Vrije Universiteit Amsterdam

Universiteit van Amsterdam

Master Thesis

Cryptographic Compilers for Linear Probabilistically Checkable Proofs

Author: Mark Bebawy

VU ID: 2620527

UvA ID: 11667230

1st supervisor: Dr. Nicolas Resch

2nd reader: Dr. Steven de Rooij

A thesis submitted in fulfillment of the requirements for the joint UvA-VU Master of Science degree in Computer Science

June 28, 2023

"A proof is whatever convinces me."

Shimon Even, 1978

Abstract

Probabilistically checkable proofs (PCPs) are proof systems in which the verifier has restricted access to the prover's proof. Their constructions are often highly technical. By allowing for linear queries in the PCP model, resulting in *linear probabilistically checkable proofs (LPCPs)*, the PCP constructions can significantly be simplified. For LPCPs to be useful, cryptographic compilers are needed that remove the idealized assumption of linear oracles by transforming an LPCP into an interactive cryptographic protocol.

Currently, the best cryptographic compiler for LPCPs is based on *linear-only encryption schemes*, which is heavy cryptographic machinery of which the existence, although believable, does not rely on standard cryptographic assumptions. The main goal of this thesis is to develop new cryptographic compilers for LPCPs that rely on standard cryptographic assumptions. We developed two such compilers that rely on the discrete logarithm assumption. Our first compiler uses a *compressed Σ-protocol* to realize the linear queries in an LPCP. When compiling *the Hadamard LPCP* for NP-relations with this compiler, we get a cryptographic proof with logarithmic communication complexity, and with the same asymptotic soundness error as the best current compiler. Our second compiler is mostly useful for LPCPs that can cleverly be transformed into *quadratic PCPs* (which are LPCPs with quadratic queries instead of linear queries) that have shorter proofs and fewer queries. We apply such a transformation to the Hadamard LPCP, resulting in the *Hadamard QPCP*, which combined with our second compiler results in a cryptographic proof with roughly one third of the communication costs of our compiled Hadamard LPCP.

In summary, our new cryptographic compilers for LPCPs allow us to reap the fruits of LPCPs, while still relying on standard cryptographic assumptions. Furthermore, this thesis highlights the value of separating the information-theoretic proof system from its cryptographic compiler to study these components in a modular fashion.

Acknowledgements

"Now to Him who is able to do exceedingly abundantly above all that we ask or think, according to the power that works in us, to Him be glory in the church by Christ Jesus to all generations, forever and ever. Amen."

– Ephesians 3:20-21

Words cannot express my gratitude to God for the immeasurable blessings that He has bestowed upon me throughout the process of writing this thesis, and throughout my entire life.

I would like to express my deepest gratitude to Dr. Nicolas Resch, who has been an outstanding thesis supervisor. He provided me with the perfect balance between freedom in conducting this research, and in guidance throughout the wide range of complex topics in this thesis. This balance allowed me to fully enjoy this research period. His enthusiasm has inspired me, and I want to thank him for his patience, kindness, feedback, our regular meetings, and for always being approachable and extremely helpful in guiding me and answering all my questions.

I am also extremely grateful to Dr. Steven de Rooij, who agreed to be the second reader of this quite lengthy thesis. I want to thank him for attending my thesis defense, and for his flexibility with the practical aspects of the examination process. Additionally, I am grateful for his supervision about one year ago during my internship at Axini. His guidance and feedback at that time have greatly contributed to the overall development of my writing and research skills.

I am also thankful for my colleagues at Axini and at Deloitte for providing me with the needed freedom and flexibility to complete this thesis, and for their support and words of encouragement.

Last but not least, I want to thank my family and friends for their unconditional love and support. Without them, the completion of this thesis would not have been possible.

Contents

List of Figures

List of Tables

1

Introduction

The concept of a proof has evolved over time. Traditionally, proofs were static objects written by a prover that could be verified by a verifier. However, with the emergence of randomized and interactive computations, and with the emergence of cryptography (where one often wants to prove that a computation has been done correctly without revealing secrets involved in the computation), the concept of a proof has significantly broadened [25]. Many *proof systems* were developed, and the goal of proof systems shifted from revealing as much knowledge about a statement as possible, to minimizing the amount of communication needed to prove a statement. In fact, the goal is often to not reveal any information beyond the fact that the statement at hand is true; such a proof system is called a *zero-knowledge proof*. Zero-knowledge proofs have numerous applications, such as privacy-preserving authentication schemes, where users prove their identity without revealing any information [21], and secure transactions in blockchains: for example, a user may wish to prove they have sufficient funds for a purchase, without revealing their account balance [15, 36].

Essentially, a proof system consists of a prover convincing a verifier of some statement. These systems can be interactive or non-interactive, and additional constraints can be imposed on the prover and the verifier. For example, we can require the verifier to only have bounded (oracle) proof access (i.e., the verifier does not read the proof in full, but only queries a few proof symbols via an oracle). Such a proof system is called a *probabilistically checkable proof (PCP)*. We can even consider a proof system where the prover publishes an oracle of their proof, and the verifier is allowed to query the oracle for *inner products* between provided *query strings* and the proof string[1]. A proof system where such a *linear-*

[1] In this information-theoretic setting, a proof is abstractly defined as a vector $\boldsymbol{\pi} = (\boldsymbol{\pi}_1, \ldots, \boldsymbol{\pi}_s) \in \mathbb{F}^s$ for a finite field $\mathbb{F}$ and some proof string length s. Therefore, the verifier can create a query string

only oracle exists is called a *linear probabilistically checkable proof (LPCP)*.

The curious reader may wonder whether such oracles could actually be constructed, or if we have delved into abstract nonsense. Indeed, it is not straightforward to implement such oracles, but it is possible by using techniques from cryptography. For this information-theoretic definition of an LPCP to be useful, we need a *cryptographic compiler*, which can be seen as a mechanism that bridges the gap between information-theoretic/abstract definitions of proof systems and cryptographic schemes that realize the abstract definitions. This thesis revolves around such cryptographic compilers for LPCPs.

There are some important properties for (cryptographic compilations of) proof systems. Firstly, we want our proof system to be *complete*. That is, for any true statement in a predetermined set of true statements, there exists a valid proof that conforms to the proof system. Secondly, we want *soundness*. That is, for any statement that is not in the predetermined set of true statements, a malicious prover should not be able to construct a proof adhering to the proof system that convinces the verifier (except with some small probability, called the *soundness error*). We also desire *zero-knowlege*, as discussed above. Furthermore, it is desirable for a proof system to minimize the amount of communication, so that the proof systems become more efficient[1].

When developing cryptographic compilers for abstract proof systems, these properties can be heavily affected for the worse. Furthermore, the compiler itself could require complex cryptographic machinery, which should be avoided as much as possible. The main goal of this thesis is to study the current cryptographic compilers for LPCPs, and to develop new cryptographic compilers that offer different pros and cons as compared to the current compilers.

1.1 Background and research goals

The context of this thesis lies in the modular approach to designing efficient zero-knowledge proof systems by separating the *information-theoretic* components from the *cryptographic* components of a proof system [12, Section 2]. The information-theoretic component describes the proof system with idealized assumptions (such as the existence of a *linear-only oracle* for LPCPs), while the cryptographic compiler removes these idealized assumptions by transforming the information-theoretic proof system into an interactive cryptographic

$\mathbf{q} = (\mathbf{q}_1, \ldots, \mathbf{q}_s) \in \mathbb{F}^s$, and obtain the inner product $\langle \boldsymbol{\pi}, \mathbf{q} \rangle = \sum_{i=1}^{s} \boldsymbol{\pi}_i \mathbf{q}_i$ from the oracle.

[1] Additionally, we desire that the (honest) prover and verifier have minimal computation times; however, we focus solely on communication in this thesis, as it is typically the main efficiency bottleneck in practical implementations of cryptographic proof systems.

protocol. This approach simplifies proof design, as each component can be studied and developed separately. Furthermore, this separation allows for flexible combinations of the two components, which could lead to more efficient and secure proof systems.

Our motivation for applying this modular design to LPCPs stems from the immense value of PCPs and their relation to LPCPs. The *PCP theorem*, widely regarded as one of the most important theorems in computer science, states that for any decision problem where purported solutions can efficiently be verified (i.e., for any NP-*problem*), there exists a PCP that can verify the solution by only looking at a *constant number* of proof symbols [3, 4]. This revolutionary result plays a pivotal role in the design of modern cryptographic proof systems, and in *computational hardness of approximation*, which is a field that concerns itself around the inherent difficulty of approximation algorithms for optimization problems [1, 18, 19, 40].

The idealized assumption in a PCP is that the verifier has restricted access to the proof string. This assumption can be dealt with by using *commitment schemes* (see Section 4.6), which are cryptographic primitives that generally rely on standard cryptographic assumptions (e.g., the *discrete logarithm (DLOG) assumption*[1]).

One major drawback of using PCPs directly, is that the proofs are often quite complicated, which in practice leads to fairly inefficient proof systems. By allowing for linear queries in the PCP model (giving us LPCPs), we can significantly simplify the proofs. For instance, the *Hadamard LPCP* (discussed in Section 6.1) is the LPCP analogue to the PCP theorem: it verifies purported solutions to NP-decision problems by only examining three linear combinations of the proof symbols via a linear-only oracle. This LPCP is significantly simpler than the PCPs that prove the PCP theorem. Besides simplifying PCP designs, LPCPs also have other applications, such as *(zero-knowledge) multi-party computation* [33, 34]. Hence, LPCPs are a natural class of information-theoretic proof systems that are worthy of study. The cost of the advantages of LPCPs is that their cryptographic compilers currently rely on heavy cryptographic machinery, namely *linear-only encryption schemes*, which belong to the field of *public-key* or *asymmetric* cryptography[2]. Roughly, linear-only encryption says that given the ciphertexts of some plaintexts, it is possible to create a ciphertext of a linear combination of the plaintexts, while only using public parameters. Although the existence of linear-only encryption schemes is believable, they have

[1] Roughly, the *discrete logarithm (DLOG) assumption* says that for a generator g of a finite cyclic group $\mathbb{G}$, it is hard to determine k from g^k; see Assumption 1.

[2] In public-key cryptography, messages are encrypted using the receiver's public key (which is public to everyone), and are decrypted using their private key (which is only known to the receiver).

not been reduced to standard cryptographic assumptions (yet), and have therefore not seen much scrutiny. Hence, we see the following trade-off: adding strength to the information-theoretic proof system seems to imply the necessity of heavier cryptographic machinery for the cryptographic compilers, at least if one is hoping to design a cryptographic proof system that is sufficiently efficient for practical implementations.

The main research goal of this thesis is to develop cryptographic compilers for LPCPs that rely on standard cryptographic assumptions. We apply two different cryptographic protocols to achieve this, namely a *compressed* Σ*-protocol* for *linear forms* (Chapter 7) and one for *arithmetic circuits* (Chapter 8). We also describe the linear-only encryption based compiler, and compare them to our new compilers. The exact research objectives of this thesis are presented in Table 1.1. By pursuing these objectives, we aim to contribute to the development of efficient cryptographic proofs. We seek to employ the advantages of LPCPs, while still relying on standard cryptographic assumptions.

Table 1.1: The research goals of this thesis.

Main goal	Cryptographically compile LPCPs using standard cryptographic assumptions.		
Subgoals	Compile LPCPs using a compressed Σ-protocol for linear forms.	Compile LPCPs using a compressed Σ-protocol for arithmetic circuits.	Compare new compilers to linear-only encryption based compiler.

1.2 Contribution

Our main contribution is the development of two new cryptographic compilers for LPCPs, based on *compressed* Σ*-protocol theory* [5]. Our compilers rely on a standard cryptographic assumption, namely the *discrete logarithm (DLOG) assumption*, instead of the heavy assumption that linear-only encryption schemes exist. Our compilers can compile any LPCP that is defined over a finite field $\mathbb{F}$ (i.e., for some $s \in \mathbb{N}$, the proof string is $\boldsymbol{\pi} \in \mathbb{F}^s$).

For our first cryptographic compiler, the Π_c*-compiler*, a verifier's query $\mathbf{q} \in \mathbb{F}^s$ and the corresponding response $\langle \mathbf{q}, \boldsymbol{\pi} \rangle$ are interpreted as an evaluation of a linear form $L(\boldsymbol{\pi})$, where L depends on $\mathbf{q}$. We employ a *compressed* Σ*-protocol* (some specific proof system) Π_c that gives a zero-knowledge proof that a value $z = L(\boldsymbol{\pi})$ is indeed obtained from the evaluation of the linear form. We apply this Π_c-protocol to each query in the LPCP, resulting in a compiled LPCP that only depends on the cryptographic assumptions underlying Π_c, which coincidentally is the DLOG assumption[1]. Furthermore, if the information-theoretic

[1] Although we focus on compressed Σ-protocols that rely on the DLOG assumption, we could replace these protocols by ones that rely on other standard cryptographic assumptions, such as the *RSA assumption* and lattice-based assumptions (the latter can be useful for post-quantum security – see Chapter 7).

LPCP is complete and (*perfect honest-verifier*) zero-knowledge, then so is the compiled LPCP. For general LPCPs, consisting of ξ queries and having soundness error $\varepsilon_{\text{LPCP}}$, the corresponding Π_c-compiled LPCP is an interactive protocol with soundness error $\varepsilon_{\text{LPCP}} + \xi \cdot \mathcal{O}(\log_2(s)/|\mathbb{F}|)$, in which $\mathcal{O}(s\xi)$ elements of $\mathbb{F}$ need to be communicated between the prover and verifier. Note that this large amount of communication can typically be reduced using an *amortization trick*, but the effectiveness has to be studied on a case-by-case basis, as the effectiveness depends on the probability distribution that the queries are sampled from.

For our second cryptographic compiler, the $\Pi_{\text{cs}}^{(1)}$*-compiler*, we introduce a strengthening of the LPCP model, namely a *quadratic probabilistically checkable proof (QPCP)*. In a QPCP, the verifier can query s-variate quadratic polynomials $q \in \mathbb{F}[X_1, \ldots, X_s]_{\leq 2}$ (with coefficients in $\mathbb{F}$) to an oracle that responds with $q(\boldsymbol{\pi}_1, \ldots, \boldsymbol{\pi}_s)$, without revealing information about the proof string $\boldsymbol{\pi} = (\boldsymbol{\pi}_1, \ldots, \boldsymbol{\pi}_s) \in \mathbb{F}^s$. Then, we interpret such quadratic queries as evaluations of *arithmetic circuits* (which we introduce in Section 3.2). We employ the compressed Σ-protocol $\Pi_{\text{cs}}^{(1)}$, which gives a zero-knowledge proof that a given circuit evaluates to a given value. We use the fact that the Hadamard LPCP, consisting of three queries, can cleverly be transformed into a *Hadamard QPCP* that only consists of a single quadratic query and has a shorter proof string. Applying the $\Pi_{\text{cs}}^{(1)}$-compiler directly to the quadratic query in the Hadamard QPCP results in a cryptographic proof that involves about one third of the communication costs of the Π_c-compiled Hadamard LPCP. Hence, the $\Pi_{\text{cs}}^{(1)}$-compiler is mostly valuable when applied to concrete LPCPs that can cleverly be transformed into QPCPs with fewer queries and a shorter proof string.

