{R}_{\text{cons}}, \mathcal{R}_{\text{inst}}, \mathcal{R}_{\text{logic}}, \mathcal{R}_{\text{arith}}, \mathcal{R}_{\text{shift}}$ specify a constant number of quadratic constraints over entries of at most two rows, comprising a constant number of field elements each. Therefore, by following [GGP+13] we can construct a QAP of constant size and constant degree and give a proof that the entries in two consecutive rows constitute a valid assignment to the QAP. Namely, we are interested in giving a proof system for the following relation.

$$\mathcal{R}_{\mathsf{QAP}} = \left\{ \begin{array}{c} (pp_{\mathsf{ILC}}, u, w) := \Big((\mathbb{F}, k),\ (\mathsf{QAP}),\ (\{a_i\}_{i=1}^{m}, \{z_j\}_{j=0}^{n-2}) \Big) : \\ \mathsf{QAP} := (\mathbb{F}, n, \{u_i(X), v_i(X), w_i(X)\}_{i=0}^{m}, t(X)) \quad \wedge \\ \forall\, i \in [m], a_i \in \mathbb{F} \quad \wedge \quad \forall\, j \in [0, n-2], z_j \in \mathbb{F} \quad \wedge \\ u(x) := u_0(X) + \sum_{i=1}^{m} a_i u_i(X) \quad \wedge \quad v(X) := v_0(X) + \sum_{i=1}^{m} a_i v_i(X) \wedge \\ w(X) := w_0(X) + \sum_{i=1}^{m} a_i w_i(X) \quad \wedge \quad z(X) := \sum_{i=0}^{n-2} z_i X^i \quad \wedge \\ u(X) \cdot v(X) = w(X) + t(X) z(X) \end{array} \right\}$$

More precisely, we will give a proof to check the validity of $T-1$ assignments to the same QAP, corresponding to the entries of the $T-1$ pairs of consecutive rows in the execution table. Since the ILC allows to commit to vectors of length k, we can extend the polynomial expression used to check the validity of one assignment to a polynomial expression with vector coefficients to check k instances of the same QAP at the same time. This is done by arranging the assignment to the same instance into the same position of vectors $\boldsymbol{a}_i \in \mathbb{F}^k$ for $1 \leq i \leq m$, and check that $t(X)$ divides each component of the following vector of polynomials

$$(\mathbf{1} \cdot u_0(X) + \textstyle\sum_{i=1}^{m} \boldsymbol{a}_i \cdot u_i(X)) \circ (\mathbf{1} \cdot v_0(X) + \sum_{i=1}^{m} \boldsymbol{a}_i \cdot v_i(X)) - (\mathbf{1} \cdot w_0(X) + \sum_{i=1}^{m} \boldsymbol{a}_i \cdot w_i(X))$$

where $\mathbf{1}$ is the vector of length k with entries all equal to 1.

Let mn be the number of vector assignments to be checked on the same QAP, i.e. $T \approx \mathsf{mn}k$. In our proof system, the prover starts by committing to vectors $\{\boldsymbol{a}_{i,j}\}_{i=1,j=1}^{m,\mathsf{mn}}$ such that for every fixed j, the vectors $\{\boldsymbol{a}_{i,j}\}_{i=1}^{m}$ correspond to a batch of k QAP assignments as described above. Furthermore, let $\{\boldsymbol{a}_{0,j}\}_{j=1}^{\mathsf{mn}}$ be the vectors with entries all equal to 1. Then, the prover computes the quotient polynomials $\boldsymbol{z}_j(X)$ for every

$j \in [\mathsf{mn}]$

$$\begin{aligned} \boldsymbol{z}_j(X) &= \sum_{k=0}^{n-2} \boldsymbol{z}_{k,j} X^i \\ &= \left[\left(\sum_{i=0}^{m} \boldsymbol{a}_{i,j} u_i(X) \right) \circ \left(\sum_{i=0}^{m} \boldsymbol{a}_{i,j} v_i(X) \right) - \sum_{i=0}^{m} \boldsymbol{a}_{i,j} w_i(X) \right] / t(X) \end{aligned} \tag{5.7}$$

and commits to their coefficient $\{\boldsymbol{z}_{k,j}\}_{k=0,j=1}^{n-2,\mathsf{mn}}$. At this point the prover shows that for each $j \in [\mathsf{mn}]$ the following holds

$$\left(\sum_{i=0}^{m} \boldsymbol{a}_{i,j} u_i(X) \right) \circ \left(\sum_{i=0}^{m} \boldsymbol{a}_{i,j} v_i(X) \right) = \sum_{i=0}^{m} \boldsymbol{a}_{i,j} w_i(X) + \boldsymbol{z}_j(X) t(X)$$

These identities can be tested using the Schwartz-Zippel Lemma. The verifier picks a random challenge $x \leftarrow \mathbb{F}$ and the prover evaluates the above polynomials in x to obtain

$$\begin{aligned} \hat{\boldsymbol{a}}_j &= \sum_{i=0}^{m} \boldsymbol{a}_{i,j} u_i(x) \\ \hat{\boldsymbol{b}}_j &= \sum_{i=0}^{m} \boldsymbol{a}_{i,j} v_i(x) \\ \hat{\boldsymbol{c}}_j &= \sum_{i=0}^{m} \boldsymbol{a}_{i,j} w_i(x) + \sum_{i=1}^{m} \boldsymbol{z}_{i,j} x^i t(x) \end{aligned}$$

At this point the prover shows the consistency of the $\boldsymbol{a}_{i,j}$ with the above evaluations and that $\hat{\boldsymbol{a}}_j \circ \hat{\boldsymbol{b}}_j = \hat{\boldsymbol{c}}_j$ for all $j \in [\mathsf{mn}]$. The full description of the proof system for a batch of $\mathsf{mn}k$ instances of the same QAP is given in Figure 5.8.

Theorem 5.4. *$(\mathcal{G}_{\mathsf{ILC}}, \mathcal{P}_{\mathsf{QAP}}, \mathcal{V}_{\mathsf{QAP}})$ is a proof system for the (batched) relation $\mathcal{R}_{\mathsf{QAP}}$ in the* ILC *model with perfect completeness, statistical strong knowledge soundness with straight-line extraction, and perfect special honest verifier zero-knowledge.*

Proof. We start by showing completeness. If the statement is true, then there exist polynomials $\boldsymbol{z}_j(X)$ of degree at most $n-2$ such that

$$\left(\sum_{i=0}^{m} \boldsymbol{a}_{i,j} u_i(X) \right) \circ \left(\sum_{i=0}^{m} \boldsymbol{a}_{i,j} v_i(X) \right) = \sum_{i=0}^{m} \boldsymbol{a}_{i,j} w_i(X) + \boldsymbol{z}_j(X) t(X)$$

$\mathcal{P}_{\mathsf{QAP}}(pp_{\mathsf{ILC}}, (\mathsf{mn}, \mathsf{QAP}), (\{\{\boldsymbol{a}_{i,j}\}_{i=1}^{m}, \{\boldsymbol{z}_{k,j}\}_{k=0}^{n-2}\}_{j=1}^{\mathsf{mn}}))$	$\mathcal{V}_{\mathsf{QAP}}(pp_{\mathsf{ILC}}, (\mathsf{mn}, \mathsf{QAP}, \{\{[\boldsymbol{a}_{i,j}]\}_{i=1}^{m}, \{[\boldsymbol{z}_{k,j}]\}_{k=0}^{n-2}\}_{j=1}^{\mathsf{mn}})$
Round 1:	**Round 1:**
• Parse QAP as $(\mathbb{F}, n, \{u_i(X), v_i(X), w_i(X)\}_{i=0}^{m}, t(X))$	
• For $j \in [\mathsf{mn}]$:	
– $\boldsymbol{a}_{0,j} := \mathbf{1}$	
– $\boldsymbol{z}_j(X) := \sum_{k=0}^{n-2} \boldsymbol{z}_{k,j} X^i$ (as in (5.7))	
/ * ILC ← • Send $(\textbf{commit}, \{\boldsymbol{a}_{i,j}\}_{i=1,j=1}^{m,\ \mathsf{mn}})$ to the ILC */	/ * ILC → ∘ Get message $m$mn from the ILC */
	ILC → ∘ Get message $n$mn from the ILC
	• $x \leftarrow \mathbb{F}$
ILC ← • Send $(\textbf{commit}, \{\boldsymbol{a}_{0,j}\}_{j=1}^{\mathsf{mn}}, \{\boldsymbol{z}_{k,j}\}_{k=0,j=1}^{n-2,\ \mathsf{mn}})$ to the ILC	ILC ← • Send $(\textbf{send}, x)$ to the ILC
Round 2 to log mn + 3:	**Round 2 to log mn + 3:**
ILC → ∘ Get message x from the ILC	ILC → ∘ Get message 3mn from the ILC
• For $j \in [\mathsf{mn}]$:	• Parse QAP as $(\mathbb{F}, n, \{u_i(X), v_i(X), w_i(X)\}_{i=0}^{m}, t(X))$
– $\hat{\boldsymbol{a}}_j := \sum_{i=0}^{m} \boldsymbol{a}_{i,j} u_i(x)$	• Run $\mathcal{V}_{\text{eq}}(pp_{\mathsf{ILC}}, (\{\mathbf{1}\}_{j=1}^{\mathsf{mn}}, \{[\boldsymbol{a}_{0,j}]\}_{j=1}^{\mathsf{mn}}))$
– $\hat{\boldsymbol{b}}_j := \sum_{i=0}^{m} \boldsymbol{a}_{i,j} v_i(x)$	• Run $\mathcal{V}_{\text{sum}}(pp_{\mathsf{ILC}}, (m\mathsf{mn}, \{u(x)_i\}_{i=0}^{m}, \{[\boldsymbol{a}_{0,j}], \ldots [\boldsymbol{a}_{m,j}], [\hat{\boldsymbol{a}}_j]\}_{j=1}^{\mathsf{mn}}))$
– $\hat{\boldsymbol{c}}_j := \sum_{i=0}^{m} \boldsymbol{a}_{i,j} w_i(x) + \boldsymbol{z}_j t(x)$	• Run $\mathcal{V}_{\text{sum}}(pp_{\mathsf{ILC}}, (m\mathsf{mn}, \{v(x)_i\}_{i=0}^{m}, \{[\boldsymbol{a}_{0,j}], \ldots, [\boldsymbol{a}_{m,j}], [\hat{\boldsymbol{b}}]_j\}_{j=1}^{\mathsf{mn}}))$
ILC ← • Send $(\textbf{commit}, \{\hat{\boldsymbol{a}}_j, \hat{\boldsymbol{b}}_j, \hat{\boldsymbol{c}}_j\}_{j=1}^{\mathsf{mn}})$ to the ILC	• Run $\mathcal{V}_{\text{sum}}(pp_{\mathsf{ILC}}, (m\mathsf{mn}, (\{w(x)_i\}_{i=0}^{m}, t(x), \{[\boldsymbol{a}_{0,j}], \ldots, [\boldsymbol{a}_{m,j}], [\boldsymbol{z}_j], [\hat{\boldsymbol{c}}_j]\}_{j=1}^{\mathsf{mn}}))$
• Run $\mathcal{P}_{\text{eq}}(pp_{\mathsf{ILC}}, \{\mathbf{1}\}_{j=1}^{\mathsf{mn}}, \{\boldsymbol{a}_{0,j}\}_{j=1}^{\mathsf{mn}})$	• Run $\mathcal{P}_{\text{prod}}(pp_{\mathsf{ILC}}, (\mathsf{mn}, \{[\hat{\boldsymbol{a}}_j], [\hat{\boldsymbol{b}}_j], [\hat{\boldsymbol{c}}_j\}_{j=1}^{\mathsf{mn}}]))$
• Run $\mathcal{P}_{\text{sum}}(pp_{\mathsf{ILC}}, (m\mathsf{mn}, \{u(x)_i\}_{i=0}^{m}), (\{\boldsymbol{a}_{0,j}, \ldots \boldsymbol{a}_{m,j}, \hat{\boldsymbol{a}}_j\}_{j=1}^{\mathsf{mn}}))$	• If all the sub-proofs accept return 1, else return 0
• Run $\mathcal{P}_{\text{sum}}(pp_{\mathsf{ILC}}, (m\mathsf{mn}, \{v(x)_i\}_{i=0}^{m}), (\{\boldsymbol{a}_{0,j}, \ldots, \boldsymbol{a}_{m,j}, \hat{\boldsymbol{b}}_j\}_{j=1}^{\mathsf{mn}}))$	
• Run $\mathcal{P}_{\text{sum}}(pp_{\mathsf{ILC}}, (m\mathsf{mn}, (\{w(x)_i\}_{i=0}^{m}, t(x)), (\{\boldsymbol{a}_{0,j}, \ldots, \boldsymbol{a}_{m,j}, \boldsymbol{z}_j, \hat{\boldsymbol{c}}_j\}_{j=1}^{\mathsf{mn}}))$	
• Run $\mathcal{P}_{\text{prod}}(pp_{\mathsf{ILC}}, (\mathsf{mn}), (\{\hat{\boldsymbol{a}}_j, \hat{\boldsymbol{b}}_j, \hat{\boldsymbol{c}}_j\}_{j=1}^{\mathsf{mn}}))$	

FIGURE 5.8: Proof of knowledge for the batched $\mathcal{R}_{\mathsf{QAP}}$ relation.

The above equality holds also when evaluating the polynomials at x. The completeness of the proof system then follows from the completeness of the sub-proofs.

Next, we show that the proof has statistical strong knowledge soundness. As usual the extraction is straight-line since the extractor sees the transcript of the communication between the prover and the ILC channel, and thus obtain a witness. It remains to show that for any deterministic malicious prover $\mathcal{P}^*$, if the committed vectors are not a valid witness for $\mathcal{R}_{\mathsf{QAP}}$, then there is negligible probability of accept. By the knowledge soundness of the equality sub-proof, the vectors $\boldsymbol{a}_{0,j}$ are equal to $\mathbf{1}$ with overwhelming probability. Also, by the knowledge-soundness of the sum proof, it must be that for each j, the values $\hat{\boldsymbol{a}}_j, \hat{\boldsymbol{b}}_j, \hat{\boldsymbol{c}}_j$ were computed correctly, i.e.

$$\begin{aligned}
\hat{\boldsymbol{a}}_j &= \sum_{i=1}^{m} \boldsymbol{a}_{i,j} u_i(x) \\
\hat{\boldsymbol{b}}_j &= \sum_{i=1}^{m} \boldsymbol{a}_{i,j} v_i(x) \\
\hat{\boldsymbol{c}}_j &= \sum_{i=1}^{m} \boldsymbol{a}_{i,j} w_i(x) + \sum_{i=1}^{m} \boldsymbol{z}_{i,j} x^i t(x)
\end{aligned}$$

In case the statement is false, then the vectors $\boldsymbol{a}_{i,j}$ do not form valid assignments for the QAP. This means that there exists at least one component for which $t(X)$ does not divide

$$\left(\sum_{i=0}^{m} \boldsymbol{a}_{i,j} u_i(X)\right) \circ \left(\sum_{i=0}^{m} \boldsymbol{a}_{i,j} v_i(X)\right) - \sum_{i=0}^{m} \boldsymbol{a}_{i,j} w_i(X)$$

By the knowledge-soundness of the product proof, we have that

$$\left(\sum_{i=1}^{m} \boldsymbol{a}_{i,j} u_i(x)\right) \circ \left(\sum_{i=1}^{m} \boldsymbol{a}_{i,j} v_i(x)\right) = \sum_{i=1}^{m} \boldsymbol{a}_{i,j} w_i(x) + \sum_{i=1}^{m} \boldsymbol{z}_{i,j} x^i t(x).$$

If the statement is false, then the left-hand side polynomial and the right-hand side polynomial are not equal. By the Schwartz-Zippel Lemma, the above check has probability at most $\frac{2n-2}{\mathbb{F}}$ to succeed. Therefore, the proof system has statistical strong knowledge soundness.

Finally, the proof system has perfect SHVZK. Given vector $\mathbf{1}$ it is possible to simulate the view of the equality sub-proof. The sub-proofs for the sum relations can be trivially simulated. The transcript of a product subproof can be simulated by picking

responses $\boldsymbol{v}_1, \boldsymbol{v}_2 \leftarrow \mathbb{F}^k$ and $\boldsymbol{v}_3 := \boldsymbol{v}_1 \circ \boldsymbol{v}_2$. By the perfect SHVZK of the sub-proofs, the transcript produced by the simulator has the same distribution of a real transcript. □

Efficiency. Here we report the efficiency of the proof system for $\mathsf{m}\mathsf{n}k$ batches of QAP of size $m = \mathcal{O}(1)$ and degree $n = \mathcal{O}(1)$. The round complexity of the proof is of one round more than the product proof of Section 4.2.3, i.e. $\log(\mathsf{m})+3$ rounds. The number of queries done by the verifier is equal to the sum of the queries of the sub-proofs, i.e. $\mathsf{qc} = 7$.

The verifier communication is $\mathcal{O}(\log(\mathsf{m}))$ challenges. The computational cost for the verifier is of $\mathcal{O}(\mathsf{m}\mathsf{n} + k)$ field multiplications for the sum and product proofs. The equality sub-proof normally requires a linear number of additions. However, here the equality is used to check that vectors have all entries all equal to 1. Therefore, the cost of verifying this query can be reduced to only check the first component of the response, and then check that all components of the response vector are equal. Thus, this only requires $\mathcal{O}(\mathsf{m}\mathsf{n})$ field multiplications to verify. For a QAP of constant size and degree, the overall computational cost of the verifier is dominated by $\mathcal{O}(\mathsf{m}\mathsf{n} + k)$ field multiplications.

The communication complexity for the prover is $\mathcal{O}(\mathsf{m}\mathsf{n})$ vectors of length k, assuming that the size of the QAP is constant. The computational complexity of the prover is dominated by the cost of the Hadamard product proof which is equal to $\mathcal{O}(\mathsf{m}\mathsf{n}k + k\mathsf{n}\log(\mathsf{n}))$. The efficiency of the proof system is summarised in Table 5.7.

Prover computation	$T_{\mathcal{P}_{\mathsf{ILC}}}$	$= \mathcal{O}(k\mathsf{n}\log\mathsf{n} + k\mathsf{m}\mathsf{n})\ \mathbb{F}^{\times}$
Verifier computation	$T_{\mathcal{V}_{\mathsf{ILC}}}$	$= \mathcal{O}(k + \mathsf{m}\mathsf{n})\ \mathbb{F}^{\times}$
Prover communication	t	$= \mathcal{O}(\mathsf{m}\mathsf{n})\ \log\lvert\mathbb{F}^k\rvert$
Verifier communication	C_{ILC}	$= \mathcal{O}(\log\mathsf{m})\ \log\lvert\mathbb{F}\rvert$
Query complexity	qc	$= 7$
Round complexity	μ	$= \log\mathsf{m} + 3$

TABLE 5.7: Efficiency of the proof of knowledge for the $\mathcal{R}_{\mathsf{QAP}}$ relation, in the case of batches of size $\mathsf{m}\mathsf{n}k$. $\mathbb{F}^{\times}$ stands for the cost of a single field multiplication, $\mathbb{F}^{+}$ stands for the cost of a single field addition, and $\log|\mathbb{F}|$ for the size of a field element.

5.5 ILC Proof for the Correct Execution of TinyRAM

In this section we combine the building-block proof introduced in the previous section to construct a proof system for the correct execution of a TinyRAM program over the ILC channel. We recall that we target programs performing intensive computation, for which the execution time dominates both the program length and the memory usage, i.e. $T > L + M$. In Section 5.3 we broke the relation $\mathcal{R}^{\mathsf{field}}_{\mathrm{TinyRAM}}$ (recalled next) down to a number of sub-relations defined over a finite field $\mathbb{F}$.

$$\mathcal{R}^{\mathsf{field}}_{\mathrm{TinyRAM}} = \left\{ \begin{array}{c} (pp, u, w) := ((W, K, \mathbb{F}, *), (P, v, T, M), \mathsf{w}) : \\ \mathsf{w} := (\mathsf{Exe}, \mathsf{Prog}, \mathsf{Mem}, \mathsf{EvenBits}, \mathsf{Pow}, \pi) \\ (pp, (P, v, T, M), \mathsf{w}) \in \mathcal{R}_{\mathrm{check}} \\ (pp, (T, M), \mathsf{w}) \in \mathcal{R}_{\mathrm{mem}} \\ (pp, \perp, \mathsf{w}) \in \mathcal{R}_{\mathrm{step}} \end{array} \right\}$$

The relation $\mathcal{R}_{\mathrm{check}}$ checks that certain entries in the extended witness w contain publicly known values included in the instance. This relation decomposes into five equality relations. The relation $\mathcal{R}_{\mathrm{mem}}$ checks the consistency of the memory throughout the execution and it is decomposed into an unknown permutation relation, a bounded lookup relation and a number of lookup and arithmetic constraints. Finally the relation $\mathcal{R}_{\mathrm{step}}$ checks the execution of each TinyRAM instruction. This is decomposed into several lookup and arithmetic constraints relations. Our strategy to prove that they are all satisfied is to commit to the extended witness w using the ILC channel. Then, we gather all the lookup relations and arithmetic constraints together and we give proof for these, as well as for the equalities, permutation and bounded lookup.

To begin we describe how to arrange the extended witness into matrices, so that they can be committed using the ILC. To facilitate the sub-proofs for arithmetic constraints, the witness is committed using a specific format. In addition to the above relations, then we also need to check that the committed matrices are committed as specified which is done by checking some additional relation $\mathcal{R}_{\mathsf{format}}$ described below.

5.5.1 Commitments to the Tables

In our proof system, the prover first commits to the extended witness w. The extended witness includes the field elements in the execution table Exe, the memory table Mem, the program table Prog, the auxiliary lookup tables EvenBits and Pow. The prover arranges these tables in multiple matrices and commits to their rows using the ILC.

The prover arranges each column of the execution table in an $\ell \times k$ matrix (such as the T entries containing the time t, the T entries containing the program counter pc_t, etc. are in the same matrix), and then she commits to each row of the resulting matrices. Entries of Exe relative to the same TinyRAM state (i.e. entries to the same row of the Exe) will be inserted in the same position across the different matrices. Furthermore, in all these matrices the last entry of each column is duplicated in the first entry of the next column. As an example, let us consider the first column of Exe which contains field elements representing the time-steps of the execution. Without loss of generality let $T = (\ell - 1)k + 1$, where T is the number of steps executed by the program and k is the vector length of the ILC. The prover organizes the field elements representing time in a matrix $\mathsf{E}_t \in \mathbb{F}^{\ell \times k}$

$$\mathsf{E}_t = \begin{pmatrix} 1 & \ell & 2\ell - 1 & \dots & \\ 2 & \ell + 1 & 2\ell & \dots & \\ \vdots & & & \ddots & \\ \ell - 1 & 2\ell - 2 & 3\ell - 3 & \dots & (\ell - 1)k \\ \ell & 2\ell - 1 & 3\ell - 2 & \dots & T \end{pmatrix}$$

Similarly, the prover organizes the rest of the Exe table into a constant number of matrices $\mathsf{E}_{\mathsf{pc}}, \mathsf{E}_{\mathsf{inst}}, \mathsf{E}_A, \dots$ one for each column. Let E be the matrix obtained by stacking all matrices on top of each other and let $\mathsf{E} = \{\boldsymbol{e}_i\}$, for $\boldsymbol{e}_i \in \mathbb{F}^k$. The prover commits to Exe by sending the command $(\textbf{commit}, \mathsf{E})$ to the ILC.

Each column of the program table is also committed to the ILC separately. In case $L \leq k$ we can store each column of Prog in one vector, i.e.

$$\mathsf{P} = \begin{pmatrix} \mathsf{P}_{\mathsf{pc}} \\ \mathsf{P}_{\mathsf{inst}} \\ \mathsf{P}_A \\ \dots \end{pmatrix} = \begin{pmatrix} 0 & 1 & \dots & L-1 \\ \mathsf{inst}_0 & \mathsf{inst}_1 & \dots & \mathsf{inst}_{L-1} \\ A_0 & A_1 & \dots & A_{L-1} \\ \dots & & \dots & \end{pmatrix}$$

otherwise, multiple rows can be used. The prover commits the program table Prog by sending $(\textbf{commit}, \mathsf{P})$ to the ILC channel.

The memory table Mem, is arranged in matrix M in the following way.

$$\mathsf{M} = \begin{pmatrix} \mathsf{M}_{\mathsf{addr}} \\ \mathsf{M}_{\mathsf{v}} \\ \mathsf{M}_{\mathsf{usd},0} \\ \mathsf{M}_{\mathsf{usd},1} \end{pmatrix} = \begin{pmatrix} 0 & 1 & \dots & M-1 \\ \mathsf{v}_0 & \mathsf{v}_1 & \dots & \mathsf{v}_{M-1} \\ 0 & 0 & \dots & 0 \\ 1 & 1 & \dots & 1 \end{pmatrix}$$

The entries of the memory table are not committed twice for each value of the flag usd. The reason we duplicated the memory in Section 5.2 is that in the bounded lookup relation we must check that for each address addr, the entries associated with $\mathsf{usd} = 0$ are accessed at most once. However in the proof we can parse the vectors and arrange them into sub-matrices without the need of recommitting to the rows. The sub-matrices used in the bounded lookup proofs are $\mathsf{M}^{\top}$, whose entries can only be accessed once, and $\mathsf{M}^{\perp}$.

$$\mathsf{M}^{\top} = \begin{pmatrix} \mathsf{M}_{\mathsf{addr}} \\ \mathsf{M}_{\mathsf{v}} \\ \mathsf{M}_{\mathsf{usd},0} \end{pmatrix} \qquad \mathsf{M}^{\perp} = \begin{pmatrix} \mathsf{M}_{\mathsf{addr}} \\ \mathsf{M}_{\mathsf{v}} \\ \mathsf{M}_{\mathsf{usd},1} \end{pmatrix}$$

The prover commits the memory table Mem by sending $(\textbf{commit}, \mathsf{M})$ to the ILC channel.

The auxiliary lookup table EvenBits and the exponent table Pow can be committed in a similar way using matrices R and S

$$\mathsf{R} = \begin{pmatrix} 0 & 1 & 4 & 5 & \dots & \sum_{i=0}^{\frac{W}{2}-1} k_i 2^{2i} \\ & & & \ddots & & \\ & & & & & \sum_{i=0}^{\frac{W}{2}-1} 2^{2i} \end{pmatrix} \qquad \mathsf{S} = \begin{pmatrix} 0 & 1 & 2 & 3 & \dots & W-1 & W \\ 1 & 2 & 4 & 8 & \ddots & 2^{W-1} & 0 \end{pmatrix}$$

where $(k_{\frac{W}{2}-1}, \dots, k_0)$ is the binary expansion of k. If any of the above tables have rows that are shorter that k, these can be filled with zeros. These additional entries are checked together with the correctness of the table by the relation $\mathcal{R}_{\text{check}}$. We also need to ensure we use an odd/even-bits decomposition that allows to store $|\mathsf{EvenBits}| \ll T$, for instance $|\mathsf{EvenBits}| = \mathcal{O}(\sqrt{T})$. This can always be done by using a decomposition into $\kappa = \mathcal{O}(1)$ words since $W = \mathcal{O}(\log(n))$.The prover commits the auxiliary lookup tables EvenBits, Pow by sending $(\textbf{commit}, \mathsf{R}, \mathsf{S})$ to the ILC channel.

Format Relation. The reason we duplicate the last entry in each column of matrix E is due to the way we check arithmetic constraints. This is done by using a QAP over pairs of consecutive rows (in the execution table Exe) that correspond to entries in consecutive rows in the sub-matrices of E. Since we batched k QAP together by using vector coefficients, entries relative to the same QAP-assignment need to be in the same component of the committed vectors. Therefore, the duplication of the last entries in the columns allows to check the transition between consecutive states that would be otherwise dislocated in different components of the committed vectors. On the other hand, this introduces an additional consistency check for on the entries of the first and last rows of matrices $\{\mathsf{E}_{\mathsf{pc}}, \mathsf{E}_{\mathsf{inst}}, \mathsf{E}_A, \dots\}$ comprising E, as described by the following relation.

$$\mathcal{R}_{\mathsf{format}} = \left\{ \begin{array}{c} (pp_{\mathsf{ILC}}, u, w) := ((W, K, \mathbb{F}, *), \perp, \mathsf{E}) : \\ \mathsf{E} := \{\mathsf{E}_{\mathsf{pc}}, \mathsf{E}_{\mathsf{inst}}, \mathsf{E}_A, \dots\} \\ \forall\, (\mathsf{t}, j) \in \{\mathsf{pc}, \mathsf{inst}, A, \dots\} \times [k-1] : (\mathsf{E}_{\mathsf{t}})_{\ell,j} = (\mathsf{E}_{\mathsf{t}})_{1,j+1} \end{array} \right\}$$

Generally, these type of statements can be proven by showing that committed matrices have entries that are shifted, which can be done similarly to a same-product relation[4]. Let $\boldsymbol{a}, \boldsymbol{b}$ be the two vectors that we want to check have shifted entries. The idea is for the verifier to pick a challenge $y \leftarrow \mathbb{F}$ and let the prover weight entries of $\boldsymbol{a}$ and $\boldsymbol{b}$ with shifted powers of y, e.g. $(1, y, y^2, \ldots,)$ and $(y, y^2, y^3, \ldots)$. Then, a same-product proof can be used to check that product of the weighted entries in the vectors are the same. We omit a description of the proof system $(\mathcal{G}_{\mathsf{ILC}}, \mathcal{P}_{\text{format}}, \mathcal{V}_{\text{format}})$ and refer to [BCG+18] for the description of a proof for general shifts of committed matrices. The proof system achieves the same features of the same-product proof: perfect completeness, statistical strong knowledge soundness with straight-line extraction and perfect SHVZK. We remark that this statement involves only two vectors for each sub-matrix of E, thus a constant number of vectors in total. The computational and communication costs for the relation $\mathcal{R}_{\mathsf{format}}$ are stated in Table 5.8

Prover computation	$T_{\mathcal{P}_{\mathsf{ILC}}}$	$= \mathcal{O}(k)\ \mathbb{F}^{\times}$
Verifier computation	$T_{\mathcal{V}_{\mathsf{ILC}}}$	$= \mathcal{O}(k)\ \mathbb{F}^{\times}$
Prover communication	t	$= \mathcal{O}(1)\ \log \lvert\mathbb{F}^k\rvert$
Verifier communication	C_{ILC}	$= \mathcal{O}(1)\ \log \lvert\mathbb{F}\rvert$
Query complexity	qc	$= 3$
Round complexity	μ	$= \mathcal{O}(1)$

TABLE 5.8: Efficiency of the proof of knowledge for the $\mathcal{R}_{\mathsf{format}}$ relation. $\mathbb{F}^{\times}$ stands for the cost of a single field multiplication, and $\log |\mathbb{F}|$ for the size of a field element.

5.5.2 Proof for the Correct TinyRAM Execution in the ILC Model

We are now in the position to describe the proof system for the correct execution of a TinyRAM program over the ILC. Given the execution trace of the program, the prover computes the extended witness w, arranges it into matrices $\mathsf{E}, \mathsf{P}, \mathsf{M}, \mathsf{R}, \mathsf{S}$ and commits to them. Then she performs the following proofs

- A proof for the correct formatting of the matrices in E storing the execution table Exe, i.e. for the relation $\mathcal{R}_{\mathsf{format}}$.
- Five equality proofs to check that public entries in the committed matrices $\mathsf{E}, \mathsf{P}, \mathsf{M}, \mathsf{R}, \mathsf{S}$ are correct as described in the relation $\mathcal{R}_{\text{check}}$.

[4] To be more precise, the same-product relation proof of [BCG+17] is based on a proof for shift relations of matrices.

- A proof for an unknown permutation of collections of four sub-matrices of E as described by the relation $\mathcal{R}_{\text{cycle}}$. In this proof we let A, B be the matrices obtained by stacking the following matrices.

$$A := \{\mathsf{E}_{\tau_{\text{link}}}, \mathsf{E}_{\mathsf{addr}}, \mathsf{E}_{\mathsf{v}_{\text{link}}}, \mathsf{E}_{\mathsf{S}} + \mathsf{E}_{\mathsf{L}}\} \qquad B := \{\mathsf{E}_{t}, \mathsf{E}_{\mathsf{addr}}, \mathsf{E}_{\mathsf{v}_{\mathsf{addr}}}, \mathsf{E}_{\mathsf{S}} + \mathsf{E}_{\mathsf{L}}\}$$

- A bounded lookup proof for the relation $\mathcal{R}_{\text{blookup}}$. Namely, to check that entries in E are contained in the lookup table formed by $\mathsf{M}^{\top}$ and $\mathsf{M}^{\perp}$, with entries in $\mathsf{M}^{\top}$ restricted to at most one access. In this proof we let E_{M} be the matrix obtained by stacking the following matrices.

$$\mathsf{E}_{\mathsf{M}} := \{\mathsf{E}_{\mathsf{addr}}, \mathsf{E}_{\mathsf{v}_{\text{init}}}, \mathsf{E}_{\mathsf{usd}}\}$$

- A lookup proof for entries in E with respect to the lookup table P. This proof checks the lookup conditions enclosed in the relation $\mathcal{R}_{\text{cons}}$. In this proof we let E_{P} be the matrix obtained by stacking the following matrices.

$$\mathsf{E}_{\mathsf{P}} := \{\mathsf{E}_{\mathsf{pc}}, \mathsf{E}_{\mathsf{inst}}, \mathsf{E}_{A}, \mathsf{E}_{\mathsf{S}}, \mathsf{E}_{\mathsf{L}}, \mathsf{E}_{\boldsymbol{s}_{\mathrm{a}}}, \mathsf{E}_{\boldsymbol{s}_{\mathrm{b}}}, \mathsf{E}_{\boldsymbol{s}_{\mathrm{c}}}, \mathsf{E}_{\boldsymbol{s}_{\mathrm{d}}}, \mathsf{E}_{\boldsymbol{s}_{\text{out}}}, \mathsf{E}_{\boldsymbol{s}_{\text{ch}}}\}$$

- A lookup proof for entries in E with respect to the lookup table R. This proof checks the lookup conditions enclosed in the relations $\mathcal{R}_{\text{time}}, \mathcal{R}_{\text{logic}}, \mathcal{R}_{\text{arith}}, \mathcal{R}_{\text{shift}}$. In this proof we let E_{R} be the matrix obtained by stacking all the sub-matrices of E on involved in range checks, i.e.

$$\mathsf{E}_{\mathsf{R}} := \{\mathsf{E}_{\mathsf{t}_{\mathsf{o}}}, \mathsf{E}_{\mathsf{t}_{\mathsf{e}}}, \mathsf{E}_{\mathsf{a}_{\mathsf{o}}}, \mathsf{E}_{\mathsf{a}_{\mathsf{e}}}, \mathsf{E}_{\mathsf{b}_{\mathsf{o}}}, \mathsf{E}_{\mathsf{b}_{\mathsf{e}}}, \mathsf{E}_{\mathsf{c}_{\mathsf{o}}}, \mathsf{E}_{\mathsf{c}_{\mathsf{e}}}, \mathsf{E}_{\mathsf{d}_{\mathsf{o}}}, \mathsf{E}_{\mathsf{d}_{\mathsf{e}}}, \mathsf{E}_{\mathsf{o}_{\mathsf{o}}}, \mathsf{E}_{\mathsf{o}_{\mathsf{e}}}, \mathsf{E}_{\mathsf{e}_{\mathsf{o}}}, \mathsf{E}_{\mathsf{e}_{\mathsf{e}}} \ldots\}$$

- A lookup proof for entries in E with respect to the lookup table S. This proof checks the lookup conditions enclosed in the relation $\mathcal{R}_{\text{shift}}$. In this proof we let E_{S} be the matrix obtained by stacking all the sub-matrices of E on involved in lookups in table Pow, i.e.

$$\mathsf{E}_{\mathsf{S}} := \{\mathsf{E}_{\mathrm{a}}, \mathsf{E}_{\mathrm{a}_{\text{power}}}\}$$

- A proof for the validity of $(\ell-1)k$ assignments in E for the same quadratic arithmetic program QAP. We let QAP check all the arithmetic constraints enclosed in the relations $\mathcal{R}_{\text{load}}, \mathcal{R}_{\text{time}}, \mathcal{R}_{\text{mux}}, \mathcal{R}_{\text{inst}}, \mathcal{R}_{\text{cons}}, \mathcal{R}_{\text{logic}}, \mathcal{R}_{\text{arith}}, \mathcal{R}_{\text{shift}}$.

The prover and verifier for the relation $\mathcal{R}^{\mathsf{field}}_{\text{TinyRAM}}$ are given in Figure 5.9.

$\mathcal{P}_{\text{TinyRAM}}(pp_{\mathsf{ILC}}, (P, v, T, M), \mathsf{w})$	$\mathcal{V}_{\text{TinyRAM}}(pp_{\mathsf{ILC}}, (P, v, T, M))$
• $\mathsf{w} := (\mathsf{Exe}, \mathsf{Prog}, \mathsf{Mem}, \mathsf{EvenBits}, \mathsf{Pow}, \pi)$ • Parse Exe, Prog, Mem, EvenBits, Pow as E, P, M, R, S ILC $\leftarrow$ • Send (**commit**, E, P, M, R, S) to the ILC • Run $\mathcal{P}_{\text{check}}(pp_{\mathsf{ILC}}, (P, v, T, M), (\mathsf{E}, \mathsf{P}, \mathsf{M}, \mathsf{R}, \mathsf{S}))$ • Run $\mathcal{P}_{\text{format}}(pp_{\mathsf{ILC}}, \perp, \mathsf{E})$ • Run $\mathcal{P}_{\text{perm}}(pp_{\mathsf{ILC}}, (\ell, 4), (A, B, \pi))$ • Run $\mathcal{P}_{\text{blookup}}(pp_{\mathsf{ILC}}, (3\ell, \frac{6M}{k}), (\mathsf{E_M}, \{\mathsf{M}^{\top}, \mathsf{M}^{\perp}\}))$ • Run $\mathcal{P}_{\text{lookup}}(pp_{\mathsf{ILC}}, (\mathcal{O}(\ell), \mathcal{O}(\frac{L}{k})), (\mathsf{E_P}, \mathsf{P}))$ • Run $\mathcal{P}_{\text{lookup}}(pp_{\mathsf{ILC}}, (\mathcal{O}(\ell), \mathcal{O}\left(\frac{\lvert\mathsf{EvenBits}\rvert}{k}\right)), (\mathsf{E_R}, \mathsf{R}))$ • Run $\mathcal{P}_{\text{lookup}}(pp_{\mathsf{ILC}}, (\ell, \mathcal{O}(1)), (\mathsf{E_S}, \mathsf{S}))$ • Run $\mathcal{P}_{\mathsf{QAP}}(pp_{\mathsf{ILC}}, \mathsf{QAP}, \mathsf{E})$	ILC $\rightarrow$ ∘ Get message $\mathcal{O}(\ell)$ from the ILC • Run $\mathcal{V}_{\text{check}}(pp_{\mathsf{ILC}}, ((P, v, T, M), [\mathsf{E}], [\mathsf{P}], [\mathsf{M}], [\mathsf{R}], [\mathsf{S}]))$ • Run $\mathcal{V}_{\text{format}}(pp_{\mathsf{ILC}}, \perp, [\mathsf{E}])$ • Run $\mathcal{V}_{\text{perm}}(pp_{\mathsf{ILC}}, (\ell, 4, [A], [B]))$ • Run $\mathcal{V}_{\text{blookup}}(pp_{\mathsf{ILC}}, (3\ell, \frac{6M}{k}, [\mathsf{E_M}], \{[\mathsf{M}^{\top}], [\mathsf{M}^{\perp}]\}))$ • Run $\mathcal{V}_{\text{lookup}}(pp_{\mathsf{ILC}}, (\mathcal{O}(\ell), \mathcal{O}\left(\frac{L}{k}\right), [\mathsf{E_P}], [\mathsf{P}]))$ • Run $\mathcal{V}_{\text{lookup}}(pp_{\mathsf{ILC}}, (\mathcal{O}(\ell), \mathcal{O}\left(\frac{\lvert\mathsf{EvenBits}\rvert}{k}\right), [\mathsf{E_R}], [\mathsf{R}]))$ • Run $\mathcal{V}_{\text{lookup}}(pp_{\mathsf{ILC}}, (\ell, \mathcal{O}(1), [\mathsf{E_S}], [\mathsf{S}]))$ • Run $\mathcal{V}_{\mathsf{QAP}}(pp_{\mathsf{ILC}}, \mathsf{QAP}, \mathsf{E})$ • If all sub-proofs accept return 1, else return 0

FIGURE 5.9: Proof of knowledge for the relation $\mathcal{R}^{\mathsf{field}}_{\text{TinyRAM}}$ in the ILC model

Theorem 5.5. *$(\mathcal{G}_{\mathsf{ILC}}, \mathcal{P}_{\text{TinyRAM}}, \mathcal{V}_{\text{TinyRAM}})$ is a proof system for $\mathcal{R}^{\mathsf{field}}_{\text{TinyRAM}}$ over the* ILC *channel with perfect completeness, statistical strong knowledge soundness with straight-line extraction, and perfect special honest-verifier zero-knowledge.*

The properties of the proof system follow from the properties achieved by each of the building-blocks used in the construction.