We could also interpret the linear queries of a general LPCP as quadratic queries without quadratic terms and compile the resulting QPCP with our $\Pi_{\text{cs}}^{(1)}$-compiler, but this approach yields poorer results when compared to the general Π_c-compiled LPCP.

All in all, our compilers make it possible to add power to information-theoretic proof systems, while still being able to cryptographically compile them into non-trivial proofs that rely on standard cryptographic assumptions. Furthermore, this thesis highlights the value of developing the cryptographic compilers separately from the underlying information-theoretic proof systems.

1.3 Related work

The main point of this thesis is to examine the value of the modular approach of separating the information-theoretic part of a proof system from the cryptographic compilers in the context of LPCPs. This modularity paradigm is made explicit by Yuval Ishai [12, 30].

Several studies have, implicitly or explicitly, applied this approach successfully. As PCPs are the objects of interest, we first discuss related works that developed cryptographic compilers for PCPs, after which we discuss cryptographic compilers for two enhancements of the PCP model:

1. We discuss related works on cryptographic compilers for *interactive oracle proofs*, which are interactive PCPs where the verifier gets bounded oracle access to each prover message.

2. We discuss cryptographic compilers for LPCPs, which are the compilers that this thesis aims to construct.

It appears to be folklore knowledge that one can convert a PCP into an efficient non-interactive proof in the *random oracle model* (i.e., in an idealized model where the prover and verifier can access a uniformly random function as a black box – see Section 4.5 for more details). Here, the idea is to have the prover compactly *commit* to its proof string[1], and subsequently open the coordinates queried by the verifier.

Other compilers exist in the *plain model* (i.e., without the assumption of a random oracle or any other such idealized functionality). In 1996, Goldreich and Kahan developed a cryptographic compiler for PCPs that gives an interactive constant-round compiled zero-knowledge proof for each NP-problem [26]. Although their separation of the information-theoretic PCPs from their cryptographic compiler is implicit, the resulting compiled PCP relies on standard cryptographic assumptions, such as the discrete logarithm assumption. Due to the implicit approach in their paper, their compiler cannot be applied to LPCPs, and therefore differs from our work. Furthermore, this compiler appears to have a poor soundness error for general PCPs.

To improve the soundness errors of compiled PCPs, another cryptographic compiler was developed that relies on different cryptographic assumptions [32], namely on the existence of a *collision resistant hash function*, which is a compressing function H (that maps bit-strings of any finite length to bit-strings of some fixed length $n \in \mathbb{N}$) such that the probability is small of finding distinct inputs x_1 and x_2 such that $H(x_1) = H(x_2)$. The problem with this compiler is that it requires much computation when applied in practice, which makes the compiler inefficient. This compiler differs from our work, as it compiles PCPs instead of LPCPs, and as collision resistant hash functions are a stronger assumption than the discrete logarithm assumption that underlies our cryptographic compilers

[1] For PCPs, the *commitment* is done by using a special data-structure called a *Merkle tree*. We discuss commitment schemes in Section 4.6

(although the existence of collision resistant hash functions is still considered a standard cryptographic assumption) [42].

To obtain cryptographic proofs for NP-problems, other models related to PCPs were considered, such as IOPs and LPCPs. In 2016, Ben-Sasson, Chiesa, and Spooner implicitly developed a cryptographic compiler that transforms IOPs[1] to non-interactive zero-knowledge proofs [9]. Their compiler basically repurposed the standard PCP compiler, and therefore also assumes the *random oracle model*. This work differs from our work, as it focuses on IOPs instead of LPCPs.

For LPCPs, the first cryptographic compiler relied on *linear homomorphic* public-key encryption schemes [31]. These encryption schemes are similar to the linear-only encryption schemes that we discussed above, with the difference being that given a ciphertext c derived from ciphertexts of some plaintexts and public parameters only, linear-only encryption schemes guarantee that c can *only* be an encryption of a linear combination of the plaintexts, whereas linear homomorphic encryption schemes do not make such a guarantee (and c could be an encryption of a non-linear function of the plaintexts). Although linear homomorphic encryption schemes can be based on standard cryptographic assumptions, using them for compiling LPCPs poses a problem for the soundness of the compiled proof. Namely, a malicious prover can commit to a *non-linear* oracle, which the linear homomorphic encryption scheme cannot detect. Therefore, an additional information-theoretic proof system is needed to guard the linearity of the prover's oracle. This work differs from our work, as our compilers do not need such an additional information-theoretic component.

To mitigate the problem of non-linear oracles, a cryptographic compiler was developed for LPCPs that relies on *linear-only encryption* [10]. This is currently the best compiler for LPCPs, as it solves the soundness problem that linear homomorphic encryption based compilers face. However, the major drawback of this compiler is that linear-only encryption schemes are, unlike our compilers, not based on standard and well-studied cryptographic assumptions. Furthermore, this compiler requires a setup by a trusted third-party (namely, the compiler requires a *common reference string*), which our compilers do not need.

All in all, existing literature lacks the application of cryptographic compilers for LPCPs that are based on standard cryptographic assumptions and do not require additional information-theoretic components. Hence, our thesis adds a valuable contribution to the field of cryptographically compiling LPCPs.

[1] This compiler assumes that the IOPs are of the correct form. That is, they must be *public-coin* (i.e., the prover can see the verifier's random coin tosses), and all queries must be *input-oblivious* (i.e., the queries do not depend on the input). Most IOPs encountered in the literature satisfy these conditions.

1.4 Thesis roadmap

In Chapter 2, we give a high-level overview of our cryptographic compilers. In Chapter 3, we revisit some definitions and theorems concerning complexity classes, we define boolean and arithmetic circuits (which are typical NP-problems to which proof systems are applied), and we give some general preliminaries. Then, in Chapter 4, we formally define probabilistic proof systems, their properties, and commitment schemes. In Chapter 5, we formally define zero-knowledge, after which we introduce the Hadamard LPCP and the state-of-the-art cryptographic compiler for LPCPs in Chapter 6. Then, our contributions appear in Chapter 7, where we compile LPCPs using a *compressed* Σ*-protocol* for linear queries, and in Chapter 8, where we transform the Hadamard LPCP into a Hadamard QPCP with a shorter proof and a single query, which we compile using a different *compressed* Σ*-protocol* for quadratic queries. In Chapter 9, we compare our new cryptographic compilers to the currently best one, and finally we conclude and give recommendations for further research in Chapter 10.

Notation and conventions

In this thesis, we will use the following notation and conventions:

- The natural numbers are taken as the set of positive integers $\mathbb{N} = \{1, 2, 3, \ldots\}$.
- For $N \in \mathbb{N}$, we write $[N]$ for the set $\{1, 2, \ldots, N\} \subset \mathbb{N}$.
- We write $\{0,1\}^*$ for the set of bit-strings of any (finite) length.
- For an n-dimensional vector $\mathbf{x}$, we denote its entries by $\mathbf{x}_1, \ldots, \mathbf{x}_n$.
- Given an n-dimensional vector $\mathbf{x}$ and an m-dimensional vector $\mathbf{y}$, we write $\mathbf{x} \| \mathbf{y} \coloneqq (\mathbf{x}_1, \ldots, \mathbf{x}_n, \mathbf{y}_1, \ldots, \mathbf{y}_m)$ for the $(n+m)$-dimensional vector obtained by concatenating $\mathbf{x}$ and $\mathbf{y}$.
- We write $c \xleftarrow{\$} \mathcal{C}$ for an element c that is sampled uniformly at random from a finite set $\mathcal{C}$.
- We write $\mathbb{F}_q$ for the finite field consisting of q elements.[1] We write $\mathbb{F}_q^*$ for the multiplicative group $\mathbb{F}_q \setminus \{0\}$. We omit the q (i.e., we write $\mathbb{F}$) when the size of the field

[1] We implicitly assume that q is a prime power, since any finite field has a prime power number of elements, and since there is a finite field of size q (which is unique up to isomorphism) for each prime power q [35, Theorem 2.5].

is clear from the context, or when we are concerned with a finite field of any size.

- We write $\mathbb{F}[X_1, \ldots, X_n]_{\leq d}$ for the set of n-variate polynomials, of total degree at most d, with coefficients in the field $\mathbb{F}$.
- We write $g(n) = \mathcal{O}(f(n))$ if there exist $C, n_0 > 0$ such that $\forall n \geq n_0$ it holds that $g(n) \leq Cf(n)$.
- In the context of proof systems, we use bold font (i.e., $\mathbf{P}$, $\mathbf{V}$) for provers and verifiers in information-theoretic proof systems, and we use calligraphic font (i.e., $\mathcal{P}$, $\mathcal{V}$) for provers and verifiers in cryptographically compiled proof systems.

2

Overview

In this chapter, we give a brief high-level outline of our cryptographic compilers for linear probabilistically checkable proofs (LPCPs). The main challenge for the compilers is to deal with the linear queries in an LPCP. We pursue two strategies, resulting in two cryptographic compilers. In Figure 2.1, we graphically represent our three approaches for improving cryptographic compilers for LPCPs using our two compilers.

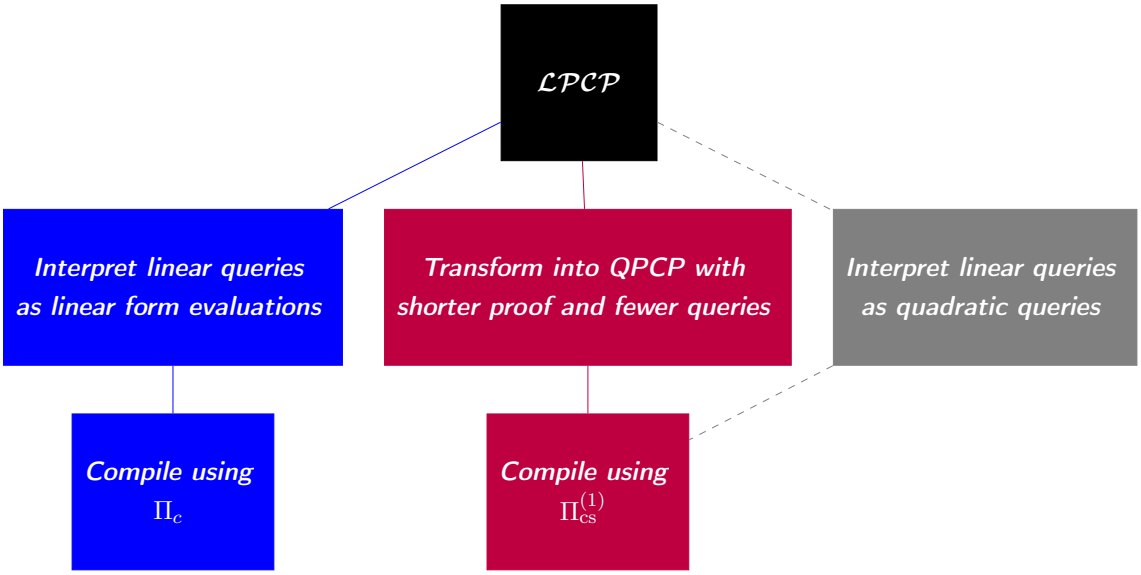

Figure 2.1: A graphical representation of our approaches for constructing cryptographic compilers for LPCPs. Starting from an information-theoretic LPCP in black, following the blue path represents the most direct and general approach. The purple path represents the best approach, but which cannot easily be generalized. The gray path represents a less useful approach that is included for completeness.

Firstly, we interpret each linear query of an information-theoretic LPCP as an evaluation

of a linear form, which we can compile using the *compressed* Σ*-protocol* (proof system) Π_c that relies on the discrete logarithm assumption, which is a standard cryptographic assumption. The details of Π_c are given in Section 7.1.

Secondly, we transform an information-theoretic LPCP into a quadratic probabilistically checkable proof (QPCP) with a shorter proof and fewer queries. We present an example of such a transformation in Section 8.1. The quadratic queries in the simpler QPCP can then be compiled using the $\Pi_{\text{cs}}^{(1)}$-protocol. The details of $\Pi_{\text{cs}}^{(1)}$ are given in Section 8.2.

A third approach, which also employs the $\Pi_{\text{cs}}^{(1)}$-protocol, is to interpret the linear queries in an LPCP as quadratic queries without quadratic terms. That is, we interpret the LPCP as a QPCP, which we can compile using the compressed Σ-protocol $\Pi_{\text{cs}}^{(1)}$. As this case applies a more powerful cryptographic compiler to the same information-theoretic LPCP, we do not expect this approach to result in more efficient proofs than the first two approaches, but we include this approach for completeness in Appendix A. Indeed, in Chapter 9, our comparison of the three approaches confirms this expectation.

3

Preliminaries

In this chapter, we discuss the preliminaries that are needed for the rest of the thesis. We start by revisiting some theory on complexity classes, then we introduce boolean and arithmetic circuits, and finally we introduce useful definitions and lemmas.

3.1 Complexity classes

In this section, we revisit the definitions of a language, a relation, and certain complexity classes, as these concepts are closely related to proof systems. These definitions follow the book *Computational Complexity* by Arora and Barak [2]. We start by defining the concept of a *language* and a *relation*.

Definition 3.1.1. Given an $n \in \mathbb{N}$ and a function $f\colon \{0,1\}^n \to \{0,1\}$, the set

$$L = \{x \in \{0,1\}^n \mid f(x) = 1\}$$

is a *language* (for f). We identify the function f with its language L.

Remark 3.1.2. Although we focus on the set $\{0,1\}^n$ for concreteness, note that any finite object in computer science can be represented as a finite bit-string. We will therefore use finite objects and bit-strings interchangeably throughout this thesis.

Remark 3.1.3. In general, when we refer to a language, we are in fact referring to an infinite family of languages, each for a given bit-length. That is, by a language L we in fact refer to a sequence $(L_{n_i})_{i=1}^{\infty}$ where each $L_{n_i} \subseteq \{0,1\}^{n_i}$ and $n_1 < n_2 < ...$ is an infinite monotonically increasing sequence of natural numbers. However, for simplicity, we will often omit this more cumbersome notation.

Many languages are obtained from *relations*.

Definition 3.1.4. A *relation* $\mathcal{R}$ is a subset $\mathcal{R} \subseteq \{0,1\}^n \times \{0,1\}^m$ for some $n, m \in \mathbb{N}$.

Remark 3.1.5. In computer science, we often interpret relations as pairs $(x; w)$ of an *instance* from a language and some *witness* for that instance. In fact, given a relation $\mathcal{R}$, we can define the corresponding language

$$L_{\mathcal{R}} = \{x \mid \exists w, (x; w) \in \mathcal{R}\}.$$

As relations and languages are related, we will use them somewhat interchangeably throughout this thesis.

For complexity classes, we will be interested in algorithms that decide whether words are in a language. The complexity of these algorithms will be expressed in the number of steps they need as a function of the *size* of the input. We formalize these concepts as follows.

Definition 3.1.6. Let L be a language. We define the *size* of $x \in L$, denoted $|x|$, as the length of the bit-string representation of x. We write $\text{poly}(|x|)$ (respectively $\exp(|x|)$) for a function of at most polynomial (respectively exponential) growth in $|x|$.

Definition 3.1.7. A language L is said to be *decided* by a deterministic algorithm $\mathcal{A}$ if, given an input x, $\mathcal{A}(x) = 1 \iff x \in L$.