Efficiency. To model feasible computation, we allow the TinyRAM program to run in a polynomial number of steps $T = \text{poly}(\lambda)$ and use a polynomial number of words of memory $M = \text{poly}(\lambda)$. Assuming $T, M \geq \lambda$, this means $\log T = \Theta(\log \lambda)$ and $\log M = \Theta(\log \lambda)$. To address all the memory used by the program we therefore need the word size to be at least $W = \Omega(\log \lambda)$. To keep the circuit size of a processor modest, it is reasonable to keep the word size low, so we will assume $W = \Theta(\log \lambda)$. Our proof system also works for larger word sizes but it is less efficient when the word

size is superlogarithmic. Note that we can store register values in memory at the cost of a constant factor overhead in computation and thus, without loss of generality, we assume $K = \mathcal{O}(1)$. Our proof system is designed for a setting where the running time is large, so we will assume $T \gg L + M$.

Let k the vector length of the ILC channel and let $T = \ell k$, so that the execution table is arranged in ℓ rows of length k. Although here our main goal is to optimise prover complexity, the optimal verification complexity is achieved when $k \approx \ell \approx \sqrt{T}$. In the next chapter, we will set the vector length in this way to also optimize the communication complexity over the standard channel.

The round complexity of the proof system is equal to the maximum round complexity of the sub-proofs, which is achieved by the unknown permutation proof counting $\mathrm{loglog}(\ell) + 4$ rounds. This round complexity is attained when all the Hadamard product sub-proofs involved in the proof system are optimized for prover complexity, i.e. for matrices consisting of $\mathcal{O}(\ell)$ rows then $\ell = \mathsf{mn}$ and $\mathsf{m} = \log(\ell)$. The query complexity is constant since all the sub-proofs use a constant number of queries. The total count is $\mathsf{qc} = 298$.

Prover communication is dominated by the communication costs of the lookup proofs, which amount to $\mathcal{O}(\ell + \frac{M+L}{k}\log(\ell))$ vectors in $\mathbb{F}^k$. When all the product subproofs used in the system are optimised towards prover computation, then also the dominant computational cost for the prover is dominated asymptotically by the lookup proofs. Overall, the cost is $\mathcal{O}(\ell k + (M + L)\log(\ell k)) = \mathcal{O}(T + (M + L)\log(T))$ field multiplications. When the running time of the program dominates the both the memory and the program length, then the computational cost for the prover is dominated by a liner number ($\mathcal{O}(T)$) of field multiplications.

The verifier communication is proportional to the round complexity of the system and counts $\mathcal{O}(\mathrm{loglog}(\ell))$ field elements. The verifier complexity is dominated by $\mathcal{O}(L+|v|)$ field additions, to check the encoding of the program table and initialisation of the memory, and $\mathcal{O}(\ell + k)$ field multiplications, to verify all the remaining sub-proofs. Therefore, when $k = \ell = \sqrt{T}$, the verification cost is a sublinear number of field multiplications, assuming that $M, L < \sqrt{T}$.

To give a more concrete figure on the computational overhead paid by the prover, we can measure the above performances in terms of TinyRAM operations, using a

TinyRAM machine with the same word size as used by the program in the instance. To get negligible knowledge error we need the field to have superpolynomial size $|\mathbb{F}| = \lambda^{\omega(1)}$. This means we need a superconstant ratio $e = \frac{\log|\mathbb{F}|}{W} = \omega(1)$. On a TinyRAM machine, field elements require e words to store and using school book arithmetic field operations can be implemented in $\alpha = \mathcal{O}(e^2)$. Therefore, the running time of the prover implemented in a TinyRAM program would terminate within $\mathcal{O}(\alpha T)$ steps, where α is an arbitrarily small superconstant function $\alpha = \omega(1)$.

The efficiency of the proof system for the execution of TinyRAM programs is given in Table 5.9

Prover computation	$T_{\mathcal{P}_{\mathsf{ILC}}}$	$= \mathcal{O}(T)\ \mathbb{F}^{\times}$
Verifier computation	$T_{\mathcal{V}_{\mathsf{ILC}}}$	$= \mathcal{O}(\sqrt{T})\ \mathbb{F}^{\times}$
Prover communication	t	$= \mathcal{O}(\sqrt{T})\ \log\|\mathbb{F}^k\|$
Verifier communication	C_{ILC}	$= \mathcal{O}(\mathrm{loglog}(T))\ \log\|\mathbb{F}\|$
Query complexity	qc	$= 298$
Round complexity	μ	$= \mathcal{O}(\mathrm{loglog}T)$

TABLE 5.9: Efficiency of the proof of knowledge for $\mathcal{R}^{\mathsf{field}}_{\mathrm{TinyRAM}}$ in the ILC model. $\mathbb{F}^{\times}$ is the cost of a single field multiplication and $\log|\mathbb{F}|$ the size of a field element. The efficiency is reported in table is for $\ell = \sqrt{T}$ and assuming that program length and memory are $L, M = \mathcal{O}(\sqrt{T})$.

Chapter 6

Compiling ILC Proofs into Standard Proofs and Arguments

In this chapter we show how to compile proofs of knowledge over the ILC channel into proofs of knowledge over the standard channel. The compiler uses a linear error correcting code and a commitment scheme to realise the functionality of the ILC channel over the standard channel. If the commitment scheme is statistically binding, the compiled proof has statistical strong knowledge soundness. If the commitment scheme is computationally binding, the compiled proof has computational strong knowledge soundness.

In the compiled proof, the prover has to open linear combinations of some encoded vectors. To preserve zero knowledge we need to ensure that these openings do not reveal any information about the witness. We achieve this by introducing a new primitive that we call exposure-resilient encoding and apply it to the codewords before revealing the openings.

We then apply the transformation to our proofs of Chapters 4 and 5 and we obtain efficient proofs for the satisfiability of arithmetic circuits and the for the correct execution of TinyRAM programs in the standard channel. We instantiate both proofs with linear-time error correcting codes and linear-time commitments. In the case of circuit satisfiability, we obtain the first proof system achieving constant overhead for the prover, optimal verification time and sublinear size proofs. In the case of TinyRAM execution, we achieve the first proof system with (arbitrarily small) superconstant overhead, sublinear verification time and sublinear size proofs.

6.1 Exposure-Resilient Encodings

In the next section we encode messages, using a linear code, and then commit to them. Later, we need to open parts of committed codewords to enable checks on relations between encoded messages. At the same time, our aim is to perform these checks without revealing anything about the encoded messages.

We achieve this by defining the notion of exposure-resilient encoding. Informally, an exposure-resilient encoding of a function $f(x)$ is a randomised function $\tilde{f}(x;r)$ such that, given $\tilde{f}(x;r)$ it is possible to re-compute $f(x)$, while revealing parts of $\tilde{f}(x;r)$ should perfectly hide $f(x)$.

This primitive has similarities with other existing primitives. Exposure-resilient functions introduced by Canetti et al. [CDH+00] are functions which are fed with random inputs. Differently from our encodings, the entire output should look random, even when part of the input is revealed. Applebaum et al. [AIK06] define randomised encodings, for which the encoded output does not reveal anything about the input other than the output of the function itself. On the one hand, we relax their definition by only revealing part of the encoded output, while on the other, we strengthen it by requiring to hide any information about the input, as well as the unrevealed parts of the output. In [ISV+13], Ishai et al. introduced zero-knowledge codes, which achieve a similar goal to our exposure-resilient codes. Their notion is used to describe codes that natively achieve a good degree of exposure-resilience as described above. Differently, we start from a code with nice properties, such as large minimum distance, and we apply an encoding to make it exposure-resilient, while preserving its original properties. As it turns out, a good encoding does not necessarily make a good code. Thus, by following our formalisation, it will be easier to argue about the properties of the resulting exposure-resilient encoding.

For $\boldsymbol{v} \in \mathbb{F}^n$ and a set $J = \{j_1, \ldots, j_k\} \subset [n]$ with $j_1 < \cdots < j_k$ we recall that the notation $\boldsymbol{v}|_J$ stands for the vector $(\boldsymbol{v}_{j_1}, \ldots, \boldsymbol{v}_{j_k})$. Similarly, for a matrix $V \in \mathbb{F}^{m \times n}$ we write $V|_J \in \mathbb{F}^{m \times k}$ for the sub-matrix of V restricted to the columns indicated in J.

Let $\boldsymbol{c} \in \mathbb{F}_q^n$, and $J \subseteq [n]$, we recall that the notation $\boldsymbol{c}|_J$ indicates the projection of $\boldsymbol{c}$ onto the coordinates in J.

Definition 6.1 (Exposure-Resilient Encoding)**.** *Let k, k', n, n' be positive integers and let*

$f : \mathbb{F}^k \to \mathbb{F}^n$ be a function. For a subset $J \subseteq [n']$ we say that $\tilde{f} : \mathbb{F}^k \times \mathbb{F}^{k'} \to \mathbb{F}^{n'}$ is a J-Exposure-Resilient Encoding (ERE) of f if it is correct and exposure resilient.

- ***Correctness:*** *There exists a deterministic polynomial time algorithm D that on input $\tilde{f}(\boldsymbol{x}; \boldsymbol{r})$ computes $f(\boldsymbol{x})$.*
- ***Exposure resilience:*** *For any $\boldsymbol{x} \in \mathbb{F}^k$ the distribution $\tilde{f}(\boldsymbol{x}; \boldsymbol{r})|_J$ induced by a uniform choice of $\boldsymbol{r} \leftarrow \mathbb{F}^{k'}$ is distributed uniformly over $\mathbb{F}^{|J|}$.*

When f is the encoding function of a linear code $\mathcal{C}$, then we say that the (minimum) distance $\widetilde{\mathsf{hd}_{\min}}$ of $\tilde{f}$ is the minimum distance between $\tilde{f}(\boldsymbol{x}; \boldsymbol{r})$ and $\tilde{f}(\boldsymbol{y}; \boldsymbol{s})$ for any distinct $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{F}^k$. We now show a simple randomised construction of an exposure-resilient encoding of a linear error-correcting code which preserves its minimum distance.

Theorem 6.1. *Let $\mathcal{C}$ be a $[n, k, \mathsf{hd}_{\min}]_{\mathbb{F}}$ code with associated encoding function $E_{\mathcal{C}}$. Then, the following function $\widetilde{E}_{\mathcal{C}} : \mathbb{F}^k \times \mathbb{F}^n \to \mathbb{F}^{2n}$*

$$\widetilde{E}_{\mathcal{C}}(\boldsymbol{x}; \boldsymbol{r}) := \big(E_{\mathcal{C}}(\boldsymbol{x}) + \boldsymbol{r}, \boldsymbol{r}\big)$$

is an J-ERE of $E_{\mathcal{C}}$ for any $J \subseteq [2n]$ satisfying $\{j \in J : j + n \in J\} = \emptyset$. Moreover, the minimum distance of $\widetilde{E}_{\mathcal{C}}$ is $\widetilde{\mathsf{hd}_{\min}} = \mathsf{hd}_{\min}$.

Proof. The above construction is clearly correct as it is possible to efficiently re-compute $E_{\mathcal{C}}(\boldsymbol{x})$ from $\widetilde{E}_{\mathcal{C}}(\boldsymbol{x}; \boldsymbol{r}) = (E_{\mathcal{C}}(\boldsymbol{x}) + \boldsymbol{r}, \boldsymbol{r})$.

The encoding is also exposure resilient. Let $J_1 = J \cap [n]$ and $J_2 = J \setminus J_1$. Then $\widetilde{E}_{\mathcal{C}}(\boldsymbol{x}; \boldsymbol{r})|_{J_1} = E_{\mathcal{C}}(\boldsymbol{x})|_{J_1} + \boldsymbol{r}|_{J_1} = E_{\mathcal{C}}(\boldsymbol{x})|_{J_1} + \widetilde{E}_{\mathcal{C}}(\boldsymbol{x}; \boldsymbol{r})|_{J_1+n}$. Since $\{j \in J : i+n \in J\} = \emptyset$, no $j \in J_1$ can satisfy $j + n \in J_2$. Given $\boldsymbol{r}|_{J_1}$ uniformly distributed on $\mathbb{F}^{|J_1|}$ independently from $\boldsymbol{r}|_{J_2}$, then $\widetilde{E}_{\mathcal{C}}(\boldsymbol{x}; \boldsymbol{r})|_{J_1}$ is also distributed uniformly and independently from $\widetilde{E}_{\mathcal{C}}(\boldsymbol{x}; \boldsymbol{r})|_{J_2}$. By construction, $\widetilde{E}_{\mathcal{C}}(\boldsymbol{x}; \boldsymbol{r})|_{J_2}$ is uniformly distributed, so $\widetilde{E}_{\mathcal{C}}(\boldsymbol{x}; \boldsymbol{r})|_J$ is uniformly distributed.

By the linearity of $\widetilde{E}_{\mathcal{C}}$, the minimum distance of $\widetilde{E}_{\mathcal{C}}$ it is equal to the minimum Hamming weight of $\widetilde{E}_{\mathcal{C}}(\boldsymbol{x}; \boldsymbol{r})$ for $\boldsymbol{x} \neq \boldsymbol{0}$. Note that for any $j \in [n]$ for which $\boldsymbol{r}|_{\{j\}}$ is non-zero, at least one of $\widetilde{E}_{\mathcal{C}}(\boldsymbol{x}; \boldsymbol{r})|_{\{j\}}$ and $\widetilde{E}_{\mathcal{C}}(\boldsymbol{x}; \boldsymbol{r})|_{\{j+n\}}$ is non-zero. Therefore, for any

$\boldsymbol{x} \neq \mathbf{0}$ and any $\boldsymbol{r}$, the following holds

$$\mathsf{wt}(\widetilde{E}_{\mathcal{C}}(\boldsymbol{x};\boldsymbol{r})) \geq \mathsf{wt}(\widetilde{E}_{\mathcal{C}}(\boldsymbol{x};\mathbf{0})) = \mathsf{wt}(E_{\mathcal{C}}(\boldsymbol{x})) \geq \mathsf{hd}_{\min}$$

□

We note that if the original code $\mathcal{C}$ admits a linear time encoding function $E_{\mathcal{C}}$, then $\widetilde{E}_{\mathcal{C}}$ is also computable in linear time.

Corollary 6.1. *Let $\widetilde{E}_{\mathcal{C}}$ be a J-ERE constructed as in Theorem 6.1 for $J \subseteq [2n]$ satisfying $\{j \in J : i + n \in J\} = \emptyset$. There exists a linear time simulator $\mathcal{S}$ that given as input $\boldsymbol{x} \in \mathbb{F}^k$, and $\boldsymbol{y} \in \mathbb{F}^{|J|}$ returns $\boldsymbol{r} \in \mathbb{F}^n$ such that the output distribution of $\boldsymbol{r} \leftarrow \mathcal{S}(\boldsymbol{x}, \boldsymbol{y})$ is uniform in $\mathbb{F}^n$, conditioned on $\widetilde{E}_{\mathcal{C}}(\boldsymbol{x};\boldsymbol{r})|_J = \boldsymbol{y}$.*

Proof. Let $J_1 := J \cap [n]$ and $J_2 := J \backslash J_1$. The simulator $\mathcal{S}(\boldsymbol{x}, \boldsymbol{y})$ simply sets $\boldsymbol{r}|_{J_1} := \boldsymbol{y}|_{J_1} - E_{\mathcal{C}}(\boldsymbol{x})|_{J_1}$ and $\boldsymbol{r}|_{j_2-n} := \boldsymbol{y}|_{j_2}$ for all $j_2 \in J_2$ and then picks the $\boldsymbol{r}_j$s for the remaining $n - |J|$ values of j uniformly and independently from $\mathbb{F}$. □

The above notion of exposure-resilience is also related to the privacy property of secret sharing schemes. It is well known that linear secret sharing schemes can be constructed from linear codes [Mas95; CCG+07; CDD+15]. In the next sections, we could just as well replace the use of our ERE with a linear-time linear secret sharing scheme. In fact, our ERE can even be seen as a 1-private linear secret sharing scheme. Even though it does not make a good secret sharing scheme, it turns out to be enough for our purposes. Moreover, our ERE results in a more efficient instantiation than using known linear-time linear secret sharing schemes [DI14; CDD+15], which use linear universal hash functions.

6.2 From the ILC Channel to the Standard Channel

In this section, we present our compiler to realise the ILC channel over standard communication channel. Recall that in the ILC channel the prover commit to vectors of length k by submitting them to the channel and the verifier can then query the ILC to open linear combinations of the committed vectors.

The idea behind the compilation of an ILC proof is that, instead of committing to vectors $V = \{\boldsymbol{v}_i\}_{i=1}^t$ using the ILC, the prover first encodes each vector $\boldsymbol{v}_i$ in V as $E_{\mathcal{C}}(\boldsymbol{v}_i)$ using a linear error-correcting code $E_{\mathcal{C}}$. In any given round, we can think of the codewords as rows $E_{\mathcal{C}}(\boldsymbol{v}_i)$ in a matrix $E_{\mathcal{C}}(V)$. Then, instead of committing to the rows of the matrix, the prover commits to the columns of $E_{\mathcal{C}}(V)$ using a non-interactive commitment scheme and sends the commitments to the verifier. When the verifier wants to open a linear combination of the original vectors, he sends the coefficients $\boldsymbol{q} = (q_1, \ldots, q_t)$ to the prover, who responds with the linear combination $\boldsymbol{v}' = \boldsymbol{q}V$. To ensure the prover is not returning a wrong response $\boldsymbol{v}'$, the verifier performs spot checks on the encoded response $E_{\mathcal{C}}(\boldsymbol{v}')$ and the committed codewords in the rows $E_{\mathcal{C}}(V)$. The verifier may request a random j-th entry of each committed codeword $E_{\mathcal{C}}(\boldsymbol{v}_i)$, corresponding to the j-th column of the error-corrected matrix $E_{\mathcal{C}}(V)$. Since the code $E_{\mathcal{C}}$ is linear, the revealed elements should satisfy

$$E_{\mathcal{C}}(\boldsymbol{v}')|_j = \sum_{i=1}^{t} q_i E_{\mathcal{C}}(\boldsymbol{v}_i)|_j = \boldsymbol{q}E_{\mathcal{C}}(V)|_j$$

The verifier repeats this check on multiple columns, so that if the code has sufficiently large minimum distance and the prover gives a wrong response $\boldsymbol{v}'$, then he has high probability to catch at least one column j where the above equality does not hold.

Revealing entries in a codeword may leak information about the encoded vector. To get SHVZK, instead of directly committing to the codewords of $E_{\mathcal{C}}$, we commit to the output of the exposure-resilient encoding $\widetilde{E}_{\mathcal{C}}$ we constructed in Theorem 6.1. This doubles the length of the rows of the committed matrix but ensures that then nothing is revealed about the encoded messages, as long as the prover only opens a set J for which $\widetilde{E}_{\mathcal{C}}$ is a J-ERE. The spot checking technique using $\widetilde{E}_{\mathcal{C}}$ is illustrated in Figure 6.1. In the following, we write $\boldsymbol{e}_i$ for the encoding of vector $\boldsymbol{v}_i$ and E for the encoding of the matrix V using the ERE, i.e.

$$\boldsymbol{e}_i := \widetilde{E}_{\mathcal{C}}(\boldsymbol{v}_i; \boldsymbol{r}_i) = (E_{\mathcal{C}}(\boldsymbol{v}_i) + \boldsymbol{r}_i, \boldsymbol{r}_i) \qquad E := \widetilde{E}_{\mathcal{C}}(V; R) = (E_{\mathcal{C}}(V) + R, R)$$

In addition to the above, the verifier queries the prover on a random linear combination $\boldsymbol{\gamma} \in \mathbb{F}^t$ and spot checks the response as for the previous queries. This is to ensure

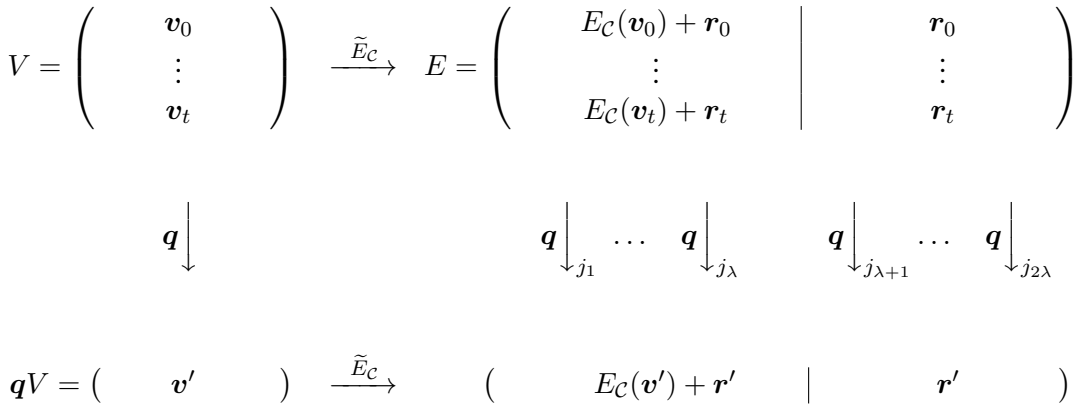

FIGURE 6.1: Vectors $\boldsymbol{v}_i$ organised in matrix V are encoded row-wise as matrix $E = \widetilde{E}_{\mathcal{C}}(V; R)$. The vertical line in the right matrix and vector denotes concatenation of matrices. The prover commits to each column of E. On query $\boldsymbol{q}$ he answers with $\boldsymbol{v}' = \boldsymbol{q}V$ and $\boldsymbol{r}' = \boldsymbol{q}R$. The verifier asks for openings to a set of columns $J = \{j_1, \ldots, j_{2\lambda}\}$ in E and checks their consistency with $\widetilde{E}_{\mathcal{C}}(\boldsymbol{v}'; \boldsymbol{r}')$.

that if the prover commits to some vectors $\boldsymbol{e}_i$ that are far from being codewords, the verifier then has high probability to detect some errors. The linear combination $\boldsymbol{q}$ queried by the ILC verifier does not necessarily suffice to ensure this, since in general the query is not chosen uniformly at random. One could, for instance, imagine that there was a vector $\boldsymbol{v}_i$ that was never queried in a non-trivial way, and hence the prover could choose it to be far from a codeword. To make sure this extra challenge $\boldsymbol{\gamma}$ does not reveal information to the verifier, the prover picks a random blinding vector $\boldsymbol{v}_0 \leftarrow \mathbb{F}^k$, which is added as the first row of V and will be added to the challenge linear combination $\boldsymbol{\gamma}$ queried by the verifier. This is similar to what is used in some of the ILC proofs in the previous chapters, e.g. the Hadamard product sub-proof of Section 4.2.3, to guarantee that the ILC responses to the opening queries made by the verifier did not reveal any information about the committed vectors.

6.2.1 The Compiler

Let $(\mathcal{G}_{\mathsf{ILC}}, \mathcal{P}_{\mathsf{ILC}}, \mathcal{V}_{\mathsf{ILC}})$ be a *non-adaptive* μ-round SHVZK proof system over the ILC for a relation $\mathcal{R}$. Recall that non-adaptive means that the verifier waits until the last round

to query linear combinations and that these are queried all at once instead of the depending on each other response.[1] All the ILC proofs presented in the previous chapters are non-adaptive. Let $\mathsf{Gen}_{E_{\mathcal{C}}}$ be a generator that, given as input the description of a finite field $\mathbb{F}$ and a length parameter k, it outputs a $[n, k, \mathsf{hd}_{\min}]_{\mathbb{F}}$ code $\mathcal{C}$ with constant rate and linear minimum distance, i.e. $n = \Theta(k)$ and $\mathsf{hd}_{\min} = \Theta(k)$. Let $\widetilde{E}_{\mathcal{C}}$ be the exposure-resilient encoding of $\mathcal{C}$ as constructed in Theorem 6.1. Finally, let $(\mathsf{CSetup}, \mathsf{CCommit})$ be a non-interactive commitment scheme.

We now give our proof system $(\mathcal{G}, \mathcal{P}, \mathcal{V})$ for the relation $\mathcal{R}$ over the standard channel. The prover starts by invoking $\mathcal{P}_{\mathsf{ILC}}$ on input the instance and the witness, to obtain vectors $\{\boldsymbol{v}_i\}_{i=1}^{t_1}$ to commit. Let these vectors be arranged in matrix V_1. The prover encodes the rows of these matrices to obtain matrix $E_1 = \widetilde{E}_{\mathcal{C}}(V_1; R_1)$. In addition to these, the prover picks a random vector $\boldsymbol{v}_0 \leftarrow \mathbb{F}^k$ and encodes it to obtain $\boldsymbol{e}_0$. The prover then commits to the columns of matrix $E_{01} := \begin{pmatrix} \boldsymbol{e}_0 \\ E_1 \end{pmatrix}$. We overload the notation and write $\boldsymbol{c}_1 \leftarrow \mathsf{CCommit}(E_{01}; \boldsymbol{s}_1)$ for the process of committing independently each column vector $E_{01}|_i$ using randomness $\boldsymbol{s}_1|_i$. The prover forwards the commitments to the verifier and terminates her move. The verifier invokes $\mathcal{V}_{\mathsf{ILC}}$ to generate a first message x_1 and forwards it to the prover.

The following rounds of the proof unfolds similarly to the first one: the prover invokes $\mathcal{P}_{\mathsf{ILC}}$ to produce a matrix $V_i \in \mathbb{F}^{t_i \times k}$, she encodes its rows and then commits to the columns of the encoded matrix E_i. Upon receipt of the commitments, the verifier replies with a challenge produced by the internal $\mathcal{V}_{\mathsf{ILC}}$. Given matrices $V_1, \ldots, V_\mu$, $R_1, \ldots, R_\mu$ and $E_1, \ldots, E_\mu$ we define

$$V = \begin{pmatrix} V_1 \\ \vdots \\ V_\mu \end{pmatrix} \qquad R = \begin{pmatrix} R_1 \\ \vdots \\ R_\mu \end{pmatrix} \qquad E = \begin{pmatrix} E_1 \\ \vdots \\ E_\mu \end{pmatrix}$$

In the last move of the ILC proof system the verifier produces a query $Q \in \mathbb{F}^{\mathsf{qc} \times t}$ for the ILC. The verifier forwards the query to the prover, together with a random challenge $\gamma \leftarrow \mathbb{F}^t$, where t is the number of rows in V, R and E. The prover answers

[1] The construction can be easily modified to an adaptive ILC proof. For each round of queries in the ILC proof, there will one extra round in the compiled proof.

the queries $\boldsymbol{\gamma}, Q$ by computing the following linear combinations

$$\boldsymbol{v}' = \boldsymbol{v}_0 + \boldsymbol{\gamma} V \qquad \boldsymbol{r}' = \boldsymbol{r}_0 + \boldsymbol{\gamma} R \qquad V' = QV \qquad R' = QR$$

Lastly, the verifier picks at random a set $J \subseteq [2n]$ of 2λ columns such that $\{j \in J : j + n \in J\} = \emptyset$ and $|J \cap [n]| = \lambda$. We refer to a set with these properties as *admissible*. The verifier challenges the prover to open the commitments indexed by J sent in every round of the proof. The verifier checks all the openings to the commitments and then checks the consistency of these with the J-entries of the encodings of $(\boldsymbol{v}', \boldsymbol{r}')$ and (V', R'), i.e.

$$\widetilde{E}_{\mathcal{C}}(\boldsymbol{v}'; \boldsymbol{r}')|_J = \boldsymbol{e}_0|_J + \boldsymbol{\gamma} E|_J \qquad \widetilde{E}_{\mathcal{C}}(V'; R')|_J = QE|_J$$

If all the checks pass, the verifier returns the response of the internal $\mathcal{V}_{\mathsf{ILC}}$ on input the opening query Q, otherwise he returns 0. The full description of the compiled proof system $(\mathcal{G}, \mathcal{P}, \mathcal{V})$ is given in Figure 6.2

6.2.2 Security Analysis

Here we show that if $(\mathcal{G}_{\mathsf{ILC}}, \mathcal{P}_{\mathsf{ILC}}, \mathcal{V}_{\mathsf{ILC}})$ is a proof of knowledge with straight-line extraction for the relation $\mathcal{R}$ over the ILC, then the compiled proof $(\mathcal{G}, \mathcal{P}, \mathcal{V})$ of Figure 6.2 is a proof of knowledge *without* straight-line extraction for the same relation over the standard channel. Assuming the commitment scheme is statistically binding and computationally hiding the compiled proof achieves statistical strong knowledge soundness and computational SHVZK. If the commitment scheme is computationally binding and statistically hiding, the output of the compilation is an argument of knowledge with computational strong knowledge soundness and statistical SHVZK.

Theorem 6.2 (Completeness)**.** *If the proof system $(\mathcal{G}_{\mathsf{ILC}}, \mathcal{P}_{\mathsf{ILC}}, \mathcal{V}_{\mathsf{ILC}})$ for the relation $\mathcal{R}$ over the* ILC *is perfectly complete, then the proof system $(\mathcal{G}, \mathcal{P}, \mathcal{V})$ in Figure 6.2 is perfectly complete for the same relation $\mathcal{R}$.*

Proof. Given the correct openings to the commitments indexed by J, the verifier's spot checks on succeed with probability 1. Since the error correcting code $\mathcal{C}$ is linear, then the exposure-resilient encoding of Theorem 6.1 is also linear. By the linearity of the

$\mathcal{P}(pp, u, w)$

- **Parse input**:
  - Parse $pp = (pp_{\mathsf{ILC}}, E_{\mathcal{C}}, ck)$
  - Parse $pp_{\mathsf{ILC}} = (\mathbb{F}, k, *)$
  - Get n from $E_{\mathcal{C}}$
- **Round** 1:
  - $\boldsymbol{v}_0, \boldsymbol{r}_0 \leftarrow \mathbb{F}^k$
  - $\boldsymbol{e}_0 \leftarrow \widetilde{E}_{\mathcal{C}}(\boldsymbol{v}_0; \boldsymbol{r}_0)$
  - $(\textbf{commit}, V_1) \leftarrow \mathcal{P}_{\mathsf{ILC}}(pp_{\mathsf{ILC}}, u, w)$
  - $R_1 \leftarrow \mathbb{F}^{t_1 \times k}$
  - $E_1 \leftarrow \widetilde{E}_{\mathcal{C}}(V_1; R_1)$
  - $E_{01} := \begin{pmatrix} \boldsymbol{e}_0 \\ E_1 \end{pmatrix}$
  - $\boldsymbol{c}_1 \leftarrow \mathsf{CCommit}(E_{01}; \boldsymbol{s}_1)$
  - Send $(\boldsymbol{c}_1, t_1)$ to the $\mathcal{V}$
- **Rounds** $2 \leq i \leq \mu$:
  - Get challenge x_{i-1} from the $\mathcal{V}$
  - $(\textbf{commit}, V_i) \leftarrow \mathcal{P}_{\mathsf{ILC}}(x_{i-1})$
  - $R_i \leftarrow \mathbb{F}^{t_i \times k}$
  - $E_i \leftarrow \widetilde{E}_{\mathcal{C}}(V_i; R_i)$
  - $\boldsymbol{c}_i \leftarrow \mathsf{CCommit}(E_i; \boldsymbol{s}_i)$
  - Send $(\boldsymbol{c}_i, t_i)$ to the $\mathcal{V}$
- **Round** $\mu + 1$:
  - Get $(\boldsymbol{\gamma}, Q)$ from the $\mathcal{V}$
  - $\boldsymbol{v}' \leftarrow \boldsymbol{v}_0 + \boldsymbol{\gamma} V$
  - $\boldsymbol{r}' \leftarrow \boldsymbol{r}_0 + \boldsymbol{\gamma} R$
  - $V' \leftarrow QV$
  - $R' \leftarrow QR$
  - Send $(\boldsymbol{v}', \boldsymbol{r}', V', R')$ to the $\mathcal{V}$
- **Round** $\mu + 2$:
  - Get J from the $\mathcal{V}$
  - Send $(E_{01}|_J, \boldsymbol{s}_1|_J, \ldots, E_\mu, \boldsymbol{s}_\mu|_J)$ to $\mathcal{V}$

$\mathcal{G}(1^\lambda)$

- $pp_{\mathsf{ILC}} \leftarrow \mathcal{G}_{\mathsf{ILC}}(1^\lambda)$
- Parse $pp_{\mathsf{ILC}} := (\mathbb{F}, k, *)$
- $E_{\mathcal{C}} \leftarrow \mathsf{Gen}_{E_{\mathcal{C}}}(\mathbb{F}, k)$
- $ck \leftarrow \mathsf{CSetup}(1^\lambda)$
- Return $pp = (pp_{\mathsf{ILC}}, E_{\mathcal{C}}, ck)$

$\mathcal{V}(pp, u)$

- **Parse input**
  - Parse $pp = (pp_{\mathsf{ILC}}, E_{\mathcal{C}}, ck)$
  - Parse $pp_{\mathsf{ILC}} = (\mathbb{F}, k, *)$
  - Get n from $E_{\mathcal{C}}$
  - Give input (pp_{ILC}, u) to the $\mathcal{V}_{\mathsf{ILC}}$
- **Rounds** $1 \leq i < \mu$:
  - Receive $(\boldsymbol{c}_i, t_i)$
  - $(\textbf{send}, x_i) \leftarrow \mathcal{V}_{\mathsf{ILC}}(t_i)$
  - Send x_i to the $\mathcal{P}$
- **Round** μ:
  - Get message $(\boldsymbol{c}_\mu, t_\mu)$ from the $\mathcal{P}$
  - $t := \sum_{i=1}^{\mu} t_i$
  - $\boldsymbol{\gamma} \leftarrow \mathbb{F}^t$
  - $(\textbf{open}, Q) \leftarrow \mathcal{V}_{\mathsf{ILC}}(t_\mu)$
  - Send $(\boldsymbol{\gamma}, Q)$ to the $\mathcal{P}$
- **Round** $\mu + 1$:
  - Get $(\boldsymbol{v}', \boldsymbol{r}', V', R')$ from the $\mathcal{P}$
  - Pick admissible $J \subset [2n]$
  - Send J to the $\mathcal{P}$
- **Round** $\mu + 2$:
  - Get $(E_{01}|_J, \boldsymbol{s}_1|_J, \ldots, E_\mu, \boldsymbol{s}_\mu|_J)$
  - Check:
    - $\widetilde{E}_{\mathcal{C}}(\boldsymbol{v}'; \boldsymbol{r}')|_J = \boldsymbol{e}_0|_J + \boldsymbol{\gamma} E|_J$
    - $\widetilde{E}_{\mathcal{C}}(V'; R')|_J = QE|_J$
    - $\boldsymbol{c}_1|_J = \mathsf{CCommit}(E_{01}|_J; \boldsymbol{s}_1|_J)$
    - $\ldots$
    - $\boldsymbol{c}_\mu|_J = \mathsf{CCommit}(E_\mu|_J; \boldsymbol{s}_\mu|_J)$
  - If any check fails return 0
  - Return $b \leftarrow \mathcal{V}_{\mathsf{ILC}}(V')$

FIGURE 6.2: Construction of $(\mathcal{G}, \mathcal{P}, \mathcal{V})$ from $(\mathcal{G}_{\mathsf{ILC}}, \mathcal{P}_{\mathsf{ILC}}, \mathcal{V}_{\mathsf{ILC}})$, commitment scheme $(\mathsf{CSetup}, \mathsf{CCommit})$ and error correcting code $\mathcal{C}$.

ERE then both $\widetilde{E}_{\mathcal{C}}(\boldsymbol{v}';\boldsymbol{r}') = \boldsymbol{e}_0 + \boldsymbol{\gamma}E$ and $\widetilde{E}_{\mathcal{C}}(V';R') = QE$ hold with probability 1. If $(pp, u, w) \in \mathcal{R}$ then $(pp_{\mathsf{ILC}}, u, w) \in \mathcal{R}$. By the completeness of ILC proof, the internal verifier $\mathcal{V}_{\mathsf{ILC}}$ accepts with probability 1 and so does verifier $\mathcal{V}$. □

Theorem 6.3 (Knowledge Soundness)**.** *If* $(\mathcal{G}_{\mathsf{ILC}}, \mathcal{P}_{\mathsf{ILC}}, \mathcal{V}_{\mathsf{ILC}})$ *is a proof system for the relation* $\mathcal{R}$ *over the* ILC *with statistically strong knowledge soundness with a straight-line extraction, and* $(\mathsf{CSetup}, \mathsf{CCommit})$ *is computationally (statistically) binding, then the proof system* $(\mathcal{G}, \mathcal{P}, \mathcal{V})$ *in Figure 6.2 has computational (statistical) strong knowledge soundness for the same relation* $\mathcal{R}$*.*

Proof. We prove the computational case, the statistical case follows similarly.

In order to argue that $(\mathcal{G}, \mathcal{P}, \mathcal{V})$ is computationally knowledge sound, we will first show that for every DPT $\mathcal{P}^*$ there exists a deterministic (but not necessarily efficient) $\mathcal{P}^*_{\mathsf{ILC}}$ convincing the verifier over the ILC with (almost) the same probability of $\mathcal{P}^*$ over the standard channel. Namely, that for every DPT $\mathcal{P}^*$ there exists $\mathcal{P}^*_{\mathsf{ILC}}$ such that for all PPT $\mathcal{A}$

$$\Pr\left[\begin{array}{c} pp \leftarrow \mathcal{G}(1^\lambda); pp = (pp_{\mathsf{ILC}}, *); (u, s) \leftarrow \mathcal{A}(pp); \\ (\mathsf{tran}_{\mathcal{P}^*}; b) \leftarrow \langle \mathcal{P}^*(s) \longleftrightarrow \mathcal{V}(pp, u; (\rho_{\mathsf{ILC}}, \rho))\rangle; \\ (\mathsf{tran}_{\mathcal{P}^*_{\mathsf{ILC}}}; b_{\mathsf{ILC}}) \leftarrow \langle \mathcal{P}^*_{\mathsf{ILC}}(u, pp; s) \overset{\mathsf{ILC}}{\longleftrightarrow} \mathcal{V}_{\mathsf{ILC}}(pp_{\mathsf{ILC}}, u; \rho_{\mathsf{ILC}})\rangle : \\ b = 1 \ \wedge \ b_{\mathsf{ILC}} = 0 \end{array}\right] \approx 0 \qquad (6.1)$$

where ρ_{ILC} is the randomness the verifier $\mathcal{V}$ gives in input to the internal verifier $\mathcal{V}_{\mathsf{ILC}}$.

Next, we will define an expected PPT *transcript extractor* $\mathcal{E}_{\mathsf{tran}}$ that given access to a rewindable oracle $\langle \mathcal{P}^*(u, pp; s) \longleftrightarrow \mathcal{V}(pp, u; (\rho_{\mathsf{ILC}}, \rho))\rangle$ outputs a view $\widetilde{\mathsf{tran}}_{\mathcal{P}^*_{\mathsf{ILC}}}$ following the same distribution of a real communication transcript between $\mathcal{P}^*_{\mathsf{ILC}}$ and the ILC channel in the execution of $\langle \mathcal{P}^*_{\mathsf{ILC}}(s, pp, u) \overset{\mathsf{ILC}}{\longleftrightarrow} \mathcal{V}_{\mathsf{ILC}}(pp_{\mathsf{ILC}}, u; \rho_{\mathsf{ILC}})\rangle$. Formally, we will show there exists an expected PPT $\mathcal{E}_{\mathsf{tran}}$ such that for all PPT $\mathcal{A}$

$$\Pr\left[\begin{array}{c} pp \leftarrow \mathcal{G}(1^\lambda); pp = (pp_{\mathsf{ILC}}, *); (u,s) \leftarrow \mathcal{A}(pp); \\ \widetilde{\mathsf{tran}}_{\mathcal{P}^*_{\mathsf{ILC}}} \leftarrow \mathcal{E}_{\mathsf{tran}}^{\langle \mathcal{P}^*(pp,u;s) \longleftrightarrow \mathcal{V}(pp,u)\rangle}(pp,u); \\ (\mathsf{tran}_{\mathcal{P}^*_{\mathsf{ILC}}}; b_{\mathsf{ILC}}) \leftarrow \langle \mathcal{P}^*_{\mathsf{ILC}}(pp,u;s) \overset{\mathsf{ILC}}{\longleftrightarrow} \mathcal{V}_{\mathsf{ILC}}(pp_{\mathsf{ILC}}, u; \rho_{\mathsf{ILC}})\rangle\colon \\ b = 1 \ \wedge \ \mathsf{tran}_{\mathcal{P}^*_{\mathsf{ILC}}} \neq \widetilde{\mathsf{tran}}_{\mathcal{P}^*_{\mathsf{ILC}}} \end{array}\right] \approx 0 \qquad (6.2)$$

where b is the output of $\mathcal{V}$ on the first execution of the oracle and ρ_{ILC} given to $\mathcal{V}_{\mathsf{ILC}}$ is the randomness used by the internal copy of $\mathcal{V}$ on the first execution of the oracle. Note that if (6.1) holds and $b = 1$, then with overwhelming probability $b_{\mathsf{ILC}} = 1$.