Definition 3.1.8. A *complexity class* is a set of languages that are decided by algorithms that adhere to some resource bounds.

Next, we define the classes P and NP.

Definition 3.1.9. The class P *(polynomial-time)* is the set of languages L for which a deterministic algorithm exists that decides L in $\text{poly}(|x|)$ steps. We say that the algorithm runs in *polynomial-time*.

Definition 3.1.10. The class NP *(nondeterministic polynomial-time)* is the set of languages L for which a deterministic algorithm exists that, given an $x \in L$ and a witness w with $|w| \leq \text{poly}(|x|)$, is able to verify in $\text{poly}(|x|)$ steps that w is a valid witness for $x \in L$. Furthermore, if $x \notin L$, then for all w the algorithm rejects.

Intuitively, the class P consists of all problems that can be solved in polynomial-time, whereas the class NP consists of all problems for which proposed solutions can be *verified* in polynomial-time. We give an example of a concrete NP language.

Example 1. We say a graph $G = (V, E)$ is 3-*colorable* if all vertices can be colored using only three colors, in such a way that each pair of adjacent vertices is colored differently. Suppose f is a function that maps finite graphs G to $\{0, 1\}$ such that

$$f(G) = 1 \iff G \text{ is 3-colorable.}$$

Then, $L = \{G \mid f(G) = 1\}$ is a *language*[1] consisting of all graphs that are 3-colorable. This language is in NP, as given a graph G and a coloring w, we can check that w is a 3-coloring of G by checking for each vertex in G that all neighboring vertices have different colors. If G has n vertices, this takes at most $\binom{n}{2}$ comparisons.

In this example, a 3-colorable graph G is an *instance*. A 3-coloring w of G is a *witness*, and the pair $(G; w)$ is an element of the relation of 3-colorable graphs.

Now, we consider algorithms that run in *probabilistic polynomial-time (PPT)*. That is, the algorithm runs in a number of steps that is polynomial in the length of the input, and the algorithm is given access to an infinite unbiased tape of random coin tosses. In other words, at the cost of one time step, the algorithm may toss a random coin. As such algorithms will depend on random coin tosses, there will be some probability for which the algorithm could fail. Adding randomness in this way allows us to extend the class P to BPP.

Definition 3.1.11. The class BPP *(bounded-error probabilistic polynomial time)* is the set of languages L for which a PPT algorithm V exists (whose random coin tosses are determined by a random variable ρ) that given an $x \in L$ outputs 1 with probability at least $\frac{2}{3}$, denoted $\mathbb{P}[V(x; \rho) = 1] \geq \frac{2}{3}$, and if $x \notin L$, then $\mathbb{P}[V(x; \rho) = 1] \leq \frac{1}{3}$.

Analogously, we can extend the notion of NP to MA by allowing for randomness.

Definition 3.1.12 ([8])**.** The class MA *(Merlin-Arthur)* is the set of languages L for which a PPT algorithm V exists such that given an $x \in L$ and a witness w for x, we have $\mathbb{P}[V(x, w; \rho) = 1] \geq \frac{2}{3}$, and given an $x \notin L$, we have $\mathbb{P}[V(x, w'; \rho) = 1] \leq \frac{1}{3}$ for all w'.

Lemma 3.1.13. *It holds that* $\mathsf{P} \subseteq \mathsf{NP} \subseteq \mathsf{MA}$*. It is widely believed that* $\mathsf{P} \subsetneq \mathsf{NP}$ *and* $\mathsf{NP} = \mathsf{MA}$*, but we have no proofs for these beliefs.*

Note that it is not known how BPP relates to NP; the former could be a subset of the latter, or the other way around. They could also be equal. The most common belief is that $\mathsf{BPP} = \mathsf{P}$, so if one believes that $\mathsf{P} \neq \mathsf{NP}$, then the standard belief is that $\mathsf{BPP} \subsetneq \mathsf{NP}$.

[1] Note that, as announced in Remark 3.1.2, here we use graphs interchangeably with any finite bit-string representation of them. Furthermore, as announced in Remark 3.1.3, the language of 3-colorable graphs is in fact a sequence of languages $(L_{n_i})_{i=1}^{\infty}$, where $n_i = \binom{i}{2}$ and L_{n_i} consists of all 3-colorable graphs that have exactly i vertices. The reason we need n_i bits to encode a graph of i vertices, is that we encode each possible edge with a 1 if it appears in the graph and otherwise with a 0.

3.2 Boolean and arithmetic circuits

In this section, we discuss *boolean circuits* and *arithmetic circuits*, which are the main objects in typical NP-problems for which *probabilistic proof systems* (which we introduce in Chapter 4) are being used. This section is based on the book *Computational Complexity* by Arora and Barak [2].

A well-known NP-problem is the *boolean satisfiability problem (SAT)*, where given a boolean formula, the question is whether there exists an assignment of the variables such that the formula evaluates to true.

A natural generalization of SAT is the *circuit SAT* problem. Roughly, a *boolean circuit* is a set of gates (representing boolean operations) and wires (representing inputs and outputs) such that each gate gets one or more inputs and produces one output by applying the boolean operation to its inputs. The entire circuit gets one or more inputs and produces one output. Figure 3.1 shows a simple boolean circuit. The circuit SAT problem then asks whether we can find inputs to the circuit such that its output is true (in Figure 3.1, inputs $\{p \mapsto 0, q \mapsto 1, r \mapsto 0\}$ make the circuit output true). If the underlying graph of a circuit is a *tree*, then one recovers the (formula) SAT problem. Thus, circuit SAT is indeed a generalization of SAT.

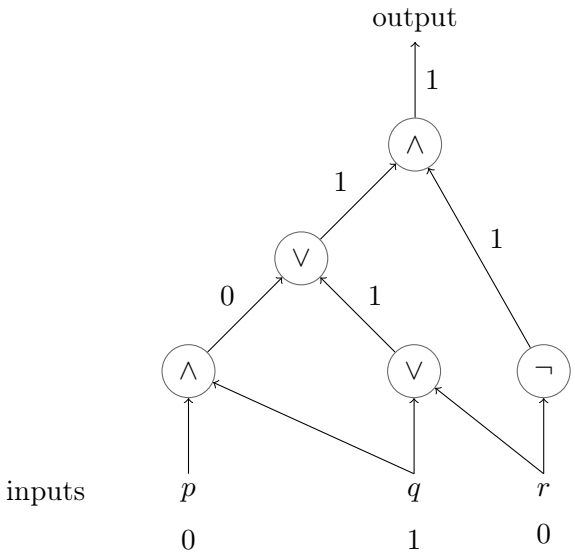

Figure 3.1: A simple boolean circuit, representing the logical formula $((p \wedge q) \vee (q \vee r)) \wedge (\neg r)$. Given inputs p and r false, and q true, the circuit outputs true.

In cryptographic applications, the main task is often to prove that some computation has been done correctly by one of the parties involved. The circuit SAT problem can be

very useful for proving that a computation has been done correctly. A computation can namely be represented by a circuit. Proving that a computation has been done correctly then boils down to proving that one knows inputs for the circuit such that the output is true. In cryptography, we often even desire proof systems where the prover can assert that such inputs exist, without revealing any of the inputs (i.e., *zero-knowledge proofs*).

For some tasks, a more natural variant of the circuit SAT problem is the *arithmetic circuit SAT* problem, where instead of boolean operations the gates represent field operations. Working with an arithmetic field instead of booleans opens the door to algebraic techniques, which often prove useful. The arithmetic circuit SAT problem then asks whether there are inputs to the circuit such that it outputs some chosen value.

When analyzing the efficiency of algorithms that solve (arithmetic) circuit SAT problems, we often analyze the number of steps the algorithm needs as a function of the *size* of the circuit, where the size is defined as the total number of gates in the circuit.

3.3 Useful definitions and lemmas

In this section, we mention some useful definitions and lemmas which we will use throughout this thesis. Most lemmas are known, but we include some proofs for completeness.

As we will often compare probability distributions, we start with the following definitions.

Definition 3.3.1. Let X and Y be random variables. We say that X and Y are *identically distributed*, denoted $X \equiv Y$, if they have the same probability distribution.

Definition 3.3.2. Let $X = \{X_\lambda\}_{\lambda\in\mathbb{N}}$ and $Y = \{Y_\lambda\}_{\lambda\in\mathbb{N}}$ be families of random variables. We say that X and Y are *computationally indistinguishable*, denoted $X \approx_c Y$, if given a *PPT* algorithm $\mathcal{A}$, any polynomial p, and sufficiently large $\lambda \in \mathbb{N}$, we have

$$|\mathbb{P}[\mathcal{A}(X_\lambda) = 1] - \mathbb{P}[\mathcal{A}(Y_\lambda) = 1]| \leq \frac{1}{p(\lambda)}.$$

Note that $\mathcal{A}$ can be interpreted as an algorithm that tries to distinguish two distributions. The above definition says that any PPT algorithm can only distinguish between the random variables with very small probability.

In many security definitions, we want to express that some function is very small. To formalize this, we need the concept of *negligible functions*.

Definition 3.3.3 (Negligible functions [2])**.** Given a domain $D \subseteq \mathbb{R}_{>0}$, a function $f\colon D \to [0,1]$ is said to be *negligible in* x, denoted $f = \text{negl}(x)$, if there exists a $C \in D$, such that for any polynomial $p\colon D \to (-\infty, \infty)$, we have $f(x) \leq \frac{1}{p(x)}$ for all $x \geq C$.

Starting from Chapter 6, we will often encounter the number $s + \binom{s}{2}$. We can write this number more concisely as follows.

Lemma 3.3.4. *Given $s \in \mathbb{N}$ (with $s \geq 2$), we have*

$$s + \binom{s}{2} = \binom{s+1}{2}.$$

Proof. We have

$$\begin{aligned} s + \binom{s}{2} &= s + \frac{s!}{2!(s-2)!} = s + \frac{1}{2}s(s-1) = \frac{1}{2}s^2 + \frac{1}{2}s \\ &= \frac{1}{2}(s+1)s = \frac{(s+1)!}{2!(s-1)!} = \binom{s+1}{2}. \end{aligned}$$

□

When proving soundness of proof systems, we typically use an important result from probability theory, namely the union bound (also known as Boole's inequality).

Lemma 3.3.5 (Union Bound)**.** *For any finite set of events $A_1, A_2, \ldots, A_n$, we have*

$$\mathbb{P}\left[\bigcup_{k=1}^{n} A_k\right] \leq \sum_{k=1}^{n} \mathbb{P}[A_k].$$

Proof. Apply the addition rule

$$\mathbb{P}[A \cup B] = \mathbb{P}[A] + \mathbb{P}[B] - \mathbb{P}[A \cap B] \leq \mathbb{P}[A] + \mathbb{P}[B]$$

inductively. □

The following three lemmas prove that additions, multiplications, and inner products with uniform randomly sampled elements are uniformly distributed. These lemmas will often be used when analyzing *zero-knowledge* and *soundness*.

Lemma 3.3.6. *Let $\mathbb{F}_q$ be a finite field and let $\alpha \xleftarrow{\$} \mathbb{F}_q$. For any $\beta \in \mathbb{F}_q$, the element $\alpha + \beta$ is uniformly distributed over $\mathbb{F}_q$.*

Proof. We prove that for any $\gamma \in \mathbb{F}_q$, $\mathbb{P}_\alpha[\alpha + \beta = \gamma] = \frac{1}{|\mathbb{F}_q|}$. Let $\beta, \gamma \in \mathbb{F}_q$. Then,

$$\mathbb{P}_\alpha[\alpha + \beta = \gamma] = \mathbb{P}_\alpha[\alpha = \gamma - \beta] = \frac{1}{|\mathbb{F}_q|},$$

as α was sampled uniformly at random from $\mathbb{F}_q$. □

Lemma 3.3.7. *Let $\mathbb{F}_q$ be a finite field and let $\alpha \xleftarrow{\$} \mathbb{F}_q$. For any $\beta \in \mathbb{F}_q^*$, the element $\alpha\beta$ is uniformly distributed over $\mathbb{F}_q$.*

Proof. We prove that for any $\gamma \in \mathbb{F}_q$, $\mathbb{P}_\alpha[\alpha\beta = \gamma] = \frac{1}{|\mathbb{F}_q|}$. Let $\beta \in \mathbb{F}_q^*$ and $\gamma \in \mathbb{F}_q$. Then,

$$\mathbb{P}_\alpha[\alpha\beta = \gamma] = \mathbb{P}_\alpha[\alpha = \gamma \cdot \beta^{-1}] = \frac{1}{|\mathbb{F}_q|},$$

as α was sampled uniformly at random from $\mathbb{F}_q$. □

Lemma 3.3.8. *Let $\mathbb{F}_q$ be a finite field and $n \in \mathbb{N}$. Let $\boldsymbol{\alpha} \xleftarrow{\$} \mathbb{F}_q^n$. For any $\boldsymbol{\beta} \in \mathbb{F}_q^n \setminus \{\mathbf{0}\}$, the element $\langle \boldsymbol{\alpha}, \boldsymbol{\beta} \rangle$ is uniformly distributed over $\mathbb{F}_q$.*

Proof. We prove this lemma by induction in the dimension n. The base case $n = 1$ is proven in Lemma 3.3.7. Now, for $n > 1$ we assume that the statement holds for $n - 1$. In other words, our induction hypothesis is that for $\boldsymbol{\alpha}' \xleftarrow{\$} \mathbb{F}_q^{n-1}$ and any $\boldsymbol{\beta}' \in \mathbb{F}_q^{n-1} \setminus \{\mathbf{0}\}$, the element $\langle \boldsymbol{\alpha}', \boldsymbol{\beta}' \rangle$ is uniformly distributed over $\mathbb{F}_q$. Now, it remains for us to prove that for $\boldsymbol{\alpha} \xleftarrow{\$} \mathbb{F}_q^n$, for any $\boldsymbol{\beta} \in \mathbb{F}_q^n \setminus \{\mathbf{0}\}$, the element $\langle \boldsymbol{\alpha}, \boldsymbol{\beta} \rangle$ is uniformly distributed over $\mathbb{F}_q$. We do this by showing that for any $\gamma \in \mathbb{F}_q$, $\mathbb{P}_{\boldsymbol{\alpha}}[\langle \boldsymbol{\alpha}, \boldsymbol{\beta} \rangle = \gamma] = \frac{1}{|\mathbb{F}_q|}$. Let $\boldsymbol{\beta} \in \mathbb{F}_q^n \setminus \{\mathbf{0}\}$ and $\gamma \in \mathbb{F}_q$. Then,

$$\mathbb{P}_{\boldsymbol{\alpha}}[\langle \boldsymbol{\alpha}, \boldsymbol{\beta} \rangle = \gamma] = \mathbb{P}_{\boldsymbol{\alpha}}\left[\sum_{i=1}^{n} \boldsymbol{\alpha}_i \cdot \boldsymbol{\beta}_i = \gamma\right] = \mathbb{P}_{\boldsymbol{\alpha}}\left[\sum_{i=1}^{n-1} \boldsymbol{\alpha}_i \cdot \boldsymbol{\beta}_i = \gamma - \boldsymbol{\alpha}_n \boldsymbol{\beta}_n\right].$$

Let $\boldsymbol{\beta}' \coloneqq (\boldsymbol{\beta}_1, \ldots, \boldsymbol{\beta}_{n-1}) \in \mathbb{F}_q^{n-1}$ and $\boldsymbol{\alpha}' \coloneqq (\boldsymbol{\alpha}_1, \ldots, \boldsymbol{\alpha}_{n-1}) \in \mathbb{F}_q^{n-1}$. We distinguish two cases, namely $\boldsymbol{\beta}' = \mathbf{0}$ and $\boldsymbol{\beta}' \neq \mathbf{0}$. If $\boldsymbol{\beta}' = \mathbf{0}$, then $\boldsymbol{\beta}_n \neq 0$ (as $\boldsymbol{\beta} \neq \mathbf{0}$), and thus

$$\mathbb{P}_{\boldsymbol{\alpha}}[\langle \boldsymbol{\alpha}, \boldsymbol{\beta} \rangle = \gamma] = \mathbb{P}_{\boldsymbol{\alpha}}\left[\sum_{i=1}^{n-1} \boldsymbol{\alpha}_i \cdot \boldsymbol{\beta}_i = \gamma - \boldsymbol{\alpha}_n \boldsymbol{\beta}_n\right] = \mathbb{P}_{\boldsymbol{\alpha}}[\boldsymbol{\alpha}_n \boldsymbol{\beta}_n = \gamma] = \frac{1}{|\mathbb{F}_q|}$$

by Lemma 3.3.7 (where we implicitly use that $\boldsymbol{\alpha} \xleftarrow{\$} \mathbb{F}_q^n$ implies $\boldsymbol{\alpha}_i \xleftarrow{\$} \mathbb{F}_q$ for each $i \in [n]$). Now, if $\boldsymbol{\beta}' \neq \mathbf{0}$, then

$$\begin{aligned}
\mathbb{P}_{\boldsymbol{\alpha}}[\langle \boldsymbol{\alpha}, \boldsymbol{\beta} \rangle = \gamma] &= \mathbb{P}_{\boldsymbol{\alpha}}\left[\sum_{i=1}^{n-1} \boldsymbol{\alpha}_i \cdot \boldsymbol{\beta}_i = \gamma - \boldsymbol{\alpha}_n \boldsymbol{\beta}_n\right] = \mathbb{P}_{\boldsymbol{\alpha}}\left[\langle \boldsymbol{\alpha}', \boldsymbol{\beta}' \rangle = \gamma - \boldsymbol{\alpha}_n \boldsymbol{\beta}_n\right] \\
&= \sum_{\eta \in \mathbb{F}_q} \mathbb{P}_{\boldsymbol{\alpha}}\left[\langle \boldsymbol{\alpha}', \boldsymbol{\beta}' \rangle = \gamma - \boldsymbol{\alpha}_n \boldsymbol{\beta}_n \mid \boldsymbol{\alpha}_n = \eta\right] \mathbb{P}_{\boldsymbol{\alpha}}[\boldsymbol{\alpha}_n = \eta] \\
&\overset{(*)}{=} \frac{1}{|\mathbb{F}_q|} \sum_{\eta \in \mathbb{F}_q} \mathbb{P}_{\boldsymbol{\alpha}'}\left[\langle \boldsymbol{\alpha}', \boldsymbol{\beta}' \rangle = \gamma - \eta \boldsymbol{\beta}_n\right] \\
&\overset{\text{(IH)}}{=} \frac{1}{|\mathbb{F}_q|} \sum_{\eta \in \mathbb{F}_q} \frac{1}{|\mathbb{F}_q|} = \frac{1}{|\mathbb{F}_q|},
\end{aligned}$$

where in step $(*)$ we again use that $\boldsymbol{\alpha} \xleftarrow{\$} \mathbb{F}_q^n$ implies $\boldsymbol{\alpha}_i \xleftarrow{\$} \mathbb{F}_q$ for each $i \in [n]$, and in step (IH) we use the induction hypothesis and the fact that $\boldsymbol{\beta}' \neq \mathbf{0}$. □

Next, we formulate the *Schwartz-Zippel Lemma*, which will play an important role in proving soundness of proof systems.

Lemma 3.3.9 (Schwartz-Zippel Lemma [37, 41, 45])**.** *Given a field $\mathbb{F}$ and a non-zero polynomial $f \in \mathbb{F}[X_1, \ldots, X_n]$ of degree $d \geq 0$, a finite subset $S \subseteq \mathbb{F}$, and independently uniformly randomly sampled elements $r_1, \ldots, r_n \xleftarrow{\$} S$, then*

$$\mathbb{P}[f(r_1, \ldots, r_n) = 0] \leq \frac{d}{|S|}.$$

Remark. Note that Lemma 3.3.8 is the degree $d = 1$ case of the Schwartz-Zippel Lemma when $S = \mathbb{F}$ for a finite field $\mathbb{F}$.

Finally, we state the *discrete logarithm (DLOG) assumption*, which is the basis of the security of many cryptographic schemes.

Assumption 1 (Discrete logarithm assumption)**.** The *Discrete Logarithm (DLOG) assumption*, relative to a cyclic group $\mathbb{G}$ of order q, says that given a generator $g \in \mathbb{G}$ and a $k \in \mathbb{Z}_q^*$, it is *hard* to determine k from g^k. More precisely, for any PPT algorithm $\mathcal{A}$, the probability (taken over uniformly random k and random coin tosses of $\mathcal{A}$) that $\mathcal{A}$ outputs k on input g^k is negligible.

Remark. Note that for the discrete logarithm assumption, we are actually referring to a growing infinite family of groups $(\mathbb{G}_\lambda)_{\lambda \in \mathbb{N}}$, where the size of the group $\mathbb{G}_\lambda$ grows with λ. Typically, $|\mathbb{G}_\lambda| \approx 2^\lambda$ [14]. However, for simplicity, we will omit this more cumbersome notation.

4

Probabilistic proof systems

In this chapter, we discuss the main objects of interest of this thesis, namely *probabilistic proof systems* and their properties. Essentially, a proof system consists of a prover that wants to prove a statement to a verifier. By imposing certain computational restrictions on the prover and/or verifier, we get different types of proof systems. Additionally, we can require that the prover leaks no information beyond the fact that the statement at hand is true, in which case we speak of a *zero-knowledge* proof (Chapter 5).

In Section 4.1, we introduce interactive proof systems generally. Then, we focus on *probabilistically checkable proofs (PCPs)* in Section 4.2, which are probabilistic proof systems with bounded access to the proof (i.e., the verifier reads only a few locations from the proof via an oracle). Thereafter, we introduce the *linear PCP (LPCP)* in Section 4.3, which is a generalization of the PCP: the oracle in an LPCP provides access to *linear combinations* of proof locations. In Section 4.4, we discuss *soundness analysis* of proof systems. Then, in Section 4.5, we discuss the *Fiat-Shamir transform*, which allows an interactive proof system to be transformed into a non-interactive one. Finally, in Section 4.6, we discuss *commitment schemes*, which serve as important building blocks for many proof systems. In particular, we focus on the *Pedersen commitment*.

4.1 Interactive proof systems

Traditionally, a proof was always considered to be a fixed static object that is verified by reading the full proof. However, since randomized and interactive computations have emerged, it was a natural development to also consider randomized and interactive proofs, where the verifier and the prover interact dynamically [25].

The general idea of an *interactive proof system* is that a computationally unbounded *prover* wants to prove to a computationally bounded (PPT) *verifier* that some proposition is true (usually that a word is a member of some language). The verifier can send *challenges* (questions) to the prover, which the prover has to answer convincingly to assure the verifier that they actually have a witness for the truth of the proposition. There may be one or poly($|x|$) rounds of challenges and responses. The interactive proof system is shown in Figure 4.1.

Note that, throughout this thesis, we separate *public input* (which is available to both prover and verifier) from *private input* (which is available to the prover only) using a semicolon. Thus, $(x_1, \ldots, x_m; w_1, \ldots, w_n)$ means the prover and verifier have access to $x_1, \ldots, x_m$, and only the prover knows $w_1, \ldots, w_n$.

To formalize the definition of an interactive proof, we first need to define the notion of a strategy and an interactive protocol.

Definition 4.1.1. A *strategy* is a function that describes the party's next move in the interactive proof as a function of the proposition to be proved (also called the *common input*), the party's internal random coin tosses, and all messages received so far.

Definition 4.1.2. A *k-round interactive protocol* is a k-round interactive game between a computationally unbounded prover and a probabilistic verifier. Each round i has the following structure:

1. The verifier sends a challenge c_i to the prover.
2. The prover responds with a message m_i, determined by the common input, the witness, all challenges that the verifier has sent so far, and internal random coin tosses (determined by a random variable ρ').
3. The verifier determines the next challenge, based on the common input, all messages previously sent by the prover, and internal random coin tosses (determined by a random variable ρ).

After k such rounds, the verifier outputs 1 if they accept and 0 if they reject.

We formalize the concept of an interactive proof as follows [25, Definition 1.1].

Definition 4.1.3 (Interactive Proof System)**.** A *k-round interactive proof system* for a relation $\mathcal{R}$ with *soundness error* $s\colon \{0,1\}^* \to [0,1]$ consists of a *prover* executing a computationally unbounded strategy, and a *verifier* executing a PPT strategy, satisfying two conditions:

- *Completeness.* For every common input $x \in L_{\mathcal{R}}$, there exist prover and verifier strategies $\mathbf{P}$ and $\mathbf{V}$ respectively, such that $\mathbf{V}$ accepts with probability 1 after engaging with $\mathbf{P}$ in the k-round interactive protocol, denoted $\mathbb{P}[\mathbf{V}(x, m_1, \ldots, m_k; \rho) = 1] = 1$.
- *Soundness.* For every common input $x \notin L_{\mathcal{R}}$ and every prover strategy, we have $\mathbb{P}[\mathbf{V}(x, m'_1, \ldots, m'_k; \rho) = 1] \leq s(x)$.

We write $\langle \mathbf{P} \leftrightarrow \mathbf{V} \rangle$ for the interactive proof system between prover strategy $\mathbf{P}$ and verifier strategy $\mathbf{V}$.

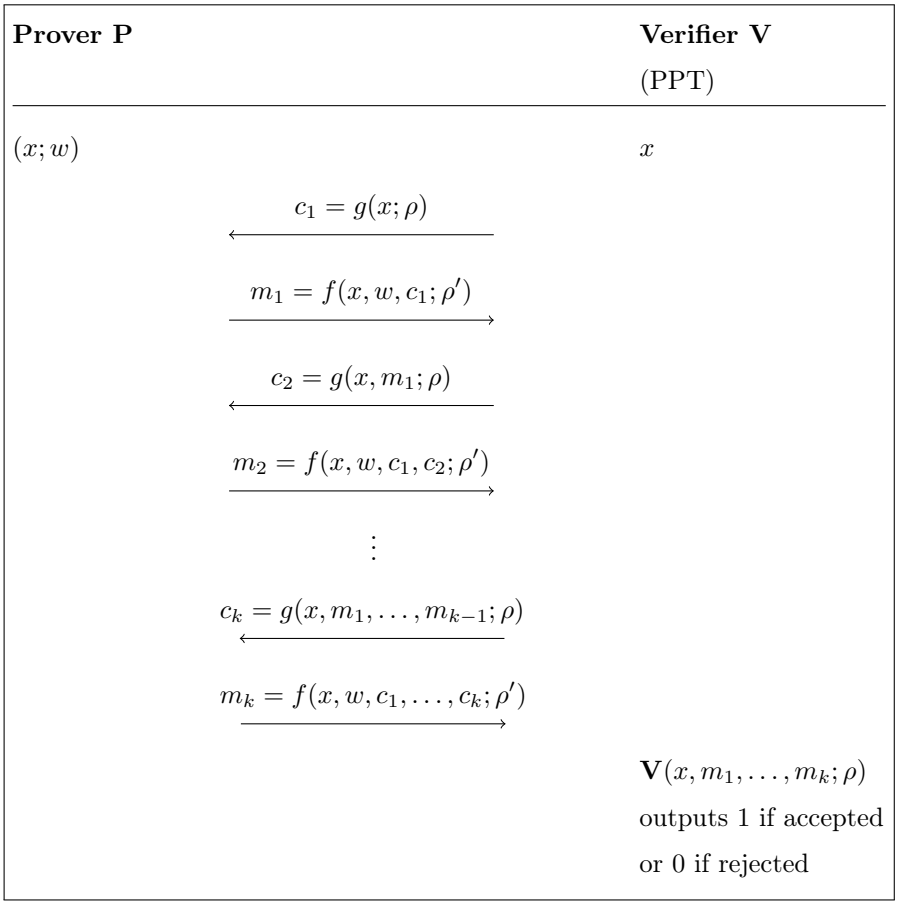

Figure 4.1: A k-round interactive proof system with common input x and purported witness w.

Remark 4.1.4. While in the definition of proof systems we allow for arbitrary soundness error functions $s\colon \{0,1\}^* \to [0,1]$, in the constructions the soundness should be thought of as a constant, say $\frac{1}{2}$, unless explicitly stated otherwise.

Remark 4.1.5. Note that in the definition above, we require *perfect completeness*. One can imagine introducing a parameter $c \in (0,1)$, such that $\mathbb{P}[\mathbf{V}(x, m_1, \ldots, m_k; \rho) = 1] \geq c$

for $x \in L_{\mathcal{R}}$. However, most proof systems in literature achieve perfect completeness ($c = 1$). In the rest of this thesis, we only consider protocols with perfect completeness.

Definition 4.1.6. We denote IP *(interactive polynomial-time)* for the set of languages for which a k-round interactive proof system exists with soundness error $\frac{1}{2}$, and with $k = \text{poly}(|x|)$.

Note that the soundness error of $\frac{1}{2}$ is an arbitrary choice. We can always repeat the interactive proof a polynomial number of times to make the soundness error smaller than any chosen constant (or even as small as $\exp(-\text{poly}(n))$, which is important for cryptographic applications). Hence, the definition of IP does not depend on the choice of soundness error. Note, however, that there is a trade-off between the number of rounds (i.e., the amount of computation and communication), and the soundness error. With more rounds, we can reduce the soundness error.

Below, we give an example of an interactive proof system that can be used to show that two graphs are *non-isomorphic* (closely following Goldreich's survey [25]).

Definition 4.1.7. We say that two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are *isomorphic*, denoted $G_1 \cong G_2$, if there exists a bijective function $\varphi \colon V_1 \to V_2$, such that $\{v_1, v_2\} \in E_1 \iff \{\varphi(v_1), \varphi(v_2)\} \in E_2$. We call φ an *isomorphism*.

Figure 4.2 presents the interactive proof system that can be used to prove that two graphs are non-isomorphic. The main idea is that the verifier challenges the prover by randomly choosing one of the graphs and then sending a *uniformly* random permutation of the selected graph. The prover then has to tell the verifier whether they received an isomorphism of G_1 or G_2. The prover can only do this reliably if the two graphs are actually non-isomorphic: if the graphs are isomorphic, then the distribution of the transmitted graph is *independent* of the graph the verifier chose, and thus from the prover's point of view, it is equally likely that the graph they received is a permutation of G_1 or G_2.

Historically, a distinction was made between interactive proof systems where the random coin tosses are public[1] (called *Arthur-Merlin proof systems* [8]), and where the random coin tosses are private [27]. However, Goldwasser and Sipser proved that these two systems are equivalent in power in terms of the languages of which they can prove membership [28].