Assuming for a moment that both (6.1) and 6.2 hold we describe the knowledge extractor $\mathcal{E}$ for $(\mathcal{G}, \mathcal{P}, \mathcal{V})$ over the standard channel. The knowledge extractor $\mathcal{E}$ runs the transcript extractor $\mathcal{E}_{\mathsf{tran}}$ and answers to the oracle queries of $\mathcal{E}_{\mathsf{tran}}$ with its own oracle. Eventually, $\mathcal{E}$ obtains a transcript $\widetilde{\mathsf{tran}}_{\mathcal{P}^*_{\mathsf{ILC}}}$ from $\mathcal{E}_{\mathsf{tran}}$. Since the ILC proof system has straight-line extractability, $\mathcal{E}$ then executes the ILC knowledge extractor on input $\widetilde{\mathsf{tran}}_{\mathcal{P}^*_{\mathsf{ILC}}}$ and receives a witness $\widetilde{w} \leftarrow \mathcal{E}_{\mathsf{ILC}}(pp_{\mathsf{ILC}}, u, \widetilde{\mathsf{tran}}_{\mathcal{P}^*_{\mathsf{ILC}}})$. Assuming (6.2), the output distribution of $\mathcal{E}_{\mathsf{tran}}$ is identically distributed to a real communication transcript between $\mathcal{P}^*_{\mathsf{ILC}}$ and the ILC, and thus $\widetilde{w}$ is distributed as the output of $\mathcal{E}_{\mathsf{ILC}}$ in its own knowledge soundness game. Assuming (6.1), if $b = 1$ then with overwhelming probability also $b_{\mathsf{ILC}} = 1$. By the strong knowledge soundness of $(\mathcal{G}_{\mathsf{ILC}}, \mathcal{P}_{\mathsf{ILC}}, \mathcal{V}_{\mathsf{ILC}})$ there is a negligible probability that $(pp, u, \widetilde{w}) \notin \mathcal{R} \ \wedge \ b_{\mathsf{ILC}} = 1$, hence $(\mathcal{G}, \mathcal{P}, \mathcal{V})$ is also strong knowledge sound.

To conclude the proof it is sufficient to show the existence of $\mathcal{P}^*_{\mathsf{ILC}}$ and $\mathcal{E}_{\mathsf{tran}}$ satisfying both (6.1) and (6.2), which we show, respectively, in Lemma 6.1 and Lemma 6.2 below.

Lemma 6.1. *If the commitment scheme* $(\mathsf{CSetup}, \mathsf{CCommit})$ *is computationally binding, then for every DPT* $\mathcal{P}^*$ *there exists* $\mathcal{P}^*_{\mathsf{ILC}}$ *such that for all PPT* $\mathcal{A}$

$$\Pr\left[\begin{array}{c} pp \leftarrow \mathcal{G}(1^\lambda); pp = (pp_{\mathsf{ILC}}, *); (u,s) \leftarrow \mathcal{A}(pp); \\ (\mathsf{tran}_{\mathcal{P}^*}; b) \leftarrow \langle \mathcal{P}^*(s) \longleftrightarrow \mathcal{V}(pp, u; (\rho_{\mathsf{ILC}}, \rho))\rangle; \\ (\mathsf{tran}_{\mathcal{P}^*_{\mathsf{ILC}}}; b_{\mathsf{ILC}}) \leftarrow \langle \mathcal{P}^*_{\mathsf{ILC}}(u, pp; s) \overset{\mathsf{ILC}}{\longleftrightarrow} \mathcal{V}_{\mathsf{ILC}}(pp_{\mathsf{ILC}}, u; \rho_{\mathsf{ILC}})\rangle\colon \\ b = 1 \ \wedge \ b_{\mathsf{ILC}} = 0 \end{array}\right] \approx 0 \qquad (6.1)$$

Proof. Our constructed $\mathcal{P}^*_{\mathsf{ILC}}$ runs an internal copy of $\mathcal{P}^*$. Recall that we do not restrict $\mathcal{P}^*_{\mathsf{ILC}}$ to run in polynomial time, this is sufficient since the proof system over the ILC achieves *statistical* strong knowledge soundness. When the internal $\mathcal{P}^*$ in round i sends a message $(\boldsymbol{c}_i, t_i)$, $\mathcal{P}^*_{\mathsf{ILC}}$ simulates $\mathcal{P}^*$ on every possible continuation of the transcript, to obtain the most frequently occurring *valid* opening $((E_i)|_j, (\boldsymbol{s}_i)|_j)$ of $(\boldsymbol{c}_i)|_j$ for each $j = 1, \ldots, 2n$. $\mathcal{P}^*_{\mathsf{ILC}}$ then uses these openings to form matrices E^*_i. For each row $\boldsymbol{e}^*_\ell$ of these matrices, $\mathcal{P}^*_{\mathsf{ILC}}$ finds a vector $\boldsymbol{v}_\ell$ and randomness $\boldsymbol{r}_\ell$ such that $\mathsf{hd}(\widetilde{E}_{\mathcal{C}}(\boldsymbol{v}_\ell, \boldsymbol{r}_\ell), \boldsymbol{e}^*_\ell) < \frac{\mathsf{hd}_{\min}}{3}$ if such a vector exists. If for some $1 \leq \ell \leq t_i$ no such vector $\boldsymbol{v}_\ell$ exists, then $\mathcal{P}^*_{\mathsf{ILC}}$ aborts. Otherwise we let V_i and R_i denote the matrices formed by the row vectors $\boldsymbol{v}_\ell$ and $\boldsymbol{r}_\ell$ in round i and $\mathcal{P}^*_{\mathsf{ILC}}$ sends V_i to the ILC. Notice that by Theorem 6.1 the minimum distance of $\widetilde{E}_{\mathcal{C}}$ is at least $\mathsf{hd}_{\min}$, so there is at most one such vector $\boldsymbol{v}_\ell$ for each $\boldsymbol{e}^*_\ell$. The prover $\mathcal{P}^*_{\mathsf{ILC}}$ proceeds in the same way for all the μ rounds in which the internal prover $\mathcal{P}^*$ sends commitments to the verifier and then terminates. Notice that the $\mathcal{V}_{\mathsf{ILC}}$ communicates over the ILC with our constructed $\mathcal{P}^*_{\mathsf{ILC}}$ and at the end he queries a challenge Q to the ILC, for which he receives $V' = QV$ from the ILC.

Next, we show that if $\mathcal{P}^*$ makes the verifier $\mathcal{V}$ accept, then with overwhelming probability $\mathcal{V}_{\mathsf{ILC}}$ interacting with our $\mathcal{P}^*_{\mathsf{ILC}}$ described above will also accept. Since the verifier $\mathcal{V}$ only accepts if its internal copy of $\mathcal{V}_{\mathsf{ILC}}$ accepts, there are only three ways we could have $\langle \mathcal{P}^*(pp, u; s) \longleftrightarrow \mathcal{V}(pp, u; (\rho_{\mathsf{ILC}}, \rho)) \rangle$ accept without $\langle \mathcal{P}^*_{\mathsf{ILC}}(pp, u; s) \overset{\mathsf{ILC}}{\longleftrightarrow} \mathcal{V}_{\mathsf{ILC}}(pp_{\mathsf{ILC}}, u; \rho_{\mathsf{ILC}}) \rangle$ being also accepting

1. If $\mathcal{P}^*$ makes an opening to a commitment that is not its most frequent opening for that commitment.

2. If $\mathcal{P}^*_{\mathsf{ILC}}$ aborts because for some ℓ there is no $\boldsymbol{v}_\ell, \boldsymbol{r}_\ell$ such that

$$\mathsf{hd}(\widetilde{E}_{\mathcal{C}}(\boldsymbol{v}_\ell, \boldsymbol{r}_\ell), \boldsymbol{e}^*_\ell) < \frac{\mathsf{hd}_{\min}}{3}$$

3. If $\mathcal{P}^*$'s response to the verifier's queries contains a matrix $V^* \neq V'$.

We will now argue that there is a negligible probability for $\mathcal{V}$ to accept in case each of the above events happen.

Case 1. The verifier $\mathcal{V}$ only accepts if the openings to the commitments indexed by the challenge set J are all valid. Since $\mathcal{P}^*$ runs in polynomial time and the commitment scheme is computationally binding, there is only negligible probability that $\mathcal{P}^*$ sends valid openings for some of the commitments in J that are not the most frequent. Therefore there is only a negligible probability of $\mathcal{V}$ accepting in this case.

Case 2. On query $(\boldsymbol{\gamma}, Q)$, the prover $\mathcal{P}^*$ answers with response $(\boldsymbol{v}^*, \boldsymbol{r}^*, V^*, R^*)$. In this case we show that if matrix E^* contains row $\boldsymbol{e}_\ell^*$ that are not close to a codeword $\widetilde{E}_{\mathcal{C}}(\boldsymbol{v}_\ell; \boldsymbol{r}_\ell)$ for some $\boldsymbol{v}_\ell, \boldsymbol{r}_\ell$, then with overwhelming probability $(\boldsymbol{e}_0^* + \gamma E^*)$ is also far from $\widetilde{E}_{\mathcal{C}}(\boldsymbol{v}^*; \boldsymbol{r}^*)$, and therefore the verifier $\mathcal{V}$ is likely to reject the proof.

Namely, we show that if there exist a linear combination $\boldsymbol{q}E^*$ for some $\boldsymbol{q} \in \mathbb{F}^t$ such that $\mathsf{hd}\left(\tilde{\mathcal{C}}, \boldsymbol{q}E^*\right) \geq \frac{\mathsf{hd}_{\min}}{3}$, then the probability that $\mathsf{hd}(\tilde{\mathcal{C}}, \boldsymbol{e}_0^* + \boldsymbol{\gamma}E^*) < \frac{\mathsf{hd}_{\min}}{6}$ is at most $\frac{1}{|\mathbb{F}|}$. This also implies the above claim about all row vectors in E^*.

Assume that $\mathsf{hd}\left(\tilde{\mathcal{C}}, \boldsymbol{q}E^*\right) \geq \frac{\mathsf{hd}_{\min}}{3}$, then for any $r \in \mathbb{F}^*$ we have

$$\mathsf{hd}\left(\tilde{\mathcal{C}}, \boldsymbol{e}_0^* + \boldsymbol{\gamma}E^*\right) + \mathsf{hd}\left(\tilde{\mathcal{C}}, \boldsymbol{e}_0^* + (\boldsymbol{\gamma} + r\boldsymbol{q})E^*\right) \geq \mathsf{hd}\left(\tilde{\mathcal{C}}, \boldsymbol{q}E^*\right) \geq \frac{\mathsf{hd}_{\min}}{3}$$

To see this, write $\boldsymbol{e}_0^* + \boldsymbol{\gamma}E^* = \boldsymbol{c}_1 + \boldsymbol{d}_1$ and $\boldsymbol{e}_0^* + (\boldsymbol{\gamma} + r\boldsymbol{q})E^* = \boldsymbol{c}_2 + \boldsymbol{d}_2$ with $\boldsymbol{c}_1, \boldsymbol{c}_2 \in \tilde{\mathcal{C}}$ and $\mathsf{wt}(\boldsymbol{d}_1) = \mathsf{hd}\left(\tilde{\mathcal{C}}, \boldsymbol{e}_0^* + \boldsymbol{\gamma}E^*\right), \mathsf{wt}(\boldsymbol{d}_2) = \mathsf{hd}\left(\tilde{\mathcal{C}}, \boldsymbol{e}_0^* + (\boldsymbol{\gamma} + r\boldsymbol{q})E^*\right)$. Now

$$\begin{aligned}
\boldsymbol{q}E^* =& (\boldsymbol{e}_0^* + (\boldsymbol{\gamma} + r\boldsymbol{q})E^* - (\boldsymbol{e}_0^* + \boldsymbol{\gamma}E^*))r^{-1} \\
=& (\boldsymbol{c}_2 + \boldsymbol{d}_2 - \boldsymbol{c}_1 - \boldsymbol{d}_1)r^{-1} \\
=& (\boldsymbol{c}_2 - \boldsymbol{c}_1)r^{-1} + (\boldsymbol{d}_2 - \boldsymbol{d}_1)r^{-1}
\end{aligned}$$

Where $(\boldsymbol{c}_2 - \boldsymbol{c}_1)r^{-1} \in \tilde{\mathcal{C}}$ and $(\boldsymbol{d}_2 - \boldsymbol{d}_1)r^{-1}$ has at most

$$\mathsf{wt}(\boldsymbol{d}_1) + \mathsf{wt}(\boldsymbol{d}_2) = \mathsf{hd}\left(\tilde{\mathcal{C}}, \boldsymbol{e}_0^* + \boldsymbol{\gamma}E^*\right) + \mathsf{hd}\left(\tilde{\mathcal{C}}, \boldsymbol{e}_0^* + (\boldsymbol{\gamma} + r\boldsymbol{q})E^*\right)$$

non-zero elements. This implies that at most one of $\boldsymbol{e}_0^* + \boldsymbol{\gamma}E^*$ and $\boldsymbol{e}_0^* + (\boldsymbol{\gamma} + r\boldsymbol{q})E^*$ can have distance less than $\frac{\mathsf{hd}_{\min}}{6}$ to $\tilde{\mathcal{C}}$. That is, there is at most one $\boldsymbol{\gamma} \in \mathbb{F}^t$ in $\{(\boldsymbol{\gamma} + r\boldsymbol{q}) : r \in \mathbb{F}\}$ that can have distance less than $\frac{\mathsf{hd}_{\min}}{6}$ to $\tilde{\mathcal{C}}$. Hence, there is probability at most $\frac{1}{|\mathbb{F}|}$ that a random $\boldsymbol{\gamma} \in \mathbb{F}^t$ satisfies $\mathsf{hd}\left(\tilde{\mathcal{C}}, \boldsymbol{e}_0^* + \boldsymbol{\gamma}E^*\right) < \frac{\mathsf{hd}_{\min}}{6}$.

Therefore with overwhelming probability we have that

$$\mathsf{hd}(\widetilde{E}_{\mathcal{C}}(\boldsymbol{v}^*;\boldsymbol{r}^*), \boldsymbol{e}_0^* + \boldsymbol{\gamma}E) \geq \frac{\mathsf{hd}_{\min}}{6}$$

This means that either in the first half of the codeword $\widetilde{E}_{\mathcal{C}}(\boldsymbol{v}^*;\boldsymbol{r}^*)$ or in the second half, there will be at least $\frac{\mathsf{hd}_{\min}}{12}$ values of j where it differs from $\boldsymbol{e}_0^* + \boldsymbol{\gamma}E^*$. It is easy to see that the λ values of J in one half of $[2n]$ are chosen uniformly and independently at random conditioned on being different. Since the minimum distance of the code is linear, then the verifier has only negligible probability of missing an index where the above words disagree. Hence, the verifier $\mathcal{V}$ will reject the proof with overwhelming probability.

Case 3. We recall that on query $(\boldsymbol{\gamma}, Q)$, the prover $\mathcal{P}^*$ answers with $(\boldsymbol{v}^*, \boldsymbol{r}^*, V^*, R^*)$. From the previous claim we have that $\mathcal{P}^*_{\mathsf{ILC}}$ does not abort and constructs matrix V which is sent to the ILC. The ILC channel answers the query Q from $\mathcal{V}_{\mathsf{ILC}}$ with matrix $V' = QV$. For the last case we show that if $\mathcal{P}^*$ answers the query with a matrix $V^* \neq V'$, then QE^* is not close to the codeword $\widetilde{E}_{\mathcal{C}}(V^*;R^*)$ and the verifier is likely to reject the proof when checking these two on indexes in J.

From the previous case, we have that for all $\boldsymbol{q} \in \mathbb{F}^t$, we have $\mathsf{hd}(\tilde{\mathcal{C}}, qE^*) < \frac{\mathsf{hd}_{\min}}{3}$, otherwise the verifier rejects with high probability. Assuming this, here we show that exists $\boldsymbol{r}^*$ for which

$$\mathsf{hd}(\widetilde{E}_{\mathcal{C}}(\boldsymbol{q}V;\boldsymbol{r}^*), qE^*) < \frac{\mathsf{hd}_{\min}}{3}$$

We prove this by induction on the number of non-zero elements $\mathsf{wt}(\boldsymbol{q})$ in $\boldsymbol{q}$. This is trivially true for $\mathsf{wt}(\boldsymbol{q}) = 0$. For $\mathsf{wt}(\boldsymbol{q}) = 1$ it follows from our definitions of vectors $\boldsymbol{v}_\ell$ which are chosen by $\mathcal{P}^*_{\mathsf{ILC}}$ such that $\mathsf{hd}(\widetilde{E}_{\mathcal{C}}(\boldsymbol{v}_\ell, \boldsymbol{r}_\ell), \boldsymbol{e}_\ell^*) < \frac{\mathsf{hd}_{\min}}{3}$. Assume by induction that it is true for all vectors in $\mathbb{F}^t$ of hamming weight less or equal κ and consider a vector $\boldsymbol{q}$ with $\mathsf{wt}(\boldsymbol{q}) \leq 2\kappa$. We can now write $\boldsymbol{q} = \boldsymbol{q}' + \boldsymbol{q}''$ where $\mathsf{wt}(\boldsymbol{q}'), \mathsf{wt}(\boldsymbol{q}'') \leq \kappa$. By the induction hypothesis, there exists $\boldsymbol{r}'$ such that $\mathsf{hd}(\widetilde{E}_{\mathcal{C}}(\boldsymbol{q}'V;\boldsymbol{r}'), \boldsymbol{q}'E^*) < \frac{\mathsf{hd}_{\min}}{3}$ and

similar for $\boldsymbol{q}''$. Since $\boldsymbol{q} = \boldsymbol{q}' + \boldsymbol{q}''$ this implies

$$\begin{aligned}\mathsf{hd}\left(\widetilde{E}_{\mathcal{C}}\left(\boldsymbol{q}V; \boldsymbol{r}' + \boldsymbol{r}''\right), \boldsymbol{q}E^*\right) &= \mathsf{hd}\left(\widetilde{E}_{\mathcal{C}}\left((\boldsymbol{q}' + \boldsymbol{q}'')V, \boldsymbol{r}' + \boldsymbol{r}''\right), (\boldsymbol{q}' + \boldsymbol{q}'')E^*\right) \\ &\leq \mathsf{hd}\left(\widetilde{E}_{\mathcal{C}}\left(\boldsymbol{q}'V; \boldsymbol{r}'\right), \boldsymbol{q}'E^*\right) + \mathsf{hd}\left(\widetilde{E}_{\mathcal{C}}\left(\boldsymbol{q}''V; \boldsymbol{r}''\right), \boldsymbol{q}''E^*\right) \\ &< 2\frac{\mathsf{hd}_{\min}}{3}\end{aligned}$$

From the previous case we know there exists $\boldsymbol{v}$ and $\boldsymbol{r}$ such that $\mathsf{hd}(\widetilde{E}_{\mathcal{C}}(\boldsymbol{v}; \boldsymbol{r}), \boldsymbol{q}E^*) < \frac{\mathsf{hd}_{\min}}{3}$. By the triangle inequality for Hamming distance, this implies

$$\begin{aligned}&\mathsf{hd}\left(\widetilde{E}_{\mathcal{C}}\left(\boldsymbol{v}; \boldsymbol{r}\right), \widetilde{E}_{\mathcal{C}}\left(\boldsymbol{q}V; \boldsymbol{r}' + \boldsymbol{r}''\right)\right) \\ &\quad\leq \mathsf{hd}\left(\widetilde{E}_{\mathcal{C}}\left(\boldsymbol{v}; \boldsymbol{r}\right), \boldsymbol{q}E^*\right) + \mathsf{hd}\left(\boldsymbol{q}E^*, \widetilde{E}_{\mathcal{C}}\left(\boldsymbol{q}V; \boldsymbol{r}' + \boldsymbol{r}''\right)\right) \\ &\quad< \frac{\mathsf{hd}_{\min}}{3} + 2\frac{\mathsf{hd}_{\min}}{3} = \mathsf{hd}_{\min}\end{aligned}$$

Since $\mathsf{hd}_{\min}$ is the minimum distance of $\widetilde{E}_{\mathcal{C}}$, we must have $\boldsymbol{v} = \boldsymbol{q}V$, and therefore $\mathsf{hd}(\widetilde{E}_{\mathcal{C}}(\boldsymbol{q}V; \boldsymbol{r}), \boldsymbol{q}E^*) < \frac{\mathsf{hd}_{\min}}{3}$. This concludes the induction argument.

The triangle inequality for Hamming distance shows that for any $(\boldsymbol{v}^*, \boldsymbol{r}^*)$ with $\boldsymbol{v}^* \neq \boldsymbol{q}V$ we have $\mathsf{hd}(\widetilde{E}_{\mathcal{C}}(\boldsymbol{v}^*; \boldsymbol{r}^*), \boldsymbol{q}E^*) \geq 2\frac{\mathsf{hd}_{\min}}{3}$. Now for any $V^* \neq V' = QV$ there is a row ℓ where the two matrices differ. Let $\boldsymbol{q}$ be the ℓ-th row of Q. Then $\mathsf{hd}(\widetilde{E}_{\mathcal{C}}(\boldsymbol{v}^*; \boldsymbol{r}^*), \boldsymbol{q}E^*) \geq 2\frac{\mathsf{hd}_{\min}}{3}$ tells us that the ℓ-th row of $\widetilde{E}_{\mathcal{C}}(V^*, R^*)$ and ℓ-th row of QE^* differs in at least $2\frac{\mathsf{hd}_{\min}}{3}$ positions.

Given that the minimum distance of the code is linear in n, if the distance between two strings of length $2n$ is at least $2\frac{\mathsf{hd}_{\min}}{3}$, then there is negligible probability that J will not contain a j such that the two strings differ in position j. Thus, the probability that $\mathcal{P}^*$ sends a matrix $V^* \neq V'$ and $\widetilde{E}_{\mathcal{C}}\left(V^*, R^*\right)|_J = QE^*|_J$ is negligible.

Altogether, the three cases show that the probability that $\mathcal{V}$ accepts while $\mathcal{V}_{\mathsf{ILC}}$ rejects is negligible. □

Lemma 6.2. *Let $\mathcal{P}^*_{\mathsf{ILC}}$ be defined as in the proof of Lemma 6.1. There exists an expected PPT $\mathcal{E}_{\mathsf{tran}}$ such that for all PPT $\mathcal{A}$*

$$\Pr\left[\begin{array}{c} pp \leftarrow \mathcal{G}(1^\lambda); pp = (pp_{\mathsf{ILC}}, *); (u,s) \leftarrow \mathcal{A}(pp); \\ \widetilde{\mathsf{tran}}_{\mathcal{P}^*_{\mathsf{ILC}}} \leftarrow \mathcal{E}_{\mathsf{tran}}^{\langle \mathcal{P}^*(pp,u;s) \longleftrightarrow \mathcal{V}(pp,u)\rangle}(pp,u); \\ (\mathsf{tran}_{\mathcal{P}^*_{\mathsf{ILC}}}; b_{\mathsf{ILC}}) \leftarrow \langle \mathcal{P}^*_{\mathsf{ILC}}(pp,u;s) \overset{\mathsf{ILC}}{\longleftrightarrow} \mathcal{V}_{\mathsf{ILC}}(pp_{\mathsf{ILC}},u;\rho_{\mathsf{ILC}})\rangle\colon \\ b = 1 \ \ \wedge \ \ \mathsf{tran}_{\mathcal{P}^*_{\mathsf{ILC}}} \neq \widetilde{\mathsf{tran}}_{\mathcal{P}^*_{\mathsf{ILC}}} \end{array}\right] \approx 0 \qquad (6.2)$$

Proof. On input (pp, u), the transcript extractor $\mathcal{E}_{\mathsf{tran}}$ first uses its oracle to get a transcript of $\langle \mathcal{P}^*(pp,u;s) \longleftrightarrow \mathcal{V}(pp,u)\rangle$. If $\mathcal{V}$ rejects, then $\mathcal{E}_{\mathsf{tran}}$ aborts. If $\mathcal{V}$ accepts, $\mathcal{E}_{\mathsf{tran}}$ rewinds the last message of $\mathcal{P}^*$ to get a transcript for a new random challenge J. $\mathcal{E}_{\mathsf{tran}}$ continues in this way until it has an accepting transcript for $2n$ independently chosen sets J. If there is only one choice of J that results in $\mathcal{V}$ accepting, $\mathcal{P}^*$ will likely have received each allowed challenge around $2n$ times and $\mathcal{E}_{\mathsf{tran}}$ will get the exact same transcript $2n$ times before it is done rewinding. Still, $\mathcal{E}_{\mathsf{tran}}$ runs in expected polynomial time: if a fraction p of all allowed sets J give accepting transcripts, the expected number of rewindings is $\frac{2n-1}{p}$, *given* that the first transcript is accepting. However, the probability that the first run accepts is p, and if it does not accept, $\mathcal{E}_{\mathsf{tran}}$ does not do any rewindings. Overall this gives $\frac{(2n-1)p}{p} = 2n-1$ rewindings in expectation.

We let $J_1, \ldots, J_{2n}$ denote the set of challenges J in the accepting transcripts obtained by $\mathcal{E}_{\mathsf{tran}}$. If $\bigcup_{i=1}^{2n} J_i$ has less than $2n - \frac{\mathsf{hd}_{\min}}{3}$ elements, $\mathcal{E}_{\mathsf{tran}}$ terminates. Otherwise, $\mathcal{E}_{\mathsf{tran}}$ is defined similarly to $\mathcal{P}^*_{\mathsf{ILC}}$: it uses the values of the openings to get at least $2n - \frac{\mathsf{hd}_{\min}}{3}$ columns of each E_i. For each of the row vectors, $\boldsymbol{e}_\ell$, it computes $\boldsymbol{v}_\ell$ and $\boldsymbol{r}_\ell$ such that $\widetilde{E}_{\mathcal{C}}(\boldsymbol{v}_\ell, \boldsymbol{r}_\ell)$ agrees with $\boldsymbol{e}_\ell$ in all entries $\boldsymbol{e}_\ell|_j$ for which the j'th column have been revealed, if such $\boldsymbol{v}$ exists. Since $\mathcal{E}_{\mathsf{tran}}$ will not correct any errors, finding such $\boldsymbol{v}_\ell$ and $\boldsymbol{r}_\ell$ corresponds to solving a linear set of equations. Notice that since the minimum distance is more than $2\frac{\mathsf{hd}_{\min}}{3}$ there is at most one such $\boldsymbol{v}_\ell$ for each $\ell \in [t]$. If for some ℓ there is no such $\boldsymbol{v}_\ell$, then $\mathcal{E}_{\mathsf{tran}}$ aborts, otherwise $\mathcal{E}_{\mathsf{tran}}$ uses the resulting vectors $\boldsymbol{v}_\ell$ as the prover messages to define $\widetilde{\mathsf{tran}}_{\mathcal{P}^*_{\mathsf{ILC}}}$.

If $|\bigcup_{i=1}^{\kappa} J_i| < 2n - \frac{\mathsf{hd}_{\min}}{3}$, there are at least $\frac{\mathsf{hd}_{\min}}{6}$ indexes in $[n] \setminus \bigcup_{i=1}^{\kappa} J_i$ or in $\{n+1, \ldots, 2n\} \setminus \bigcup_{i=1}^{\kappa} J_i$. In either case, a random allowed J has negligible probability of being contained in $\bigcup_{i=1}^{\kappa} J_i$. Since $\mathcal{E}_{\mathsf{tran}}$ runs in expected polynomial time, this implies

by induction that there is only negligible probability that $|\bigcup_{i=1}^{\kappa} J_i| < \min(\kappa, 2n - \frac{\mathsf{hd}_{\min}}{3})$ and therefore $|\bigcup_{i=1}^{2n} J_i| < 2n - \frac{\mathsf{hd}_{\min}}{3}$.

Finally, we need to show there is at most negligible probability that for some ℓ there are no $\boldsymbol{v}_\ell$ and $\boldsymbol{r}_\ell$ such that $\widetilde{E}_{\mathcal{C}}(\boldsymbol{v}_\ell, \boldsymbol{r}_\ell)$ agrees with $\boldsymbol{e}_\ell$ on all the opened $j \in \bigcup_{i=1}^{2n} J_i$ columns and that $b = 1$. In particular, this shows that the probability that $b = 1$ and $\mathcal{E}_{\mathsf{tran}}$ does not extract the transcript of $\mathcal{P}^*_{\mathsf{ILC}}$ is negligible.

Since the expected number of rewindings is polynomial, we can assume that in all the rewindings, $\mathcal{P}^*$ only makes openings to the most frequent openings. Note that if this is not the case, after sufficiently many rewindings the extractor also observes the most frequent openings and can thus break the binding property of the commitment scheme. In Lemma 6.1 we showed that the probability that $b = 1$ but $\mathcal{P}^*$ sends a $V^* \neq V'$ is negligible and by following the same argument, the probability that $b = 1$ but $\mathcal{P}^*$ sends $\boldsymbol{v}^* \neq \boldsymbol{v}'$ is negligible. Therefore, in the following we will assume $\boldsymbol{v}^* = \boldsymbol{v}'$.

Now suppose that there is some $\boldsymbol{e}_\ell$ such that the opened values are inconsistent with $\widetilde{E}_{\mathcal{C}}(\boldsymbol{v}_\ell, \boldsymbol{r}_\ell)$ for any $\boldsymbol{r}_\ell$. That is, there is some j such that $j, n+j \in \bigcup_{i=1}^{2n} J_i$ and $\boldsymbol{e}_\ell|_j - \boldsymbol{e}_\ell|_{n+j} \neq E_{\mathcal{C}}(\boldsymbol{v})|_j$. For uniformly chosen $\boldsymbol{\gamma}_\ell \in \mathbb{F}$, we get that $\boldsymbol{\gamma}_\ell(\boldsymbol{e}_\ell|_j - \boldsymbol{e}_\ell|_{n+j} - E_{\mathcal{C}}(\boldsymbol{v})|_j)$ is uniformly distributed in $\mathbb{F}$. Hence for a random $\boldsymbol{\gamma} \in \mathbb{F}^t$, we have that $\boldsymbol{\gamma}(E|_j - E|_{n+j} - E_{\mathcal{C}}(\boldsymbol{v})|_j)$ is uniformly distributed. When $\mathcal{V}$ queries $\boldsymbol{\gamma}$, $\mathcal{P}^*$ will respond with $\boldsymbol{v}^* = \boldsymbol{v}'$ and some $\boldsymbol{r}^*$. $\mathcal{V}$ will only accept on a challenge J if for all $j \in J$ we have $(\boldsymbol{e}_0 + \boldsymbol{\gamma}E)|_j = \widetilde{E}_{\mathcal{C}}(\boldsymbol{v}', \boldsymbol{r}^*)|_j$. Since $j, n+j \in \bigcup_{i=1}^{2n} J_i$ we have $(\boldsymbol{e}_0 + \boldsymbol{\gamma}E)|_j = \widetilde{E}_{\mathcal{C}}(\boldsymbol{v}', \boldsymbol{r}^*)|_j$ and $(\boldsymbol{e}_0 + \boldsymbol{\gamma}E)|_{n+j} = \widetilde{E}_{\mathcal{C}}(\boldsymbol{v}', \boldsymbol{r}^*)|_{n+j}$. Consider the following

$$\begin{aligned}
\boldsymbol{e}_0|_j - \boldsymbol{e}_0|_{n+j} + \boldsymbol{\gamma}E|_j - \boldsymbol{\gamma}E|_{n+j} &= \widetilde{E}_{\mathcal{C}}(\boldsymbol{v}', \boldsymbol{r}^*)|_j - \widetilde{E}_{\mathcal{C}}(\boldsymbol{v}', \boldsymbol{r}^*)|_{n+j} \\
&= E_{\mathcal{C}}(\boldsymbol{v}')|_j \\
&= (E_{\mathcal{C}}(\boldsymbol{v}_0) + \boldsymbol{\gamma}E_{\mathcal{C}}(\boldsymbol{v}))|_j
\end{aligned}$$

By rearranging the two sides of the first and the last term we have

$$\boldsymbol{\gamma}E|_j - \boldsymbol{\gamma}E|_{n+j} - \boldsymbol{\gamma}E_{\mathcal{C}}(\boldsymbol{v})_j = E_{\mathcal{C}}(\boldsymbol{v}_0)_j - (\boldsymbol{e}_0)_j + (\boldsymbol{e}_0)_{n+j}$$

For random $\boldsymbol{\gamma}$ the left-hand side is uniform and the right-hand side is fixed, hence equality only happens with negligible probability. This concludes the proof of the

lemma and of Theorem 6.3. . □

Theorem 6.4 (SHVZK). *If* $(\mathcal{G}_{\mathsf{ILC}}, \mathcal{P}_{\mathsf{ILC}}, \mathcal{V}_{\mathsf{ILC}})$ *is a public coin proof system for the relation* $\mathcal{R}$ *over the* ILC *with perfect SHVZK and* $(\mathsf{CSetup}, \mathsf{CCommit})$ *is computationally (statistically) hiding then the public coin proof system* $(\mathcal{G}, \mathcal{P}, \mathcal{V})$ *of Figure 6.2 is computationally (statistically) SHVZK.*

Proof. To prove SHVZK, we describe the simulator $\mathcal{S}(pp, u, \rho)$ that produces a simulated view, which is indistinguishable from the real view of the honest verifier $\mathcal{V}$.

Given the randomness ρ, the simulator learns both the messages and the queries $\rho_{\mathsf{ILC}} = (x_1, \ldots, x_{\mu-1}, Q)$ produced by the internal $\mathcal{V}_{\mathsf{ILC}}$ and forwarded by the honest $\mathcal{V}$. $\mathcal{S}$ can therefore run $\mathcal{S}_{\mathsf{ILC}}(pp_{\mathsf{ILC}}, u, \rho_{\mathsf{ILC}})$ to simulate the view of the internal $\mathcal{V}_{\mathsf{ILC}}$. This gives it $(t_1, \ldots, t_\mu, V')$. By the SHVZK property of $(\mathcal{G}_{\mathsf{ILC}}, \mathcal{P}_{\mathsf{ILC}}, \mathcal{V}_{\mathsf{ILC}})$ this simulated transcript is identically distributed to the real view of the internal $\mathcal{V}_{\mathsf{ILC}}$. The simulator $\mathcal{S}_{\mathsf{ILC}}$ returns a simulated query response V' as part of the simulated transcript.

Then, $\mathcal{S}$ reads the rest of ρ to also learn the challenges $\boldsymbol{\gamma}$ and J that $\mathcal{V}$ sends to the prover. The simulator picks uniformly at random $\boldsymbol{v}' \leftarrow \mathbb{F}^k$. In a real proof $\boldsymbol{v}_0$ is chosen uniformly at random, therefore the simulated $\boldsymbol{v}'$ is identically distributed as in a real proof. Since the verifier is honest, the set J is an admissible set and the exposure resilience of $\widetilde{E}_{\mathcal{C}}$ guarantees that $E_{01}|_J, \ldots, E_\mu|_J$ are distributed uniformly at random, assuming the randomness used in each execution of $\widetilde{E}_{\mathcal{C}}$ is sampled uniformly and independently at random, as done in a real execution of the proof. Therefore, the simulator also picks this part of the transcript uniformly at random. Given these, the simulator computes both $\boldsymbol{e}_0|_J + \boldsymbol{\gamma} E|_J$ and $QE|_J$. The simulator $\mathcal{S}$ then invokes the simulator described in Corollary 6.1. Given an admissible set J and inputs $\boldsymbol{v}', \boldsymbol{e}_0|_J + \boldsymbol{\gamma} E|_J$ (resp. $V', QE|_J$), this returns randomness $\boldsymbol{r}'$ (resp. R') that is uniformly distributed conditioned on satisfying $\widetilde{E}_{\mathcal{C}}(\boldsymbol{v}'; \boldsymbol{r}')|_j = \boldsymbol{e}_0|_J + \boldsymbol{\gamma} E|_J$ (resp. $\widetilde{E}_{\mathcal{C}}(V'; R')|_j = QE|_J$). This part of the transcript is identically distributed as in a real proof.

Finally, $\mathcal{S}$ defines $E_{01}|_{\bar{J}}, \ldots, E_\mu|_{\bar{J}}$ to be 0 matrices for $\bar{J} = [2n] \setminus J$. It then picks $\boldsymbol{s}_1, \ldots, \boldsymbol{s}_\mu$ at random and makes the commitments $\boldsymbol{c}_1, \ldots, \boldsymbol{c}_\mu$ as in the protocol. We see that all the $\boldsymbol{c}_i|_J$ commitments are computed as in the real execution from values that are identically distributed as in a real proof. The $\boldsymbol{c}_i|_{\bar{J}}$ are commitments to different values than in a real proof. However, by the computational (statistical) hiding

property of the commitment scheme, they have a distribution that is computationally (statistically) indistinguishable from the correct distribution. Overall, the simulated view produced by the $\mathcal{S}$ is computationally (statistically) indistinguishable from the real view of the honest verifier. □

6.3 Efficiency and Instantiations

Here we report the general efficiency of a compiled proof of knowledge and then discuss the cases of our ILC proofs for the satisfiability of arithmetic circuits and for the execution of TinyRAM programs from Chapter 4 and 5. We recall that the only requirement of our compiler is that the error correcting code has to be linear, with constant rate, and linear minimum distance. The commitment scheme does not require any additional properties other than the standard notion of hiding and binding. Therefore, any combination of error correcting codes and commitments with these features can be used to realise our ILC proofs over the standard channel. However, our main goal in the previous chapter has been to minimise the asymptotical overhead incurred by the prover. Therefore, we are interested in optimal choices of error correcting codes and commitment schemes that allow to preserve the efficiency of our proofs also over the standard channel. For the error correcting codes we opt for the family by Druk and Ishai [DI14], which we recalled in Theorem 3.1. These are defined over a generic finite field $\mathbb{F}$ and only require a linear number of field additions to compute. For the commitment scheme we consider two possible instantiations which we described in Section 3.5. The first one is based on the hash function of Applebaum et al. [AHI+17b]. The resulting commitment is computationally binding assuming the bSVP problem and statistically hiding. The second one is based on the linear-time computable PRG of Ishai et al. [IKO+08], which is statistically binding and computationally hiding, and relies on the hardness of decoding sparsely generated linear codes. By using the first commitment scheme we obtain statistical special honest verifier zero-knowledge arguments of knowledge. With the second scheme we obtain computational special honest verifier zero-knowledge proofs of knowledge.

6.3.1 Efficiency of the Compilation

We recall the notation used to indicate the efficiency measures of an ILC proof system: μ is the number of rounds, $t = \sum_{i=1}^{\mu} t_i$ is the prover's communication complexity, C_{ILC} is the verifier's communication complexity, qc the verifier's query complexity, $T_{\mathcal{P}_{\mathsf{ILC}}}$ is the running time of the prover, and $T_{\mathcal{V}_{\mathsf{ILC}}}$ the running time of the verifier. Let $\mathcal{C}$ be a $[n, k, \mathsf{hd}_{\min}]_{\mathbb{F}}$ code. We write $T_{\widetilde{E}_{\mathcal{C}}}(k)$ for the cost of encoding a vector in $\mathbb{F}^k$, $T_{\mathsf{Com}}(t_i)$ for the cost of committing to t_i field elements, $T_{\text{Mmul}}(\mathsf{qc}, t, b)$ for the cost of multiplying matrices a matrix in $\mathbb{F}^{\mathsf{qc} \times t}$ by a matrix in $\mathbb{F}^{t \times b}$, and let $C_{\mathsf{Com}}(t_i)$ be the combined size of the commitments to t_i field elements. We give the dominant factors of efficiency of the compiled proof in Table 6.1.