[1] If the verifier's random coin tosses are visible to the prover, we say that the random coin tosses are public.

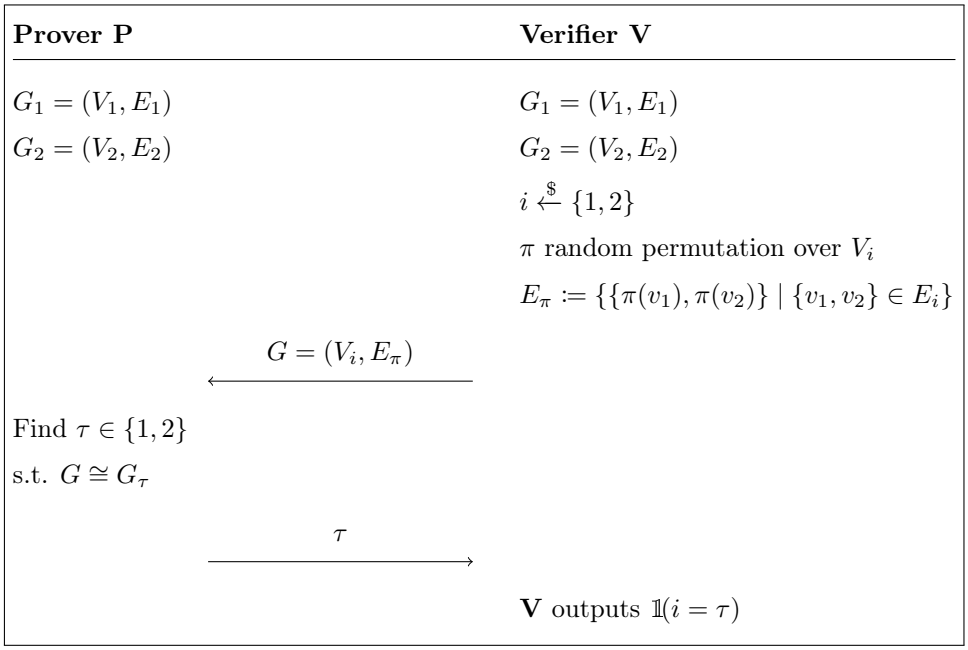

Figure 4.2: A 1-round interactive proof system where the prover proves that two graphs are non-isomorphic. Note that if $G \cong G_1 \cong G_2$, the prover will send $\tau \xleftarrow{\$} \{1, 2\}$ to the verifier.

4.2 Probabilistically checkable proofs

An important design principle for proof systems is the construction of *efficient* verifiers. That is, we desire proof systems in which the communication consists of as few bits as possible. To this end, proof systems have been developed in which the verifier has *bounded* access to the proof (i.e., the verifier gets an oracle with which they can query a few proof symbols).

Note that a *proof* should not be confused with a *witness*; a proof is the set of messages sent by the prover to the verifier, which typically differs from a witness for the membership of a word in a language.

In this section, we will discuss *probabilistically checkable proofs (PCPs)*, which are non-interactive proof systems in which the verifier gets *oracle access* to a proof, to which they only do a few queries to determine whether the proof is valid. Essentially, this means that the verifier can determine whether a purported witness is valid by only reading a few proof symbols. This is made possible by adding much redundancy to a proof. The PCP system is represented graphically in Figure 4.3. The idea of an oracle is represented graphically in Figure 4.4, where besides the PCP oracle, the oracles in an *LPCP* (Section 4.3) and a *QPCP* (Section 8.1) are represented as well.

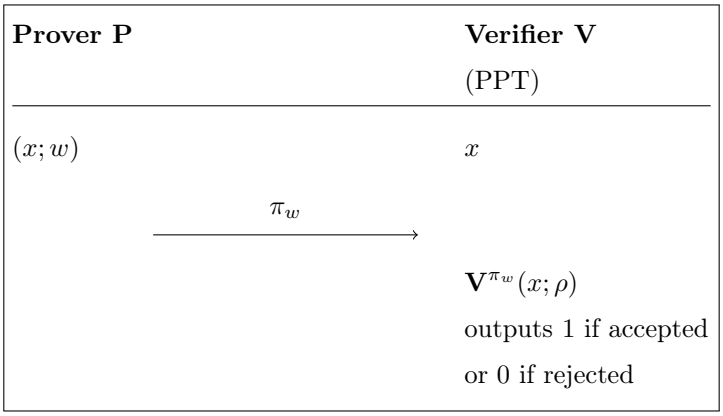

Figure 4.3: A probabilistically checkable proof with common input x and purported witness w.

For a PCP, we will be interested in the number of locations that need to be queried from the proof, and a measure for the amount of randomness that is involved in the proof system. This measure of randomness is closely related to the length of the proofs.[1]

We formally define PCPs as follows.

Definition 4.2.1 (Probabilistically Checkable Proof [25])**.** A *probabilistically checkable proof (PCP) system* for a relation $\mathcal{R}$ with *soundness error* $s\colon \{0,1\}^* \to [0,1]$ is a *verifier* $\mathbf{V}$ executing a PPT strategy, satisfying two conditions:

- *Completeness.* For every $(x; w) \in \mathcal{R}$ (where x is common input), there exists a proof oracle π_w, such that $\mathbf{V}$ accepts with probability 1 on input x and access to π_w. We denote this as $\mathbb{P}[\mathbf{V}^{\pi_w}(x; \rho) = 1] = 1$, where ρ is the random variable determining the verifier's internal random coin tosses.
- *Soundness.* For all $x \notin L_{\mathcal{R}}$ and for all oracles π', $\mathbb{P}[\mathbf{V}^{\pi'}(x; \rho) = 1] \leq s(x)$.

As was the case for interactive proof systems (Definition 4.1.3), we can reduce the soundness error by doing extra rounds of the proving process. Hence, we have a trade-off between the soundness error and the total number of proof symbols that are read.

At first glance, the PCP model might seem a bit unnatural. For example, how can we force a PPT verifier to only read a few locations from the proof? This might not seem

[1] Suppose the PCP can do q queries and toss r coins. There are 2^r different possible outcomes of the sequence of coin tosses. If, for each of these sequences, the PCP queries q unique locations of the proof, then the total number of locations that can possibly be queried is $2^r q$. Therefore, $2^r q$ is an upper bound for the effective proof length. Technically, we assume here that the verifier's queries are *non-adaptive* (i.e., the query locations only depend on the verifier's internal random coin tosses, and not on what is read from previous queries). Most PCP constructions are non-adaptive PCPs.

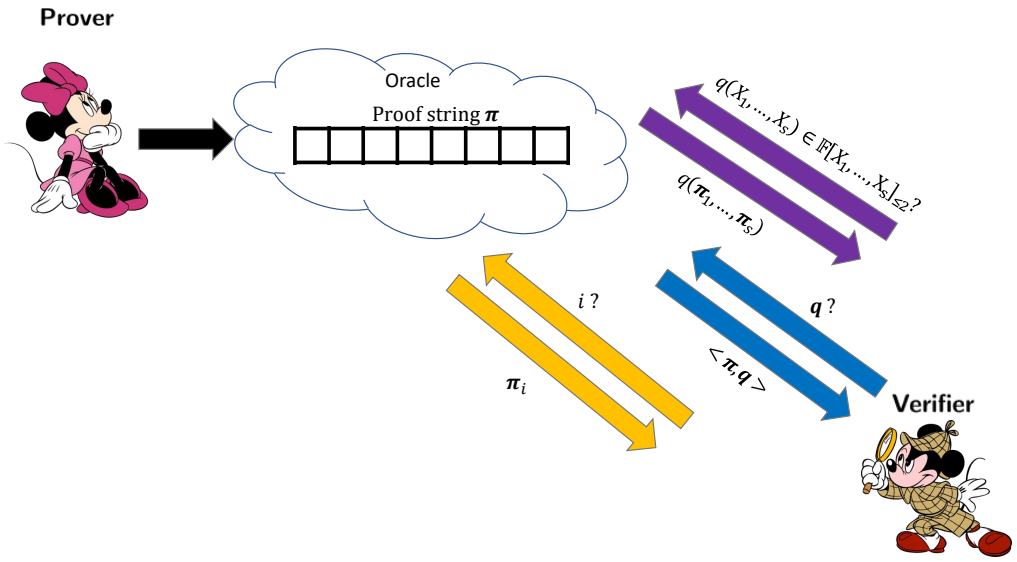

Figure 4.4: A graphical representation of three types of oracles for a proof string $\boldsymbol{\pi}$. In orange, we have the PCP oracle that gives access to individual proof symbols $\boldsymbol{\pi}_i$ (Section 4.2). In blue, we have the LPCP oracle that gives access to inner products with the proof string (Section 4.3). In purple, we have the QPCP oracle that gives access to s-variate quadratic polynomial queries evaluated in the proof string (Section 8.1).

like an important detail at this point, but for cryptographic applications (where we often wish to limit how much a verifier can learn from an interaction), it will be crucial that the verifier's access to the proof is indeed restricted. Fortunately, cryptographic techniques can indeed implement a PCP system. The basic idea is to have the prover commit to a proof string, after which the verifier responds by sending the desired query locations. The prover then reveals the query locations, together with a proof that they are indeed the values they had committed to. Note that this does cost a bit of interaction between the prover and verifier, but it is quite minimal and can furthermore be removed later via the *Fiat-Shamir transform* (see Section 4.5). We make this usage of cryptographic techniques more explicit in Chapter 6, Chapter 7, and Chapter 8.

Furthermore, one might initially perceive the PCP model as limiting, since restricting the verifier's access to the proof may appear to restrict our ability to prove interesting statements. However, the *PCP theorem* [3, 4], which is one of the most celebrated theorems in computer science, reveals the opposite to be true. The PCP theorem shows that the PCP model is powerful enough to prove all NP-statements with only a *constant number* of

queries.

Theorem 4.2.2 (PCP theorem [3, 4])**.** *For each language L in* NP*, there exists a PCP that proves membership in L. The PCP only needs $\mathcal{O}(\log |x|)$ random coin tosses (where x is the input) and a* constant number *of oracle queries.*

4.3 Linear probabilistically checkable proofs

We can strengthen the PCP model by replacing the single-point queries by *linear* queries. That is, given a proof string $\boldsymbol{\pi} \in \mathbb{F}^s$ (where $\mathbb{F}$ is some finite field), instead of querying a single point $\boldsymbol{\pi}_i \in \mathbb{F}$, the verifier can query linear combinations of points in $\boldsymbol{\pi}$. That is, the verifier can send a query $\mathbf{q} \in \mathbb{F}^s$ to the oracle to obtain $\langle \boldsymbol{\pi}, \mathbf{q} \rangle$. Such an oracle is represented in blue in Figure 4.4. The resulting proof system is called a *linear PCP (LPCP)* [31] and is represented graphically in Figure 4.5.

Prover P		**Verifier V** (PPT)
$(x; w)$		x
	$\xrightarrow{\pi_w^{\mathrm{L}}}$	
		$\mathbf{V}^{\pi_w^{\mathrm{L}}}(x; \rho)$ outputs 1 if accepted or 0 if rejected

Figure 4.5: A linear probabilistically checkable proof with common input x and purported witness w.

One major advantage of this strengthening is that it is relatively easy (when compared to PCPs) to construct an LPCP for NP-relations, as we shall see in Section 6.1. The simplicity of these constructions often results in proof lengths that tend to be much shorter in practical constructions. The cost of this advantage is that the cryptographic compilers for information-theoretic LPCPs rely on stronger cryptographic assumptions, namely on the existence of *linear-only encryption schemes* [30]. Linear-only encryption says that, given access to ciphertexts only, it is possible to *homomorphically*[1] derive encryptions

[1] Homomorphic encryption means that computations can be performed on encrypted data without decrypting the data first.

of linear combinations of the corresponding plaintexts, but of no other functions of the plaintexts.

Currently, the methods to compile LPCPs to cryptographic proofs require this heavy machinery (linear-only encryption). The constructions of linear-only encryption are based on fairly well-understood principles for cryptographic designs; however, they are heuristic in the sense that none offer a formal reduction of the linear-only property to more standard cryptographic assumptions. Therefore, the main contribution of this thesis is the ability to compile LPCPs using a much simpler assumption (e.g., the *discrete logarithm assumption*; see Assumption 1), which is the topic of Chapter 7 and Chapter 8.

Note that we can further strengthen the PCP model by allowing for higher-degree queries. That is, our queries could be s-variate polynomials q of some degree, to which the oracle responds with $q(\boldsymbol{\pi}_1, \ldots, \boldsymbol{\pi}_s)$, where $\boldsymbol{\pi} \in \mathbb{F}^s$ is the proof string. In Section 8.1, we will introduce *quadratic PCPs (QPCPs)* (where the queries have total degree at most 2). The QPCP queries are represented in purple in Figure 4.4.

4.4 Soundness analysis

Generally, there are two properties that need to hold for a proof system to be useful, namely *completeness* (all propositions from a predefined set of true statements have a proof), and *soundness* (a malicious prover only has a small probability of convincing the verifier that a false statement is true). Completeness is usually easy to prove, as it mostly involves tying together the different steps of the proof system and applying some calculations. We will see an example in Chapter 5 for which completeness is indeed easy to prove (Example 3). Soundness, however, is much harder to prove when only working with the definition. Therefore, *extractor analysis* has been introduced to be able to prove stronger statements that imply soundness, some of which are easier to address than soundness. In this section, we discuss three extractor analysis techniques, namely *proofs of knowledge* or *knowledge soundness* (which is stronger than statistical soundness), *special soundness* (which implies knowledge soundness) and *arguments of knowledge (computational soundness)*.

Statistical soundness: proof of knowledge

Statistical soundness is the definition of soundness we have used until this section: a proof system for a relation $\mathcal{R}$ is *statistically sound* if, for every $x \notin L_{\mathcal{R}}$ and every (possibly malicious and computationally unbounded) prover $\mathbf{P}'$, the probability that the verifier

accepts is bounded by some *soundness error*. Throughout this thesis, when we do not specify the type of soundness, we are referring to statistical soundness.

Proofs of knowledge give us a stronger condition that implies statistical soundness. A *proof of knowledge* is essentially a proof in which the prover convinces the verifier that they have *knowledge* of some property (i.e., the prover proves that they have a witness for some $x \in L_{\mathcal{R}}$, instead of just proving that such a witness exists). To formally define a proof of knowledge, we first need to define what it means to know something. In the context of a proof of knowledge, we say intuitively that a prover *knows* an object if that object can be outputted by an efficient algorithm that uses that prover as a black-box. Such an algorithm is called a *knowledge extractor* and must satisfy the following property[1]: if $\mathbf{P}'$ is a (possibly malicious and computationally unbounded) prover for which $\mathbf{V}$ accepts with high probability after interacting with $\mathbf{P}'$ on input x, then there exists a probabilistic *knowledge extractor* E with oracle access to $\mathbf{P}'$ that outputs a witness w for x in expected polynomial time[2] [24, 25].

Note that if the knowledge extractor is able to output a witness w for x, then indeed $x \in L_{\mathcal{R}}$ (otherwise such a witness would not exist). Hence, if no knowledge extractor exists and $x \notin L_{\mathcal{R}}$, then for every (possibly malicious and computationally unbounded) prover $\mathbf{P}'$, the verifier will only accept with small probability. In other words, a proof of knowledge (or *knowledge soundness*) implies statistical soundness.