Prover Comp.	$T_{\mathcal{P}_{\mathsf{ILC}}} + t \cdot T_{\widetilde{E}_{\mathcal{C}}}(k) + 2n \cdot \sum_{i=1}^{\mu} T_{\mathsf{Com}}(t_i) + T_{\text{Mmul}}(\mathsf{qc}+1, t, k+n)$
Verifier Comp.	$T_{\mathcal{V}_{\mathsf{ILC}}} + (\mathsf{qc}+1) \cdot T_{\widetilde{E}_{\mathcal{C}}}(k) + 2\lambda \cdot \sum_{i=1}^{\mu} T_{\mathsf{Com}}(t_i) + T_{\text{Mmul}}(\mathsf{qc}+1, t, 2\lambda)$
Communication	$C_{\mathsf{ILC}} + 2n \cdot \sum_{i=1}^{\mu} C_{\mathsf{Com}}(t_i) + (\mathsf{qc}+1) \cdot (k+n+t) + 2\lambda \cdot t$
Round Comp.	$\mu + 2$

TABLE 6.1: Efficiency of a compiled proof of knowledge $(\mathcal{G}, \mathcal{P}, \mathcal{V})$ for $(pp, u, w) \in \mathcal{R}$.

6.3.2 Proofs and Arguments for the Satisfiability of Arithmetic Circuits

We now move to analyse the efficiency of the compiled proofs of knowledge for the satisfiability of arithmetic circuit we gave in Figure 4.10. By instantiating the codes with the ones in Theorem 3.1 we get $T_{\widetilde{E}_{\mathcal{C}}}(k) = \mathcal{O}(k)$ field additions. Let us now plug in the efficiency of our ILC proof given in Table 4.5 into the efficiency formulas in Table 6.1. We use $k \approx \sqrt{N}$, $n = \mathcal{O}(k)$, $t = \mathcal{O}(\sqrt{N})$, $\mu = \mathcal{O}(\log \log N)$, $\mathsf{qc} = 20 = \mathcal{O}(1)$ and assume $k \gg \lambda$. The resulting efficiency is given in Table 6.2

Prover Computation	$T_{\mathcal{P}}$	$= \mathcal{O}(N)\ \mathbb{F}^{\times} + 2n \cdot \sum_{i=1}^{\mu} T_{\mathsf{Com}}(t_i)$
Verifier Computation	$T_{\mathcal{V}}$	$= \mathcal{O}(N)\ \mathbb{F}^{+} + 2\lambda \cdot \sum_{i=1}^{\mu} T_{\mathsf{Com}}(t_i)$
Communication	t	$= \mathcal{O}(\lambda\sqrt{N})\ \log \lvert\mathbb{F}\rvert + 2n \cdot \sum_{i=1}^{\mu} C_{\mathsf{Com}}(t_i)$
Round Complexity	μ	$= \mathcal{O}(\log \log(N))$

TABLE 6.2: Efficiency of our proof of knowledge for the relation $\mathcal{R}_{\text{AC}}$ over the standard channel. $\mathbb{F}^{\times}, \mathbb{F}^{+}$ are the costs of field multiplications and additions, respectively. $\log \lvert\mathbb{F}\rvert$ is the size of a field element.

Instantiating with the commitment scheme from Applebaum et al. [AHI+17b] we get computational knowledge soundness and statistical SHVZK. The commitments

are compact, i.e. a commitment has size $C_{\mathsf{Com}}(t_i) = \mathrm{poly}(\lambda)$ regardless of the message length, giving us sub-linear communication. The commitments can be computed with a linear number of bit operations, i.e. $T_{\mathsf{Com}}(t_i) = \mathrm{poly}(\lambda) + \mathcal{O}(t_i)$ bit operations. Although we do not have an accurate estimation on the number of the *arithmetic* operations required to evaluate this commitment scheme, we notice that the overall computational cost for prover and verifier remains linear. To give a fair account of the performances of the proof system we measure the efficiency in a unified model which can perform both natively Boolean and arithmetic operations, and we consider prover and verifier implemented as RAM programs, e.g. TinyRAM programs.

In Chapter 5 we considered TinyRAM machines with word size equal to $W = \Theta(\log(\lambda))$. Since the soundness of the proof system requires a large field, i.e. $|\mathbb{F}| = \lambda^{\omega(1)}$, we have that field elements require $e = \frac{\log|\mathbb{F}|}{W} = \omega(1)$ words for storing. Field additions then cost $\mathcal{O}(e)$ TinyRAM operations and field multiplications cost at most $\mathcal{O}(e^2)$ TinyRAM operations. As the hash function operates over bit-strings, if we stored each bit in a separate word of size $W = \Theta(\log \lambda)$ we would incur a logarithmic overhead. However, the hash function is computable by a linear-size boolean circuit and we can therefore apply a bit-slicing technique. We view the hash of an n-word string as W parallel hashes of n-bit strings. Each of the bit-strings is processed with the same boolean circuit, which means they can be computed in parallel in one go by a TinyRAM program using a linear number of steps, e.g hashing one field element requires $\mathcal{O}(e)$ TinyRAM steps. Therefore, in this model hashing field elements has a computational cost comparable to field additions.

Expressing the costs of Table 6.2 in TinyRAM operations, we have that the computational cost of the prover is dominated by $\mathcal{O}(e^2 N)$ TinyRAM steps, which is the same asymptotic efficiency of a program evaluating the circuit up to a constant overhead. Regarding verifier computation, the cost of field additions dominates the cost of checking 2λ committed columns, assuming $k \gg \lambda$, and thus the overall complexity is dominated by $\mathcal{O}(eN)$ TinyRAM steps. Asymptotically this is comparable to the cost of reading the statement in the memory, therefore the verifier's cost is optimal up to a constant factor overhead, unless more compact representations of the statement are used. Communication complexity amounts to $\mathrm{poly}(\lambda)\sqrt{N}$ words.

Instantiating with the commitment from Ishai et al. [IKO+08] we get statistical

knowledge soundness and computational SHVZK. The commitments have linear size $C_{\mathsf{Com}}(t_i) = \mathrm{poly}(\lambda) + t_i \log(|\mathbb{F}|)$ bits, giving us linear communication overall, i.e. $\mathcal{O}(eN)$ words of length $\Theta(\log(\lambda))$. By expressing the performances in TinyRAM, the commitments can be computed in linear time at a cost of $T_{\mathsf{Com}}(t_i) = \mathrm{poly}(\lambda) + e\mathcal{O}(t_i)$ TinyRAM steps, again giving us $\mathcal{O}(e^2 N)$ TinyRAM steps for the prover, which is comparable to the number of steps for computing $\mathcal{O}(N)$ multiplications. The computational cost for the verifier is dominated again by $\mathcal{O}(eN)$ TinyRAM steps, comparable to the cost of computing $\mathcal{O}(N)$ additions on a TinyRAM machine.

We summarise the costs of the instantiated proofs and arguments in Table 6.3, reporting the efficiency both in terms of TinyRAM operations and the equivalent amount of field operations when performed in TinyRAM.

Measure\Inst.	Using [AHI+17b]			Using [IKO+08]		
Prover Comp.	$\mathcal{O}(N)\ \mathbb{F}^{\times}$	$\approx$	$\mathcal{O}(e^2 N)$ TR	$\mathcal{O}(N)\ \mathbb{F}^{\times}$	$\approx$	$\mathcal{O}(e^2 N)$ TR
Verifier Comp.	$\mathcal{O}(N)\ \mathbb{F}^{+}$	$\approx$	$\mathcal{O}(eN)$ TR	$\mathcal{O}(N)\ \mathbb{F}^{+}$	$\approx$	$\mathcal{O}(eN)$ TR
Communication	$\mathrm{poly}(\lambda)\sqrt{N}\ \log\|\mathbb{F}\|$	$\approx$	$\mathrm{poly}(\lambda)\sqrt{N}\ \|W\|$	$\mathcal{O}(N)\ \log\|\mathbb{F}\|$	$\approx$	$\mathcal{O}(eN)\ \|W\|$
Round Comp.	$\mathcal{O}(\log\log N)$			$\mathcal{O}(\log\log N)$		
Completeness	Perfect			Perfect		
K. Soundness	Computational			Statistical		
SHVZK	Statistical			Computational		

TABLE 6.3: Efficiency of two instantiations of our SHVZK proofs and arguments for arithmetic circuit satisfiability both in terms of TinyRAM operations and field operations. $\mathbb{F}^{\times}, \mathbb{F}^{+}$ are the costs of field multiplications and additions, respectively. $\log|\mathbb{F}|, |W|$ are the size of field elements and words, respectively. TinyRAM operations are denoted as TR and $e = \frac{\log|\mathbb{F}|}{W}$.

To summarise, we constructed the first proofs and arguments of knowledge for the satisfiability of arithmetic circuits with linear complexity for prover and verifier, which are asymptotically optimal up to constant factor. Our arguments of knowledge also achieve sublinear communication complexity.

6.3.3 Proofs and Arguments for the Correct Program Execution

We now move on to analyse the efficiency of the compiled version of the proof of knowledge for the correct program execution of a TinyRAM program we described in Figure 5.9. As in Chapter 5, we consider programs of length $L = \mathrm{poly}(\lambda)$ terminating within $T = \mathrm{poly}(\lambda)$ steps and using $M = \mathrm{poly}(\lambda)$ word of memory on a TinyRAM machine using $W = \Theta(\log(\lambda))$ and $K = \mathcal{O}(1)$ registers. The soundness of the proof

Measure\Inst.	Using [AHI+17b]	Using [IKO+08]
Prover Comp.	$\mathcal{O}(e^2T)$ TR	$\mathcal{O}(e^2N)$ TR
Verifier Comp.	$\text{poly}(\lambda)\mathcal{O}(L+\lvert v\rvert+\sqrt{T})$ TR	$\text{poly}(\lambda)\mathcal{O}(L+\lvert v\rvert+\sqrt{T})$ TR
Communication	$\text{poly}(\lambda)\mathcal{O}(\sqrt{T})\ \lvert W\rvert$	$\mathcal{O}(eT)\ \lvert W\rvert$
Round Comp.	$\mathcal{O}(\log\log T)$	$\mathcal{O}(\log\log T)$
Completeness	Perfect	Perfect
K. Soundness	Computational	Statistical
SHVZK	Statistical	Computational

TABLE 6.4: Efficiency of two instantiations of our SHVZK proofs and arguments for the execution of TinyRAM programs. L is the length of the program and $|v|$ is the size of the public inputs of the program. TR stands for TinyRAM operations, $|W|$ is the word size of the TinyRAM machine and $e = \frac{\log|\mathbb{F}|}{W} = \omega(1)$ is the overhead of field operations in TinyRAM.

system requires the field size to be $|\mathbb{F}| = \lambda^{\omega(1)}$, which results in a superconstant ratio $e = \frac{\log|\mathbb{F}|}{W} = \omega(1)$, which can be made arbitrarily small. Our proofs are designed for programs executing intensive computation and for which we consider $T \gg L + M$.

For the error correcting codes we use again the ones by Druk and Ishai [DI14] which are computable by a linear number of TinyRAM steps. Instantiating the construction with the commitment scheme from Applebaum et al. [AHI+17b] we get computational knowledge soundness, statistical SHVZK and sublinear communication complexity. Instantiating the construction with the commitments from Ishai et al. [IKO+08] we get statistical knowledge soundness, computational SHVZK and linear communication. As all the primitives are computable in linear time, our compiled proofs achieve the same asymptotics as over the ILC channel.

Let us now combine the efficiency of our ILC proofs given in Table 5.9 with the efficiency formulas in Table 6.1. In this case we use $k \approx \sqrt{T}$, $n = \mathcal{O}(k)$, $t = \mathcal{O}(\sqrt{T})$, $\mu = \mathcal{O}(\log\log T)$, $\mathsf{qc} = 298 = \mathcal{O}(1)$ and assume $k \gg \lambda$. We summarise the costs of the instantiated proofs and arguments in Table 6.4, reporting the efficiency in terms of TinyRAM operations, taking into account the overhead of computing field operations in TinyRAM.

To summarise, we constructed the first proofs and arguments for the correct execution of programs achieving arbitrarily small superconstant overhead for the prover, i.e. $\mathcal{O}(e^2)$. The verification cost of our proofs is very efficient: aside from the cost of reading the instance, the verifier only pays a sublinear number of steps in the execution time of the program. Our arguments of knowledge additionally achieve sublinear

communication complexity.

Chapter 7

Foundations of Fully Dynamic Group Signatures

In this chapter we diverge from the topics of the previous ones and take a look at group signatures, a compelling application of zero-knowledge proofs. Group signatures are a fundamental cryptographic primitive allowing a member of a group to anonymously sign messages on behalf of the group: nothing about the identity of the signer is disclosed apart from her membership in the group. Group membership is administered by a designated authority which is referred to as group manager. In case of a dispute, the group manager or a designated opening authority have the ability to identify the signer and attribute the signature to her.

Group signatures are typically divided into static, partially dynamic, and fully dynamic, depending on how flexible the group membership of the users is. In static group signatures [BMW03], the group members are fixed once and for all during the setup phase. Partially dynamic group signatures [BSZ05; KY06] allow the enrolment of members in the group at any time but members cannot leave or be removed from the group once they have joined. In many settings, however, it is desirable to offer full flexibility in joining and leaving the group, which is the case covered by fully dynamic group signatures.

In this chapter we address the fundamental question of defining security for fully dynamic group signatures. While the security of the static and partially dynamic group settings has been rigorously formulated [BMW03; BSZ05; KY06; SSE+12] and is now well understood, the security of their fully dynamic groups counterpart, which is more relevant to practice, has on the other hand received less attention and is still

lacking at present. Particularly, the different design paradigms assume different, and sometimes informal models, which do not necessarily generalise to other design approaches. This resulted in various models, the majority of which lack rigour. Consequently, it can be difficult to compare the merits of the different constructions in terms of their security guarantees. Moreover, existing models place a large amount of trust in the setup and assume that keys are generated honestly, which does not necessarily reflect real-world scenarios.

Towards a General Model. In the process of formulating a general model we identify a subtle difference on the security of models following the revocation list based approach and other approaches, e.g. based on cryptographic accumulators. In these models the manager periodically publishes some information about members excluded from the group, i.e. revocation lists. The life of the scheme spans over different intervals, or epochs, at the start of which the manager updates the revocation lists. Signatures are bound to a specific epoch and it is vital for functionality that old valid signatures are accepted by the verification algorithm. We observe that constructions that follow this approach allow group members to sign with respect to an epoch unless they are explicitly revoked at that time. However, this allows them to sign with respect to an epoch predating their joining epoch, as they were not revoked at that time. In a sense this may be considered an attack against traceability, as those members were not in the group at that interval. Since accumulator-based constructions are generally not susceptible to this issue, they can be seen to achieve a slightly stronger notion of traceability.

One could, on the other hand, dismiss this attack by arguing that, in these models, it is implicit that epochs are not references to configurations of the group at a specific time. The underlying issue is a gap between one's interpretation of group signatures and what the definition implies.

To address this, our traceability definition models a more general security notion: users are not authorised to sign unless they are *active* members of the group. We regard the time span when a non-revoked user is considered active to be design-specific and think of it as the group manager's official *policy* upon when users are allowed to sign.

Following this interpretation accumulator-based constructions can be seen to satisfy our traceability notion with respect to stronger policies.

Chapter Outline. In Section 7.1 we provide a rigorous security model for fully dynamic group signatures. We consider both the case of a single group manager and the case where the role of the group manager who can grant or revoke membership is separated from the role of the opening authority who can identify a signer. Our security definitions are general and applicable across different design paradigms. In particular, our model covers both accumulator-based and revocation list based designs. Our model offers stringent security definitions, including the case where the authorities' keys are adversarially generated.

We then show that suitable restrictions of our security definitions yield definitions related to existing formal security definitions for partially dynamic group signatures (Section 7.2) and static group signatures (Section 7.3).

In Section 7.4 we present a generic construction of fully dynamic group signatures from accountable ring signatures and show that this satisfies the strongest variant of our security definitions, namely the ones with separate authorities and adversarial key generation.

Finally, in Section 7.5 we clarify the difference between the different flavours of traceability and the use of activation policies in revocation list based constructions.

Notation. In this chapter we adopt a specific notation to model interactions between algorithms. We recall that for algorithms A and B, $(x; y) \leftarrow \langle A(a); B(b)\rangle$ denotes the the joint execution of A (with input a) and B (with input b) where at the end A outputs x and B outputs y. Delving into the details of the interaction, it may take place over several rounds during which the algorithms send messages to each other and after which they halt and return their outputs. For an interactive algorithm, a move in the protocol is generated as $(\mathsf{out}; M') \leftarrow A(M)$, where M is the message just received from the other participant or $M = \mathsf{init}$ if this is the first move in the protocol, and M' is the message to send to the other participant. We use the convention that when out is empty ($\mathsf{out} = \varepsilon$), it means that A intends for the interaction to continue, but when out is not empty A will send the last message M' (unless empty) and then terminate the

interaction with output out. We note that in the setting of group signatures, the group manager may be involved in multiple concurrent interactions. The group manager's state may therefore change between two rounds in any given joint execution. We write $(\mathsf{out}; M'; \mathsf{st}_A) \leftarrow A(M; \mathsf{st}_A)$ when we explicitly want to indicate the state of an algorithm A may be updated. We use $\perp$ as an error symbol. Algorithms do not return $\perp$ unless they explicitly want to indicate an error. Conversely, we use the symbol $\top$ to indicate success.

7.1 Definitions for Fully Dynamic Group Signatures

A Fully Dynamic Group Signature ($\mathcal{FDGS}$) scheme involves a set of users who are potential group members and a group manager GM who is in charge of issuing and revoking group membership. The group signature scheme enables group members to sign messages on behalf of the group in an anoymous way, but in case of abuse the group manager can revoke the anonymity and open the signature to reveal the signer.

We are interested in the fully dynamic setting where users can join and leave the group at any time at the discretion of the group manager. In static or partially dynamic group signatures, where members cannot leave, it is possible to fix the group information associated with the group at initialization. For a fully dynamic group, however, there has to be a way to prevent a revoked member from using her old key to sign messages. This means the group information associated with the group must change after revocation. We divide the group information into a permanent group public key gpk and temporary group information info_τ, associated with an index τ referred to as an epoch. The group information depicts changes to the group, for instance, it could include the current members of the group (as in accumulator-based constructions) or those who have been excluded from the group (as in constructions based on revocation lists). As in existing models, we assume that anyone can verify the authenticity of the published group information.

Unlike existing security models for group signatures that assume trusted key generation, we separate key generation from trusted parameter setup. This allows us to define stringent security that protects against adversarial group managers who might generate their keys maliciously. Our definitions can easily be adapted to work for the

weaker setting where the group manager's keys are generated honestly as in the case of existing models.

We give two flavours of our definitions. We start by providing a definition where the roles of opening signatures and administering group membership are overseen by the same authority. Subsequently, we specialise the definition to the setting where each of those roles is overseen by a separate authority.

7.1.1 Syntax

A fully dynamic group signature scheme $\mathcal{FDGS}$ involves a group manager GM and a set of users. Additionally, there might be the presence of a trusted third party that generates some initial parameters used in the scheme. The scheme consists of the following algorithms and data structures:

- Interactive polynomial time protocol run by GM and a user: **Join**
- Probabilistic polynomial time algorithms: **GSetup**, **GKGen**, **UpdateGroup**, **Sign**, **Open**
- Deterministic polynomial time algorithms: **IsActive**, **Verify**, **Judge**
- Data structure: **Reg**.

We will describe in greater details the data structure, the algorithms and their usage in a $\mathcal{FDGS}$ scheme.

Reg: The registry is a data structure, which is filled as users join the group. The group manager associates any joining group members with session identifiers $\mathsf{i} = 1, 2, 3, \ldots$. When user i joins, she is able to store a record reg_i in the registry. Once a record is stored, it cannot be changed. The group manager has read access to the registry and may store the information and use it during opening when tracing the originator of a signature. We model access to the registry with the following two algorithms/oracles:

- $\mathsf{ReadReg}(\mathsf{i})$: On input a session identifier i, it returns the corresponding entry in the registry reg_i. If no record reg_i is stored, it returns $\perp$.

- $\mathsf{WriteReg}(\mathsf{i}, M)$: On input a session identifier i and a message M, it sets $\mathsf{reg}_\mathsf{i} := M$. This oracle can only be used once for every identifier i, further calls with the same session identifier are ignored.

One way to instantiate the registry is with a PKI, which is done explicitly in e.g. [BSZ05]. The registry $\mathbf{Reg}$ can be hosted by GM but it requires each entry reg_i to be signed by the user involved in the i-th instance of the protocol. Whether one instantiates the registry with a PKI or in a different way it is out of the scope of this chapter as long as it gives us the desired functionality.

$\mathbf{GSetup}(1^\lambda) \to \mathsf{param}$: There may be a trusted third party that runs this algorithm to generate public parameters param. In case a trusted setup is not required, this algorithm can be simply regarded as setting $\mathsf{param} := 1^\lambda$.

$\mathbf{GKGen}(\mathsf{param}) \to (\mathsf{out}_\mathsf{GM}; \mathsf{st}_\mathsf{GM})$: The group manager uses this algorithm to generate $\mathsf{out}_\mathsf{GM} := (\mathsf{mpk}, \mathsf{info}_0)$, which consists of the manager's public key and the initial group information, and the resulting state st_GM of the group manager. The group public key is $\mathsf{gpk} := (\mathsf{param}, \mathsf{mpk})$.

Join: To enroll a user as a member, the GM may run the interactive joining protocol with her. Their respective algorithms are:

- $\mathbf{Join}_{\mathsf{User}}^{\mathsf{WriteReg}(\mathsf{i},\cdot)}(M; \mathsf{st}) \to (\mathsf{out}; M_\mathsf{GM}; \mathsf{st})$: This algorithm specifies the user's execution of the interactive joining protocol with the GM. Given an input message from the GM and the user's internal state st it returns a message for the GM and a new state. In its first call, the algorithm is executed on initial input $(\mathsf{init}; \mathsf{gpk})$. In each instance of the protocol, the user is allowed a single call to the oracle $\mathsf{WriteReg}(\mathsf{i}, \cdot)$ for writing into the registration table an entry reg_i corresponding to its identifier i. Joining session i terminates after at most $k(\lambda)$ rounds by a call returning $(\mathsf{gsk}; M_\mathsf{GM}; \mathsf{st})$, which includes the user's secret key $\mathsf{gsk}_\mathsf{i} := \mathsf{gsk}$, an optional final message for the issuer including a termination message done, and the user final state. If it terminates with $\mathsf{gsk} = \perp$, the user will consider it as a fail to join, and on failure it will always be the case that it ends with $M_\mathsf{GM} = (\mathsf{done}, \perp)$. After termination, the user will ignore all future inputs to $\mathbf{Join}_{\mathsf{User}}$.

- $\mathbf{Join}_{\mathsf{GM}}^{\mathsf{ReadReg(i)}}(\mathsf{i}, M_{\mathsf{GM}}; \mathsf{st}_{\mathsf{GM}}) \rightarrow (\mathsf{out}_{\mathsf{GM}}; M; \mathsf{st}_{\mathsf{GM}})$: This algorithm specifies the GM's execution in the interactive joining protocol with a user. The GM keeps track of distinct instances of the protocol using unique identifiers i, which we without loss of generality assume are numbered 1, 2, 3, etc. The algorithm receives as input a session identifier i, a message M_{GM} received from the user, and the GM's internal state and it returns a message M for the user interacting in session i and updates the state $\mathsf{st}_{\mathsf{GM}}$. The algorithm has access to the oracle $\mathsf{ReadReg}(\mathsf{i})$ to read the entry $\mathsf{reg}_{\mathsf{i}}$ in the registration table $\mathbf{Reg}$. Each joining session will terminate after at most a polynomial number of rounds. We let $k(\lambda)$ be the maximal number of rounds before termination. Termination will be indicated in the local output $\mathsf{out}_{\mathsf{GM}}$ of the GM and can be successful ($\top$) or fail ($\bot$), and if it fails the output message will be $M_{\mathsf{i}} = (\mathsf{done}, \bot)$. After termination the GM will ignore future calls with the same i.

For conciseness we will often refer to the user involved in the i-th session of the **Join** protocol with the manager as user i. Please observe though that the user may not be aware of her own session identifier i, since she may not be aware of how many other users are joining or have already joined the group.

UpdateGroup$(\mathcal{R}; \mathsf{st}_{\mathsf{GM}}) \rightarrow (\mathsf{info}; \mathsf{st}_{\mathsf{GM}})$**:** The group manager runs this algorithm to update the group information, where the set $\mathcal{R}$ consists of session identifiers associated with users to be revoked. The algorithm returns new public group information info and updates the state of GM. The group information info may or may not depend on the set of newly joined members of the group, which the group manager records in its internal state. The group information info is intended as group information pertaining to the group and we will in general assume anybody may have access to the sequence $\mathsf{info}_0, \mathsf{info}_1, \ldots$ the group manager creates during the lifetime of the group signature scheme.

IsActive$(\mathsf{i}, \tau, \mathsf{st}_{\mathsf{GM}}) \rightarrow 1/0$**:** The **Join** protocol and the **UpdateGroup** algorithm describe how an honest GM adds and revokes group members. The exact moment when a member is activated and able to sign is design specific. In some constructions, group members are implicitly activated after successfully terminating the

Join protocols and may even be able to sign with respect to previous epochs; in others they are explicitly activated by GM when a new group information info_τ is published. Consequently, different design choices lead to different time spans where members are allowed to sign. In order to take into account these differences in the security definitions without favouring a particular design paradigm, we use the **IsActive** procedure, which should be interpreted as the group manager's policy for when a member is considered active.

The **IsActive** algorithm takes as input the state of the group manager, a session identifier i, and an epoch τ associated with group information info_τ the group manager has published earlier. We refer to a user as an *active* member of the group at epoch τ if and only if the algorithm returns 1. We place the following constraints on the policies an honest group manager can have for when a user is active:

- If τ is not associated with any info_τ the group manager has published, the algorithm returns 0.
- If i is not associated with a joining session where the group manager has terminated succesfully, the algorithm returns 0.
- If i was revoked when creating info_τ for this epoch or earlier, the algorithm returns 0.
- If i is associated with a joining session where the group manager ended her part successfully before info_τ was created, and user i is not revoked at or before epoch τ, the algorithm returns 1.

Sign$(\mathsf{gsk}, \mathsf{info}, m) \to \Sigma$**:** Given a user's group signing key gsk, group information info, and a message m, the signing algorithm outputs a group signature Σ.

Verify$(\mathsf{gpk}, \mathsf{info}, m, \Sigma) \to 1/0$**:** The verification algorithm checks whether Σ is a valid group signature on m with respect to the group information info and outputs a bit: 1 for accept and 0 for reject.

Open$(\mathsf{gpk}, \mathsf{st}_{\mathsf{GM}}, \mathsf{info}, m, \Sigma) \to (\mathsf{i}, \pi)$**:** The opening algorithm receives as input the group public key gpk, the state of the group manager, some public group information info, a message, and a signature. It returns a session identifier i together with a

proof π attributing Σ to user i. If the algorithm is unable to open the signature to a particular group member, it returns $(\perp, \pi)$ to indicate that it could not attribute the signature.

Judge$(\mathsf{gpk}, \mathsf{info}, \mathsf{reg}, m, \Sigma, \pi) \rightarrow 1/0$**:** The judge algorithm checks the validity of a proof π attributing the signature Σ on m under group information info to a user with registry record reg. It outputs 1 for accept and 0 for reject.

Separating the Role of Group Manager and Opening Authority

Following the suggestion of Camenisch and Michels [CM98], Bellare, Shi and Zhang [BSZ05] separate the group manager role we described above into two parts: a group manager GM (who they call the Issuer) administrating group membership, and an opening authority OA (who they call the Opener) capable of tracing the signer of a message. There are natural settings where such a division of roles may be called for; an organization may for instance consider group management the task of the human resources department, and opening signatures the domain of the fraud department. While the separation of the group manager and opening authority roles complicate definitions a little, they also have the advantage of permitting more fine-grained security notion that allow for adversarial behaviour in either the group manager or the opening authority.

When separating out the role of the opening authority, the syntax of a group signature scheme changes. Key generation can now be seen as a joint process that involves both the group manager and the opening authority. Sometimes it may be desirable to minimise interaction and allow authorities to generate their own keys independently, but for maximal generality we define key generation as an interactive protocol between them, where independent key generation is a special case. Another change is that since the opening algorithm is run by the opening authority it needs access to read the registry to know who to attribute a given signature to. We describe these changes below:

GKGen: To generate the group public key the GM and OA may run an interactive protocol. Their respective algorithms are:

- $\mathbf{GKGen}_{\mathsf{GM}}(M_{\mathsf{GM}}; \mathsf{st}_{\mathsf{GM}}) \rightarrow (\mathsf{out}_{\mathsf{GM}}; M_{\mathsf{OA}}; \mathsf{st}_{\mathsf{GM}})$: This algorithm specifies the GM's execution in the interactive key generation protocol with the OA. It gets as input a message M_{GM} received from the OA and the GM's internal state and returns an output $\mathsf{out}_{\mathsf{GM}}$, a message M_{OA} for the OA and updates the state to $\mathsf{st}_{\mathsf{GM}}$. The state of the group manager is initialised as $\mathsf{st}_{\mathsf{GM}} := \mathsf{param}$. If the GM initiates the protocol, the input message is initialised as $M_{\mathsf{GM}} := \mathsf{init}$. In a successful execution of the protocol, the last call of the algorithm returns a non-empty output value $\mathsf{out}_{\mathsf{GM}} := (\mathsf{mpk}, \mathsf{info}_0)$ consisting of the GM public key and the initial group information. After termination, subsequent calls to the algorithm will be ignored.
- $\mathbf{GKGen}_{\mathsf{OA}}(M_{\mathsf{OA}}; \mathsf{st}_{\mathsf{OA}}) \rightarrow (\mathsf{out}_{\mathsf{OA}}; M_{\mathsf{GM}}; \mathsf{st}_{\mathsf{OA}})$: This algorithm specifies the OA's execution in the interactive key generation protocol with the GM. It gets as input a message from the GM and the OA's internal state and it returns an output $\mathsf{out}_{\mathsf{GM}}$, a message M_{GM} for the GM and updates the state $\mathsf{st}_{\mathsf{OA}}$. The state of the OA is initialised as $\mathsf{st}_{\mathsf{OA}} := \mathsf{param}$. If the OA initiates the protocol the input message is initialised as $M_{\mathsf{OA}} := \mathsf{init}$. In a successful execution of the protocol, the last call of the algorithm returns an non-empty output value $\mathsf{out}_{\mathsf{OA}} := (\mathsf{opk}, \mathsf{osk})$ consisting of the OA's public and secret keys. Subsequent calls of the algorithm are ignored.

We denote an entire execution of the key generation protocol as

$$((\mathsf{mpk}, \mathsf{info}_0; \mathsf{st}_{\mathsf{GM}}); (\mathsf{opk}, \mathsf{osk})) \leftarrow \langle \mathbf{GKGen}_{\mathsf{GM}}(\mathsf{param}); \mathbf{GKGen}_{\mathsf{OA}}(\mathsf{param}) \rangle$$

and let $\mathsf{gpk} := (\mathsf{param}, \mathsf{mpk}, \mathsf{opk})$ be the group public key.

$\mathbf{Open}^{\mathsf{ReadReg}(\cdot)}(\mathsf{gpk}, \mathsf{osk}, \mathsf{info}, m, \Sigma) \rightarrow (\mathsf{i}, \pi)$**:** The opening algorithm receives as input the group public key gpk, the opening key osk, group information info, a message, and a signature. It returns a session identifier i together with a proof π attributing Σ to user i. If the algorithm is unable to open the signature to a particular group member, it returns $(\perp, \pi)$ to indicate that it could not attribute the signature.

Relation Between Definitions with Single and Separate Authorities

Group signatures with and without separate authorities are closely related. Specifically, given a group signature scheme $\mathcal{FDGS}$ for separate GM and OA, we can define a single authority group signature scheme $\mathcal{FDGS}'$, where the group manager GM′ first runs the interaction $((\mathsf{mpk}, \mathsf{info}_0; \mathsf{st}_{\mathsf{GM}}); (\mathsf{opk}, \mathsf{osk})) \leftarrow \langle \mathbf{GKGen}_{\mathsf{GM}}(\mathsf{param}); \mathbf{GKGen}_{\mathsf{OA}}(\mathsf{param})\rangle$ and returns the manager public key $\mathsf{mpk}' := (\mathsf{mpk}, \mathsf{opk})$ and sets the state to be $\mathsf{st}'_{\mathsf{GM}} := (\mathsf{st}_{\mathsf{GM}}, \mathsf{osk})$. Whenever a new user joins the group, GM has access to an oracle that allows it to read the corresponding record in the registry, and we imagine GM′ keeps track of these registry entries so it has its own virtual ReadReg oracle. Whenever GM′ has to run the opening algorithm, it will then just use osk. All other algorithms in $\mathcal{FDGS}'$ are defined in the natural way from $\mathcal{FDGS}$. It is easy to see that this transformation preserves the efficiency of $\mathcal{FDGS}$ and we will in the following argue that security is preserved as well.

7.1.2 Security Definitions

Definition 7.1. *An $\mathcal{FDGS}$ with the syntax in Section 7.1.1 is a fully dynamic group signature if it is correct, anonymous, traceable and non-frameable as defined below.*

Correctness

Correctness guarantees that an honest user can enrol in the group and produce signatures that are accepted by the **Verify** algorithm. We assume in the correctness definition that GM is honest and willing to enrol the user, since otherwise it could just refuse membership. However, there may be other users that are malicious. We therefore model correctness as an adversarial game, where the adversary acts on behalf of all other users and we require that the honest user can successfully enrol and sign messages. The correctness definition captures three aspects in this setting:

- An honest user interacting with an honest GM should be able to enroll in at most $k(\lambda)$ rounds after which the user terminates successfully with a key gsk_i.
- The group manager should before or in the same round terminate with success indicator $\top$ and activate the user no later than the next update

- Once activated the user should be able to sign messages.

These three properties should hold as long as the user is not revoked.

In the game $\mathbf{Exp}^{\mathsf{Corr}}_{\mathcal{FDGS},\mathcal{A}}$ (shown in Figure 7.1), we grant the adversary $\mathcal{A}$ access to the following oracles, details of which are given in Figure 7.2. We maintain several global counters: h is the joining session identifier of the honest user which is initialised as $\perp$, N is the number of users that initiated the **Join** protocol with the GM, and τ_{Current} is the current epoch, τ_{Join} is the epoch during which the user created her key, τ_{Revoke} is the time the user was revoked (if ever), and K is the number of calls to the AddHU oracle, i.e., the number of rounds executed by the honest user in the **Join** protocol.

AddHU(): This oracle adds a single honest user to the group. Each call of the oracle executes the next round of interaction in the **Join** protocol between the honest user and the honest group manager. It returns the exchanged messages as well as the outputs of both parties. Note that the adversary learns the group signing key of the honest user at the successful conclusion of the interaction.

SndToM($\mathsf{i}, M_{\mathsf{GM}}$): This oracle allows the adversary to add a corrupt user to the group. The adversary can deviate from the **Join** protocol by sending arbitrary messages M_{GM} to the GM. Each oracle call executes the next move of an honest GM on input message M_{GM} in the i-th instance of the **Join** protocol. It returns the GM response message and output.

Update($\mathcal{R}$): This oracle allows the adversary to update the public group information. Here $\mathcal{R}$ is the set of the group members to be removed from the group. Calling this oracle triggers a new epoch.

Write(i, M): Given a session identifier and a message M, the oracle sets $\mathsf{reg}_{\mathsf{i}} := M$. The oracle can only be used once for every identifier i, further calls to it return $\perp$ without producing changes to the registry. It cannot be called on the identifier corresponding to the joining session initiated by AddHU.

State(): This oracle returns the current GM's state $\mathsf{st}_{\mathsf{GM}}$.

In the correctness definition, we restrict the adversary to enrolling a single honest user into the group via calling the AddHU oracle. This generalises to the case of multiple honest users via a standard hybrid argument.

Experiment: $\mathbf{Exp}^{\mathsf{Corr}}_{\mathcal{FDGS},\mathcal{A}}(\lambda)$

– $\mathsf{h} := \bot; N := 0; K := 0; \tau_{\mathsf{Current}} := 0; \tau_{\mathsf{Join}} := \infty; \tau_{\mathsf{Revoke}} := \infty$
– $\mathsf{param} \leftarrow \mathbf{GSetup}(1^\lambda)$
– $(\mathsf{mpk}, \mathsf{info}_0; \mathsf{st}_{\mathsf{GM}}) \leftarrow \mathbf{GKGen}(\mathsf{param})$
– $\mathsf{gpk} := (\mathsf{param}, \mathsf{mpk})$
– $(m, \tau) \leftarrow \mathcal{A}^{\mathsf{AddHU},\mathsf{SndToM},\mathsf{Update},\mathsf{Write},\mathsf{State}}(\mathsf{gpk}, \mathsf{info}_0)$
– If $K = k(\lambda)$ and $\tau_{\mathsf{Revoke}} = \infty$ and $\tau_{\mathsf{Join}} = \infty$ return 0
– If $\mathsf{h} = \bot$ or $\tau > \tau_{\mathsf{Current}}$ return 1
– If $\tau_{\mathsf{Join}} < \tau < \tau_{\mathsf{Revoke}}$ and $\mathbf{IsActive}(\mathsf{h}, \tau, \mathsf{st}_{\mathsf{GM}}) = 0$ return 0
– If $\mathbf{IsActive}(\mathsf{h}, \tau, \mathsf{st}_{\mathsf{GM}}) = 0$ return 1
– $\Sigma \leftarrow \mathbf{Sign}(\mathsf{gsk}_{\mathsf{h}}, \mathsf{info}_\tau, m)$
– Return $\mathbf{Verify}(\mathsf{gpk}, \mathsf{info}_\tau, m, \Sigma)$

FIGURE 7.1: Correctness game.

AddHU()
. If $K = k(\lambda)$ return $\bot$
. $K := K + 1$
. If $\mathsf{h} = \bot$:
  ∘ $N := N + 1; \mathsf{h} := N$
  ∘ $M_{\mathsf{h}} := \mathsf{init}; \mathsf{st}_{\mathsf{h}} := \mathsf{gpk}$
  ∘ $\mathsf{gsk}_{\mathsf{h}} := \bot$
. $(\mathsf{out}_{\mathsf{h}}; M_{\mathsf{GM}}; \mathsf{st}_{\mathsf{h}}) \leftarrow \mathbf{Join}^{\mathsf{WriteReg}(\mathsf{h},\cdot)}_{\mathsf{User}}(M_{\mathsf{h}}; \mathsf{st}_{\mathsf{h}})$
. If $\mathsf{out}_{\mathsf{h}} \neq \varepsilon$:
  ∘ $\mathsf{gsk}_{\mathsf{h}} := \mathsf{out}_{\mathsf{h}}$
  ∘ $\tau_{\mathsf{Join}} = \tau_{\mathsf{Current}}$
  ∘ $K = k(\lambda)$ // maximal number of rounds
. $(\mathsf{out}_{\mathsf{GM}}; M_{\mathsf{h}}; \mathsf{st}_{\mathsf{GM}}) \leftarrow \mathbf{Join}^{\mathsf{ReadReg}(\mathsf{h})}_{\mathsf{GM}}(\mathsf{h}, M_{\mathsf{GM}}; \mathsf{st}_{\mathsf{GM}})$
. Return $(\mathsf{out}_{\mathsf{h}}, M_{\mathsf{GM}}), (\mathsf{out}_{\mathsf{GM}}, M_{\mathsf{h}})$

State()
. Return $\mathsf{st}_{\mathsf{GM}}$

SndToM($\mathsf{i}, M_{\mathsf{GM}}$)
. If $\mathsf{i} \notin [N+1] \vee \mathsf{i} = \mathsf{h}$ return $\bot$
. If $\mathsf{i} = N + 1$:
  ∘ $N := N + 1$
. $(\mathsf{out}_{\mathsf{i}}; M_{\mathsf{i}}; \mathsf{st}_{\mathsf{GM}}) \leftarrow \mathbf{Join}^{\mathsf{ReadReg}(\mathsf{i})}_{\mathsf{GM}}(\mathsf{i}, M_{\mathsf{GM}}; \mathsf{st}_{\mathsf{GM}})$
. Return $(\mathsf{out}_{\mathsf{i}}, M_{\mathsf{i}})$

Update($\mathcal{R}$)
. If $\mathcal{R} \not\subseteq [N]$ return $\bot$
. $(\mathsf{info}; \mathsf{st}_{\mathsf{GM}}) \leftarrow \mathbf{UpdateGroup}(\mathcal{R}; \mathsf{st}_{\mathsf{GM}})$
. $\tau_{\mathsf{Current}} := \tau_{\mathsf{Current}} + 1$
. If $\mathsf{h} \in \mathcal{R}$ and $\tau_{\mathsf{Revoke}} = \infty$ set $\tau_{\mathsf{Revoke}} = \tau_{\mathsf{Current}}$
. Return $\mathsf{info}_{\tau_{\mathsf{Current}}} := \mathsf{info}$

Write(i, M)
. If $\mathsf{i} = \mathsf{h}$ or $\mathsf{reg}_{\mathsf{i}} \neq \bot$ return $\bot$
. Set $\mathsf{reg}_{\mathsf{i}} := M$

FIGURE 7.2: Oracles used in the correctness game.

Experiment: $\mathbf{Exp}^{\mathsf{Corr}}_{\mathcal{FDGS},\mathcal{A}}(\lambda)$

- $\mathsf{h} := \perp; N := 0; K := 0; \tau_{\mathsf{Current}} := 0; \tau_{\mathsf{Join}} := \infty; \tau_{\mathsf{Revoke}} := \infty$
- $\mathsf{param} \leftarrow \mathbf{GSetup}(1^\lambda)$
- $\Big((\mathsf{mpk}, \mathsf{info}_0; \mathsf{st}_{\mathsf{GM}}); (\mathsf{opk}; \mathsf{st}_{\mathcal{A}})\Big) \leftarrow \langle \mathbf{GKGen}_{\mathsf{GM}}(\mathsf{param}); \mathcal{A}(\mathsf{param})\rangle$
- $\mathsf{gpk} := (\mathsf{param}, \mathsf{mpk}, \mathsf{opk})$
- $(m, \tau) \leftarrow \mathcal{A}^{\mathsf{AddHU},\mathsf{SndToM},\mathsf{Update},\mathsf{Write},\mathsf{State}}(\mathsf{gpk}, \mathsf{info}_0; \mathsf{st}_{\mathcal{A}})$
- If $K = k(\lambda)$ and $\tau_{\mathsf{Revoke}} = \infty$ and $\tau_{\mathsf{Join}} = \infty$ return 0
- If $\mathsf{h} = \perp$ or $\tau > \tau_{\mathsf{Current}}$ return 1
- If $\tau_{\mathsf{Join}} < \tau < \tau_{\mathsf{Revoke}}$ and $\mathbf{IsActive}(\mathsf{h}, \tau, \mathsf{st}_{\mathsf{GM}}) = 0$ return 0
- If $\mathbf{IsActive}(\mathsf{h}, \tau, \mathsf{st}_{\mathsf{GM}}) = 0$ return 1
- $\Sigma \leftarrow \mathbf{Sign}(\mathsf{gsk}_{\mathsf{h}}, \mathsf{info}_\tau, m)$
- Return $\mathbf{Verify}(\mathsf{gpk}, \mathsf{info}_\tau, m, \Sigma)$

FIGURE 7.3: Correctness game for separate GM and OA.

Definition 7.2 (Correctness)**.** *An $\mathcal{FDGS}$ scheme is* correct *if for any PPT adversary $\mathcal{A}$*

$$\Pr[\mathbf{Exp}^{\mathsf{Corr}}_{\mathcal{FDGS},\mathcal{A}}(\lambda) = 1] \approx 1$$

If the definition holds also for unbounded adversaries, we say the $\mathcal{FDGS}$ scheme is statistically correct, and if the probability is exactly 1 we say the $\mathcal{FDGS}$ scheme is perfectly correct.

Variations of Correctness. In the correctness definition we give the adversary access to the state of the group manager. This means even if the honest group manager's secret data is leaked, an honest user can still enroll and sign messages as long as the group manager considers her active. A more relaxed but still reasonable definition of correctness would be to assume the group manager keeps her state secret. In this latter case, it is then natural to give the adversary access to an opening oracle to model that the group manager may sometimes trace a member who produced a signature.

In the case where there is a separation between the group manager and an opening authority, we still need the group manager to be honest for correctness to make sense. However, we may want correctness to hold even in the presence of a malicious opening authority; since it is the sovereign domain of the group manager to decide who is an active member and should be able to sign messages. In Fig. 7.3 we therefore define the correctness game with an adversarial opening authority. It is straightforward to see that given a group signature scheme $\mathcal{FDGS}$ with separate GM and OA satisfying correctness, the transformation from Section 7.1.1 gives a group signature scheme

$\mathcal{FDGS}'$ with a single group manager that is correct too.

Many definitions of correctness found in the literature [BMW03; BSZ05; KY06] encompass not just that an honestly generated signature is accepted by the verification algorithm but also add other requirements such as an honestly generated signature should be opened to the honest signer who generated it. In our definition of correctness we only require that honestly generated signatures are accepted and refer to other security definitions to handle additional requirements that we consider less central. In particular, for the property of honestly generated signatures opening to the correct signer, it is captured by the traceability and opening soundness properties we later define, and captured in a much stronger sense than usually done in correctness definitions since we explicitly consider opening soundness in a highly adversarial setting instead of just considering an honest interaction. This leaves a small definitional gap when considering *perfect* correctness since traceability and opening soundness may only hold computationally. However, we find the difference to be insignificant and have therefore deliberately opted for the minimal and simplest definition of correctness that only demands honest generated signatures to be accepted.

Anonymity

Group signatures should be anonymous and not reveal the identity of the group member who produced them. Since the group manager has the ability to trace signers we must assume the group manager to be honest for anonymity to hold but some of the other users may be malicious.

In the game $\mathbf{Exp}^{\mathsf{Anon-b}}_{\mathcal{FDGS},\mathcal{A}}$ (shown in Figure 7.4), we maintain the following counter and lists: N is the number of users that initiated the **Join** protocol with the GM, $\mathcal{H}$ is a list of honest users, and $\mathcal{C}$ is a list of challenge signatures obtained from the challenge oracle. We give the adversary access to the following oracles, details of which are given in Figure 7.5

AddHU(i) : This oracle allows the adversary to add honest users to the group by going through the join protocol one round at a time. The oracle models full key exposure, both communication and the user's signing key are leaked to the adversary.

$\mathsf{SndToM}(\mathsf{i}, M_{\mathsf{GM}})$: This oracle allows the adversary to add corrupt users to the group. The adversary can deviate from the **Join** protocol by sending arbitrary messages M_{GM} to the GM. Each oracle call executes the next move of an honest GM on input message M_{GM} in the i-th instance of the **Join** protocol. It returns the GM response message and output.

$\mathsf{Chal}_b(\mathsf{info}, m, \mathsf{i}_0, \mathsf{i}_1)$: This a left-right oracle for defining anonymity. It takes as input some group information info, a message m, and two honest users $\mathsf{i}_0, \mathsf{i}_1$. It returns a group signature on the message using key $\mathsf{gsk}_{\mathsf{i}_b}$ for $b \leftarrow \{0,1\}$ and the given group information. It is required that both challenge users are able to sign with respect to info. The adversary can only call this oracle once.

$\mathsf{Open}(\mathsf{info}, m, \Sigma)$: Returns the session identifier i of the signer who produced signature Σ on m with respect to info, together with a proof π. The oracle cannot be called on a signature obtained from the Chal_b oracle.

$\mathsf{ReadReg}(\mathsf{i})$: Given a session identifier i, it returns the corresponding entry in the registry $\mathsf{reg}_{\mathsf{i}}$.

$\mathsf{Update}(\mathcal{R})$: Allows the adversary to prompt a group information update and increment the epoch. Here $\mathcal{R}$ is a set of the group members to be revoked from the group.

The adversary can interact with honest users and join corrupt users. At some point, the adversary picks two honest members of the group at a chosen epoch, gets a signature from one of them, and tries to learn which of them has signed a chosen message. The goal of the adversary is to guess which member signed the message. For simplicity, the adversary is only allowed a single challenge query but a standard hybrid argument (similar to that used in [BSZ05]) shows this is equivalent to seeing many challenge signatures. Our definition covers full key exposure attacks by allowing $\mathcal{A}$ to learn the secret keys of all users in the group.

Definition 7.3 (Anonymity)**.** *An $\mathcal{FDGS}$ scheme is* anonymous *if for all PPT adversaries $\mathcal{A}$ the following advantage is negligible*

$$\mathbf{Adv}^{\mathsf{Anon}}_{\mathcal{FDGS},\mathcal{A}}(\lambda) := \left|\Pr[\mathbf{Exp}^{\mathsf{Anon}-0}_{\mathcal{FDGS},\mathcal{A}}(\lambda) = 1] - \Pr[\mathbf{Exp}^{\mathsf{Anon}-1}_{\mathcal{FDGS},\mathcal{A}}(\lambda) = 1]\right|.$$

Experiment: $\mathbf{Exp}^{\mathsf{Anon}-b}_{\mathcal{FDGS},\mathcal{A}}(\lambda)$
– $\mathsf{param} \leftarrow \mathbf{GSetup}(1^\lambda); N := 0; \mathcal{H} := \emptyset; \mathcal{C} := \emptyset$
– $(\mathsf{mpk}, \mathsf{info}_0; \mathsf{st}_{\mathsf{GM}}) \leftarrow \mathbf{GKGen}(\mathsf{param})$
– $\mathsf{gpk} := (\mathsf{param}, \mathsf{mpk})$
– Return $b^* \leftarrow \mathcal{A}^{\mathsf{AddHU},\mathsf{SndToM},\mathsf{Open},\mathsf{Chal}_b,\mathsf{ReadReg},\mathsf{Update}}(\mathsf{gpk}, \mathsf{info}_0)$

FIGURE 7.4: Anonymity game.

AddHU(i)
. If $\mathsf{i} \notin [N+1]$ return $\perp$
. If $\mathsf{i} = N+1$
∘ $\mathcal{H} := \mathcal{H} \cup \{\mathsf{i}\}$
∘ $N := N+1$
∘ $M_\mathsf{i} := \mathsf{init}$
∘ $\mathsf{st}_\mathsf{i} := \mathsf{gpk}$
∘ $\mathsf{gsk}_\mathsf{i} := \perp$
. $(\mathsf{out}_\mathsf{i}; M_{\mathsf{GM}}; \mathsf{st}_\mathsf{i}) \leftarrow \mathbf{Join}^{\mathsf{WriteReg}(\mathsf{i},\cdot)}_{\mathsf{User}}(M_\mathsf{i}; \mathsf{st}_\mathsf{i})$
. If $\mathsf{out}_\mathsf{i} \neq \varepsilon$:
∘ $\mathsf{gsk}_\mathsf{i} := \mathsf{out}_\mathsf{i}$
. $(\mathsf{out}_{\mathsf{GM}}; M_\mathsf{i}; \mathsf{st}_{\mathsf{GM}}) \leftarrow \mathbf{Join}^{\mathsf{ReadReg}(\mathsf{i})}_{\mathsf{GM}}(\mathsf{i}, M_{\mathsf{GM}}; \mathsf{st}_{\mathsf{GM}})$
. Return $(\mathsf{out}_\mathsf{i}, M_{\mathsf{GM}}), (\mathsf{out}_{\mathsf{GM}}, M_\mathsf{i})$

ReadReg(i)
. Return reg_i

Update($\mathcal{R}$)
. If $\mathcal{R} \not\subseteq [N]$ return $\perp$
. $(\mathsf{info}; \mathsf{st}_{\mathsf{GM}}) \leftarrow \mathbf{UpdateGroup}(\mathcal{R}; \mathsf{st}_{\mathsf{GM}})$
. $\tau_{\mathsf{Current}} := \tau_{\mathsf{Current}} + 1$
. Return $\mathsf{info}_{\tau_{\mathsf{Current}}} := \mathsf{info}$

SndToM(i, M_{GM})
. If $\mathsf{i} \notin [N+1] \vee \mathsf{i} \in \mathcal{H}$ return $\perp$
. If $\mathsf{i} = N+1$:
∘ $N := N+1$
. $(\mathsf{out}_\mathsf{i}; M_\mathsf{i}; \mathsf{st}_{\mathsf{GM}}) \leftarrow \mathbf{Join}^{\mathsf{ReadReg}(\mathsf{i})}_{\mathsf{GM}}(\mathsf{i}, M_{\mathsf{GM}}; \mathsf{st}_{\mathsf{GM}})$
. Return $(\mathsf{out}_\mathsf{i}, M_\mathsf{i})$

$\mathsf{Chal}_b(\mathsf{info}, m, \mathsf{i}_0, \mathsf{i}_1)$
. If $\{\mathsf{i}_0, \mathsf{i}_1\} \not\subseteq \mathcal{H}$ return $\perp$
. $\Sigma_0 \leftarrow \mathbf{Sign}(\mathsf{gsk}_{\mathsf{i}_0}, \mathsf{info}, m)$
. $\Sigma_1 \leftarrow \mathbf{Sign}(\mathsf{gsk}_{\mathsf{i}_1}, \mathsf{info}, m)$
. If $\mathbf{Verify}(\mathsf{gpk}, \mathsf{info}, m, \Sigma_0) = 0$ return $\perp$
. If $\mathbf{Verify}(\mathsf{gpk}, \mathsf{info}, m, \Sigma_1) = 0$ return $\perp$
. $\mathcal{C} := \{(\mathsf{info}, m, \Sigma_b)\}$
. Return Σ_b

Open(info, m, Σ)
. If $(\mathsf{info}, m, \Sigma) \in \mathcal{C}$ return $\perp$
. If $\mathbf{Verify}(\mathsf{gpk}, \mathsf{info}, m, \Sigma) = 0$ return $\perp$
. Return $\mathbf{Open}(\mathsf{gpk}, \mathsf{st}_{\mathsf{GM}}, \mathsf{info}, m, \Sigma)$.

FIGURE 7.5: Oracles used in the anonymity game.

Variations of Anonymity. Our definition of anonymity corresponds to what Bellare et al. [BMW03] call full anonymity. Their usage of *full* anonymity emphasizes that anonymity holds even in the presence of an adversary that sees the signing keys of honest users and has access to an opening oracle. Our definition captures full anonymity, not only does the adversary see every honest user's signing key but she also sees the entire joining transcript. Full anonymity gives strong security guarantees since it ensures that even if a user's secret key is leaked her past or future signatures still do not reveal her identity.

Group signatures with full anonymity imply the existence of IND-CPA and IND-CCA secure public-key encryption, as shown in [AW04] and [CG04], respectively. The anonymity notion therefore has to be relaxed if one wants to build group signatures based on one-way functions as is done in [CG04]. Such a relaxation can consist in not giving out_i to the adversary in the AddHU oracle but instead give the adversary access to a signing oracle that will allow it to get signatures from honest users on any

message of its choosing.

Boneh, Boyen and Shacham [BBS04] define another relaxed form of anonymity where the adversary does not have access to the Open oracle. This relaxation is analogous to the distinction between IND-CPA and IND-CCA secure public-key encryption, and indeed they refer to the notion as CPA-anonymity.

Let us now consider the case where we separate the roles of managing the group, GM, and the role of opening signatures, OA. For anonymity to hold, we need the opening authority to be honest, however, we may desire security against a malicious group manager. We give the corresponding anonymity game in Fig. 7.6. Please observe that the oracles the adversary has access to are different because when the adversary runs the group manager it can directly manage the joining interaction with honest users, simulate the enrolment of corrupt users, and compute group information updates by itself. Therefore we remove the Update, SndToM oracles and replace the AddHU oracle with an SndToU oracle that lets the adversarially controlled group manager communicate with an honest user trying to join the group.

Experiment: $\mathbf{Exp}^{\mathsf{Anon}-b}_{\mathcal{FDGS},\mathcal{A}}(\lambda)$
- $\mathsf{param} \leftarrow \mathbf{GSetup}(1^\lambda); N := 0; \mathcal{C} := \emptyset$
- $\Big((\mathsf{mpk}; \mathsf{st}_\mathcal{A}); (\mathsf{opk}, \mathsf{osk})\Big) \leftarrow \langle \mathcal{A}(\mathsf{param}); \mathbf{GKGen}_{\mathsf{OA}}(\mathsf{param})\rangle$
- $\mathsf{gpk} := (\mathsf{param}, \mathsf{mpk}, \mathsf{opk})$
- Return $b^* \leftarrow \mathcal{A}^{\mathsf{SndToU},\mathsf{Open},\mathsf{Chal}_b,\mathsf{ReadReg}}(\mathsf{gpk}; \mathsf{st}_\mathcal{A})$

FIGURE 7.6: Anonymity game for separate GM and OA.

$\underline{\mathsf{SndToU}(\mathsf{i}, M_\mathsf{i})}$
- If $\mathsf{i} \notin [N+1]$ return $\perp$
- If $\mathsf{i} = N+1$:
  - $N := N+1$
  - $M_\mathsf{i} := \mathsf{init}$
  - $\mathsf{st}_\mathsf{i} := \mathsf{gpk}$
- $(\mathsf{out}_\mathsf{i}; M_{\mathsf{GM}}; \mathsf{st}_\mathsf{i}) \leftarrow \mathbf{Join}^{\mathsf{WriteReg}(\mathsf{i},\cdot)}_{\mathsf{User}}(M_\mathsf{i}; \mathsf{st}_\mathsf{i})$
- If $\mathsf{out}_\mathsf{i} \neq \varepsilon \wedge \mathsf{out}_\mathsf{i} \neq \perp$:
  - $\mathsf{gsk}_\mathsf{i} := \mathsf{out}_\mathsf{i}$
- Return $(\mathsf{out}_\mathsf{i}, M_{\mathsf{GM}})$

$\underline{\mathsf{ReadReg}(\mathsf{i})}$
- Return reg_i

$\underline{\mathsf{Chal}_b(\mathsf{info}, m, \mathsf{i}_0, \mathsf{i}_1)}$
- If $\{\mathsf{i}_0, \mathsf{i}_1\} \not\subseteq [N]$ return $\perp$
- $\Sigma_0 \leftarrow \mathbf{Sign}(\mathsf{gsk}_{\mathsf{i}_0}, \mathsf{info}, m)$
- $\Sigma_1 \leftarrow \mathbf{Sign}(\mathsf{gsk}_{\mathsf{i}_1}, \mathsf{info}, m)$
- If $\mathbf{Verify}(\mathsf{gpk}, \mathsf{info}, m, \Sigma_0) = 0$ return $\perp$
- If $\mathbf{Verify}(\mathsf{gpk}, \mathsf{info}, m, \Sigma_1) = 0$ return $\perp$
- $\mathcal{C} := \{(\mathsf{info}, m, \Sigma_b)\}$
- Return Σ_b

$\underline{\mathsf{Open}(\mathsf{info}, m, \Sigma)}$
- If $(\mathsf{info}, m, \Sigma) \in \mathcal{C}$ return $\perp$
- If $\mathbf{Verify}(\mathsf{gpk}, \mathsf{info}, m, \Sigma) = 0$ return $\perp$
- Return $\mathbf{Open}^{\mathsf{ReadReg}}(\mathsf{gpk}, \mathsf{osk}, \mathsf{info}, m, \Sigma)$

FIGURE 7.7: Oracles used in the anonymity game for separate GM and OA.

Let $\mathcal{FDGS}$ be a group signature scheme with separate GM and OA with full anonymity, and let $\mathcal{FDGS}'$ be the resulting single authority group signature scheme resulting from the transformation given in Section 7.1.1. Then $\mathcal{FDGS}'$ is anonymous according to the single authority definition because an $\mathcal{FDGS}$ adversary can as a special case run an honest GM algorithm and simulate everything that happens in an attack against $\mathcal{FDGS}'$. In particular, by running GM honestly, it can simulate the SndToM oracle, and use the SndToU oracle to build a simulated AddHU oracle.

Traceability

Traceability protects the group manager by ensuring that all signatures that are valid for a given epoch can be opened to an active member of the group. In the traceability game $\mathbf{Exp}^{\mathsf{Trace}}_{\mathcal{FDGS},\mathcal{A}}$ shown in Figure 7.8, we maintain a counter N for the number of users that initiated the **Join** protocol with the GM and the adversary $\mathcal{A}$ has access to the following oracles, details of which are given in Figure 7.9.

$\mathsf{SndToM}(\mathsf{i}, M_{\mathsf{GM}})$: This oracle allows the adversary to add corrupt users to the group. She can deviate from the **Join** protocol by sending arbitrary messages M_{GM} to GM. Each oracle call executes the next move of an honest GM on input message M_{GM} in the i-th instance of the **Join** protocol. It returns the GM output and response message.

$\mathsf{Update}(\mathcal{R})$: This oracle allows the adversary to trigger a group information update and increment the epoch. Here $\mathcal{R}$ is the set of the group members to be removed from the group.

$\mathsf{WriteReg}(\mathsf{i}, M)$: Given a session identifier and a message M, the oracle sets $\mathsf{reg}_{\mathsf{i}} := M$. The oracle can only be used once for every identifier i.

$\mathsf{Open}(\mathsf{info}, m, \Sigma)$: Returns the session identifier i of the signer who produced signature Σ on m with respect to info, together with a proof π.

The adversary can add and revoke corrupt users to the group and request the OA to open any signature. The goal of the adversary in the traceability game is to produce a valid signature on a message for a given epoch, such that either the signature traces

back to a user which was not active at that epoch, or the proof of opening produced by the honest OA does not verify.

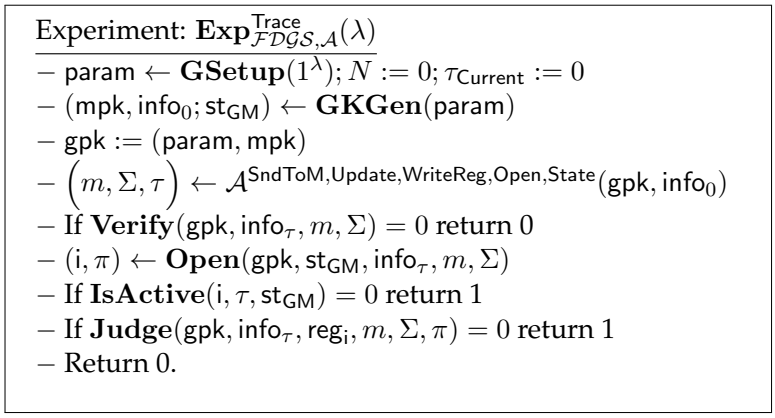

FIGURE 7.8: Traceability game.

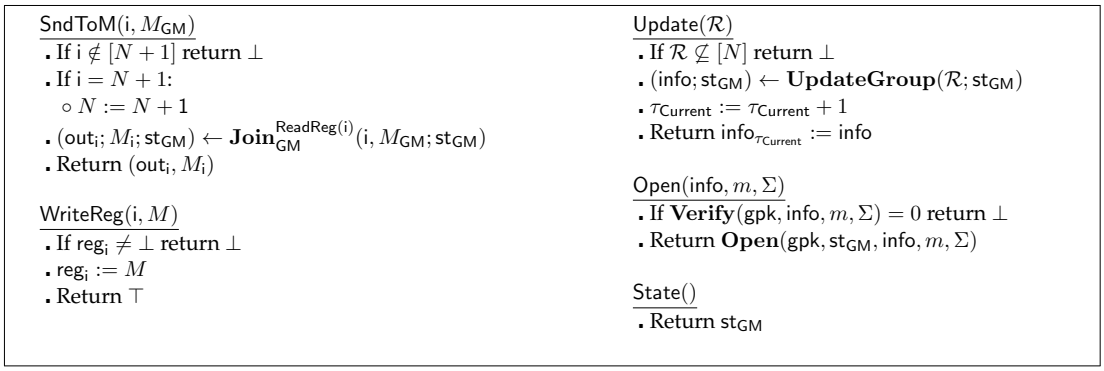

FIGURE 7.9: Oracles used in the traceability game.

Definition 7.4 (Traceability)**.** *An $\mathcal{FDGS}$ scheme is* traceable *if for all PPT adversaries $\mathcal{A}$, the following advantage is negligible*

$$\mathbf{Adv}^{\mathsf{Trace}}_{\mathcal{FDGS},\mathcal{A}}(\lambda) := \Pr[\mathbf{Exp}^{\mathsf{Trace}}_{\mathcal{FDGS},\mathcal{A}}(\lambda) = 1].$$

Variations of Traceability. Bellare, Micciancio and Warinschi [BMW03] defined traceability purely with respect to identifying a signer but did not require a proof of correct opening. Bellare, Shi and Zhang [BSZ05] included the use of a proof for correct opening that can be verified by anybody using the **Judge** algorithm. If we trust the group manager instead of requiring a proof of correct opening, the defining game in Fig. 7.8 can be simplified by eliminating the proof and the **Judge** algorithm.

By necessity, the group manager needs to be at least partially trusted since otherwise it can just enrol some dummy member that can then sign arbitrary messages and

act as a scapegoat. However, we have defined traceability such that it holds even if the honest group managers secret state is leaked. A reasonable relaxation of our definitions would be to trust the group manager to keep its state secret, in which case we would remove the State oracle.

Let us consider the case where we separate the roles of group manager and opening authority. The opening authority is the primary stakeholder that wants to ensure signers can be traced. However, we define security strongly by requiring traceability even in case its secret opening key is leaked. As in the single authority setting, we still need to have some trust in the group manager to keep track of who is active in the group and not to enrol dummy members, however, again we opt for a strong definition of security where its state may be leaked.

Experiment: $\mathbf{Exp}^{\mathsf{Trace}}_{\mathcal{FDGS},\mathcal{A}}(\lambda)$
- $\mathsf{param} \leftarrow \mathbf{GSetup}(1^\lambda); N := 0; \tau_{\mathsf{Current}} := 0$
- $\Big((\mathsf{mpk}, \mathsf{info}_0; \mathsf{st}_{\mathsf{GM}}); (\mathsf{opk}, \mathsf{osk})\Big) \leftarrow \langle \mathbf{GKGen}_{\mathsf{GM}}(\mathsf{param}); \mathbf{GKGen}_{\mathsf{OA}}(\mathsf{param})\rangle$
- $\mathsf{gpk} := (\mathsf{param}, \mathsf{mpk}, \mathsf{opk})$
- $\Big(m, \Sigma, \tau\Big) \leftarrow \mathcal{A}^{\mathsf{SndToM},\mathsf{Update},\mathsf{WriteReg},\mathsf{State}}(\mathsf{gpk}, \mathsf{info}_0, \mathsf{osk})$
- If $\mathbf{Verify}(\mathsf{gpk}, \mathsf{info}_\tau, m, \Sigma) = 0$ return 0
- $(\mathsf{i}, \pi) \leftarrow \mathbf{Open}^{\mathsf{ReadReg}}(\mathsf{gpk}, \mathsf{osk}, \mathsf{info}_\tau, m, \Sigma)$
- If $\mathbf{IsActive}(\mathsf{i}, \tau, \mathsf{st}_{\mathsf{GM}}) = 0$ return 1
- If $\mathbf{Judge}(\mathsf{gpk}, \mathsf{info}_\tau, \mathsf{reg}_\mathsf{i}, m, \Sigma, \pi) = 0$ return 1
- Return 0.

FIGURE 7.10: Traceability game for separate group manager and opening authority.

$\underline{\mathsf{SndToM}(\mathsf{i}, M_{\mathsf{GM}})}$
- If $\mathsf{i} \notin [N+1]$: Return $\perp$
- If $\mathsf{i} = N+1$:
  - $N := N+1$
- $(\mathsf{out}_\mathsf{i}; M_\mathsf{i}; \mathsf{st}_{\mathsf{GM}}) \leftarrow \mathbf{Join}^{\mathsf{ReadReg}(\mathsf{i})}_{\mathsf{GM}}(\mathsf{i}, M_{\mathsf{GM}}; \mathsf{st}_{\mathsf{GM}})$
- Return $(\mathsf{out}_\mathsf{i}, M_\mathsf{i})$

$\underline{\mathsf{State}}$
- Return $\mathsf{st}_{\mathsf{GM}}$

$\underline{\mathsf{Update}(\mathcal{R})}$
- If $\mathcal{R} \not\subseteq [N]$ return $\perp$
- $(\mathsf{info}; \mathsf{st}_{\mathsf{GM}}) \leftarrow \mathbf{UpdateGroup}(\mathcal{R}; \mathsf{st}_{\mathsf{GM}})$
- $\tau_{\mathsf{Current}} := \tau_{\mathsf{Current}} + 1$
- Return $\mathsf{info}_{\tau_{\mathsf{Current}}} := \mathsf{info}$

$\underline{\mathsf{WriteReg}(\mathsf{i}, M)}$
- If $\mathsf{reg}_\mathsf{i} \neq \perp$ return $\perp$
- Return $\mathsf{reg}_\mathsf{i} := M$

FIGURE 7.11: Oracles used in the traceability game for separate GM and OA.

The two games for traceability are very similar, and it is easy to see that the transformation in Sec. 7.1.1 of an $\mathcal{FDGS}$ for separate GM and OA to a single group manager scheme $\mathcal{FDGS}'$ preserves traceability.

Non-Frameability

Non-frameability is a security notion that says even if the rest of the group as well as the group manager are fully corrupt, they cannot falsely attribute a signature to an honest member who did not produce it. In the non-frameability game $\mathbf{Exp}^{\mathsf{Non-Frame}}_{\mathcal{FDGS},\mathcal{A}}$ shown in Figure 7.12, we grant the adversary access to the oracles described below and detailed in Figure 7.13, and we keep a global list $\mathcal{S}$ of signatures produced by an honest user. Please note that the adversary controls the group manager and hence session identifiers no longer carry much meaning, the adversary can pretend the user has any session identifier. Instead without loss of generality we simply identify the honest user with a generic record reg and require that only the honest user is able to write in this record.

$\mathsf{SndToHU}(M_{\mathsf{h}})$: This oracle allows the adversary to interact with a single honest user in an instance of the join protocol. Each call executes the next move of the honest user on input a message M_{h} provided by the adversary (playing the role of the corrupt group manager) and returns the user response message. The user may write a message into the register using oracle Write, which can be accessed only once by the user. Messages received by the Write oracle are revealed to the adversary. The user output $\mathsf{out}_{\mathsf{h}}$, i.e., the signing key, is *not* disclosed to the adversary.

$\mathsf{SignHU}(\mathsf{info}, m)$: This oracle is used by the adversary against non-frameability to obtain signatures from an honest group member h added to the group via SndToHU calls. It returns a group signature on the message m using key $\mathsf{gsk}_{\mathsf{h}}$ and group information info.

The adversary can add a single honest user in the group and request him to sign arbitrary messages. The goal of the adversary in the non-frameability game is to produce a valid signature on a message that was not requested to the signing oracle, together with an accepting proof attributing the signature to the honest user.

Definition 7.5 (Non-frameability)**.** *An $\mathcal{FDGS}$ scheme is* non-frameable *if for all PPT adversaries $\mathcal{A}$, the following advantage is negligible*

$$\mathbf{Adv}^{\mathsf{Non-Frame}}_{\mathcal{FDGS},\mathcal{A}}(\lambda) := \Pr[\mathbf{Exp}^{\mathsf{Non-Frame}}_{\mathcal{FDGS},\mathcal{A}}(\lambda) = 1].$$

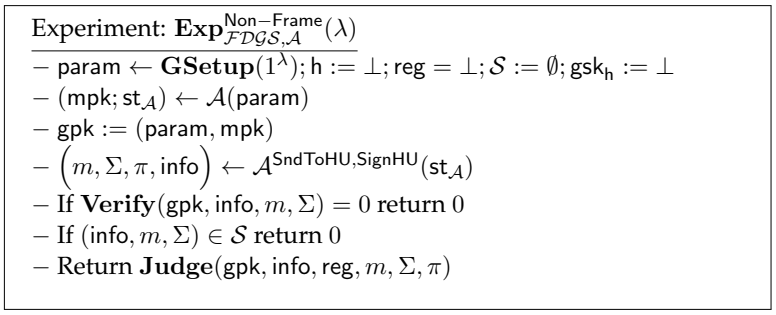

FIGURE 7.12: Non-Frameability game.

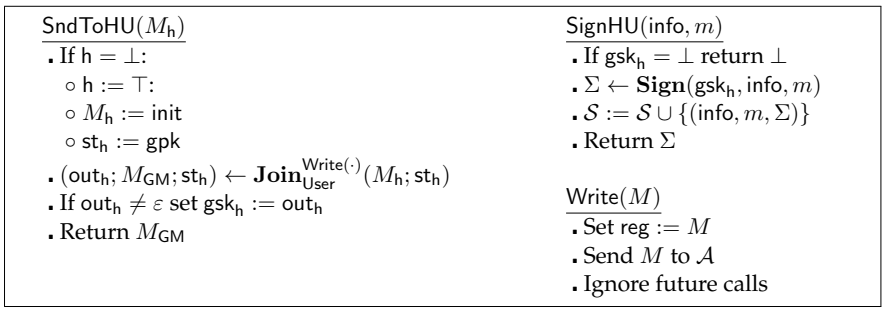

FIGURE 7.13: Oracles used in the non-frameability game.

In our definition of non-frameability the adversary controls the group manager during the key generation process. Thus, our definition is stronger than existing definitions, which only allow the group manager to be corrupted after the group keys have been honestly generated.

We allow only a single honest user in the group and ask the adversary to frame her. It can be shown that this implies the more general case involving several honest users in the group by a standard hybrid argument since the adversary can simulate the actions of additional honest users.

Variations of Non-Frameability. While traceability still makes sense without proofs and the **Judge** algorithm, non-frameability does not since it is about whether a corrupt group manager would be able to *prove* instead of just falsely accusing an honest user of having signed a message. If we consider the setting with no proofs and no **Judge** algorithm, we can therefore completely eliminate the non-frameability notion from our definitions.

In Fig. 7.14 we give a variation of the non-frameability game suitable for the setting where the roles of group manager and opening authority are separated. Since both are under adversarial control, the game is equivalent to the previous non-frameability

game where the group manager has both roles, so it is easy to see the transformation from Sec. 7.1.1 of a two authority $\mathcal{FDGS}$ into a single authority $\mathcal{FDGS}'$ preserves non-frameability.

Experiment: $\mathbf{Exp}^{\mathsf{Non-Frame}}_{\mathcal{FDGS},\mathcal{A}}(\lambda)$

- $\mathsf{param} \leftarrow \mathbf{GSetup}(1^\lambda); \mathsf{h} := \perp; \mathcal{S} := \emptyset; \mathsf{gsk_h} := \perp$
- $(\mathsf{mpk}, \mathsf{opk}; \mathsf{st}_\mathcal{A}) \leftarrow \mathcal{A}(\mathsf{param})$
- $\mathsf{gpk} := (\mathsf{param}, \mathsf{mpk}, \mathsf{opk})$
- $\Big(m, \Sigma, \pi, \mathsf{info}\Big) \leftarrow \mathcal{A}^{\mathsf{SndToHU},\mathsf{SignHU}}(\mathsf{st}_\mathcal{A})$
- If $\mathbf{Verify}(\mathsf{gpk}, \mathsf{info}, m, \Sigma) = 0$ return 0
- If $(\mathsf{info}, m, \Sigma) \in \mathcal{S}$ return 0
- Return $\mathbf{Judge}(\mathsf{gpk}, \mathsf{info}, \mathsf{reg}, m, \Sigma, \pi)$

FIGURE 7.14: Non-Frameability game for separate GM and OA.

7.1.3 Additional Security Definitions

In addition to the core security properties correctness, anonymity, traceability and non-frameability, a group signature scheme may have additional security guarantees. In the above definitions the opening provided by either the group manager or the opening authority can be generally thought as a deterrent against misbehaviours of users. It is not hard, however, to envision applications in which openings could be also used to create incentives. In such scenarios it may become crucial to prevent an attacker exploiting the opening mechanism to her own advantage rather than eluding it.

Opening Binding

Opening binding[1] defined by Sakai et al. [SSE+12] in the context of partially dynamic group signatures guarantees that even if all authorities and users collude they should not be able to produce a valid signature that can be selectively attributed to two different members. Consider for instance a contest to find the best stock market analyst. Group signatures are used to sign stock market predictions by the experts, who should remain anonymous in order not to influence the markets, and later we use the opening algorithm in order to tally up who is the best expert. There may be a financial

[1] Sakai et al. [SSE+12] refer to this notion as opening soundness.

incentive to become the leading expert, so we could imagine a collusion where an “expert" and a “fool" enrol and then collaborate to attribute all correct predictions to the “expert" and all wrong predictions to the “amateur".

We define opening binding game in Fig. 7.15, where the goal of the adversary is to create a signature and two distinct attributions to who signed it. We consider a strongly adversarial setting, where both the authorities and users may be adversarial but want the guarantee that each signature must be attributed to a unique record in the registry.

Definition 7.6 (Opening Binding). *An $\mathcal{FDGS}$ scheme is* opening binding *if for all PPT adversaries $\mathcal{A}$*

$$\mathbf{Adv}^{\mathsf{Opening-Bind}}_{\mathcal{FDGS},\mathcal{A}}(\lambda) := \Pr[\mathbf{Exp}^{\mathsf{Opening-Bind}}_{\mathcal{FDGS},\mathcal{A}}(\lambda) = 1] \approx 0.$$

Experiment: $\mathbf{Exp}^{\mathsf{Opening-Bind}}_{\mathcal{FDGS},\mathcal{A}}(\lambda)$
- $\mathsf{param} \leftarrow \mathbf{GSetup}(1^\lambda)$
- $(\mathsf{mpk}, \mathsf{info}, \mathsf{opk}, m, \Sigma, \mathsf{reg}, \pi, \mathsf{reg}', \pi') \leftarrow \mathcal{A}(\mathsf{param})$
  //This is for separate GM and OA, use $\mathsf{opk} = \varepsilon$ if only single GM
- $\mathsf{gpk} := (\mathsf{param}, \mathsf{mpk}, \mathsf{opk})$
- If $\mathbf{Verify}(\mathsf{gpk}, \mathsf{info}, m, \Sigma) = 0$ return 0
- If $\mathbf{Judge}(\mathsf{gpk}, \mathsf{info}, \mathsf{reg}, m, \Sigma, \pi) = 0$ return 0
- If $\mathbf{Judge}(\mathsf{gpk}, \mathsf{info}, \mathsf{reg}', m, \Sigma, \pi') = 0$ return 0
- If $\mathsf{reg} \neq \mathsf{reg}'$ return 1, else return 0

FIGURE 7.15: Opening binding game.

Opening Soundness

Consider again the example of a competition to determine who is a the best stock market prediction expert, this time from the perspective of an honest expert. It would be problematic if it was possible to “steal" the group signatures by the honest user, i.e., falsely attribute them to somebody else. The worst-case scenario here is that dishonest authorities are collaborating with a malicious user to attribute the honest user’s signature to the malicious user instead.

We define the opening soundness experiment in Fig. 7.16, where the adversary is trying to attribute a signature Σ of an honest user to a different registry. The experiment uses an oracle to enrol and honest user and an oracle that provides an honestly generated signature described in Fig. 7.17.

Definition 7.7 (Opening soundness)**.** *An $\mathcal{FDGS}$ scheme is* opening sound *if for all PPT adversaries $\mathcal{A}$*

$$\mathbf{Adv}^{\mathsf{Opening-Sound}}_{\mathcal{FDGS},\mathcal{A}}(\lambda) := \Pr[\mathbf{Exp}^{\mathsf{Opening-Sound}}_{\mathcal{FDGS},\mathcal{A}}(\lambda) = 1] \approx 0.$$

Related notions of opening soundness were previously considered by Kiayias and Yung [KY06], as a requirement for correctness, and Sakai et al. [SSE+12], under the name of *weak* opening soundness. Our definition considers highly adversarial settings and thus captures a much more stringent notion.