Although designing a knowledge extractor is typically more difficult than just proving soundness, it is often desirable to prove knowledge soundness, especially in cases where the *existence* of a witness is obvious, but the *knowledge* of a witness is of interest. For example, suppose $H\colon \{0,1\}^{200} \to \{0,1\}^{100}$ is a *collision-resistant hash function* (i.e., a function where the probability is small of finding distinct x and y such that $H(x) = H(y)$). By the pigeonhole principle, there must exist a witness (x, y) (with $x \neq y$) such that $H(x) = H(y)$, because H is a compressing function. However, it is much more interesting for a prover to be able to assert that they have *knowledge* of a witness (x, y) (for which they probably needed to do some presumably difficult computation).

[1] Although in practice proofs of knowledge are mostly studied in the context of efficient provers (an unbounded prover may find a witness through an exhaustive search), we define proofs of knowledge for general provers, as the definition does not depend on the runtime of the prover.

[2] A probabilistic algorithm runs in *expected* polynomial time if the expectation of the runtime (taken over the randomness of the algorithm) is polynomial in the input size.

Knowledge soundness: special soundness

Similar to (but slightly simpler than) knowledge extractors is the concept of *special soundness*. A 3-move (public-coin) proof system for a relation $\mathcal{R}$ has the *special soundness* property if, given two distinct accepting proof transcripts with the same first message (m, c_1, z_1) and (m, c_2, z_2) for the same input x, with $c_1 \neq c_2$, there exists a PPT *extractor* E that outputs a witness w for x [16]. Note that we assume here that in the 3-message (or 3-move) proof system the prover first sends a message, after which the verifier responds with a challenge, to which the prover sends an answer. Such proof systems are called Σ-protocols and are discussed further in Section 5.3.

We see that for a proof system that has special soundness, if $x \notin L_{\mathcal{R}}$ (i.e., if there exists no witness for x), intuitively there exists at most one challenge c that the verifier can send to the prover while still being convinced that $x \in L_{\mathcal{R}}$. Suppose, towards contradiction, that there were multiple such challenges, then by special soundness there exists an extractor that outputs a witness w for x, but such a witness does not exist; we have a contradiction. Hence, special soundness implies statistical soundness with soundness error $\frac{1}{|\mathcal{C}|}$, where $\mathcal{C}$ is the (finite) set of possible challenges the verifier can send. Special soundness even implies knowledge soundness, which we introduced in the previous section. Note that it requires much work to formalize this intuition.

The implication from special soundness to knowledge (and statistical) soundness is extremely useful, as special soundness can be easier to prove than knowledge soundness: one has to define an extractor that extracts witnesses from two proof transcripts. This task can be much easier than proving (knowledge) soundness from the definition. In Chapter 5, we will discuss an example where it is indeed easy to prove special soundness (Example 3).

Now, special soundness can be generalized as follows. Let $\mu \in \mathbb{N}$ and $k_1, \ldots, k_\mu \in \mathbb{N}$ be given. A $(2\mu + 1)$-move public-coin proof system for a relation $\mathcal{R}$ has the $(k_1, \ldots, k_\mu)$-*special soundness* property if, given a tree of distinct accepting proof transcripts as shown in Figure 4.6, there exists a PPT *extractor* E that outputs a witness w for x [6]. Note that special soundness as defined above is a special case of $(k_1, \ldots, k_\mu)$-special soundness. Namely, special soundness is 2-special soundness.

As was the case for special soundness, $(k_1, \ldots, k_\mu)$-special soundness also implies statistical soundness with a certain soundness error (see Lemma 7.1.5) [6, Theorem 1].

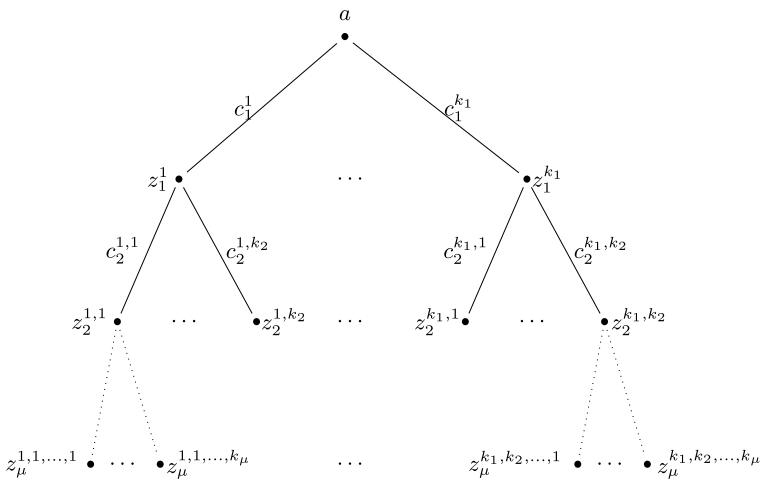

Figure 4.6: This tree represents distinct proof transcripts for the same input in a $(2\mu+1)$-move public-coin proof system. Namely, each path from the root to a leaf represents a single proof transcript, where each node on the path is a prover message, and each edge on the path is a verifier message. *Reprinted from Attema, Cramer, and Kohl [6].*

Computational soundness: argument of knowledge

We can relax the notion of statistical soundness by requiring only that it be *computationally unfeasible* for a prover to convince a verifier that a false statement is true (whereas for statistical soundness, we do not take the computational power of the prover into account). We call this relaxation *computational soundness* [24]. Thus, a proof system for a relation $\mathcal{R}$ is computationally sound if for every $x \notin L_{\mathcal{R}}$ *and every PPT prover* $\mathbf{P}'$, the probability that the verifier accepts after interacting with $\mathbf{P}'$ on input x is small. In this context, we replace the term *proof of knowledge* by *argument of knowledge* to highlight the fact that we require a less strong notion of soundness: essentially, arguments are computationally sound proofs. As we have weakened the prover, it can be easier to prove computational soundness than statistical soundness. Nonetheless, computational soundness is sufficient for many applications [24].

We can also define computational variants of knowledge and special soundness. *Computational knowledge soundness* says that if a possibly malicious *PPT* prover $\mathbf{P}'$ convinces the verifier $\mathbf{V}$ to accept on input x, then there exists a probabilistic knowledge extractor E with oracle access to $\mathbf{P}'$ that outputs a witness w for x in expected polynomial time. Similarly, *computational special soundness* says that if a possibly malicious *PPT* prover is able to construct two (or a tree of) valid proof transcripts for the same input x and the same first message, then there exists a PPT extractor E that outputs a witness w for x.

4.5 Fiat-Shamir transform: from interactive to non-interactive

Although interactive proof systems can increase power (e.g., Figure 4.2 presented an interactive protocol for testing graph non-isomorphism, which is a relation that is not believed to be in NP), interactivity can, however, be costly in terms of added communication complexity. Furthermore, interactivity can be problematic in terms of latency, as both the prover and the verifier need to be online during the entire proof. The *Fiat-Shamir Transform* [20] is a method that translates interactive proofs into non-interactive proofs, thereby reducing the communication complexity while preserving the power of interactivity.

Any interactive public-coin interactive proof can be translated into a non-interactive proof by applying the *Fiat-Shamir Transform*. Intuitively, this follows from the fact that in the public-coin setting, the prover can see the tape of random coin tosses of the verifier. Therefore, instead of receiving a challenge from the verifier, the prover can execute the computation that the verifier would have done to compute the challenge.

To understand the Fiat-Shamir transform, we need to define *cryptographic hash functions*.

Definition 4.5.1 ([43])**.** A *cryptographic hash function* is a function $H\colon \{0,1\}^* \to \{0,1\}^n$ for some $n \in \mathbb{N}$ that satisfies the following properties:

- *One-way.* Given an $x \in \{0,1\}^*$, it is easy to compute $H(x)$. However, given $H(x)$, it is hard to find x.
- *Collision-resistant.* It is hard to find distinct $x_1, x_2 \in \{0,1\}^*$ such that $H(x_1) = H(x_2)$.

Remark. The word 'hard' in the above definition could be made more precise, but for our discussion this informal definition suffices.

The idea is that a cryptographic hash function produces *hashes* for their inputs that look like random bit-strings. The properties of the Fiat-Shamir transform depend on the *Random Oracle Model (ROM)*, which generalizes collision-resistant hash functions. Currently, there is no consensus on the validity of the ROM.

Definition 4.5.2. In the *Random Oracle Model (ROM)*, all parties have access to a *perfect* hash function $H\colon \{0,1\}^* \to \{0,1\}^n$, meaning:

- for distinct $x_1, x_2 \in \{0,1\}^*$, $H(x_1)$ and $H(x_2)$ are independent and uniformly random;
- H is a function (i.e., it produces the same output for the same input);

- H is interpreted as a black-box random oracle (i.e., we have no knowledge of the internal state of H).

The interaction in an interactive proof consists of random challenges that the verifier poses to the prover, to which the prover gives convincing answers. The main idea of the Fiat-Shamir transform is to replace these random challenges by hashes of the prover's messages. That means the prover will act as if they receive a challenge c from the verifier, whereas in reality c is a hash of the prover's previous message(s). Any verifier can check the proof by calculating the hashes of the prover's messages and checking whether they agree with the challenges calculated by the prover. If the hash function is modeled as a random oracle, the transform will be sound and complete (assuming the interactive proof system is sound and complete). After the transform, all messages go in one direction (from prover to verifier), and thus can be sent as one large message. The Fiat-Shamir transform is shown schematically below, where Figure 4.7 represents the proof system before the transform, and Figure 4.8 represents the system after the transform.

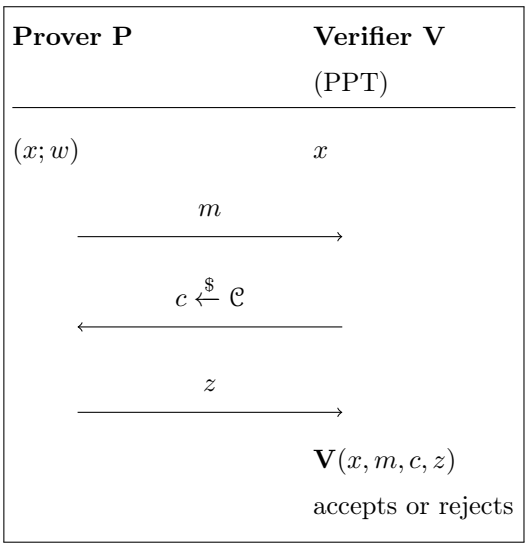

Figure 4.7: A simple interactive proof before the Fiat-Shamir transform. The verifier randomly samples some challenge c from a finite set $\mathcal{C}$.

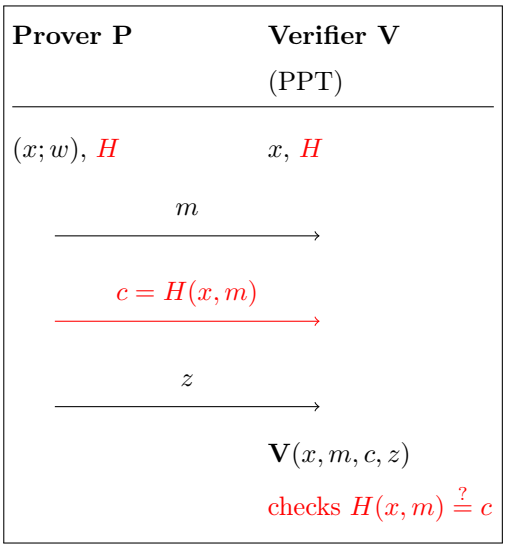

Figure 4.8: The result of applying the Fiat-Shamir transform to a simple interactive proof. Differences with the interactive proof are highlighted in red.

4.6 Commitment schemes

The overarching theme of this thesis is compiling information-theoretic proof systems into concrete cryptographic constructions. Such constructions are often centered around *commitment schemes*, which is the topic of this section. We follow *Commitment Schemes and Zero-Knowledge Protocols* by Ivan Damgård [17], and we focus on the famous *Pedersen commitment* [38].

Essentially, a commitment scheme allows a player P to commit to a secret value in such a way that this value cannot be changed at a later point in time. A toy-example of a commitment scheme is a game between players P and V, where P commits to a secret message by writing the message on a piece of paper, putting that paper in a box, and locking the box. The player P gives this locked box to V. Now, P cannot change the message they committed to, and if P wants, they can reveal this message to V by giving them the key of the lock. There are two important properties that make this game a commitment scheme:

- *binding:* once P has chosen a message and committed to it (by locking the box and giving it to V), P cannot change this value at a later point in time.
- *hiding:* when V receives the commitment (i.e., the locked box), V cannot know what the secret message is, unless P reveals it to V by giving V the key.

We now define these properties more formally.

Definition 4.6.1 (Commitment scheme)**.** Let $\mathcal{R}$ be a general set of randomness, let $\mathcal{M}$ be a message space, and let $\mathcal{C}$ be a commitment space. Let $\mathcal{G}$ be a PPT algorithm, called the *generator*, that outputs a function $\texttt{commit}\colon \mathcal{R} \times \mathcal{M} \to \mathcal{C}$. A *commitment scheme* $(\mathcal{R}, \mathcal{M}, \mathcal{C}, \mathcal{G}, P, V)$ is a game between PPT algorithms P and V consisting of three phases:

1. *set-up phase.* P runs $\mathcal{G}$ and sends a description of $\texttt{commit}$ to V (or vice versa);
2. *commit phase.* P decides to commit to $x \in \mathcal{M}$: they sample $r \xleftarrow{\$} \mathcal{R}$, and they send $C \coloneqq \texttt{commit}(r, x) \in \mathcal{C}$ to V;
3. *reveal phase.* P sends r and x to V, and V checks whether $C = \texttt{commit}(r, x)$.

The *binding* and *hiding* properties come in a *perfect* variant and in a *computational* variant. A commitment scheme is *computationally binding* if the probability is small that a PPT algorithm P is able to find two different openings that induce the same commitment, and the scheme is *perfectly binding* if this probability is zero, even for an unbounded algorithm P.

Similarly, a commitment scheme is *computationally hiding* if the probability is small for a PPT algorithm V to guess correctly what value is inside a commitment, and the scheme is *perfectly hiding* if this probability is zero, even for an unbounded algorithm V.

More formally, we define these properties as follows.

Definition 4.6.2 (Binding)**.** A commitment scheme $(\mathcal{R}, \mathcal{M}, \mathcal{C}, \mathcal{G}, P, V)$ is:

- *perfectly binding* if, for all $r, r' \in \mathcal{R}$ and all $x, x' \in \mathcal{M}$, we have

$$\texttt{commit}(r, x) = \texttt{commit}(r', x') \implies x = x'.$$

- *computationally binding* if, for all PPT algorithms P^* that take as input a function $\texttt{commit} \leftarrow \mathcal{G}$, $\varepsilon(\lambda)$ is negligible in λ, where $\varepsilon(\lambda)$ is the probability that P^* outputs $r, r' \in \mathcal{R}$, $x, x' \in \mathcal{M}$, and $C \in \mathcal{C}$, such that $\texttt{commit}(r, x) = C = \texttt{commit}(r', x')$ and $x \neq x'$.