Experiment: $\mathbf{Exp}^{\mathsf{Opening-Sound}}_{\mathcal{FDGS},\mathcal{A}}(\lambda)$

- $\mathsf{param} \leftarrow \mathbf{GSetup}(1^\lambda)$; $\mathsf{h} = \bot$; $\mathsf{reg} := \bot$; $\mathsf{gsk_h} = \bot$; $\Sigma := \bot$
- $(\mathsf{mpk}, \mathsf{opk}; \mathsf{st}_\mathcal{A}) \leftarrow \mathcal{A}(\mathsf{param})$
  //This is for separate GM and OA, use $\mathsf{opk} = \varepsilon$ if only single GM
- $\mathsf{gpk} := (\mathsf{param}, \mathsf{mpk}, \mathsf{opk})$
- $(\mathsf{info}^*, \mathsf{reg}^*, m^*, \pi^*) \leftarrow \mathcal{A}^{\mathsf{SndToHU},\mathsf{SignHU}}(\mathsf{st}_\mathcal{A})$
- If $\mathbf{Judge}(\mathsf{gpk}, \mathsf{info}^*, \mathsf{reg}^*, m^*, \Sigma, \pi^*) = 0$ return 0
- If $\mathsf{reg} \neq \mathsf{reg}^*$ return 1, else return 0

FIGURE 7.16: Opening soundness game.

$\underline{\mathsf{SndToHU}(M_\mathsf{h})}$
- If $\mathsf{h} = \bot$:
  - $\mathsf{h} := \top$:
  - $M_\mathsf{h} := \mathsf{init}$
  - $\mathsf{st_h} := \mathsf{gpk}$
- $(\mathsf{out_h}; M_\mathsf{GM}; \mathsf{st_h}) \leftarrow \mathbf{Join}^{\mathsf{Write}(\cdot)}_{\mathsf{User}}(M_\mathsf{h}; \mathsf{st_h})$
- If $\mathsf{out_h} \neq \varepsilon$ set $\mathsf{gsk_h} := \mathsf{out_h}$
- Return $(\mathsf{out_h}, M_\mathsf{GM}, \mathsf{st_h})$

$\underline{\mathsf{SignHU}(\mathsf{info}, m)}$
- $\Sigma \leftarrow \mathbf{Sign}(\mathsf{gsk_h}, \mathsf{info}, m)$
- Return Σ
- Ignore future calls

$\underline{\mathsf{Write}(M)}$
- Set $\mathsf{reg} := M$
- Send M to $\mathcal{A}$
- Ignore future calls

FIGURE 7.17: Oracles used in the opening soundness game.

7.2 Partially Dynamic Group Signatures

In the previous section, we defined fully dynamic group signatures. Earlier formal definitions covered static groups, where the membership is fixed at initialisation [BMW03], and partially dynamic groups where new members may enrol but without revocation [BSZ05; KY06]. We will now discuss how our definitions for fully dynamic group signatures can be relaxed to the partially dynamic settings, and we will show that

aside from minor differences earlier formal definitions of group signatures can be seen as restrictions of our definitions to special cases.

7.2.1 Restriction to Partially Dynamic Signatures

In prior works, the term dynamic group signature refers to the case where new members may enrol. These definitions do not cover revocation though, the group can grow but it will never shrink. This partially dynamic setting is simply a special case of our fully dynamic group signatures, where we never revoke members, i.e., in all calls to Update we have revocation set $\mathcal{R} = \emptyset$.

For some designs of group signatures, the group information does not change as new members are enrolled. This is unlike the fully dynamic setting, where by necessity the group information must change to prevent revoked signers from using their current keys to produce valid signatures. When the group information is immutable, we can eliminate the Update function entirely from our scheme and remove the corresponding oracle in the security definitions. This in turn means we only have one epoch $\tau = 0$ and also the **IsActive** policy now simply says that an enrolled user is active immediately after the group manager considers her joining procedure to have ended successfully. Since info_0 is generated by GM together with the manager public key mpk, we can without loss of generality assume $\mathsf{info}_0 = \varepsilon$. For the special case of partially dynamic group signatures with immutable group information, the definitions can therefore be simplified by excluding the epoch $\tau = 0$, the public group information $\mathsf{info}_0 = \varepsilon$ and the Update procedure.

These notational simplifications lead us to the following syntax for a partially dynamic group signature scheme with immutable group information:

Reg: Data structure with records reg_i for joining session identifiers $\mathsf{i} = 1, 2, 3, \ldots$ with algorithms/oracles.

- $\mathsf{ReadReg}(\mathsf{i})$: Return reg_i (or $\perp$ if no such record exists).
- $\mathsf{WriteReg}(\mathsf{i}, M)$: Set $\mathsf{reg}_\mathsf{i} := M$ and ignore further calls with the same i.

GSetup$(1^\lambda) \to \mathsf{param}$**:** PPT algorithm generating trusted parameters (or 1^λ if there is no trusted setup).

$\mathbf{GKGen}(\mathsf{param}) \to (\mathsf{mpk}; \mathsf{st}_{\mathsf{GM}})$**:** PPT algorithm for group manager key generation. The group public key is $\mathsf{gpk} := (\mathsf{param}, \mathsf{mpk})$.

If we separate the roles of group manager GM and opening authority OA, they instead run the interactive key generation protocol

$$((\mathsf{mpk}; \mathsf{st}_{\mathsf{GM}}); (\mathsf{opk}, \mathsf{osk})) \leftarrow \langle \mathbf{GKGen}_{\mathsf{GM}}(\mathsf{param}); \mathbf{GKGen}_{\mathsf{OA}}(\mathsf{param}) \rangle$$

and let $\mathsf{gpk} := (\mathsf{param}, \mathsf{mpk}, \mathsf{opk})$ be the group public key.

Join: Interactive protocol for enrolling user in group. Defined by PPT algorithms

- $\mathbf{Join}_{\mathsf{User}}^{\mathsf{WriteReg}(\mathsf{i},\cdot)}(M; \mathsf{st}) \to (\boldsymbol{out}; M_{\mathsf{GM}}; \mathsf{st})$.
- $\mathbf{Join}_{\mathsf{GM}}^{\mathsf{ReadReg}(\mathsf{i})}(\mathsf{i}, M_{\mathsf{GM}}; \mathsf{st}_{\mathsf{GM}}) \to (\boldsymbol{out}_{\mathsf{GM}}; M; \mathsf{st}_{\mathsf{GM}})$.

$\mathbf{IsActive}(\mathsf{i}, \mathsf{st}_{\mathsf{GM}}) \to 1/0$: DPT algorithm defining when user is considered active. In the partially dynamic case, the policy is that the user joining in session i is active if and only if the group manager terminated with success symbol $\top$.

$\mathbf{Sign}(\mathsf{gsk}, m) \to \Sigma$**:** PPT signing algorithm.

$\mathbf{Verify}(\mathsf{gpk}, m, \Sigma) \to 1/0$**:** DPT verification algorithm.

$\mathbf{Open}(\mathsf{gpk}, \mathsf{st}_{\mathsf{GM}}, m, \Sigma) \to (\mathsf{i}, \pi)$**:** PPT opening algorithm.

If we separate the roles of group manager and opening authority, OA instead uses the opening key osk to run $\mathbf{Open}^{\mathsf{ReadReg}(\cdot)}(\mathsf{gpk}, \mathsf{osk}, m, \Sigma) \to (\mathsf{i}, \pi)$.

$\mathbf{Judge}(\mathsf{gpk}, \mathsf{reg}, m, \Sigma, \pi) \to 1/0$**:** DPT algorithm determining if signature was correctly attributed to registry record reg.

The corresponding simplified security experiments for partially dynamic group signature scheme with immutable group information is given in Fig. 7.18. We list the experiments for the single group manager setting, and note in the comments indicated by // how to adapt them to the separate GM and OA setting. The oracles are defined exactly as in the case of fully dynamic group signatures and simplified by excluding the group information $\mathsf{info} = \varepsilon$ and epochs $\tau = 0$.

Experiment: $\mathbf{Exp}^{\mathsf{Corr}}_{\mathcal{PDGS},\mathcal{A}}(\lambda)$

- $\mathsf{param} \leftarrow \mathbf{GSetup}(1^\lambda)$; $\mathsf{h} := \bot$; $N := 0$; $K := 0$
- $(\mathsf{mpk}; \mathsf{st}_{\mathsf{GM}}) \leftarrow \mathbf{GKGen}(\mathsf{param})$; $\mathsf{gpk} := (\mathsf{param}, \mathsf{mpk})$
  // $\Big((\mathsf{mpk}; \mathsf{st}_{\mathsf{GM}}); (\mathsf{opk}; \mathsf{st}_{\mathcal{A}})\Big) \leftarrow \langle \mathbf{GKGen}_{\mathsf{GM}}(\mathsf{param}); \mathcal{A}(\mathsf{param})\rangle$
  // $\mathsf{gpk} := (\mathsf{param}, \mathsf{mpk}, \mathsf{opk})$
- $m \leftarrow \mathcal{A}^{\mathsf{AddHU},\mathsf{SndToM},\mathsf{Write},\mathsf{State}}(\mathsf{gpk})$
- If $K = k(\lambda)$ and $\mathbf{IsActive}(\mathsf{h}, \mathsf{st}_{\mathsf{GM}}) = 0$ return 0
- If $\mathbf{IsActive}(\mathsf{h}, \mathsf{st}_{\mathsf{GM}}) = 0$ return 1
- $\Sigma \leftarrow \mathbf{Sign}(\mathsf{gsk}_{\mathsf{h}}, m)$
- Return $\mathbf{Verify}(\mathsf{gpk}, m, \Sigma)$

Experiment: $\mathbf{Exp}^{\mathsf{Anon}-b}_{\mathcal{PDGS},\mathcal{A}}(\lambda)$

- $\mathsf{param} \leftarrow \mathbf{GSetup}(1^\lambda)$; $N := 0$; $\mathcal{H} := \emptyset$; $\mathcal{C} := \emptyset$
- $(\mathsf{mpk}; \mathsf{st}_{\mathsf{GM}}) \leftarrow \mathbf{GKGen}(\mathsf{param})$; $\mathsf{gpk} := (\mathsf{param}, \mathsf{mpk})$
  // $\Big((\mathsf{mpk}; \mathsf{st}_{\mathcal{A}}); (\mathsf{opk}, \mathsf{osk})\Big) \leftarrow \langle \mathcal{A}(\mathsf{param}); \mathbf{GKGen}_{\mathsf{OA}}(\mathsf{param})\rangle$
  // $\mathsf{gpk} := (\mathsf{param}, \mathsf{mpk}, \mathsf{opk})$
- Return $b^* \leftarrow \mathcal{A}^{\mathsf{AddHU},\mathsf{SndToM},\mathsf{Open},\mathsf{Chal}_b,\mathsf{ReadReg}}(\mathsf{gpk})$
  // Return $b^* \leftarrow \mathcal{A}^{\mathsf{SndToU},\mathsf{Open},\mathsf{Chal}_b,\mathsf{ReadReg}}\Big(\mathsf{gpk}; \mathsf{st}_{\mathcal{A}}\Big)$

Experiment: $\mathbf{Exp}^{\mathsf{Trace}}_{\mathcal{PDGS},\mathcal{A}}(\lambda)$

- $\mathsf{param} \leftarrow \mathbf{GSetup}(1^\lambda)$; $N := 0$
- $(\mathsf{mpk}; \mathsf{st}_{\mathsf{GM}}) \leftarrow \mathbf{GKGen}(\mathsf{param})$; $\mathsf{gpk} := (\mathsf{param}, \mathsf{mpk})$
  // $\Big((\mathsf{mpk}; \mathsf{st}_{\mathsf{GM}}); (\mathsf{opk}, \mathsf{osk})\Big) \leftarrow \langle \mathbf{GKGen}_{\mathsf{GM}}(\mathsf{param}); \mathbf{GKGen}_{\mathsf{OA}}(\mathsf{param})\rangle$
  // $\mathsf{gpk} := (\mathsf{param}, \mathsf{mpk}, \mathsf{opk})$
- $(m, \Sigma) \leftarrow \mathcal{A}^{\mathsf{SndToM},\mathsf{WriteReg},\mathsf{Open},\mathsf{State}}(\mathsf{gpk})$
  // $\Big(m, \Sigma\Big) \leftarrow \mathcal{A}^{\mathsf{SndToM},\mathsf{WriteReg},\mathsf{State}}(\mathsf{gpk}, \mathsf{osk})$
- If $\mathbf{Verify}(\mathsf{gpk}, m, \Sigma) = 0$ return 0
- $(\mathsf{i}, \pi) \leftarrow \mathbf{Open}(\mathsf{gpk}, \mathsf{st}_{\mathsf{GM}}, m, \Sigma)$
  // $(\mathsf{i}, \pi) \leftarrow \mathbf{Open}^{\mathsf{ReadReg}}(\mathsf{gpk}, \mathsf{osk}, m, \Sigma)$
- If $\mathbf{IsActive}(\mathsf{i}, \mathsf{st}_{\mathsf{GM}}) = 0$ return 1
- If $\mathbf{Judge}(\mathsf{gpk}, \mathsf{reg}_{\mathsf{i}}, m, \Sigma, \pi) = 0$ return 1
- Return 0

Experiment: $\mathbf{Exp}^{\mathsf{Non}-\mathsf{Frame}}_{\mathcal{PDGS},\mathcal{A}}(\lambda)$

- $\mathsf{param} \leftarrow \mathbf{GSetup}(1^\lambda)$; $\mathsf{h} := \bot$; $\mathsf{reg} = \bot$; $\mathcal{S} := \emptyset$; $\mathsf{gsk}_{\mathsf{h}} := \bot$
- $(\mathsf{mpk}; \mathsf{st}_{\mathcal{A}}) \leftarrow \mathcal{A}(\mathsf{param})$; $\mathsf{gpk} := (\mathsf{param}, \mathsf{mpk})$
  // $(\mathsf{mpk}, \mathsf{opk}; \mathsf{st}_{\mathcal{A}}) \leftarrow \mathcal{A}(\mathsf{param})$; $\mathsf{gpk} := (\mathsf{param}, \mathsf{mpk}, \mathsf{opk})$
- $\Big(m, \Sigma, \pi\Big) \leftarrow \mathcal{A}^{\mathsf{SndToHU},\mathsf{SignHU}}\Big(\mathsf{st}_{\mathcal{A}}\Big)$
- If $\mathbf{Verify}(\mathsf{gpk}, m, \Sigma) = 0$ return 0
- If $(m, \Sigma) \in \mathcal{S}$ return 0
- Return $\mathbf{Judge}(\mathsf{gpk}, \mathsf{reg}, m, \Sigma, \pi)$

FIGURE 7.18: Security experiments for partially dynamic group signatures.

7.2.2 Comparison to Bellare, Shi and Zhang [BSZ05]

Bellare, Shi and Zhang [BSZ05] define partially dynamic group signatures with separate group manager (called Issuer) and opening authority (called Opener). Their definition is a specific type of partially dynamic group signature with immutable group information. We will now describe restrictions to our definition of partially dynamic group signatures that yields a definition similar to their definition.

Bellare, Shi and Zhang consider a group manager state formed as $\mathsf{st}_{\mathsf{GM}} = (\mathsf{msk}, \{\mathsf{st}^i_{\mathsf{GM}}\})$. The state is therefore compartmentalized to consist of a fixed part msk, which we call the manager's secret key, and other parts $\mathsf{st}^{\mathsf{i}}_{\mathsf{GM}}$ that are specific to the joining sessions. It is assumed joining session states are independent of each other, which makes it easier to reason about concurrent joins. In particular, since the joins are independent of each other, we can under the same assumption of compartmentalized group manager state define correctness in terms of a single joining session and ignore any concurrent joining sessions.

Bellare, Shi and Zhang assume a trusted key generation procedure. If we assume keys to be honestly generated, we can combine the trusted parameter generation and the joining protocol into a single algorithm, which first runs the parameter generation and then honestly execute the interactive key generation protocol. So there is little loss of generality in assuming a single trusted algorithm that generates all keys.

Taken together, for a partially dynamic group signature scheme with compartmentalized group manager state and trusted key generation we can simplify the syntax and security experiments as described below. A group signature scheme with compartmentalized group manager state satisfying our definition in Fig. 7.18 directly yields a partially dynamic group signature scheme satisfying the definition in Fig. 7.19 by letting the key generation procedure run the setup algorithm and the interactive key generation protocol honestly and outputting the resulting keys.

Reg : Data structure with records $\mathsf{reg}_{\mathsf{i}}$ for joining session identifiers $\mathsf{i} = 1, 2, 3, \ldots$ with algorithms/oracles.

- $\mathsf{ReadReg}(\mathsf{i})$: Return $\mathsf{reg}_{\mathsf{i}}$ (or $\perp$ if no such record exists).
- $\mathsf{WriteReg}(\mathsf{i}, M)$: Set $\mathsf{reg}_{\mathsf{i}} := M$ and ignore further calls with the same i.

$\mathbf{GKGen}(1^\lambda) \to (\mathsf{gpk}, \mathsf{msk}, \mathsf{osk})$**:** Trusted PPT algorithm for key generation.

Join : Interactive protocol for enrolling user in group. Defined by PPT algorithms.

- $\mathbf{Join}_{\mathsf{User}}^{\mathsf{WriteReg(i,\cdot)}}(M; \mathsf{st}) \to (\boldsymbol{out}; M_{\mathsf{GM}}; \mathsf{st})$.
- $\mathbf{Join}_{\mathsf{GM}}^{\mathsf{ReadReg(i)}}(\mathsf{i}, M_{\mathsf{GM}}, \mathsf{msk}; \mathsf{st}_{\mathsf{GM}}^{\mathsf{i}}) \to (\boldsymbol{out}_{\mathsf{GM}}; M; \mathsf{st}_{\mathsf{GM}}^{\mathsf{i}})$.

$\mathbf{IsActive}(\mathsf{i}, \mathsf{st}_{\mathsf{GM}}) \to 1/0$: DPT algorithm defining when user is considered active. In the partially dynamic case, the policy is that the user joining in session i is active if and only if the group manager terminated with success symbol $\top$.

$\mathbf{Sign}(\mathsf{gsk}, m) \to \Sigma$**:** PPT signing algorithm.

$\mathbf{Verify}(\mathsf{gpk}, m, \Sigma) \to 1/0$**:** DPT verification algorithm.

$\mathbf{Open}^{\mathsf{ReadReg}(\cdot)}(\mathsf{gpk}, \mathsf{osk}, m, \Sigma) \to (\mathsf{i}, \pi)$: PPT opening algorithm.

$\mathbf{Judge}(\mathsf{gpk}, \mathsf{reg}, m, \Sigma, \pi) \to 1/0$**:** DPT algorithm determining if signature was correctly attributed to registry record reg.

The matching simplified security experiments are given in Fig. 7.19. The oracles are defined exactly as in the previous definitions.

Bellare, Shi and Zhang assume a confidential communication channel between the group manager and joining users, while we assume an open channel. A scheme satisfying our security definition will of course also satisfy the weaker security definition that assumes confidential communication channels. Conversely, the group manager and user can use public-key cryptography to establish a confidential channel, so this definitional difference is immaterial. Bellare, Shi and Zhang also consider a slightly stronger definition of correctness where an honestly generated signature must always open to identify the signer, which as discussed in Sec. 7.1.2 is covered in a computational sense by traceability and therefore in our opinion immaterial.

It can now be seen by direct comparison with [BSZ05] that the experiments in Fig. 7.19 yield a security definition very similar to the one given by Bellare, Shi and Zhang. One remaining difference is that Bellare, Shi and Zhang explicitly include a public key infrastructure where each user has an identity $j \in \mathbb{N}$ and generates a key pair $(\mathsf{upk}[j], \mathsf{usk}[j])$, and they consider the registry to be under the jurisdiction of the group manager. However, as discussed in Section 7.1.1 we can eliminate this

Experiment: $\mathbf{Exp}^{\mathsf{Corr}}_{\mathcal{BSZ},\mathcal{A}}(\lambda)$
– $(\mathsf{gpk}, \mathsf{msk}, \mathsf{osk}) \leftarrow \mathbf{GKGen}(1^\lambda)$; $\mathsf{h} := \perp$; $N := 0$; $K := 0$
– $m \leftarrow \mathcal{A}^{\mathsf{AddHU},\mathsf{Write}}(\mathsf{gpk}, \mathsf{msk}, \mathsf{osk})$
– If $K = k(\lambda)$ and $\mathbf{IsActive}(\mathsf{h}, \mathsf{st}_{\mathsf{GM}}) = 0$ return 0
– If $\mathbf{IsActive}(\mathsf{h}, \mathsf{st}_{\mathsf{GM}}) = 0$ return 1
– $\Sigma \leftarrow \mathbf{Sign}(\mathsf{gsk}_{\mathsf{h}}, m)$
– Return $\mathbf{Verify}(\mathsf{gpk}, m, \Sigma)$

Experiment: $\mathbf{Exp}^{\mathsf{Anon}-b}_{\mathcal{BSZ},\mathcal{A}}(\lambda)$
– $(\mathsf{gpk}, \mathsf{msk}, \mathsf{osk}) \leftarrow \mathbf{GKGen}(1^\lambda)$; $N := 0$; $\mathcal{H} := \emptyset$; $\mathcal{C} := \emptyset$
– Return $b^* \leftarrow \mathcal{A}^{\mathsf{SndToU},\mathsf{Open},\mathsf{Chal}_b,\mathsf{ReadReg}}(\mathsf{gpk}, \mathsf{msk})$

Experiment: $\mathbf{Exp}^{\mathsf{Trace}}_{\mathcal{BSZ},\mathcal{A}}(\lambda)$
– $(\mathsf{gpk}, \mathsf{msk}, \mathsf{osk}) \leftarrow \mathbf{GKGen}(1^\lambda)$; $N := 0$
– $\Big(m, \Sigma\Big) \leftarrow \mathcal{A}^{\mathsf{SndToM},\mathsf{WriteReg}}(\mathsf{gpk}, \mathsf{msk}, \mathsf{osk})$
– If $\mathbf{Verify}(\mathsf{gpk}, m, \Sigma) = 0$ return 0
– $(\mathsf{i}, \pi) \leftarrow \mathbf{Open}^{\mathsf{ReadReg}}(\mathsf{gpk}, \mathsf{osk}, m, \Sigma)$
– If $\mathbf{IsActive}(\mathsf{i}, \mathsf{st}_{\mathsf{GM}}) = 0$ return 1
– If $\mathbf{Judge}(\mathsf{gpk}, \mathsf{reg}_{\mathsf{i}}, m, \Sigma, \pi) = 0$ return 1
– Return 0

Experiment: $\mathbf{Exp}^{\mathsf{Non}-\mathsf{Frame}}_{\mathcal{BSZ},\mathcal{A}}(\lambda)$
– $(\mathsf{gpk}, \mathsf{msk}, \mathsf{osk}) \leftarrow \mathbf{GKGen}(1^\lambda)$; $\mathsf{h} := \perp$; $\mathsf{reg} = \perp$; $\mathcal{S} := \emptyset$; $\mathsf{gsk}_{\mathsf{h}} := \perp$
– $\Big(m, \Sigma, \pi\Big) \leftarrow \mathcal{A}^{\mathsf{SndToHU},\mathsf{SignHU}}(\mathsf{gpk}, \mathsf{msk}, \mathsf{osk})$
– If $\mathbf{Verify}(\mathsf{gpk}, m, \Sigma) = 0$ return 0
– If $(m, \Sigma) \in \mathcal{S}$ return 0
– Return $\mathbf{Judge}(\mathsf{gpk}, \mathsf{reg}, m, \Sigma, \pi)$

FIGURE 7.19: Security experiments for partially dynamic group signatures akin to Bellare, Shi and Zhang [BSZ05]

extra step to simplify definitions and just consider the registry record to be under control of the user. For concreteness, this can be done by letting the user generate a key pair $(\mathsf{upk}, \mathsf{usk})$ for an sEUF-CMA secure digital signature scheme and create a signature σ on her intended record together with her identity j and public key $\mathsf{upk}[j]$. I.e., in Bellare et al. [BSZ05] let the record created when she is joining in session i be $\mathsf{reg}[\mathsf{i}] = (j, \mathsf{upk}[j], \mathsf{reg}_\mathsf{i}, \sigma)$. The user can now send this record to the group manager when she wants to write to the registry. Since the group manager cannot forge the user's signature, this is equivalent to letting the user have one-time write access to the registry, and given the record index i it is easy to map to the user identity j. Our definition in Fig. 7.19 is mostly similar to the definition of Bellare et al. [BSZ05] modulo this difference.

7.2.3 Comparison to Kiayias and Yung [KY06]

Kiayias and Yung [KY06] also give a formal definition of partially dynamic group signatures with immutable group information. Their definition is in the single authority setting and deviates from our definition and Bellare, Shi and Zhang [BSZ05] by trusting the group manager to open honestly and not requiring proofs of correct attribution to a signer. This simplification is easy to implement given a single authority partially dynamic group signature; the opening algorithm can simply discard the proof of correct attribution. Kiayias and Yung assume the key generation process is honest and they also assume the group manager's internal state can be compartmentalized as $\mathsf{st}_\mathsf{GM} = (\mathsf{msk}, \{\mathsf{st}^\mathsf{i}_\mathsf{GM}\})$, both of which are special cases of our definition.[2] In their mode, opening takes place against a public record, so we let the opening algorithm have access to the registry, while it is of course easy to just incorporate it into the state of the group manager. We present the syntax and security experiments below.

Reg : Data structure with records reg_i for joining session identifiers $\mathsf{i} = 1, 2, 3, \ldots$ with algorithms/oracles.

- $\mathsf{ReadReg}(\mathsf{i})$: Return reg_i (or $\perp$ if no such record exists).
- $\mathsf{WriteReg}(\mathsf{i}, M)$: Set $\mathsf{reg}_\mathsf{i} := M$ and ignore further calls with the same i.

[2] Kiayias and Yung use very different notation. What they call 'state' is what we call 'registry'. Their public state contains also the transcripts of the execution of the **Join** protocol.

$\mathbf{GKGen}(1^\lambda) \to (\mathsf{gpk}; \mathsf{msk})$**:** PPT algorithm for group manager key generation.

Join : Interactive protocol for enrolling user in group. Defined by PPT algorithms.

- $\mathbf{Join}_{\mathsf{User}}^{\mathsf{WriteReg}(\mathsf{i},\cdot)}(M; \mathsf{st}) \to (\boldsymbol{out}; M_{\mathsf{GM}}; \mathsf{st})$.
- $\mathbf{Join}_{\mathsf{GM}}^{\mathsf{ReadReg}(\mathsf{i})}(\mathsf{i}, M_{\mathsf{GM}}, \mathsf{msk}; \mathsf{st}_{\mathsf{GM}}^{\mathsf{i}}) \to (\boldsymbol{out}_{\mathsf{GM}}; M; \mathsf{st}_{\mathsf{GM}}^{\mathsf{i}})$.

$\mathbf{IsActive}(\mathsf{i}, \mathsf{st}_{\mathsf{GM}}) \to 1/0$: DPT algorithm defining when user is considered active. In the partially dynamic case, the policy is that the user joining in session i is active if and only if the group manager terminated with success symbol $\top$.

$\mathbf{Sign}(\mathsf{gsk}, m) \to \Sigma$**:** PPT signing algorithm.

$\mathbf{Verify}(\mathsf{gpk}, m, \Sigma) \to 1/0$**:** DPT verification algorithm.

$\mathbf{Open}^{\mathsf{ReadReg}}(\mathsf{gpk}, \mathsf{msk}, m, \Sigma) \to \mathsf{i}$**:** DPT opening algorithm.

The matching security experiments are given in Fig. 7.20. The oracles are defined exactly as in the previous definitions.

The security definition given in Fig. 7.20 is close to the security definition given by Kiayias and Yung. There are some immaterial differences, such as allowing arbitrary identifier i versus numbering them consecutively, and their specifying that the registry must have entries of a specific form corresponding to the joining transcript. There is one significant difference in the anonymity experiment. Here Kiayias and Yung require indistinguishability of two signers as long as the keys are consistent with the protocol, i.e., could plausibly have been generated. We on the other hand, just require indistinguishability for honestly generated keys. In our opinion, the stronger anonymity notion of Kiayias and Yung is overkill; it is reasonable to assume honest signers will follow the protocol and therefore our anonymity experiment suffices to protect them. Aside from this difference, inspection reveals that there are mainly notational and terminological differences between our security definition in Fig. 7.20 and Kiayias and Yung [KY06].

Experiment: $\mathbf{Exp}^{\mathsf{Corr}}_{\mathcal{KY},\mathcal{A}}(\lambda)$
– $(\mathsf{gpk}, \mathsf{msk}) \leftarrow \mathbf{GKGen}(1^\lambda)$; $\mathsf{h} := \perp$; $N := 0$; $K := 0$
– $m \leftarrow \mathcal{A}^{\mathsf{AddHU},\mathsf{SndToM},\mathsf{WriteReg},\mathsf{State}}(\mathsf{gpk})$
– If $K = k(\lambda)$ and $\mathbf{IsActive}(\mathsf{h}, \mathsf{st}_{\mathsf{GM}}) = 0$ return 0
– If $\mathbf{IsActive}(\mathsf{h}, \mathsf{st}_{\mathsf{GM}}) = 0$ return 1
– $\Sigma \leftarrow \mathbf{Sign}(\mathsf{gsk}_{\mathsf{h}}, m)$
– Return $\mathbf{Verify}(\mathsf{gpk}, m, \Sigma)$

Experiment: $\mathbf{Exp}^{\mathsf{Anon}-b}_{\mathcal{KY},\mathcal{A}}(\lambda)$
– $(\mathsf{gpk}, \mathsf{msk}) \leftarrow \mathbf{GKGen}(1^\lambda)$; $N := 0$; $\mathcal{H} := \emptyset$; $\mathcal{C} := \emptyset$
– Return $b^* \leftarrow \mathcal{A}^{\mathsf{AddHU},\mathsf{SndToM},\mathsf{Open},\mathsf{Chal}_b,\mathsf{ReadReg}}(\mathsf{gpk})$

Experiment: $\mathbf{Exp}^{\mathsf{Trace}}_{\mathcal{KY},\mathcal{A}}(\lambda)$
– $(\mathsf{gpk}, \mathsf{msk}) \leftarrow \mathbf{GKGen}(1^\lambda)$
– $(m, \Sigma) \leftarrow \mathcal{A}^{\mathsf{SndToM},\mathsf{WriteReg},\mathsf{Open}}(\mathsf{gpk})$
– If $\mathbf{Verify}(\mathsf{gpk}, m, \Sigma) = 0$ return 0
– $\mathsf{i} \leftarrow \mathbf{Open}^{\mathsf{ReadReg}}(\mathsf{gpk}, \mathsf{msk}, m, \Sigma)$
– If $\mathbf{IsActive}(\mathsf{i}, \mathsf{st}_{\mathsf{GM}}) = 0$ return 1, else return 0

Experiment: $\mathbf{Exp}^{\mathsf{Non}-\mathsf{Frame}}_{\mathcal{KY},\mathcal{A}}(\lambda)$
– $(\mathsf{gpk}, \mathsf{msk}) \leftarrow \mathbf{GKGen}(1^\lambda)$; $\mathsf{h} := \perp$; $\mathcal{S} := \emptyset$
– $\Big(m, \Sigma\Big) \leftarrow \mathcal{A}^{\mathsf{SndToHU},\mathsf{SignHU}}(\mathsf{gpk}, \mathsf{msk})$
– If $\mathbf{Verify}(\mathsf{gpk}, m, \Sigma) = 0$ return 0
– If $(m, \Sigma) \in \mathcal{S}$ return 0
– If $\mathbf{Open}^{\mathsf{ReadReg}}(\mathsf{gpk}, \mathsf{msk}, m, \Sigma) = \mathsf{h}$ return 1, else return 0

FIGURE 7.20: Security experiments for partially dynamic group signatures akin to Kiayias and Yung [KY06].

7.3 Static Group Signatures

Similarly to the previous section we now discuss further relaxations of our definitions to the static settings and show that definitions of [BMW03] can be seen as special case of ours.

7.3.1 Restriction to Static Group Signatures

In static group signatures, the set of group members is fixed from the start and never changes. This means the setup procedure is different, since it includes the key generation for all the group members, so strictly speaking static group signatures are not just a definitional restriction of dynamic group signatures but a distinct definition altogether. However, we can easily convert a dynamic group signature into a static one by incorporating the join procedure for the users into the key generation process and then at the end hand them their secret keys.

In our definition of static group signatures, we will replace the group manager key generation with a trusted combined key generation protocol that also generates keys for the group members. Since the set of group members is static, there is no management operations taking place so the sole purpose of the group manager is to be able to open group signatures and identify the signer. We therefore only consider the single authority setting. Also, since membership does not change, the group information does not need to change either and we only have a single epoch, so we can without loss of generality fix $\mathsf{info}_0 = \varepsilon$ and $\tau = 0$ and omit them from the definition. We can also eliminate reference to the **IsActive** procedure since all group members are automatically considered active. Finally, since enrolment of users takes place during setup we can only get non-frameability if the registry is honestly generated. This means that in all the security experiments, we must assume trusted key generation, and therefore we can incorporate the trusted parameters generation into the key generation procedure.

With these changes in mind we get the following security experiments for static group signatures, where N is an arbitrary polynomially bounded function of the security parameter.

Reg: Data structure with records reg_i for members numbered $\mathsf{i} = 1, 2, 3, \ldots, N$.

- $\mathsf{ReadReg}(\mathsf{i})$: Return reg_i (or $\perp$ if no such record exists).
- $\mathsf{WriteReg}(\mathsf{i}, M)$: Set $\mathsf{reg}_\mathsf{i} := M$ and ignore further calls with the same i.

$\mathbf{GKGen}^{\mathsf{WriteReg}}(1^\lambda, N) \to (\mathsf{gpk}, \mathsf{gsk}_1, \ldots, \mathsf{gsk}_N; \mathsf{st}_\mathsf{GM})$**:** PPT algorithm for group manager key generation, which depends on the number of desired group members N. Returns the group public key, secret keys for the users, writes registry entries $\mathsf{reg}_1, \ldots, \mathsf{reg}_N$, and sets the state of the group manager (which can be interpreted as a group manager secret key).

$\mathbf{Sign}(\mathsf{gsk}, m) \to \Sigma$**:** PPT signing algorithm.

$\mathbf{Verify}(\mathsf{gpk}, m, \Sigma) \to 1/0$**:** DPT verification algorithm.

$\mathbf{Open}(\mathsf{gpk}, \mathsf{st}_\mathsf{GM}, m, \Sigma) \to (\mathsf{i}, \pi)$**:** PPT opening algorithm.

$\mathbf{Judge}(\mathsf{gpk}, \mathsf{reg}, m, \Sigma, \pi) \to 1/0$**:** DPT algorithm determining if signature was correctly attributed to registry record reg.

The matching simplified security experiments for static group signature scheme and oracle Keys are given in Fig. 7.21. The remaining oracles are defined exactly as in the case of fully dynamic group signatures and simplified by excluding the group information $\mathsf{info} = \varepsilon$ and epochs $\tau = 0$.

7.3.2 Comparison to Bellare, Micciancio and Warinschi [BMW03]

Bellare, Micciancio and Warinschi [BMW03] gave a formal definition of static group signatures where group membership is fixed at the beginning of the protocol. Their definition does not include proof of correct opening, they just require the group manager to identify the signer. This can be seen as a special case of our static group signature scheme, where we omit the **Judge** algorithm and simply trust the judgment of the group manager. We still need the core properties of traceability, i.e., we can open all valid signatures to one of the members, and still want from non-frameability that a signature will not be opened to a member who did not sign it. We present the simplified definition of static group signatures in Fig. 7.22 and the syntax below. Oracle are defined as for the previous definitions. Bellare et al. [BMW03] combine the traceability and non-frameability notions into a combined notion they call full-traceability. It

Experiment: $\mathbf{Exp}^{\mathsf{Corr}}_{\mathcal{SGS},\mathcal{A},N}(\lambda)$
– $(\mathsf{gpk}, \mathsf{gsk}_1, \ldots, \mathsf{gsk}_N; \mathsf{st}_{\mathsf{GM}}) \leftarrow \mathbf{GKGen}^{\mathsf{WriteReg}}(1^\lambda, N)$
– $(\mathsf{h}, m) \leftarrow \mathcal{A}^{\mathsf{ReadReg}}(\mathsf{gpk}, \mathsf{gsk}_1, \ldots, \mathsf{gsk}_N, \mathsf{st}_{\mathsf{GM}})$
– If $\mathsf{h} \notin [N]$ return 1
– $\Sigma \leftarrow \mathbf{Sign}(\mathsf{gsk}_{\mathsf{h}}, m)$
– Return $\mathbf{Verify}(\mathsf{gpk}, m, \Sigma)$

Experiment: $\mathbf{Exp}^{\mathsf{Anon}-b}_{\mathcal{SGS},\mathcal{A},N}(\lambda)$
– $(\mathsf{gpk}, \mathsf{gsk}_1, \ldots, \mathsf{gsk}_N; \mathsf{st}_{\mathsf{GM}}) \leftarrow \mathbf{GKGen}^{\mathsf{WriteReg}}(1^\lambda, N)$
– Return $b^* \leftarrow \mathcal{A}^{\mathsf{Open},\mathsf{Chal}_b,\mathsf{ReadReg}}(\mathsf{gpk}, \mathsf{gsk}_1, \ldots, \mathsf{gsk}_N)$

Experiment: $\mathbf{Exp}^{\mathsf{Trace}}_{\mathcal{SGS},\mathcal{A},N}(\lambda)$
– $(\mathsf{gpk}, \mathsf{gsk}_1, \ldots, \mathsf{gsk}_N; \mathsf{st}_{\mathsf{GM}}) \leftarrow \mathbf{GKGen}^{\mathsf{WriteReg}}(1^\lambda, N)$
– $(m, \Sigma) \leftarrow \mathcal{A}^{\mathsf{ReadReg}}(\mathsf{gpk}, \mathsf{gsk}_1, \ldots, \mathsf{gsk}_N, \mathsf{st}_{\mathsf{GM}})$
– If $\mathbf{Verify}(\mathsf{gpk}, m, \Sigma) = 0$ return 0
– $(\mathsf{i}, \pi) \leftarrow \mathbf{Open}(\mathsf{gpk}, \mathsf{st}_{\mathsf{GM}}, m, \Sigma)$
– If $\mathbf{Judge}(\mathsf{gpk}, \mathsf{reg}_{\mathsf{i}}, m, \Sigma, \pi) = 0$ return 1
– Return 0

Experiment: $\mathbf{Exp}^{\mathsf{Non}-\mathsf{Frame}}_{\mathcal{SGS},\mathcal{A},N}(\lambda)$
– $\mathcal{S} = \emptyset, \mathsf{h} = \perp$
– $(\mathsf{gpk}, \mathsf{gsk}_1, \ldots, \mathsf{gsk}_N; \mathsf{st}_{\mathsf{GM}}) \leftarrow \mathbf{GKGen}^{\mathsf{WriteReg}}(1^\lambda, N)$
– $\Big(m, \Sigma, \pi\Big) \leftarrow \mathcal{A}^{\mathsf{ReadReg},\mathsf{Keys},\mathsf{SignHU}}\Big(\mathsf{gpk}, \mathsf{st}_{\mathsf{GM}}\Big)$
– If $\mathbf{Verify}(\mathsf{gpk}, m, \Sigma) = 0$ return 0
– If $(m, \Sigma) \in \mathcal{S}$ return 0
– Return $\mathbf{Judge}(\mathsf{gpk}, \mathsf{reg}_{\mathsf{h}}, m, \Sigma, \pi)$

Oracle: $\mathsf{Keys}(\mathsf{h})$
. If $\mathsf{h} \notin [N]$ return $\perp$
. Return $\{\mathsf{gsk}_i\}_{i \neq \mathsf{h}}$ and ignore future calls

FIGURE 7.21: Security experiments for static group signatures.

is not hard to see though that the definition given in Fig. 7.22 is almost equivalent to their security definition for static group signatures.

$\mathbf{GKGen}(1^\lambda, N) \to (\mathsf{gpk}, \mathsf{gsk}_1, \ldots, \mathsf{gsk}_N; \mathsf{st}_{\mathsf{GM}})$**:** PPT algorithm for group manager key generation, which depends on the number of desired group members N. Returns the group public key, secret keys for the users, and sets the state of the group manager (which can be interpreted as a group manager secret key).

$\mathbf{Sign}(\mathsf{gsk}, m) \to \Sigma$**:** PPT signing algorithm

$\mathbf{Verify}(\mathsf{gpk}, m, \Sigma) \to 1/0$**:** DPT verification algorithm.

$\mathbf{Open}(\mathsf{gpk}, \mathsf{st}_{\mathsf{GM}}, m, \Sigma) \to \mathsf{i}$**:** DPT opening algorithm.

Experiment: $\mathbf{Exp}^{\mathsf{Corr}}_{\mathcal{BMW},\mathcal{A},N}(\lambda)$
- $(\mathsf{gpk}, \mathsf{gsk}_1, \ldots, \mathsf{gsk}_N; \mathsf{st}_{\mathsf{GM}}) \leftarrow \mathbf{GKGen}(1^\lambda, N)$
- $(\mathsf{h}, m) \leftarrow \mathcal{A}(\mathsf{gpk}, \mathsf{gsk}_1, \ldots, \mathsf{gsk}_N, \mathsf{st}_{\mathsf{GM}})$
- If $\mathsf{h} \notin [N]$ return 1
- $\Sigma \leftarrow \mathbf{Sign}(\mathsf{gsk}_{\mathsf{h}}, m)$
- Return $\mathbf{Verify}(\mathsf{gpk}, m, \Sigma)$

Experiment: $\mathbf{Exp}^{\mathsf{Anon}-b}_{\mathcal{BMW},\mathcal{A},N}(\lambda)$
- $(\mathsf{gpk}, \mathsf{gsk}_1, \ldots, \mathsf{gsk}_N; \mathsf{st}_{\mathsf{GM}}) \leftarrow \mathbf{GKGen}(1^\lambda, N)$
- Return $b^* \leftarrow \mathcal{A}^{\mathsf{Open},\mathsf{Chal}_b}(\mathsf{gpk}, \mathsf{gsk}_1, \ldots, \mathsf{gsk}_N)$

Experiment: $\mathbf{Exp}^{\mathsf{Trace}}_{\mathcal{BMW},\mathcal{A},N}(\lambda)$
- $(\mathsf{gpk}, \mathsf{gsk}_1, \ldots, \mathsf{gsk}_N; \mathsf{st}_{\mathsf{GM}}) \leftarrow \mathbf{GKGen}(1^\lambda, N)$
- $(m, \Sigma) \leftarrow \mathcal{A}(\mathsf{gpk}, \mathsf{gsk}_1, \ldots, \mathsf{gsk}_N, \mathsf{st}_{\mathsf{GM}})$
- If $\mathbf{Verify}(\mathsf{gpk}, m, \Sigma) = 0$ return 0
- $\mathsf{i} \leftarrow \mathbf{Open}(\mathsf{gpk}, \mathsf{st}_{\mathsf{GM}}, m, \Sigma)$
- If $\mathsf{i} \notin [N]$ return 1
- Return 0

Experiment: $\mathbf{Exp}^{\mathsf{Non}-\mathsf{Frame}}_{\mathcal{BMW},\mathcal{A},N}(\lambda)$
- $\mathcal{S} = \emptyset, \mathsf{h} = \perp$
- $(\mathsf{gpk}, \mathsf{gsk}_1, \ldots, \mathsf{gsk}_N; \mathsf{st}_{\mathsf{GM}}) \leftarrow \mathbf{GKGen}(1^\lambda, N)$
- $\Big(m, \Sigma\Big) \leftarrow \mathcal{A}^{\mathsf{Keys},\mathsf{SignHU}}\Big(\mathsf{gpk}, \mathsf{st}_{\mathsf{GM}}\Big)$
- If $\mathbf{Verify}(\mathsf{gpk}, m, \Sigma) = 0$ return 0
- If $(m, \Sigma) \in \mathcal{S}$ return 0
- If $\mathbf{Open}(\mathsf{gpk}, \mathsf{st}_{\mathsf{GM}}, m, \Sigma) = \mathsf{h}$ return 1, else return 0

FIGURE 7.22: Security experiments for static group signatures akin to Bellare, Micciancio and Warinschi [BMW03].

If we have a static group signature scheme with proofs of correct opening satisfying the definition in Fig. 7.21, then it is easy to convert it into a similar group signature without proofs of correct opening satisfying the definition in Fig. 7.22. The group key generation algorithm can include registry record reg_i in the secret signing key gsk_i and the group manager state $\mathsf{st}_{\mathsf{GM}}$ may include the entire registry (we could in principle instead include the registry information in the group public key, but this might lead to an undesirable increase in the size of gpk). When the **Open** algorithm is called, it runs the original opening algorithm to get (i, π), verifies the proof using the **Judge** algorithm, and returns i. It follows from the security definitions in Fig. 7.21 that this simple modification leads to a static group signature scheme without proof of correct opening that satisfies the definitions in Fig. 7.22.

7.4 Fully Dynamic Group Signatures from Accountable Ring Signatures

Bootle et al. [BCC+15] give a generic construction of accountable ring signatures, where every signature can be traced back to a user in the ring. Differently from group signatures, accountable ring signatures lack of appointed authorities, and thus signers choose their designated opener at signing time. In addition, they give an efficient instantiation in the random oracle model that is based on the DDH assumption. Their instantiation yields signatures of logarithmic size (in the size of the ring), they cost quasi-linear time to generate, and linear time to verify.

In this section we show that accountable ring signatures imply fully dynamic group signatures. We start by recalling the security definitions of accountable ring signatures and then present a generic construction of fully dynamic group signatures from accountable ring signatures. Combined with [BCC+15], this gives a generic construction of fully dynamic group signatures from one-way functions, IND-CPA encryption, and non-interactive zero-knowledge proofs. We then show that our construction satisfies the strongest variant of our definitions, i.e. with respect to separate authorities and adversarial key generation.

7.4.1 Accountable Ring Signatures

Bootle et al. [BCC+15] define an accountable ring signature scheme over a PPT setup ARS_{Setup} as a tuple of polynomial time algorithms $(\mathsf{ARS}_{OKGen}, \mathsf{ARS}_{UKGen}, \mathsf{ARS}_{Sign}, \mathsf{ARS}_{Vfy}, \mathsf{ARS}_{Open}, \mathsf{ARS}_{Judge})$.

$\mathsf{ARS}_{Setup}(1^\lambda)$**:** Given the security parameter, produces public parameters pp used (sometimes implicitly) by the rest of the scheme. The public parameters define key spaces PK, DK, VK, SK with efficient algorithms for sampling and deciding membership.

$\mathsf{ARS}_{OKGen}(pp)$**:** Given the public parameters pp, it produces a public key $pk \in PK$ and secret key $dk \in DK$ for an opener. Without loss of generality, we assume dk defines pk deterministically and write $pk = \mathsf{ARS}_{OKGen}(pp, dk)$ when computing pk from dk.

$\mathsf{ARS}_{UKGen}(pp)$**:** Given the public parameters pp, it produces a verification key $vk \in VK$ and a secret signing key $sk \in SK$ for a user. We can assume sk deterministically determines vk and write $vk = \mathsf{ARS}_{UKGen}(pp, sk)$ when computing vk from sk.

$\mathsf{ARS}_{Sign}(pk, sk, R, m)$**:** Given an opener's public key, a message, a ring (i.e. a set of verification keys) and a secret key, it produces a ring signature σ. The algorithm returns the error symbol $\perp$ if $pk \notin PK, R \not\subset VK, sk \notin SK$ or $vk = \mathsf{ARS}_{UKGen}(pp, sk) \notin R$.

$\mathsf{ARS}_{Vfy}(pk, R, m, \sigma)$**:** Given an opener's public key, a message, a ring and a signature, it returns 1 if accepting the signature and 0 otherwise. We assume the algorithm always returns 0 if $pk \notin PK$ or $R \not\subset VK$.

$\mathsf{ARS}_{Open}(dk, R, m, \sigma)$**:** Given a message, a ring, a ring signature and an opener's secret key, it returns a verification key vk and a proof ψ that the owner of vk produced the signature. If $dk \notin DK$ or σ is not a valid signature using $pk = \mathsf{ARS}_{OKGen}(pp, dk)$, the algorithm returns $\perp$.

$\mathsf{ARS}_{Judge}(pk, R, vk, m, \sigma, \psi)$**:** Given an opener's public key, a message, a ring, a signature, a verification key and a proof, it returns 1 if accepting the proof and 0 otherwise. We assume the algorithm returns 0 if σ is invalid or $vk \notin R$.

Accountable ring signatures should be correct, anonymous, traceable, fully unforgeable and tracing sound. We recall [BCC+15] definitions of all these properties below.

Definition 7.8 (Correctness)**.** *An accountable ring signature scheme is correct if for any PPT adversary* $\mathcal{A}$

$$\Pr\left[\begin{array}{c} pp \leftarrow \mathsf{ARS}_{Setup}(1^\lambda); (vk, sk) \leftarrow \mathsf{ARS}_{UKGen}(pp); \\ (pk, R, m) \leftarrow \mathcal{A}(pp, sk); \sigma \leftarrow \mathsf{ARS}_{Sign}(pk, sk, R, m) : \\ \text{If } pk \in PK, R \subset VK, vk \in R \text{ then } \mathsf{ARS}_{Vfy}(pk, R, m, \sigma) = 1 \end{array}\right] \approx 1$$

Anonymity ensures that a signature does not reveal the identity of the ring member who produced it without the opener explicitly wanting to open the particular signature. The definition below implies anonymity against full key exposure attacks ([BKM09]) since in the game the adversary is allowed to choose the secret signing keys of the users.

Definition 7.9 (Anonymity)**.** *An accountable ring signature scheme is anonymous if for any PPT adversary* $\mathcal{A}$

$$\Pr\left[\begin{array}{c} pp \leftarrow \mathsf{ARS}_{Setup}(1^\lambda); b \leftarrow \{0,1\}; (pk, dk) \leftarrow \mathsf{ARS}_{OKGen}(pp): \\ \mathcal{A}^{\mathrm{Chal}_b, \mathrm{Open}}(pp, pk) = b \end{array}\right] \approx \frac{1}{2}$$

- Chal_b*: is an oracle that the adversary can only call once. On query* (R, m, sk_0, sk_1) *it runs* $\sigma_0 \leftarrow \mathsf{ARS}_{Sign}(pk, sk_0, R, m)$; $\sigma_1 \leftarrow \mathsf{ARS}_{Sign}(pk, sk_1, R, m)$*. If* $\sigma_0 \neq \perp$ *and* $\sigma_1 \neq \perp$ *it returns* σ_b*, otherwise it returns* $\perp$*.*

- Open*: is an oracle that on a query* (R, m, σ) *returns* $\mathsf{ARS}_{Open}(dk, R, m, \sigma)$*. If* σ *was obtained by calling* Chal_b *on* $(R, m, \cdot, \cdot)$*, the oracle returns* $\perp$*.*

Traceability ensures that the specified opener can always identify the ring member who produced a signature and that she is able to produce a valid proof for her decision.

Definition 7.10 (Traceability)**.** *An accountable ring signature scheme is traceable if for any PPT adversary $\mathcal{A}$*

$$\Pr\left[\begin{array}{c} pp \leftarrow \mathsf{ARS}_{Setup}(1^\lambda); (dk, R, m, \sigma) \leftarrow \mathcal{A}(pp); \\ pk \leftarrow \mathsf{ARS}_{OKGen}(pp, dk); (vk, \psi) \leftarrow \mathsf{ARS}_{Open}(dk, R, m, \sigma): \\ \mathsf{ARS}_{Vfy}(pk, R, m, \sigma) = 1 \ \wedge \ \mathsf{ARS}_{Judge}(pk, R, vk, m, \sigma, \psi) = 0 \end{array}\right] \approx 0$$

Full unforgeability ensures that an adversary, who may control the opener, can neither falsely accuse an honest user of producing a ring signature nor forge ring signatures on behalf of an honest ring. The former should hold even when all other users in the ring are corrupt.

Definition 7.11 (Full Unforgeability)**.** *An accountable ring signature scheme is fully unforgeable if for any PPT adversary $\mathcal{A}$*

$$\Pr\left[\begin{array}{c} pp \leftarrow \mathsf{ARS}_{Setup}(1^\lambda); (pk, vk, R, m, \sigma, \psi) \leftarrow \mathcal{A}^{\mathrm{UKGen,Sign,RevealU}}(pp): \\ \Big(vk \in Q_{\mathrm{UKGen}} \setminus Q_{\mathrm{RevealU}} \ \wedge \ (pk, vk, R, m, \sigma) \notin Q_{\mathrm{Sign}} \\ \wedge \mathsf{ARS}_{Judge}(pk, R, vk, m, \sigma, \psi) = 1\Big) \\ \vee \ \Big(R \subset Q_{\mathrm{UKGen}} \setminus Q_{\mathrm{RevealU}} \ \wedge \ (pk, \cdot, R, m, \sigma) \notin Q_{\mathrm{Sign}} \\ \wedge \ \mathsf{ARS}_{Vfy}(pk, R, m, \sigma) = 1\Big) \end{array}\right] \approx 0$$

- UKGen*: runs $(vk, sk) \leftarrow \mathsf{ARS}_{UKGen}(pp)$ and returns vk. Q_{UKGen} is the set of verification keys vk that have been generated by this oracle.*
- Sign*: is an oracle that on query (pk, vk, R, m) checks if $vk \in R \cap Q_{\mathrm{UKGen}}$, in which case returns $\sigma \leftarrow \mathsf{ARS}_{Sign}(pk, sk, R, m)$. Q_{Sign} contains the queries and responses (pk, vk, R, m, σ).*
- RevealU*: is an oracle that when queried on $vk \in Q_{\mathrm{UKGen}}$ returns the corresponding signing key sk. Q_{RevealU} is the list of verification keys vk for which the corresponding signing key has been revealed.*

Tracing soundness is analogous to our opening binding definition for group signatures and ensures that a signature cannot be traced to two different users in the ring. That is, only one person can be identified as the signer, even when all users, as well as the opener, are fully corrupt.

Definition 7.12 (Tracing Soundness)**.** *An accountable ring signature scheme satisfies tracing soundness if for any PPT adversary $\mathcal{A}$*

$$\Pr\left[\begin{array}{l} pp \leftarrow \mathsf{ARS}_{Setup}(1^\lambda); (pk, R, m, \sigma, vk_1, \psi_1, vk_2, \psi_2) \leftarrow \mathcal{A}(pp): \\ \forall i \in \{1,2\},\ \mathsf{ARS}_{Judge}(pk, R, vk_i, m, \sigma, \psi_i) = 1 \wedge\ vk_1 \neq vk_2 \end{array}\right] \approx 0.$$

7.4.2 Generic Construction from Accountable Ring Signatures

Next, we show a generic construction of a fully dynamic group signature scheme obtained from an accountable ring signature. The main difference between the two primitives is that the latter does not feature designated authorities. In order to minimise the amount of trust placed in the authority, we set our group signature in the case of separate group manager and opening authority.

The key generation protocol of our group signature consists of two independent processes run by the group manager and the opening authority, respectively. The opening authority computes a pair of keys by executing the opener key generation algorithm ARS_{OKGen}, and announces her public key. The group manager simply initialise her internal state and the group information, which are initially set equal to empty sets.

A user initiates a joining session with the group manager by generating a key pair (vk, sk), using ARS_{UKGen}, and then she adds the verification key vk into the registry. The manager reads the registry entry and checks if the same key had already being registered, in which case the joining fails. We do not spell out the details of the registry, but we assume users can write once into the registry and that the manager can read the content of it. We recall that a registry offering these features can be instantiated with a PKI and thus we abstract it out to simplify the construction.

The group manager stores the outcome of all the joining sessions as well as the lists $\mathcal{I}_\tau$ of active users at each epoch. To activate new members, the manager keeps a list of users that have joined in the current epoch and updates the group information info, which triggers a new epoch. The information of the group info consists of the verification keys of all active members. A group signature consists of an accountable ring signature using the current information info as the ring, and the opening authority public key as the opener's key.

Details of our fully-dynamic group signature from an accountable ring signature are given in Figure 7.23.

7.4.3 Security in our Separate Authorities Model

Theorem 7.1. *The generic group signature scheme construction from accountable ring signatures of Figure 7.23 satisfies our separate authority definitions for a secure, fully-dynamic group signature scheme, and is additionally opening binding.*

Proof. We use a similar proof strategy for all properties: we assume the existence of an adversary $\mathcal{A}$ against the corresponding property of the group signature scheme. We then show how to build an adversary $\mathcal{B}$ that uses $\mathcal{A}$ to break the same property of the accountable ring signature.

For correctness, we start with $\mathcal{B}$ in the correctness experiment of Definition 7.8. The adversary receives pp and the secret key sk of the target user. Given pp, the adversary $\mathcal{B}$ provides param to $\mathcal{A}$ and let her pick the public key opk for the opening authority of the group signature, while $\mathcal{B}$ plays the role of the honest group manager in the game of Figure 7.3. The adversary $\mathcal{B}$ generates the initial group information info_0 and provides it to $\mathcal{A}$. When simulating AddHU for $\mathcal{A}$, $\mathcal{B}$ uses the secret key sk for the challenge user; note that vk can be efficiently obtained given sk. The oracle calls to $\mathsf{SndToM}, \mathsf{Update}, \mathsf{Write}, \mathsf{State}$ used by $\mathcal{A}$ are also simulateable by $\mathcal{B}$, which keeps the internal state of the group manager. Once $\mathcal{A}$ returns a message and epoch pair (m, τ), $\mathcal{B}$ retrieves the group information info_τ, consisting of the public keys of active group members at epoch τ, and returns $(\mathsf{opk}, m, \mathsf{info}_\tau)$. We observe that $\mathcal{A}$ only wins the game of Figure 7.3 when either the registration protocol fails to complete successfully for the target user, or the target user is flagged as inactive even though she has joined and is not revoked, or if the produced signature fails to verify. The registration of the target user fails only in case $\mathcal{A}$ has already successfully registered the same verification key vk in a previous session. Since vk is not exposed to $\mathcal{A}$ before AddHU is called, this corresponds to guessing the target key vk. Assuming the key space VK is large enough, this only happens with small probability and we can thus assume the target user to successfully complete the joining protocol. When $\mathcal{B}$ updates the group information, the target user verification key is included in info until explicitly removed by another

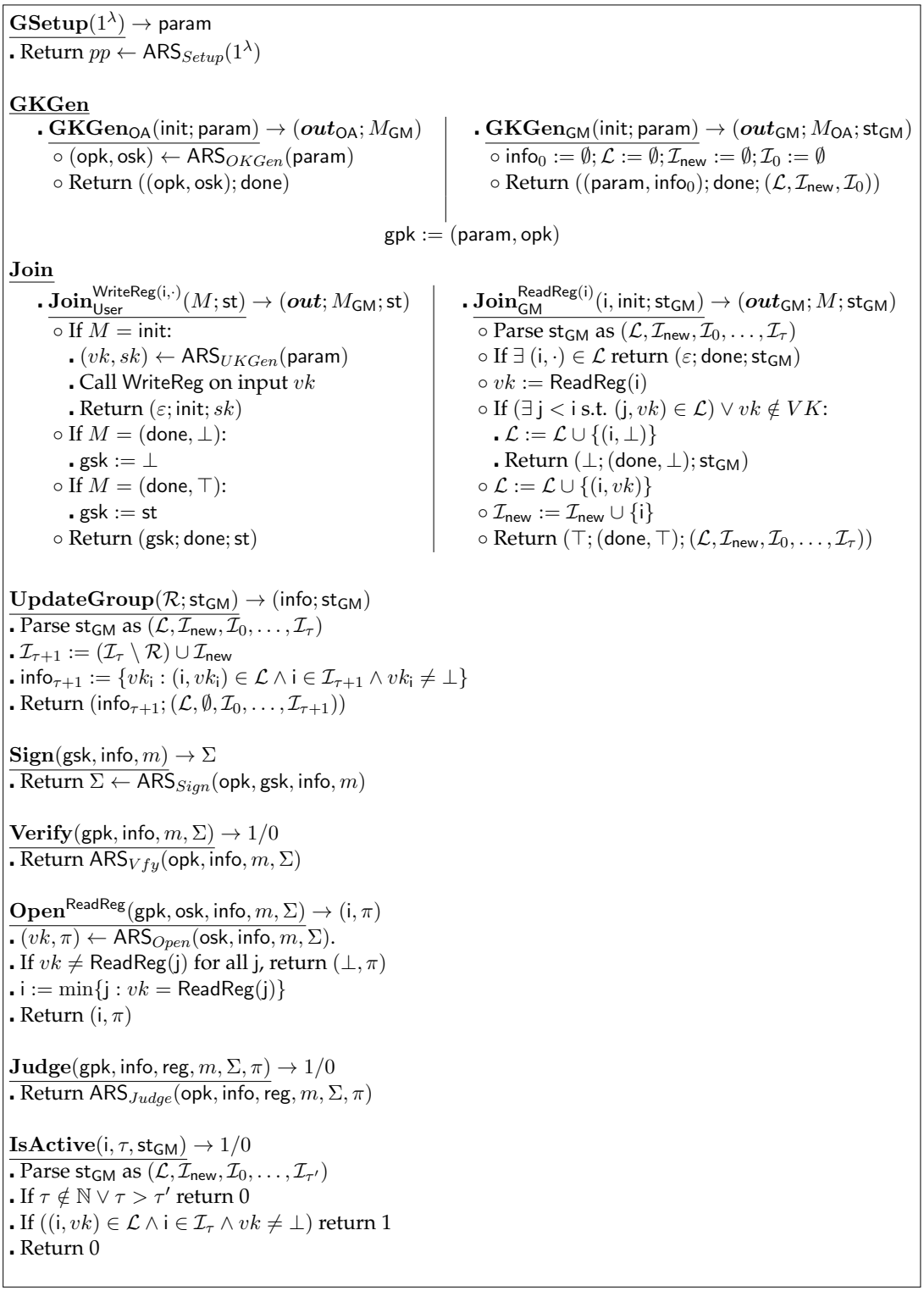

FIGURE 7.23: Construction of a fully dynamic group signature from an accountable ring signature [BCC+15] .

update. The verification key can only be removed either by revoking the target user or if the adversary adds another group member with the same vk and then requests to revoke her. However, registering the same verification key in a later session of the joining protocol would cause the new session to fail. The remaining winning condition of $\mathcal{A}$ in the game of Figure 7.3 is captured by the the last line of Definition 7.8.

For anonymity, we follow the same strategy: the adversary $\mathcal{B}$ plays the game of Definition 7.9 and obtains the public parameters pp and a public key pk. Then he proceeds to simulate the key generation protocol for the group signature (Figure 7.6): he lets $\mathcal{A}$ generate the group manager public key and sets pk as the opening public key of the opening authority. Note that the key generation protocol in Figure 7.23 consists of each authority independently producing and announcing their own keys. This implies that pk is sufficient for $\mathcal{B}$ to simulate the protocol. Finally, $\mathcal{B}$ needs to simulate $\mathcal{A}$'s oracle calls. For the challenge and opening oracles, this is done by using his own oracles, whereas for $\mathsf{SndToU}, \mathsf{ReadReg}$ it can simply answer directly. We note that $\mathcal{B}$ will correctly guess the challenge if and only if $\mathcal{A}$'s guess is also correct.

For traceability, $\mathcal{B}$ starts by receiving pp and internally running both sides of the key generation protocol. He then starts the group signature traceability game of Figure 7.10 and calls $\mathcal{A}$. As $\mathcal{B}$ plays the role of the group manager in $\mathcal{A}$'s game, he can directly reply to her oracle queries. To complete the proof, we examine when $\mathcal{A}$ wins her game: it must be the case that user i is inactive or that the **Judge** algorithm fails. In the accountable ring signature definition the first case folds into the second: i being inactive implies $vk_i \notin R$ which will explicitly cause the judging algorithm to fail. In either case, $\mathcal{A}$ succeeding in the game of Figure 7.10 implies $\mathcal{B}$ succeeding according to Definition 7.10.

For non-frameability, let $\mathcal{B}$ play in the full unforgeability game of Definition 7.11. The adversary $\mathcal{B}$ receives pp and initialises $\mathcal{A}$ in the game of Figure 7.14, which generates the group public key gpk. $\mathcal{B}$ uses his oracles UKGen and Sign to respond to the queries of the $\mathsf{SndToHU}$ and SignHU, respectively. Adversary $\mathcal{B}$ forwards $\mathcal{A}$'s output together with the opener public key and the honest user verification key. A wining output by $\mathcal{A}$ in her game will cause $\mathcal{B}$ to win via the first branch on his own game: note that $\mathcal{B}$ does not make use of his $\mathrm{RevealU}$ oracle and that $\mathcal{A}$ does not output a user identity since she only includes a single honest user in the group.

Reduction to opening binding is near-trivial from tracing soundness: $\mathcal{B}$ simply passes the setup parameters and converts the output of $\mathcal{A}$ into a ring. We complete the proof by pointing out that under the accountable ring signatures definitions ARS_{Judge} implies correct verification. □

Instantiating the Register. The joining protocol specified in Figure 7.23 assumes the existence of an ideal register which could be instantiated in different ways. We do point out however, that for the traceability proof to go through, it is necessary that the manager does not accept keys that have already been registered. At the same time, to avoid the opening authority to misattribute signatures, it is important for the registry to be robust enough, such that attempts to modify or copy its entries will fail. The former is captured by our idealization of the registry, which we recall can be realised with a PKI. The latter is accomplished by considering valid only the first occurrence of a key in the registry, and ignoring all later occurrences as the group manager should have rejected the corresponding joining sessions.

7.5 On the Security of Constructions Based on Revocation Lists

A common approach for designing efficient fully dynamic group signatures is by using revocation lists. Users interacts with the manager to obtain a certificate for their group membership. The group information, periodically updated by the manager, consists of a revocation list which stores information about the revoked users. To sign, users have to show they hold a valid certificate and that they are not part of the set of revoked users. Examples of efficient schemes following this approach include the ones of Libert et al. [LPY12b; LPY12a] and Nakanishi et al. [NFH+10].

In these constructions, a user is considered authorised to sign unless the manager explicitly includes her in the revocation list. However, this means that a user can also sign with respect to old epochs, including those predating her joining to the group. As the user was not yet part of the group at that time, it raises the question to whether this represents an issue for the security of the scheme.

At first glance, this is the dual of a well known issue with many revocation systems. If a user is revoked and anonymity is maintained, the revoked user is able to produce

back-dated signatures that still verify. The difference here is that while the revoked user *was* authorised to be part of the group for the epoch in question, in the situation described above, however, the signing user was in fact *not* part of the group at that time. If the adversary is able to obstruct or delay the opening of this signature (e.g. via legal action), its existence would implicitly frame the group's past membership as the signature would be attributed to them.

The issue resides mainly on the interpretation one gives to the epochs in the lifespan of the scheme. Namely, whether epochs reference the state of the group at the time the corresponding group information was created. Our model does not opt for a specific choice, as different design paradigms may have different takes on this. To capture different options, our model includes the **IsActive** algorithm for spelling out the conditions that makes a user active. This helps to clear potential ambiguities a construction may have regarding the timespan users are meant to be authorised sign. Moreover, it also enables to compare the security achieved by different schemes based on how strong their underlying **IsActive** policy is.

The **IsActive** policy has to satisfy some necessary requirements, imposed by our model, which ensure the definitions capture the intended security notions. One of these requirements is the following: if user i is associated with a joining session where the group manager ended her part successfully *before* info_τ was created, and user i is not revoked at or before epoch τ, the algorithm returns 1. However, the policy does not impose a specific outcome in case the joining session terminates *after* the info_τ was created, which could then be set equal to either 0 or 1. For example, we can include the following condition into the policy

- If i is associated with a joining session where the group manager ended her part *after* info_τ was created, and user i is not revoked at or before epoch τ, the algorithm returns 0.

Observe that this requirement is achieved by the policy of the construction in Figure 7.23. On the other hand, this policy may be too strict for constructions following the revocation list approach, as members can typically sign with respect to epochs predating their enrolment into the group. The violation of the above condition translates into a trivial attack against traceability: an adversary can simply enrol a user and then

return a signature produced by the same user with respect to an epoch predating her joining epoch. If the signature is valid, this represents a breach of traceability because the user is not regarded as active with respect to that epoch. We notice that in case such a policy is adopted, construction such as [LPY12b; LPY12a; NFH+10] are all susceptible to this attack. For such constructions we can then replace the above with the following condition, which was implicitly assumed in their respective models

- If i is associated with a joining session where the group manager ended her part *successfully* after info_τ was created, and user i is not revoked at or before epoch τ, the algorithm returns 1.

Note that the **IsActive** algorithm receives as input the internal state of the group manager, thus the outcome of the policy can depend on whether the joining session for user i has currently terminated. This enables to capture the immediate activation of certificate based constructions, i.e users become active as soon as they successfully terminate the join protocol, without requiring further action from the group manager.

Given the above trivial attack, it seems that constructions based on revocation lists can only achieve a slightly weaker notion of traceability than the construction presented in the previous section. Whether the difference on the two notions is substantial may depend on the intended applications of the primitive. In the case of [LPY12b; LPY12a; NFH+10], these constructions can be easily adjusted to support stronger policies. At an high level, these schemes fix the maximal number of users of the system in the setup of the primitive. Therefore, one could initialise all users that have not yet joined the protocol as revoked. In this way, the revocation list is used to effectively store the configuration of the group at each epoch. However, this modification may affect the efficiency of these constructions, introducing dependencies on the maximal number of users rather than the number of revoked members.

Chapter 8

Conclusions

In the first part of this work we addressed the question of whether it is possible to construct zero-knowledge proofs and arguments with constant computational overhead for the prover and gave a positive answer to it. In the process, we introduced an information theoretic model, the ILC model, in which we assembled the core components of our proofs. The level of abstraction encompassed by the model contributed to isolate the key features of the arguments presented by Groth in [Gro09], decoupling them from the underlying commitment scheme used in their construction. This was a pivotal process towards the achievement of our efficiency improvements since it opened up the possibility to instantiate our construction with different primitives and to relax the requirements imposed on these.

Our zero-knowledge proofs and arguments for the satisfiability of arithmetic circuits are highly efficient. They achieve constant computational overhead for the prover, optimal verification time (up to a constant factor), sublinear communication and small round complexity. Our zero-knowledge proofs and arguments for the correct execution of TinyRAM programs do not reach our designated goal of constant computational overhead for the prover. Nonetheless, they achieve good performances and they improve upon the efficiency of existing arguments, which brings us one step closer towards the result. In our arguments the overhead incurred by the prover is an arbitrarily small superconstant function with respect to the time it takes to execute the program directly, while both the verification and communication costs are sublinear.

These contributions leave space for improvements and indicate future research directions. Firstly, our proofs achieve the desired efficiency only for arithmetic circuits

over large enough fields. An interesting question is whether our results can be generalised for Boolean circuits or arithmetic circuits over a small field, without introducing undesirable overheads in computation. As already mentioned above, another interesting question is on the possibility to achieve constant overhead for zero-knowledge proofs for the execution of programs. The challenge underpinning both these questions is related to the reduction of the field size, without affecting the soundness of the proofs.

The other important challenge left open by this work is to achieve good concrete efficiency. Our proofs are not optimised for this, and the constant inside the asymptotic complexity are not yet practical. The main bottleneck here is due to the choice of the primitives we used to instantiate our constructions, which are optimal from an asymptotic point of view. Since our approach is general and imposes minimal requirements on the primitives, other instantiations could offer different efficiency tradeoffs.

Another interesting question comes from the observation that various recent zero-knowledge proofs seem to share similarities, despite they use different approaches, e.g. from the MPC and IOP settings. It is an interesting question to see if these approaches can be combined within a unified model and to exploit the several improvements stemming from the different directions.

In the second part of this work, we proposed a formal security model for fully dynamic group signatures. We think that our definitions provide a good level of abstraction of the primitive, and that this can be used to capture the security of constructions which follow very different approaches from each other. Our model covers a wide spectrum of security definitions and offers several relaxations to help capturing security in different settings. Finally, our model offers stringent security properties which require minimal trust in the designated authorities.

Bibliography

[AW04] M. Abdalla and B. Warinschi, "On the minimal assumptions of group signature schemes", in *Information and Communications Security, 6th International Conference, ICICS 2004, Malaga, Spain, October 27-29, 2004, Proceedings*, 2004, pp. 1–13.

[AFG+16] M. Abe, G. Fuchsbauer, J. Groth, K. Haralambiev, and M. Ohkubo, "Structure-preserving signatures and commitments to group elements", *J. Cryptology*, vol. 29, no. 2, pp. 363–421, 2016.

[AH91] W. Aiello and J. Håstad, "Statistical zero-knowledge languages can be recognized in two rounds", *J. Comput. Syst. Sci.*, vol. 42, no. 3, pp. 327–345, 1991.

[Ale11] M. Alekhnovich, "More on average case vs approximation complexity", *Computational Complexity*, vol. 20, no. 4, pp. 755–786, 2011.

[AHI+17a] S. Ames, C. Hazay, Y. Ishai, and M. Venkitasubramaniam, "Ligero: Lightweight sublinear arguments without a trusted setup", in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, 2017, pp. 2087–2104.

[ABG+13] P. Ananth, R. Bhaskar, V. Goyal, and V. Rao, "On the (in)security of Fischlin's paradigm", in *Theory of Cryptography - 10th Theory of Cryptography Conference, TCC 2013, Tokyo, Japan, March 3-6, 2013. Proceedings*, 2013, pp. 202–221.

[AHI+17b] B. Applebaum, N. Haramaty, Y. Ishai, E. Kushilevitz, and V. Vaikuntanathan, "Low-complexity cryptographic hash functions", in *8th Innovations in Theoretical Computer Science Conference, ITCS 2017, January 9-11, 2017, Berkeley, CA, USA*, 2017, 7:1–7:31.

[AIK06] B. Applebaum, Y. Ishai, and E. Kushilevitz, "Cryptography in NC^0", *SIAM J. Comput.*, vol. 36, no. 4, pp. 845–888, 2006.

[AIK08] ——, "On pseudorandom generators with linear stretch in NC^0", *Computational Complexity*, vol. 17, no. 1, pp. 38–69, 2008.

[AS92] S. Arora and S. Safra, "Probabilistic checking of proofs; A new characterization of NP", in *33rd Annual Symposium on Foundations of Computer Science, Pittsburgh, Pennsylvania, USA, 24-27 October 1992*, 1992, pp. 2–13.

[ACH+05] G. Ateniese, J. Camenisch, S. Hohenberger, and B. de Medeiros, "Practical group signatures without random oracles", *IACR Cryptology ePrint Archive*, vol. 2005, p. 385, 2005.

[ACJ+00] G. Ateniese, J. Camenisch, M. Joye, and G. Tsudik, "A practical and provably secure coalition-resistant group signature scheme", in *Advances in Cryptology - CRYPTO 2000, 20th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 2000, Proceedings*, 2000, pp. 255–270.

[Bab85] L. Babai, "Trading group theory for randomness", in *Proceedings of the 17th Annual ACM Symposium on Theory of Computing, May 6-8, 1985, Providence, Rhode Island, USA*, 1985, pp. 421–429.

[BHS18] M. Backes, L. Hanzlik, and J. Schneider, "Membership privacy for fully dynamic group signatures", *IACR Cryptology ePrint Archive*, vol. 2018, p. 641, 2018.

[BP97] N. Bari and B. Pfitzmann, "Collision-free accumulators and fail-stop signature schemes without trees", in *Advances in Cryptology - EUROCRYPT '97, International Conference on the Theory and Application of Cryptographic Techniques, Konstanz, Germany, May 11-15, 1997, Proceeding*, 1997, pp. 480–494.

[BBC+18] C. Baum, J. Bootle, A. Cerulli, R. del Pino, J. Groth, and V. Lyubashevsky, "Sub-linear lattice-based zero-knowledge arguments for arithmetic circuits", in *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part II*, 2018, pp. 669–699.

[Bay14] S. Bayer, "Practical zero-knowledge protocols based on the discrete logarithm assumption", PhD thesis, University College London, 2014.

[BG92] M. Bellare and O. Goldreich, "On defining proofs of knowledge", in *Advances in Cryptology - CRYPTO '92, 12th Annual International Cryptology Conference, Santa Barbara, California, USA, August 16-20, 1992, Proceedings*, 1992, pp. 390–420.

[BJY97] M. Bellare, M. Jakobsson, and M. Yung, "Round-optimal zero-knowledge arguments based on any one-way function", in *Advances in Cryptology - EUROCRYPT '97, International Conference on the Theory and Application of Cryptographic Techniques, Konstanz, Germany, May 11-15, 1997, Proceeding*, 1997, pp. 280–305.

[BMW03] M. Bellare, D. Micciancio, and B. Warinschi, "Foundations of group signatures: Formal definitions, simplified requirements, and a construction based on general assumptions", in *Advances in Cryptology - EUROCRYPT 2003, International Conference on the Theory and Applications of Cryptographic Techniques, Warsaw, Poland, May 4-8, 2003, Proceedings*, 2003, pp. 614–629.

[BR93] M. Bellare and P. Rogaway, "Random oracles are practical: A paradigm for designing efficient protocols", in *CCS '93, Proceedings of the 1st ACM Conference on Computer and Communications Security, Fairfax, Virginia, USA, November 3-5, 1993.*, 1993, pp. 62–73.

[BSZ05] M. Bellare, H. Shi, and C. Zhang, "Foundations of group signatures: The case of dynamic groups", in *Topics in Cryptology - CT-RSA 2005, The Cryptographers' Track at the RSA Conference 2005, San Francisco, CA, USA, February 14-18, 2005, Proceedings*, 2005, pp. 136–153.

[BGG+88] M. Ben-Or, O. Goldreich, S. Goldwasser, J. Håstad, J. Kilian, S. Micali, and P. Rogaway, "Everything provable is provable in zero-knowledge", in *Advances in Cryptology - CRYPTO '88, 8th Annual International Cryptology Conference, Santa Barbara, California, USA, August 21-25, 1988, Proceedings*, 1988, pp. 37–56.

[BBH+18] E. Ben-Sasson, I. Bentov, Y. Horesh, and M. Riabzev, "Scalable, transparent, and post-quantum secure computational integrity", *IACR Cryptology ePrint Archive*, vol. 2018, p. 46, 2018.

[BCG+16a] E. Ben-Sasson, A. Chiesa, A. Gabizon, M. Riabzev, and N. Spooner, "Short interactive oracle proofs with constant query complexity, via composition and sumcheck", *Electronic Colloquium on Computational Complexity (ECCC)*, vol. 23, p. 46, 2016.

[BCG+16b] E. Ben-Sasson, A. Chiesa, A. Gabizon, and M. Virza, "Quasi-linear size zero knowledge from linear-algebraic PCPs", in *Theory of Cryptography - 13th International Conference, TCC 2016-A, Tel Aviv, Israel, January 10-13, 2016, Proceedings, Part II*, 2016, pp. 33–64.

[BCG+13a] E. Ben-Sasson, A. Chiesa, D. Genkin, E. Tromer, and M. Virza, "SNARKs for C: verifying program executions succinctly and in zero knowledge", in *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part II*, 2013, pp. 90–108.

[BCG+13b] E. Ben-Sasson, A. Chiesa, D. Genkin, E. Tromer, and M. Virza, *TinyRAM architecture specification, v0.991*, 2013. [Online]. Available: `http://www.scipr-lab.org/doc/TinyRAM-spec-0.991.pdf` (visited on 09/14/2018).

[BCR+18] E. Ben-Sasson, A. Chiesa, M. Riabzev, N. Spooner, M. Virza, and N. P. Ward, "Aurora: Transparent succinct arguments for R1CS", *IACR Cryptology ePrint Archive*, vol. 2018, p. 62, 2018. [Online]. Available: `http://eprint.iacr.org/2018/828`.

[BCS16] E. Ben-Sasson, A. Chiesa, and N. Spooner, "Interactive oracle proofs", in *Theory of Cryptography - 14th International Conference, TCC 2016-B, Beijing, China, October 31 - November 3, 2016, Proceedings, Part II*, 2016, pp. 31–60.

[BCT+14] E. Ben-Sasson, A. Chiesa, E. Tromer, and M. Virza, "Succinct non-interactive zero knowledge for a von Neumann architecture", in *Proceedings of the 23rd USENIX Security Symposium, San Diego, CA, USA, August 20-22, 2014.*, 2014, pp. 781–796.

[BCT+17] ——, "Scalable zero knowledge via cycles of elliptic curves", *Algorithmica*, vol. 79, no. 4, pp. 1102–1160, 2017.

[BRS17] E. Ben-Sasson, N. Ron-Zewi, and M. Sudan, "Sparse affine-invariant linear codes are locally testable", *Computational Complexity*, vol. 26, no. 1, pp. 37–77, 2017. DOI: 10.1007/s00037-015-0115-6. [Online]. Available: https://doi.org/10.1007/s00037-015-0115-6.

[BdM93] J. C. Benaloh and M. de Mare, "One-way accumulators: A decentralized alternative to digital sinatures (extended abstract)", in *Advances in Cryptology - EUROCRYPT '93, Workshop on the Theory and Application of of Cryptographic Techniques, Lofthus, Norway, May 23-27, 1993, Proceedings*, 1993, pp. 274–285.