Definition 4.6.3 (Hiding)**.** A commitment scheme $(\mathcal{R}, \mathcal{M}, \mathcal{C}, \mathcal{G}, P, V)$ is:

- *perfectly hiding* if, for all $x, x' \in \mathcal{M}$, the probability distributions of $\texttt{commit}(r, x)$ and $\texttt{commit}(r', x')$ (taken over r and r') are identically distributed.
- *computationally hiding* if, for all $x, x' \in \mathcal{M}$, the probability distributions of $\texttt{commit}(r, x)$ and $\texttt{commit}(r', x')$ (taken over r and r') are computationally indistinguishable (Definition 3.3.2).

Pedersen commitment

We now define the *Pedersen commitment*, which is computationally binding (under the discrete logarithm assumption) and perfectly hiding [38].

Definition 4.6.4 (Pedersen commitment)**.** Let $\mathbb{G}$ be a cyclic Abelian group of prime order, such that the discrete logarithm assumption (Assumption 1) holds. Given two generators $g, h \in \mathbb{G}$, the *Pedersen commitment* works as follows. The committer wants to commit to some $x \in \mathbb{Z}_q$. They sample $\gamma \xleftarrow{\$} \mathbb{Z}_q$, and they publish the commitment $g^x h^\gamma$.

A slightly different Pedersen commitment, which we will use in Chapter 7 and Chapter 8, allows for the commitment of multiple messages $x_1, \ldots, x_n \in \mathbb{Z}_q$ (or a vector $\mathbf{x} \in \mathbb{Z}_q^n$) by using $n + 1$ generators $g_1, \ldots, g_n, h \in \mathbb{G}$, sampling a $\gamma \xleftarrow{\$} \mathbb{Z}_q$, and committing to $g_1^{x_1} \cdots g_n^{x_n} h^\gamma$. This variation of the Pedersen commitment is also computationally binding (under the discrete logarithm assumption) and perfectly hiding [5].

5

Zero-knowledge

Fascinatingly, proof systems exist that do not leak more information about a proposition than the fact that it is true. Such proofs are called *zero-knowledge proofs*. They have numerous applications. For example, zero-knowledge proofs can allow inspectors to confirm whether an object is a nuclear weapon, without gaining knowledge about the internal workings of the possible weapon [22]. This application is particularly useful for nuclear disarmament. Zero-knowledge proofs are also desirable in cryptographic settings. They can be used for authentication schemes, where users prove their identity without revealing any information [21]. Finally, zero-knowledge proofs can be applied to blockchains: for example, a user may wish to prove they have sufficient funds for a purchase, without revealing their account balance [15, 36].

However, the concept of a zero-knowledge proof feels somehow paradoxical: how can we prove a statement without revealing any information? The following toy-example, inspired by the *Zero-Knowledge Proofs MOOC* [13], gives a sense of how zero-knowledge proofs could be possible.

Example 2. Suppose Paul has two candies and wants to prove to Vanessa that the candies have different colors, without revealing the colors of those candies. Paul gets two non-transparent cups, hides each candy under one of the cups, and puts the cups in front of Vanessa. Paul looks away, after which Vanessa tosses a random coin. If the coin lands on heads, she switches the cups. Otherwise, she leaves the cups as they are. After she is finished, Paul will take a peak under the cups and tell Vanessa whether she switched the cups or not.

We see that if the two candies indeed have different colors, Paul will know with certainty whether Vanessa switched the cups. If, however, the candies have the same color, Paul has a probability of $\frac{1}{2}$ of guessing correctly whether the cups were switched. If Paul and

Vanessa repeat this game k times, the probability that Paul guesses correctly each time drops to $\frac{1}{2^k}$.

This example shows that Vanessa can be convinced that Paul indeed has two different colored candies, without learning the colors of the candies. Furthermore, Paul can trick Vanessa with only a small probability as the number of rounds increases.

The aim of this chapter is to formally define zero-knowledge proofs. We define *perfect zero-knowledge* in Section 5.1, we define *non-interactive zero-knowledge proofs* in Section 5.2, and we define Σ-*protocols*, which is a special class of zero-knowledge proofs consisting of three moves, in Section 5.3.

5.1 Perfect zero-knowledge

To formally define zero-knowledge proofs, we need a definition of what it means for the verifier to learn nothing beyond the fact that the assertion is true. We follow the *simulation paradigm* as described in Goldreich's book *Foundations of Cryptography* [24, Chapter 4]. Intuitively, the verifier learns nothing if what they can compute before the interaction is the same as what they can compute after the interaction. This is certainly true if the verifier can simulate each possible proof transcript without interacting with the prover: anything they could compute after the interaction can be computed without talking to the prover by generating a transcript using a *simulator*.

We model the simulation by comparing two random variables, namely the *proof transcript random variable* (depending on the common input and witness), and a *simulator random variable* (depending only on the common input). If these random variables are *close* to each other, the proof system is *zero-knowledge*. There are multiple ways to define this closeness, which give rise to different definitions of zero-knowledge. For this thesis, it suffices to only define *perfect zero-knowledge*, where we require the two random variables to be *identically distributed*. In the literature, we also find *statistical zero-knowledge*, where the distributions of the two random variables are required to be *statistically close* (i.e., the distributions are indistinguishable for any algorithm), and *computational zero-knowledge*, where the distributions are required to be *computationally indistinguishable* (i.e., the distributions are indistinguishable for any PPT algorithm).

We define the *proof transcript* and *simulator* random variables that need to be compared when analyzing zero-knowledge.

Definition 5.1.1. Let **P** and **V** be a prover and verifier in a proof system.

- The *proof transcript* $\Pi\langle \mathbf{P} \leftrightarrow \mathbf{V} \rangle(x; w)$ is a random variable over proof transcripts that depends on the common input x, a witness w, and the internal random coin tosses of $\mathbf{P}$ and $\mathbf{V}$.
- The *simulator* or *simulated transcript* $\texttt{Sim}(x)$ is a random variable that only depends on the common input x, and the internal random coin tosses of $\mathbf{V}$.

Note that the sample space of these random variables consists of possible transcripts of the proof system (including the outcome of the coin tosses).

Formally, perfect zero-knowledge for a verifier that adheres to the underlying proof system is defined as follows.

Definition 5.1.2 (Perfect honest-verifier zero-knowledge [24])**.** A prover in a proof system $\langle \mathbf{P} \leftrightarrow \mathbf{V} \rangle$ for a relation $\mathcal{R}$ is *perfect honest-verifier zero-knowledge* if there exists a PPT simulator $\texttt{Sim}$ such that for each $(x; w) \in \mathcal{R}$, we have

$$\Pi\langle \mathbf{P} \leftrightarrow \mathbf{V} \rangle(x; w) \equiv \texttt{Sim}(x).$$

The definition above assumes that the verifier is honest (i.e., that the verifier adheres to the rules of the proof system). We can generalize this definition to include all PPT verifier strategies as follows.

Definition 5.1.3 (Perfect malicious-verifier zero-knowledge [24])**.** A prover in a proof system $\langle \mathbf{P} \leftrightarrow \mathbf{V} \rangle$ for a relation $\mathcal{R}$ is *perfect malicious-verifier zero-knowledge* if for every PPT verifier strategy $\mathbf{V}'$ and $(x; w) \in \mathcal{R}$, there exists a PPT simulator $\texttt{Sim}$ such that

$$\Pi\langle \mathbf{P} \leftrightarrow \mathbf{V}' \rangle(x; w) \equiv \texttt{Sim}(x, \mathbf{V}').$$

5.2 Non-interactive perfect zero-knowledge

In this section, we give a definition of a special type of zero-knowledge proofs, namely *non-interactive* zero-knowledge proofs. We follow Goldreich's survey on zero-knowledge proofs [23], although non-interactive zero-knowledge was originally introduced by Blum, Feldman, and Micali [11]. We follow the so-called *common reference string (CRS) model*, which assumes the existence of a trusted third-party that provides a reference string following a publicly known distribution to the prover and verifier.

Definition 5.2.1 (Non-interactive zero-knowledge)**.** A *non-interactive zero-knowledge proof* consists of a prover strategy $\mathbf{P}$, a verifier strategy $\mathbf{V}$, and a reference string σ (provided by a trusted third-party). $\mathbf{P}$ and $\mathbf{V}$ can read σ and do additional random coin tosses. $\mathbf{P}$ can send one message m to $\mathbf{V}$, after which $\mathbf{V}$ either accepts or rejects the proof. Such a model is *zero-knowledge* if the following two random variables are identically distributed:

- *Proof transcript.* (m, σ) is a random variable consisting of the proof message and the reference string.

- *Simulator.* The simulator $\texttt{Sim}(\sigma)$ is a random variable that only depends on the reference string σ.

It turns out that by sharing a common, short, random string, any interactive zero-knowledge proof can be replaced by a non-interactive zero-knowledge proof [11]. Furthermore, the Fiat-Shamir transform can be proved secure in the Random Oracle Model [39]. Specifically, the Fiat-Shamir transform preserves zero-knowledge (i.e., if the original proof system is zero-knowledge, then the transform will also be zero-knowledge), assuming the hash function does not leak any information.

5.3 Σ-Protocols

We often encounter interactive zero-knowledge proofs consisting of three messages. Therefore, such proofs are given a special name, namely Σ-protocols. Σ-protocols are a special class of interactive zero-knowledge proofs, formally defined as follows.

Definition 5.3.1 (Σ-protocol [29])**.** Let $\mathcal{R}$ be a relation such that each $x \in L_{\mathcal{R}}$ has a witness w which has length at most $\text{poly}(|x|)$. Let $\mathcal{C}$ be a finite set, called the *challenge space*. Let $x \in L_{\mathcal{R}}$ and let w be a polynomial length witness for x. A *Σ-protocol* for $\mathcal{R}$ is a 3-move interactive proof between a prover $\mathbf{P}$ and a verifier $\mathbf{V}$, both executing PPT strategies, where the prover is given $(x; w)$ as input and the verifier is given x, satisfying the following conditions:

- *Three moves.* There are three moves with the following structure. Firstly, $\mathbf{P}$ sends a *commitment* message m to $\mathbf{V}$. Secondly, $\mathbf{V}$ responds with a uniformly randomly chosen *challenge* $c \xleftarrow{\$} \mathcal{C}$. Thirdly, $\mathbf{P}$ sends an *opening* message z to $\mathbf{V}$. The verifier either accepts or rejects (based on a *deterministic* computation on x and the proof transcript (m, c, z)).

- *Completeness.* For every common input $x \in L_{\mathcal{R}}$, there exist PPT prover and verifier strategies $\mathbf{P}$ and $\mathbf{V}$ such that the verifier $\mathbf{V}$ accepts after executing the 3-move protocol with $\mathbf{P}$ on input x.

- *Special soundness.* There exists a probabilistic polynomial-time algorithm, called an *extractor*, that outputs a witness w for an $x \in L_{\mathcal{R}}$, when given two accepting transcripts (m, c_1, z_1) and (m, c_2, z_2) for x, with $c_1 \neq c_2$.

- *Perfect honest-verifier zero-knowledge.* There exists a PPT simulator $\texttt{Sim}$ that outputs a transcript (m, c, z) for an input $x \in L_{\mathcal{R}}$ that is identically distributed to the proof transcript for honest players executing the 3-move protocol on input x. In other words, $\Pi\langle \mathbf{P} \leftrightarrow \mathbf{V} \rangle(x; w) \equiv \texttt{Sim}(x)$.

We secretly already introduced Σ-protocols in Figure 4.7. Recall from Section 4.4 that special soundness implies statistical soundness with soundness error $\frac{1}{|\mathcal{C}|}$ [16]. With special soundness, it becomes easier to design and analyze Σ-protocols, as special soundness can be easier to prove than (statistical) soundness. Typically, the most challenging part for Σ-protocols is formally proving (honest-verifier) zero-knowledge.

Example Σ-protocol: Schnorr's identification scheme

We give an example of a Σ-protocol for a *discrete logarithm* in a cyclic group $\mathbb{G}$.

Example 3. Let $\mathbb{G}$ be a cyclic Abelian group of prime order q for which the discrete logarithm assumption holds. The common input consists of a generator $g \in \mathbb{G}$, and a group element $x = g^w \in \mathbb{G}$. Paul wants to prove to Vanessa that he knows $w \in \mathbb{Z}_q$ without revealing w. He does this through the following interactive zero-knowledge proof:

1. Paul picks a $v \xleftarrow{\$} \mathbb{Z}_q$ uniformly at random and sends $m = g^v$ to Vanessa.
2. Vanessa picks a challenge $c \xleftarrow{\$} \mathbb{Z}_q$ uniformly at random and sends c to Paul.
3. Paul calculates $z = v - cw$ and sends z to Vanessa.
4. Vanessa checks whether $m = g^z x^c$. She accepts if this is the case and rejects otherwise.

Completeness follows from the fact that

$$g^z x^c = g^z g^{wc} = g^{v-cw+wc} = g^v = m.$$

Soundness follows from special soundness: given two accepting proof transcripts $(m = g^v, c_1, z_1 = v - c_1 w)$ and $(m = g^v, c_2, z_2 = v - c_2 w)$ (with $c_1 \neq c_2$), we can extract the witness w by calculating

$$\frac{z_2 - z_1}{c_1 - c_2} = \frac{-c_2 w + c_1 w}{c_1 - c_2} = w.$$

This shows that special soundness holds, and since the challenge space $\mathcal{C}$ in this case is $\mathbb{Z}_q$, this proof system is statistically sound with soundness error $\frac{1}{|\mathbb{Z}_q|} = \frac{1}{q}$.

The proof system is zero-knowledge, as Paul does not reveal any information about w. We can construct a simple simulator algorithm $\texttt{Sim}$: we start by selecting a z and c uniformly at random ($c \xleftarrow{\$} \mathbb{Z}_q, z \xleftarrow{\$} \mathbb{Z}_q$), after which we set $m = g^z x^c$. Then, (m, c, z) is

a valid proof transcript, as $m = g^{z+cw} = g^v$ with $v := z + cw$. Note that by the discrete logarithm assumption (Assumption 1), the input $x = g^w$ reveals nothing about w.

Now, we can convert this into a non-interactive zero-knowledge proof by applying the Fiat-Shamir transform. The common input still contains the generator $g \in \mathbb{G}$ and a group element $x = g^w \in \mathbb{G}$, but here it also contains a perfect cryptographic hash function[1] H. Again, Paul wants to prove that he knows w. Paul executes the following protocol; we highlight the differences with the interactive protocol in red.

1. Paul picks a $v \xleftarrow{\$} \mathbb{Z}_q$ uniformly at random and calculates $m = g^v$.
2. Paul calculates $c = H(g, x, m)$, which looks like a random element in $\mathbb{Z}_q$ by the properties of a perfect cryptographic hash function.
3. Paul calculates $z = v - cw$ and sends (m, z) to Vanessa.
4. Vanessa calculates $c = H(g, x, m)$ and checks whether $m = g^z x^c$. She accepts if this is the case and rejects otherwise.

We see that the interaction is removed by letting Paul create his own challenge. Both versions of Schnorr's identification scheme are presented in Figure 5.1 and Figure 5.2.