[BKM09] A. Bender, J. Katz, and R. Morselli, "Ring signatures: Stronger definitions, and constructions without random oracles", *J. Cryptology*, vol. 22, no. 1, pp. 114–138, 2009.

[Ben65] V. E. Beneš, *Mathematical theory of connecting networks and telephone traffic*. Academic press, 1965, vol. 17.

[BCN+10] P. Bichsel, J. Camenisch, G. Neven, N. P. Smart, and B. Warinschi, "Get shorty via group signatures without encryption", in *Security and Cryptography for Networks, 7th International Conference, SCN 2010, Amalfi, Italy, September 13-15, 2010. Proceedings*, 2010, pp. 381–398.

[BCC+12] N. Bitansky, R. Canetti, A. Chiesa, and E. Tromer, "From extractable collision resistance to succinct non-interactive arguments of knowledge,

and back again", in *Innovations in Theoretical Computer Science 2012, Cambridge, MA, USA, January 8-10, 2012*, 2012, pp. 326–349.

[BCC+13] ——, "Recursive composition and bootstrapping for SNARKs and proof-carrying data", in *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, 2013, pp. 111–120.

[BCI+13] N. Bitansky, A. Chiesa, Y. Ishai, R. Ostrovsky, and O. Paneth, "Succinct non-interactive arguments via linear interactive proofs", in *Theory of Cryptography - 10th Theory of Cryptography Conference, TCC 2013, Tokyo, Japan, March 3-6, 2013. Proceedings*, 2013, pp. 315–333.

[BFM88] M. Blum, P. Feldman, and S. Micali, "Non-interactive zero-knowledge and its applications (extended abstract)", in *Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA*, 1988, pp. 103–112.

[BBS04] D. Boneh, X. Boyen, and H. Shacham, "Short group signatures", in *Advances in Cryptology - CRYPTO 2004, 24th Annual International Cryptology-Conference, Santa Barbara, California, USA, August 15-19, 2004, Proceedings*, 2004, pp. 41–55.

[BEF19] D. Boneh, S. Eskandarian, and B. Fisch, "Post-quantum EPID group signatures from symmetric primitives", in *Topics in Cryptology - CT-RSA 2019 - The Cryptographers' Track at the RSA Conference 2019, San Francisco, CA, USA, March 4-8, 2019, Proceedings*, Springer, 2019.

[BS04] D. Boneh and H. Shacham, "Group signatures with verifier-local revocation", in *Proceedings of the 11th ACM Conference on Computer and Communications Security, CCS 2004, Washington, DC, USA, October 25-29, 2004*, 2004, pp. 168–177.

[BCC+16a] J. Bootle, A. Cerulli, P. Chaidos, E. Ghadafi, and J. Groth, "Foundations of fully dynamic group signatures", in *Applied Cryptography and Network Security - 14th International Conference, ACNS 2016, Guildford, UK, June 19-22, 2016. Proceedings*, 2016, pp. 117–136.

[BCC+15] J. Bootle, A. Cerulli, P. Chaidos, E. Ghadafi, J. Groth, and C. Petit, "Short accountable ring signatures based on DDH", in *Computer Security - ESORICS 2015 - 20th European Symposium on Research in Computer Security, Vienna, Austria, September 21-25, 2015, Proceedings, Part I*, 2015, pp. 243–265.

[BCC+16b] J. Bootle, A. Cerulli, P. Chaidos, J. Groth, and C. Petit, "Efficient zero-knowledge arguments for arithmetic circuits in the discrete log setting", in *Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II*, 2016, pp. 327–357.

[BCG+17] J. Bootle, A. Cerulli, E. Ghadafi, J. Groth, M. Hajiabadi, and S. K. Jakobsen, "Linear-time zero-knowledge proofs for arithmetic circuit satisfiability", in *Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part III*, 2017, pp. 336–365.

[BCG+18] J. Bootle, A. Cerulli, J. Groth, S. K. Jakobsen, and M. Maller, "Arya: Nearly linear-time zero-knowledge proofs for correct program execution", in *Advances in Cryptology - ASIACRYPT 2018 - 24th International Conference on the Theory and Applications of Cryptology and Information Security*, 2018.

[BG18] J. Bootle and J. Groth, "Efficient batch zero-knowledge arguments for low degree polynomials", in *Public-Key Cryptography - PKC 2018 - 21st IACR International Conference on Practice and Theory of Public-Key Cryptography, Rio de Janeiro, Brazil, March 25-29, 2018, Proceedings, Part II*, 2018, pp. 561–588.

[BCN18] C. Boschini, J. Camenisch, and G. Neven, "Floppy-sized group signatures from lattices", in *Applied Cryptography and Network Security - 16th International Conference, ACNS 2018, Leuven, Belgium, July 2-4, 2018, Proceedings*, 2018, pp. 163–182.

[BW06] X. Boyen and B. Waters, "Compact group signatures without random oracles", in *Advances in Cryptology - EUROCRYPT 2006, 25th Annual International Conference on the Theory and Applications of Cryptographic Techniques, St. Petersburg, Russia, May 28 - June 1, 2006, Proceedings*, 2006, pp. 427–444.

[BW07] ——, "Full-domain subgroup hiding and constant-size group signatures", in *Public Key Cryptography - PKC 2007, 10th International Conference on Practice and Theory in Public-Key Cryptography, Beijing, China, April 16-20, 2007, Proceedings*, 2007, pp. 1–15.

[BCC88] G. Brassard, D. Chaum, and C. Crépeau, "Minimum disclosure proofs of knowledge", *J. Comput. Syst. Sci.*, vol. 37, no. 2, pp. 156–189, 1988.

[BCY91] G. Brassard, C. Crépeau, and M. Yung, "Constant-round perfect zero-knowledge computationally convincing protocols", *Theor. Comput. Sci.*, vol. 84, no. 1, pp. 23–52, 1991.

[BFR+13] B. Braun, A. J. Feldman, Z. Ren, S. T. V. Setty, A. J. Blumberg, and M. Walfish, "Verifying computations with state", in *ACM SIGOPS 24th Symposium on Operating Systems Principles, SOSP '13, Farmington, PA, USA, November 3-6, 2013*, 2013, pp. 341–357.

[BS01] E. Bresson and J. Stern, "Efficient revocation in group signatures", in *Public Key Cryptography, 4th International Workshop on Practice and Theory in Public Key Cryptography, PKC 2001, Cheju Island, Korea, February 13-15, 2001, Proceedings*, 2001, pp. 190–206.

[Bri04] E. Brickell, "An efficient protocol for anonymously providing assurance of the container of a private key", *Submitted to the Trusted Computing Group*, 2004.

[BCC04] E. F. Brickell, J. Camenisch, and L. Chen, "Direct anonymous attestation", in *Proceedings of the 11th ACM Conference on Computer and Communications Security, CCS 2004, Washington, DC, USA, October 25-29, 2004*, 2004, pp. 132–145.

[CG04] J. Camenisch and J. Groth, "Group signatures: Better efficiency and new theoretical aspects", in *Security in Communication Networks, 4th International Conference, SCN 2004, Amalfi, Italy, September 8-10, 2004, Revised Selected Papers*, 2004, pp. 120–133.

[CL02] J. Camenisch and A. Lysyanskaya, "Dynamic accumulators and application to efficient revocation of anonymous credentials", in *Advances in Cryptology - CRYPTO 2002, 22nd Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 2002, Proceedings*, 2002, pp. 61–76.

[CL04] ——, "Signature schemes and anonymous credentials from bilinear maps", in *Advances in Cryptology - CRYPTO 2004, 24th Annual International CryptologyConference, Santa Barbara, California, USA, August 15-19, 2004, Proceedings*, 2004, pp. 56–72.

[CM98] J. Camenisch and M. Michels, "A group signature scheme with improved efficiency", in *Advances in Cryptology - ASIACRYPT '98, International Conference on the Theory and Applications of Cryptology and Information Security, Beijing, China, October 18-22, 1998, Proceedings*, 1998, pp. 160–174.

[CNR12] J. Camenisch, G. Neven, and M. Rückert, "Fully anonymous attribute tokens from lattices", in *Security and Cryptography for Networks - 8th International Conference, SCN 2012, Amalfi, Italy, September 5-7, 2012. Proceedings*, 2012, pp. 57–75.

[CDH+00] R. Canetti, Y. Dodis, S. Halevi, E. Kushilevitz, and A. Sahai, "Exposure-resilient functions and all-or-nothing transforms", in *Advances in Cryptology - EUROCRYPT 2000, International Conference on the Theory and Application of Cryptographic Techniques, Bruges, Belgium, May 14-18, 2000, Proceeding*, 2000, pp. 453–469.

[CGH04] R. Canetti, O. Goldreich, and S. Halevi, "The random oracle methodology, revisited", *J. ACM*, vol. 51, no. 4, pp. 557–594, 2004.

[CRV+02] M. R. Capalbo, O. Reingold, S. P. Vadhan, and A. Wigderson, "Randomness conductors and constant-degree lossless expanders", in *Proceedings*

of the 17th Annual IEEE Conference on Computational Complexity, Montréal, Québec, Canada, May 21-24, 2002, 2002, p. 15.

[CW77] L. Carter and M. N. Wegman, "Universal classes of hash functions (extended abstract)", in *Proceedings of the 9th Annual ACM Symposium on Theory of Computing, May 4-6, 1977, Boulder, Colorado, USA*, 1977, pp. 106–112.

[CDD+16] I. Cascudo, I. Damgård, B. David, N. Döttling, and J. B. Nielsen, "Rate-1, linear time and additively homomorphic UC commitments", in *Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part III*, 2016, pp. 179–207.

[CDG+17] M. Chase, D. Derler, S. Goldfeder, C. Orlandi, S. Ramacher, C. Rechberger, D. Slamanig, and G. Zaverucha, "Post-quantum zero-knowledge and signatures from symmetric-key primitives", in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, 2017, pp. 1825–1842.

[CGM16] M. Chase, C. Ganesh, and P. Mohassel, "Efficient zero-knowledge proof of algebraic and non-algebraic statements with applications to privacy preserving credentials", in *Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part III*, 2016, pp. 499–530.

[CvH91] D. Chaum and E. van Heyst, "Group signatures", in *Advances in Cryptology - EUROCRYPT '91, Workshop on the Theory and Application of of Cryptographic Techniques, Brighton, UK, April 8-11, 1991, Proceedings*, 1991, pp. 257–265.

[CCG+07] H. Chen, R. Cramer, S. Goldwasser, R. de Haan, and V. Vaikuntanathan, "Secure computation from random error correcting codes", in *Advances in Cryptology - EUROCRYPT 2007, 26th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Barcelona, Spain, May 20-24, 2007, Proceedings*, 2007, pp. 291–310.

[CTV15] A. Chiesa, E. Tromer, and M. Virza, "Cluster computing in zero knowledge", in *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part II*, 2015, pp. 371–403.

[CPS+16] M. Ciampi, G. Persiano, L. Siniscalchi, and I. Visconti, "A transform for NIZK almost as efficient and general as the Fiat-Shamir transform without programmable random oracles", in *Theory of Cryptography - 13th International Conference, TCC 2016-A, Tel Aviv, Israel, January 10-13, 2016, Proceedings, Part II*, 2016, pp. 83–111.

[CMT12] G. Cormode, M. Mitzenmacher, and J. Thaler, "Practical verified computation with streaming interactive proofs", in *Innovations in Theoretical Computer Science 2012, Cambridge, MA, USA, January 8-10, 2012*, 2012, pp. 90–112.

[CD98] R. Cramer and I. Damgård, "Zero-knowledge proofs for finite field arithmetic; or: Can zero-knowledge be for free?", in *Advances in Cryptology - CRYPTO '98, 18th Annual International Cryptology Conference, Santa Barbara, California, USA, August 23-27, 1998, Proceedings*, 1998, pp. 424–441.

[CDD+15] R. Cramer, I. B. Damgård, N. Döttling, S. Fehr, and G. Spini, "Linear secret sharing schemes from error correcting codes and universal hash functions", in *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part II*, 2015, pp. 313–336.

[CDP12] R. Cramer, I. Damgård, and V. Pastro, "On the amortized complexity of zero knowledge protocols for multiplicative relations", in *Information Theoretic Security - 6th International Conference, ICITS 2012, Montreal, QC, Canada, August 15-17, 2012. Proceedings*, 2012, pp. 62–79.

[CDS94] R. Cramer, I. Damgård, and B. Schoenmakers, "Proofs of partial knowledge and simplified design of witness hiding protocols", in *Advances*

in Cryptology - CRYPTO '94, 14th Annual International Cryptology Conference, Santa Barbara, California, USA, August 21-25, 1994, Proceedings, 1994, pp. 174–187.

[Dam00] I. Damgård, "Efficient concurrent zero-knowledge in the auxiliary string model", in *Advances in Cryptology - EUROCRYPT 2000, International Conference on the Theory and Application of Cryptographic Techniques, Bruges, Belgium, May 14-18, 2000, Proceeding*, 2000, pp. 418–430.

[DF02] I. Damgård and E. Fujisaki, "A statistically-hiding integer commitment scheme based on groups with hidden order", in *Advances in Cryptology - ASIACRYPT 2002, 8th International Conference on the Theory and Application of Cryptology and Information Security, Queenstown, New Zealand, December 1-5, 2002, Proceedings*, 2002, pp. 125–142.

[DI06] I. Damgård and Y. Ishai, "Scalable secure multiparty computation", in *Advances in Cryptology - CRYPTO 2006, 26th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 2006, Proceedings*, 2006, pp. 501–520.

[DIK10] I. Damgård, Y. Ishai, and M. Krøigaard, "Perfectly secure multiparty computation and the computational overhead of cryptography", in *Advances in Cryptology - EUROCRYPT 2010, 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Monaco / French Riviera, May 30 - June 3, 2010. Proceedings*, 2010, pp. 445–465.

[DLO+18] I. Damgård, J. Luo, S. Oechsner, P. Scholl, and M. Simkin, "Compact zero-knowledge proofs of small hamming weight", in *Public-Key Cryptography - PKC 2018 - 21st IACR International Conference on Practice and Theory of Public-Key Cryptography, Rio de Janeiro, Brazil, March 25-29, 2018, Proceedings, Part II*, 2018, pp. 530–560.

[DZ13] I. Damgård and S. Zakarias, "Constant-overhead secure computation of boolean circuits using preprocessing", in *Theory of Cryptography - 10th Theory of Cryptography Conference, TCC 2013, Tokyo, Japan, March 3-6, 2013. Proceedings*, 2013, pp. 621–641.

[DP06] C. Delerablée and D. Pointcheval, "Dynamic fully anonymous short group signatures", in *Progressin Cryptology - VIETCRYPT 2006, First International Conferenceon Cryptology in Vietnam, Hanoi, Vietnam, September 25-28, 2006, Revised Selected Papers*, 2006, pp. 193–210.

[DS18] D. Derler and D. Slamanig, "Highly-efficient fully-anonymous dynamic group signatures", in *Proceedings of the 2018 on Asia Conference on Computer and Communications Security, AsiaCCS 2018, Incheon, Republic of Korea, June 04-08, 2018*, 2018, pp. 551–565.

[DKN+04] Y. Dodis, A. Kiayias, A. Nicolosi, and V. Shoup, "Anonymous identification in ad hoc groups", in *Advances in Cryptology - EUROCRYPT 2004, International Conference on the Theory and Applications of Cryptographic Techniques, Interlaken, Switzerland, May 2-6, 2004, Proceedings*, 2004, pp. 609–626.

[DI14] E. Druk and Y. Ishai, "Linear-time encodable codes meeting the Gilbert-Varshamov bound and their cryptographic applications", in *Innovations in Theoretical Computer Science, ITCS'14, Princeton, NJ, USA, January 12-14, 2014*, 2014, pp. 169–182.

[EKS18] A. El Kaafarani, S. Katsumata, and R. Solomon, "Anonymous reputation systems achieving full dynamicity from lattices", in *Financial Cryptography and Data Security - 22st International Conference, FC 2018*, Mar. 2018.

[FFS88] U. Feige, A. Fiat, and A. Shamir, "Zero-knowledge proofs of identity", *J. Cryptology*, vol. 1, no. 2, pp. 77–94, 1988.

[FS89] U. Feige and A. Shamir, "Zero knowledge proofs of knowledge in two rounds", in *Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings*, 1989, pp. 526–544.

[FS86] A. Fiat and A. Shamir, "How to prove yourself: Practical solutions to identification and signature problems", in *Advances in Cryptology - CRYPTO '86, Santa Barbara, California, USA, 1986, Proceedings*, 1986, pp. 186–194.

[Fis05] M. Fischlin, "Communication-efficient non-interactive proofs of knowledge with online extractors", in *Advances in Cryptology - CRYPTO 2005: 25th Annual International Cryptology Conference, Santa Barbara, California, USA, August 14-18, 2005, Proceedings*, 2005, pp. 152–168.

[For89] L. Fortnow, "The complexity of perfect zero-knowledge", *Advances in Computing Research*, vol. 5, pp. 327–343, 1989.

[FNO15] T. K. Frederiksen, J. B. Nielsen, and C. Orlandi, "Privacy-free garbled circuits with applications to efficient zero-knowledge", in *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part II*, 2015, pp. 191–219.

[FI06] J. Furukawa and H. Imai, "An efficient group signature scheme from bilinear maps", *IEICE Transactions*, vol. 89-A, no. 5, pp. 1328–1338, 2006.

[FY04] J. Furukawa and S. Yonezawa, "Group signatures with separate and distributed authorities", in *Security in Communication Networks, 4th International Conference, SCN 2004, Amalfi, Italy, September 8-10, 2004, Revised Selected Papers*, 2004, pp. 77–90.

[GGP10] R. Gennaro, C. Gentry, and B. Parno, "Non-interactive verifiable computing: Outsourcing computation to untrusted workers", in *Advances in Cryptology - CRYPTO 2010, 30th Annual Cryptology Conference, Santa Barbara, CA, USA, August 15-19, 2010. Proceedings*, 2010, pp. 465–482.

[GGP+13] R. Gennaro, C. Gentry, B. Parno, and M. Raykova, "Quadratic span programs and succinct NIZKs without PCPs", in *Advances in Cryptology - EUROCRYPT 2013, 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings*, 2013, pp. 626–645.

[Gen09] C. Gentry, "A fully homomorphic encryption scheme", PhD thesis, Stanford University, 2009.

[GGI+15] C. Gentry, J. Groth, Y. Ishai, C. Peikert, A. Sahai, and A. D. Smith, "Using fully homomorphic hybrid encryption to minimize non-interative zero-knowledge proofs", *J. Cryptology*, vol. 28, no. 4, pp. 820–843, 2015.

[GMO16] I. Giacomelli, J. Madsen, and C. Orlandi, "ZKBoo: Faster zero-knowledge for boolean circuits", in *25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016.*, 2016, pp. 1069–1083.

[GH98] O. Goldreich and J. Håstad, "On the complexity of interactive proofs with bounded communication", *Inf. Process. Lett.*, vol. 67, no. 4, pp. 205–214, 1998.

[GK96a] O. Goldreich and A. Kahan, "How to construct constant-round zero-knowledge proof systems for NP", *J. Cryptology*, vol. 9, no. 3, pp. 167–190, 1996.

[GK96b] O. Goldreich and H. Krawczyk, "On the composition of zero-knowledge proof systems", *SIAM J. Comput.*, vol. 25, no. 1, pp. 169–192, 1996.

[GMW91] O. Goldreich, S. Micali, and A. Wigderson, "Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems", *J. ACM*, vol. 38, no. 3, pp. 691–729, 1991.

[GSV98] O. Goldreich, A. Sahai, and S. P. Vadhan, "Honest-verifier statistical zero-knowledge equals general statistical zero-knowledge", in *Proceedings of the Thirtieth Annual ACM Symposium on the Theory of Computing, Dallas, Texas, USA, May 23-26, 1998*, 1998, pp. 399–408.

[GVW02] O. Goldreich, S. P. Vadhan, and A. Wigderson, "On interactive proofs with a laconic prover", *Computational Complexity*, vol. 11, no. 1-2, pp. 1–53, 2002.

[GK03] S. Goldwasser and Y. T. Kalai, "On the (in)security of the Fiat-Shamir paradigm", in *44th Symposium on Foundations of Computer Science (FOCS 2003), 11-14 October 2003, Cambridge, MA, USA, Proceedings*, 2003, pp. 102–113.

[GKR15] S. Goldwasser, Y. T. Kalai, and G. N. Rothblum, "Delegating computation: Interactive proofs for muggles", *J. ACM*, vol. 62, no. 4, 27:1–27:64, 2015.

[GMR89] S. Goldwasser, S. Micali, and C. Rackoff, "The knowledge complexity of interactive proof systems", *SIAM J. Comput.*, vol. 18, no. 1, pp. 186–208, 1989.

[GS89] S. Goldwasser and M. Sipser, "Private coins versus public coins in interactive proof systems", *Advances in Computing Research*, vol. 5, pp. 73–90, 1989.

[GKV10] S. D. Gordon, J. Katz, and V. Vaikuntanathan, "A group signature scheme from lattice assumptions", in *Advances in Cryptology - ASIACRYPT 2010 - 16th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 5-9, 2010. Proceedings*, 2010, pp. 395–412.

[Gro04] J. Groth, "Honest verifier zero-knowledge arguments applied", PhD thesis, University of Aarhus, 2004.

[Gro06] ——, "Simulation-sound NIZK proofs for a practical language and constant size group signatures", in *Advances in Cryptology - ASIACRYPT 2006, 12th International Conference on the Theory and Application of Cryptology and Information Security, Shanghai, China, December 3-7, 2006, Proceedings*, 2006, pp. 444–459.

[Gro07] ——, "Fully anonymous group signatures without random oracles", in *Advances in Cryptology - ASIACRYPT 2007, 13th International Conference on the Theory and Application of Cryptology and Information Security, Kuching, Malaysia, December 2-6, 2007, Proceedings*, 2007, pp. 164–180.

[Gro09] ——, "Linear algebra with sub-linear zero-knowledge arguments", in *Advances in Cryptology - CRYPTO 2009, 29th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2009. Proceedings*, 2009, pp. 192–208.

[Gro10] ——, "Short pairing-based non-interactive zero-knowledge arguments", in *Advances in Cryptology - ASIACRYPT 2010 - 16th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 5-9, 2010. Proceedings*, 2010, pp. 321–340.

[Gro16] ——, "On the size of pairing-based non-interactive arguments", in *Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II*, 2016, pp. 305–326.

[GKM+18] J. Groth, M. Kohlweiss, M. Maller, S. Meiklejohn, and I. Miers, "Updatable and universal common reference strings with applications to zk-SNARKs", in *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part III*, 2018, pp. 698–728.

[GM17] J. Groth and M. Maller, "Snarky signatures: Minimal signatures of knowledge from simulation-extractable SNARKs", in *Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part II*, 2017, pp. 581–612.

[GOS12] J. Groth, R. Ostrovsky, and A. Sahai, "New techniques for non-interactive zero-knowledge", *J. ACM*, vol. 59, no. 3, 11:1–11:35, 2012.

[GI01] V. Guruswami and P. Indyk, "Expander-based constructions of efficiently decodable codes", in *42nd Annual Symposium on Foundations of Computer Science, FOCS 2001, 14-17 October 2001, Las Vegas, Nevada, USA*, 2001, pp. 658–667.

[GI02] ——, "Near-optimal linear-time codes for unique decoding and new list-decodable codes over smaller alphabets", in *Proceedings on 34th Annual ACM Symposium on Theory of Computing, May 19-21, 2002, Montréal, Québec, Canada*, 2002, pp. 812–821.

[GI03] ——, "Linear time encodable and list decodable codes", in *Proceedings of the 35th Annual ACM Symposium on Theory of Computing, June 9-11, 2003, San Diego, CA, USA*, 2003, pp. 126–135.

[GI05] ——, "Linear-time encodable/decodable codes with near-optimal rate", *IEEE Trans. Information Theory*, vol. 51, no. 10, pp. 3393–3400, 2005.

[HM96] S. Halevi and S. Micali, "Practical and provably-secure commitment schemes from collision-free hashing", in *Advances in Cryptology - CRYPTO '96, 16th Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 1996, Proceedings*, 1996, pp. 201–215.

[HMR15] Z. Hu, P. Mohassel, and M. Rosulek, "Efficient zero-knowledge proofs of non-algebraic statements with sublinear amortized cost", in *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part II*, 2015, pp. 150–169.

[IY87] R. Impagliazzo and M. Yung, "Direct minimum-knowledge computations", in *Advances in Cryptology - CRYPTO '87, A Conference on the Theory and Applications of Cryptographic Techniques, Santa Barbara, California, USA, August 16-20, 1987, Proceedings*, 1987, pp. 40–51.

[IKO07] Y. Ishai, E. Kushilevitz, and R. Ostrovsky, "Efficient arguments without short PCPs", in *22nd Annual IEEE Conference on Computational Complexity (CCC 2007), 13-16 June 2007, San Diego, California, USA*, 2007, pp. 278–291.

[IKO+08] Y. Ishai, E. Kushilevitz, R. Ostrovsky, and A. Sahai, "Cryptography with constant computational overhead", in *Proceedings of the 40th Annual ACM Symposium on Theory of Computing, Victoria, British Columbia, Canada, May 17-20, 2008*, 2008, pp. 433–442.

[IKO+09] ——, "Zero-knowledge proofs from secure multiparty computation", *SIAM J. Comput.*, vol. 39, no. 3, pp. 1121–1152, 2009.

[ISV+13] Y. Ishai, A. Sahai, M. Viderman, and M. Weiss, "Zero knowledge LTCs and their applications", in *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques - 16th International Workshop, APPROX 2013, and 17th International Workshop, RANDOM 2013, Berkeley, CA, USA, August 21-23, 2013. Proceedings*, 2013, pp. 607–622.

[JKO13] M. Jawurek, F. Kerschbaum, and C. Orlandi, "Zero-knowledge using garbled circuits: How to prove non-algebraic statements efficiently", in *2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13, Berlin, Germany, November 4-8, 2013*, 2013, pp. 955–966.

[KR08] Y. T. Kalai and R. Raz, "Interactive PCP", in *Automata, Languages and Programming, 35th International Colloquium, ICALP 2008, Reykjavik, Iceland, July 7-11, 2008, Proceedings, Part II - Track B: Logic, Semantics, and Theory of Programming & Track C: Security and Cryptography Foundations*, 2008, pp. 536–547.

[Kat12] J. Katz, "Which languages have 4-round zero-knowledge proofs?", *J. Cryptology*, vol. 25, no. 1, pp. 41–56, 2012.

[KTY04] A. Kiayias, Y. Tsiounis, and M. Yung, "Traceable signatures", in *Advances in Cryptology - EUROCRYPT 2004, International Conference on the Theory and Applications of Cryptographic Techniques, Interlaken, Switzerland, May 2-6, 2004, Proceedings*, 2004, pp. 571–589.

[KY05] A. Kiayias and M. Yung, "Group signatures with efficient concurrent join", in *Advances in Cryptology - EUROCRYPT 2005, 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005, Proceedings*, 2005, pp. 198–214.

[KY06] ——, "Secure scalable group signature with dynamic joins and separable authorities", *IJSN*, vol. 1, no. 1/2, pp. 24–45, 2006.

[Kil92] J. Kilian, "A note on efficient zero-knowledge proofs and arguments (extended abstract)", in *Proceedings of the 24th Annual ACM Symposium on Theory of Computing, May 4-6, 1992, Victoria, British Columbia, Canada*, 1992, pp. 723–732.

[KP17] S. Kumawat and S. Paul, "A new constant-size accountable ring signature scheme without random oracles", in *Information Security and Cryptology - 13th International Conference, Inscrypt 2017, Xi'an, China, November 3-5, 2017, Revised Selected Papers*, 2017, pp. 157–179.

[LLL+13] F. Laguillaumie, A. Langlois, B. Libert, and D. Stehlé, "Lattice-based group signatures with logarithmic signature size", in *Advances in Cryptology - ASIACRYPT 2013 - 19th International Conference on the Theory and Application of Cryptology and Information Security, Bengaluru, India, December 1-5, 2013, Proceedings, Part II*, 2013, pp. 41–61.

[LZC+16] R. W. F. Lai, T. Zhang, S. S. M. Chow, and D. Schröder, "Efficient sanitizable signatures without random oracles", in *Computer Security - ESORICS 2016 - 21st European Symposium on Research in Computer Security, Heraklion, Greece, September 26-30, 2016, Proceedings, Part I*, 2016, pp. 363–380.

[LLM+16] B. Libert, S. Ling, F. Mouhartem, K. Nguyen, and H. Wang, "Signature schemes with efficient protocols and dynamic group signatures from lattice assumptions", in *Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part II*, 2016, pp. 373–403.

[LLN+16] B. Libert, S. Ling, K. Nguyen, and H. Wang, "Zero-knowledge arguments for lattice-based accumulators: Logarithmic-size ring signatures and group signatures without trapdoors", in *Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II*, 2016, pp. 1–31.

[LMN16] B. Libert, F. Mouhartem, and K. Nguyen, "A lattice-based group signature scheme with message-dependent opening", in *Applied Cryptography and Network Security - 14th International Conference, ACNS 2016, Guildford, UK, June 19-22, 2016. Proceedings*, 2016, pp. 137–155.

[LPY12a] B. Libert, T. Peters, and M. Yung, "Group signatures with almost-for-free revocation", in *Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings*, 2012, pp. 571–589.

[LPY12b] ——, "Scalable group signatures with revocation", in *Advances in Cryptology - EUROCRYPT 2012 - 31st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cambridge, UK, April 15-19, 2012. Proceedings*, 2012, pp. 609–627.

[LV09] B. Libert and D. Vergnaud, "Group signatures with verifier-local revocation and backward unlinkability in the standard model", in *Cryptology and Network Security, 8th International Conference, CANS 2009, Kanazawa, Japan, December 12-14, 2009. Proceedings*, 2009, pp. 498–517.

[Lin03] Y. Lindell, "Parallel coin-tossing and constant-round secure two-party computation", *J. Cryptology*, vol. 16, no. 3, pp. 143–184, 2003.

[Lin15] ——, "An efficient transform from sigma protocols to NIZK with a CRS and non-programmable random oracle", in *Theory of Cryptography - 12th Theory of Cryptography Conference, TCC 2015, Warsaw, Poland, March 23-25, 2015, Proceedings, Part I*, 2015, pp. 93–109.

[LNR+18] S. Ling, K. Nguyen, A. Roux-Langlois, and H. Wang, "A lattice-based group signature scheme with verifier-local revocation", *Theor. Comput. Sci.*, vol. 730, pp. 1–20, 2018.

[LNW15] S. Ling, K. Nguyen, and H. Wang, "Group signatures from lattices: Simpler, tighter, shorter, ring-based", in *Public-Key Cryptography - PKC 2015 - 18th IACR International Conference on Practice and Theory in Public-Key Cryptography, Gaithersburg, MD, USA, March 30 - April 1, 2015, Proceedings*, 2015, pp. 427–449.

[LNW+18] S. Ling, K. Nguyen, H. Wang, and Y. Xu, "Lattice-based group signatures: Achieving full dynamicity (and deniability) with ease", *CoRR*, vol. abs/1801.08737, 2018.

[Mas95] J. L. Massey, "Some applications of coding theory in cryptography", in *Codes and Ciphers: Cryptography and Coding IV*, 1995, pp. 33–47.

[Mic94] S. Micali, "CS proofs (extended abstracts)", in *35th Annual Symposium on Foundations of Computer Science, Santa Fe, New Mexico, USA, 20-22 November 1994*, 1994, pp. 436–453.

[MP03] D. Micciancio and E. Petrank, "Simulatable commitments and efficient concurrent zero-knowledge", in *Advances in Cryptology - EUROCRYPT 2003, International Conference on the Theory and Applications of Cryptographic Techniques, Warsaw, Poland, May 4-8, 2003, Proceedings*, 2003, pp. 140–159.

[MRS17] P. Mohassel, M. Rosulek, and A. Scafuro, "Sublinear zero-knowledge arguments for RAM programs", *IACR Cryptology ePrint Archive*, vol. 2017, p. 129, 2017.

[NFH+10] T. Nakanishi, H. Fujii, Y. Hira, and N. Funabiki, "Revocable group signature schemes with constant costs for signing and verifying", *IEICE Transactions*, vol. 93-A, no. 1, pp. 50–62, 2010.

[NF07] T. Nakanishi and N. Funabiki, "Verifier-local revocation group signature schemes with backward unlinkability from bilinear maps", *IEICE Transactions*, vol. 90-A, no. 1, pp. 65–74, 2007.

[NNL01] D. Naor, M. Naor, and J. Lotspiech, "Revocation and tracing schemes for stateless receivers", in *Advances in Cryptology - CRYPTO 2001, 21st Annual International Cryptology Conference, Santa Barbara, California, USA, August 19-23, 2001, Proceedings*, 2001, pp. 41–62.

[Nao03] M. Naor, "On cryptographic assumptions and challenges", in *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings*, 2003, pp. 96–109.

[NOV+92] M. Naor, R. Ostrovsky, R. Venkatesan, and M. Yung, "Perfect zero-knowledge arguments for NP can be based on general complexity assumptions (extended abstract)", in *Advances in Cryptology - CRYPTO '92, 12th Annual International Cryptology Conference, Santa Barbara, California, USA, August 16-20, 1992, Proceedings*, 1992, pp. 196–214.

[Nef01] C. A. Neff, "A verifiable secret shuffle and its application to e-voting", in *CCS 2001, Proceedings of the 8th ACM Conference on Computer and Communications Security, Philadelphia, Pennsylvania, USA, November 6-8, 2001.*, 2001, pp. 116–125.

[Ngu05] L. Nguyen, "Accumulators from bilinear pairings and applications", in *Topics in Cryptology - CT-RSA 2005, The Cryptographers' Track at the RSA Conference 2005, San Francisco, CA, USA, February 14-18, 2005, Proceedings*, 2005, pp. 275–292.

[NS04] L. Nguyen and R. Safavi-Naini, "Efficient and provably secure trapdoor-free group signature schemes from bilinear pairings", in *Advances in Cryptology - ASIACRYPT 2004, 10th International Conference on the Theory and Application of Cryptology and Information Security, Jeju Island, Korea, December 5-9, 2004, Proceedings*, 2004, pp. 372–386.

[NOV06] M. Nguyen, S. J. Ong, and S. P. Vadhan, "Statistical zero-knowledge arguments for NP from any one-way function", *Electronic Colloquium on Computational Complexity (ECCC)*, vol. 13, no. 075, 2006.

[NZZ15] P. Q. Nguyen, J. Zhang, and Z. Zhang, "Simpler efficient group signatures from lattices", in *Public-Key Cryptography - PKC 2015 - 18th IACR International Conference on Practice and Theory in Public-Key Cryptography, Gaithersburg, MD, USA, March 30 - April 1, 2015, Proceedings*, 2015, pp. 401–426.

[PHG+16] B. Parno, J. Howell, C. Gentry, and M. Raykova, "Pinocchio: Nearly practical verifiable computation", *Commun. ACM*, vol. 59, no. 2, pp. 103–112, 2016.

[Ped91] T. P. Pedersen, "Non-interactive and information-theoretic secure verifiable secret sharing", in *Advances in Cryptology - CRYPTO '91, 11th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1991, Proceedings*, 1991, pp. 129–140.

[RST01] R. L. Rivest, A. Shamir, and Y. Tauman, "How to leak a secret", in *Advances in Cryptology - ASIACRYPT 2001, 7th International Conference on the Theory and Application of Cryptology and Information Security, Gold Coast, Australia, December 9-13, 2001, Proceedings*, 2001, pp. 552–565.

[SV00] A. Sahai and S. P. Vadhan, "A complete problem for statistical zero knowledge", *Electronic Colloquium on Computational Complexity (ECCC)*, vol. 7, no. 84, 2000.

[SSE+12] Y. Sakai, J. C. N. Schuldt, K. Emura, G. Hanaoka, and K. Ohta, "On the security of dynamic group signatures: Preventing signature hijacking", in *Public Key Cryptography - PKC 2012 - 15th International Conference on Practice and Theory in Public Key Cryptography, Darmstadt, Germany, May 21-23, 2012. Proceedings*, 2012, pp. 715–732.

[Sch91] C. Schnorr, "Efficient signature generation by smart cards", *J. Cryptology*, vol. 4, no. 3, pp. 161–174, 1991.

[Sch80] J. T. Schwartz, "Fast probabilistic algorithms for verification of polynomial identities", *J. ACM*, vol. 27, no. 4, pp. 701–717, 1980.

[Sha92] A. Shamir, "IP = PSPACE", *J. ACM*, vol. 39, no. 4, pp. 869–877, 1992.

[Son01] D. X. Song, "Practical forward secure group signature schemes", in *CCS 2001, Proceedings of the 8th ACM Conference on Computer and Communications Security, Philadelphia, Pennsylvania, USA, November 6-8, 2001.*, 2001, pp. 225–234.

[Spi96] D. A. Spielman, "Linear-time encodable and decodable error-correcting codes", *IEEE Trans. Information Theory*, vol. 42, no. 6, pp. 1723–1731, 1996.

[Tha13] J. Thaler, "Time-optimal interactive proofs for circuit evaluation", in *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part II*, 2013, pp. 71–89.

[TW87] M. Tompa and H. Woll, "Random self-reducibility and zero knowledge interactive proofs of possession of information", in *28th Annual Symposium on Foundations of Computer Science, Los Angeles, California, USA, 27-29 October 1987*, 1987, pp. 472–482.

[TX03] G. Tsudik and S. Xu, "Accumulating composites and improved group signing", in *Advances in Cryptology - ASIACRYPT 2003, 9th International Conference on the Theory and Application of Cryptology and Information Security, Taipei, Taiwan, November 30 - December 4, 2003, Proceedings*, 2003, pp. 269–286.

[Unr12] D. Unruh, "Quantum proofs of knowledge", in *Advances in Cryptology - EUROCRYPT 2012 - 31st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cambridge, UK, April 15-19, 2012. Proceedings*, 2012, pp. 135–152.

[VSB+13] V. Vu, S. T. V. Setty, A. J. Blumberg, and M. Walfish, "A hybrid architecture for interactive verifiable computation", in *2013 IEEE Symposium on Security and Privacy, SP 2013, Berkeley, CA, USA, May 19-22, 2013*, 2013, pp. 223–237.

[WHG+16] R. S. Wahby, M. Howald, S. Garg, A. Shelat, and M. Walfish, "Verifiable ASICs", in *IEEE Symposium on Security and Privacy, SP 2016, San Jose, CA, USA, May 22-26, 2016*, 2016, pp. 759–778.

[WJB+17] R. S. Wahby, Y. Ji, A. J. Blumberg, A. Shelat, J. Thaler, M. Walfish, and T. Wies, "Full accounting for verifiable outsourcing", in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, 2017, pp. 2071–2086.

[WSR+15] R. S. Wahby, S. T. V. Setty, Z. Ren, A. J. Blumberg, and M. Walfish, "Efficient RAM and control flow in verifiable outsourced computation", in *22nd Annual Network and Distributed System Security Symposium, NDSS 2015, San Diego, California, USA, February 8-11, 2015*, 2015.

[WTS+18] R. S. Wahby, I. Tzialla, A. Shelat, J. Thaler, and M. Walfish, "Doubly-efficient zkSNARKs without trusted setup", in *2018 IEEE Symposium on Security and Privacy, SP 2018, Proceedings, 21-23 May 2018, San Francisco, California, USA*, 2018, pp. 926–943.

[XY04] S. Xu and M. Yung, "Accountable ring signatures: A smart card approach", in *Smart Card Research and Advanced Applications VI, IFIP 18th World Computer Congress, TC8/WG8.8 & TC11/WG11.2 Sixth International Conference on Smart Card Research and Advanced Applications (CARDIS), 22-27 August 2004, Toulouse, France*, 2004, pp. 271–286.

[ZGK+17] Y. Zhang, D. Genkin, J. Katz, D. Papadopoulos, and C. Papamanthou, "A zero-knowledge version of vSQL", *IACR Cryptology ePrint Archive*, vol. 2017, p. 1146, 2017.

[ZGK+18] ——, "vRAM: Faster verifiable RAM with program-independent preprocessing", in *2018 IEEE Symposium on Security and Privacy, SP 2018, Proceedings, 21-23 May 2018, San Francisco, California, USA*, 2018, pp. 908–925.

[Zip79] R. Zippel, "Probabilistic algorithms for sparse polynomials", in *Symbolic and Algebraic Computation, EUROSAM '79, An International Symposiumon Symbolic and Algebraic Computation, Marseille, France, June 1979, Proceedings*, 1979, pp. 216–226.