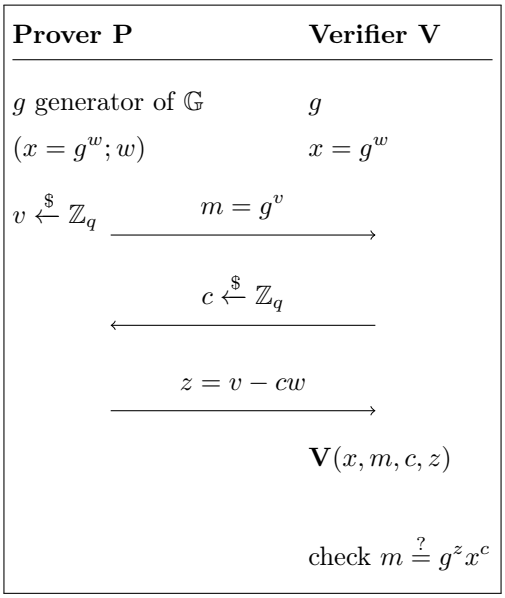

Figure 5.1: Schnorr's identification scheme before the Fiat-Shamir transform.

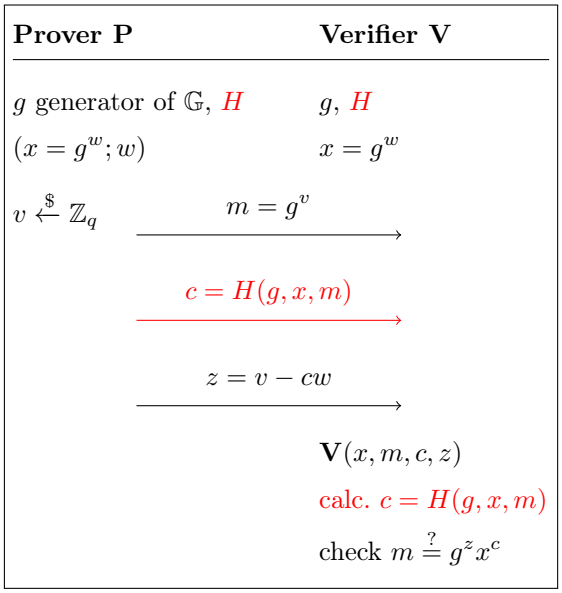

Figure 5.2: Schnorr's identification scheme after the Fiat-Shamir transform.

[1] Although we defined cryptographic hash functions in terms of bit-strings, here we identify $\mathbb{Z}_q$ with any binary representation of it.

6

Hadamard LPCP and state-of-the-art compiler

The main purpose of this thesis is to cryptographically compile LPCPs, without relying on heavy cryptographic machinery, so that LPCPs may be used in practice. In this chapter, we study the best current cryptographic compiler for LPCPs, to which we will compare our new compilers. We first start by introducing an LPCP for NP-relations in Section 6.1, the *Hadamard LPCP*, which will serve as a concrete example to apply the compilers to and compare them. Then, in Section 6.2, we introduce *linear-only encryption* which the state-of-the-art cryptographic compiler for LPCPs relies on. Although the constructions of linear-only encryption are based on fairly well-understood principles for cryptographic designs, none of the constructions currently offer a formal reduction of the linear-only property to more standard cryptographic assumptions. Therefore, we present our main contributions in Chapter 7 and Chapter 8: namely two new cryptographic compilers that compile LPCPs using much simpler assumptions. We thoroughly compare the resulting security parameters and communication overhead in Chapter 9.

6.1 LPCP for NP-relations: the Hadamard LPCP

In this section, we introduce the *Hadamard LPCP*, which is an LPCP that proves membership of NP-relations (see Figure 6.2). We closely follow the blog post *Zero-Knowledge Proofs from Information-Theoretic Proof Systems* by Yuval Ishai [30]. The Hadamard LPCP is a natural adaptation of the *Hadamard PCP*, which is implicit but important in the proof of the PCP theorem [3], although the Hadamard PCP was probably known to the community prior to the PCP theorem. In the PCP version, the proof length is ex-

ponential in the input, whereas the linear queries in the LPCP allow for shorter proofs, as we shall see below. The name stems from the *Hadamard code*, which is a very famous error-correcting code that inspired the Hadamard PCP construction.

An NP-relation $\mathcal{R}$ (for a fixed statement length n) can be written as a polynomial-sized arithmetic circuit C over a finite field $\mathbb{F}$ (we defined circuits in Section 3.2). The arithmetic circuit essentially describes the polynomial-time NP-verifier of the relation, which takes as input a statement $\mathbf{x} \in \mathbb{F}^n$ and a purported witness $\mathbf{w} \in \mathbb{F}^m$. The circuit outputs zero if and only if the verifier accepts on input $(\mathbf{x}; \mathbf{w})$. Since $\mathbf{x}$ is public, we fill in the values of $\mathbf{x}$ in the circuit and consider the resulting arithmetic circuit $C_{\mathbf{x}}$ that takes a $\mathbf{w} \in \mathbb{F}^m$ as input. The Hadamard LPCP then consists of a prover and verifier who are given the common input $C_{\mathbf{x}}$. The prover wants to show that $\exists \mathbf{w} \in \mathbb{F}^m$ such that $C_{\mathbf{x}}(\mathbf{w}) = 0$ (i.e., that $(\mathbf{x}; \mathbf{w}) \in \mathcal{R}$). This circuit satisfiability problem captures natural questions. For example, $\mathbf{x}$ could be the value of a payment the prover wants to make, and $\mathbf{w}$ the amount of money the prover has in their bank account; the circuit checks whether $\mathbf{x} \leq \mathbf{w}$ (in which case $C_{\mathbf{x}}(\mathbf{w}) = 0$).

First, the common input $C_{\mathbf{x}}$ is converted to a collection of multivariate quadratic polynomials (which are then known to both prover and verifier). Note that $C_{\mathbf{x}}$ only takes m inputs, but contains n pre-filled values that coincide with the common input $\mathbf{x}$ (see Figure 6.1 for an example). If $s \in \mathbb{N}$ is the number of gates of C (including the input gates), then this conversion produces s quadratic polynomials $Q_1(Y_1, \ldots, Y_s), \ldots, Q_s(Y_1, \ldots, Y_s)$ in such a way that they are all equal to zero if and only if $\exists \mathbf{w} \in \mathbb{F}^m$ such that the output of the circuit is $C_{\mathbf{x}}(\mathbf{w}) = 0$.

Definition 6.1.1 (Conversion of arithmetic circuit to quadratic polynomials)**.** Let $C_{\mathbf{x}}$ be an arithmetic circuit of size s over a finite field $\mathbb{F}$, where the common input $\mathbf{x} \in \mathbb{F}^n$ is pre-filled, and which takes as input a $\mathbf{w} \in \mathbb{F}^m$ (i.e., $s > m + n$). The conversion of $C_{\mathbf{x}}$ produces s quadratic polynomials in s variables. Each gate of $C_{\mathbf{x}}$ corresponds to a single variable Y_i.

The variables $Y_1, \ldots, Y_m$ correspond to the unknown input gates (where $\mathbf{w}$ is filled in), the variables $Y_{m+1}, \ldots, Y_{m+n}$ correspond to the internal gates where the common input $\mathbf{x} \in \mathbb{F}^n$ is filled in, the variable Y_s corresponds to the output gate, and all other variables correspond to multiplication and addition gates.

For each variable (except for the input variables $Y_1, \ldots, Y_m$), we get one polynomial describing the corresponding gate. For the output gate, we have an additional polynomial describing the output value. Finally, for notational purposes, we add $m - 1$ dummy polynomials that all equal zero, so that the number of variables coincides with the number of polynomials. The details of the conversion are as follows:

- For each common input $\mathbf{x}_i$, where $i \in \{1, \ldots, n\}$ (i.e., $Y_{m+i} = \mathbf{x}_i$), we set $Q_i(Y_1, \ldots, Y_s) = Y_{m+i} - \mathbf{x}_i$.
- For each multiplication gate i (i.e., $Y_i = Y_j Y_k$), we set $Q_{i-m}(Y_1, \ldots, Y_s) = Y_i - Y_j Y_k$. Note that $i > m$, as $Y_1, \ldots, Y_m$ correspond to the unknown input gates, and thus cannot be a multiplication or addition gate.
- For each addition gate $i > m$ (i.e., $Y_i = Y_j + Y_k$), we set $Q_{i-m}(Y_1, \ldots, Y_s) = Y_i - (Y_j + Y_k)$.
- For the output gate, we set $Q_{s+1-m}(Y_1, \ldots, Y_s) = Y_s$.
- We define $m-1$ dummy polynomials $Q_{s+2-m}(Y_1, \ldots, Y_s) = \ldots = Q_s(Y_1, \ldots, Y_s) = 0$.

By construction, all polynomials are zero if and only if $\exists \mathbf{w} \in \mathbb{F}^m$ such that $C_{\mathbf{x}}(\mathbf{w}) = 0$. Note that for the output gate Y_s we have two polynomials, namely $Q_{s+1-m}(Y_1, \ldots, Y_s) = Y_s$ and the polynomial that depends on the type of the output gate (addition or multiplication). Below, we give an example of converting the arithmetic circuit in Figure 6.1 to quadratic polynomials.

Example 4. Given the arithmetic circuit in Figure 6.1. We have $s = 9$ gates, of which $m = 2$ gates are input gates (indicated in red). Applying the conversion as described in Definition 6.1.1 gives us the following 9 quadratic polynomials:

- *Common input* $\mathbf{x} = (-3, 6)$. $Q_1(Y_1, \ldots, Y_9) = Y_3 + 3$ and $Q_2(Y_1, \ldots, Y_9) = Y_4 - 6$.
- *Multiplication.* $Q_4(Y_1, \ldots, Y_9) = Y_6 - Y_2 Y_3$, $Q_5(Y_1, \ldots, Y_9) = Y_7 - Y_3 Y_4$, and $Q_7(Y_1, \ldots, Y_9) = Y_9 - Y_7 Y_8$.
- *Addition.* $Q_3(Y_1, \ldots, Y_9) = Y_5 - (Y_1 + Y_2)$ and $Q_6(Y_1, \ldots, Y_9) = Y_8 - (Y_5 + Y_6)$.
- *Output.* $Q_8(Y_1, \ldots, Y_9) = Y_9$.
- *Dummy.* $Q_9(Y_1, \ldots, Y_9) = 0$.

The variables in red correspond to the unknown inputs.

The prover now needs to convince the verifier that there exists a $\mathbf{y} \in \mathbb{F}^s$ such that $Q_1(\mathbf{y}_1, \ldots, \mathbf{y}_s) = \ldots = Q_s(\mathbf{y}_1, \ldots, \mathbf{y}_s) = 0$. By design, this $\mathbf{y}$ simply consists of the values of each gate of $C_{\mathbf{x}}(\mathbf{w})$. Therefore, the prover first computes the values $\mathbf{y} \in \mathbb{F}^s$ of all gates in $C_{\mathbf{x}}(\mathbf{w})$. Since an LPCP only supports linear queries, the prover needs to construct a

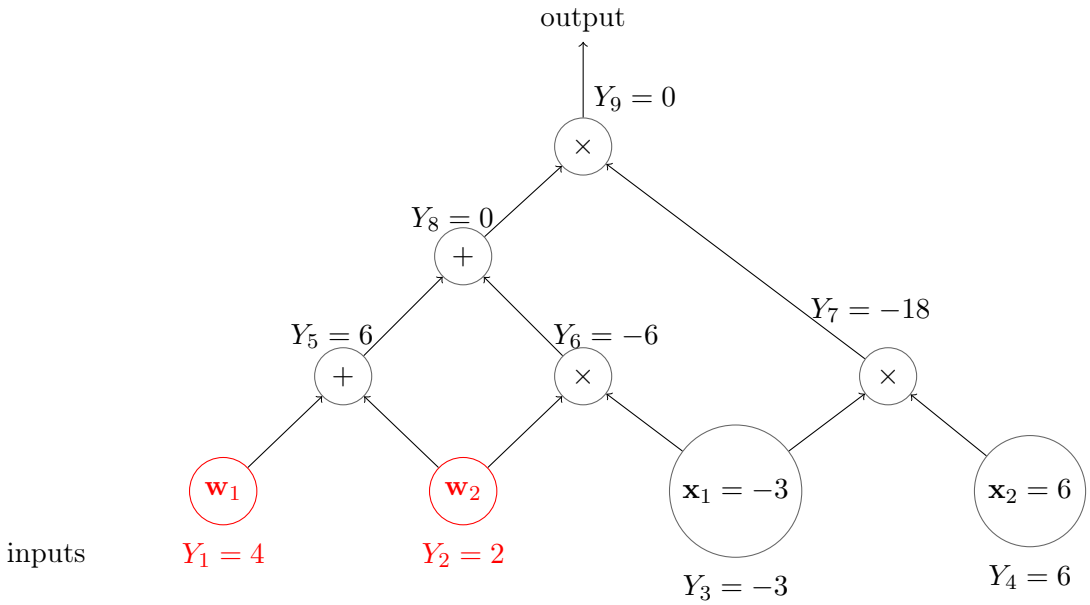

Figure 6.1: An arithmetic circuit with input $\mathbf{w}$ for the (pre-filled) common input $\mathbf{x} = (-3, 6)$, representing the formula $(\mathbf{w}_1 + \mathbf{w}_2 + \mathbf{w}_2\mathbf{x}_1)\mathbf{x}_1\mathbf{x}_2 = -18(\mathbf{w}_1 + \mathbf{w}_2 - 3\mathbf{w}_2)$. Given inputs $\mathbf{w}_1 = 4$ and $\mathbf{w}_2 = 2$, the circuit outputs 0.

proof string that allows verifying $\mathbf{y}$ by only using linear queries. To this end, the prover computes[1]

$$\mathbf{y} \,(\times)\, \mathbf{y} \coloneqq (\mathbf{y}_i^2)_{1 \le i \le s} \| (\mathbf{y}_i \mathbf{y}_j)_{1 \le i < j \le s} = (\mathbf{y}_i \mathbf{y}_j)_{1 \le i \le j \le s} \in \mathbb{F}^{\binom{s+1}{2}},$$

where $\|$ denotes vector concatenation. Note that our newly defined unary[2] operator $(\times)$ resembles the tensor product $\otimes$ (recall that $\mathbf{y} \otimes \mathbf{y} = (\mathbf{y}_i \mathbf{y}_j)_{i,j \in [s]}$), where the only difference is that $(\times)$ includes only one of $\mathbf{y}_i \mathbf{y}_j$ and $\mathbf{y}_j \mathbf{y}_i$ in its entries, whereas $\otimes$ includes both. Given $\widehat{\mathbf{y}} \coloneqq \mathbf{y} \,(\times)\, \mathbf{y}$, the prover constructs the proof string $\boldsymbol{\pi} \coloneqq (\mathbf{y}, \widehat{\mathbf{y}}) \in \mathbb{F}^{s + \binom{s+1}{2}}$.

Now, the verifier needs to check two facts by querying $\boldsymbol{\pi}$:

1. *consistency of proof string:* whether the $\widehat{\mathbf{y}}$ part of $\boldsymbol{\pi}$ actually equals $\mathbf{y} \,(\times)\, \mathbf{y}$.
2. *polynomials equal zero:* whether $Q_1(\mathbf{y}_1, \ldots, \mathbf{y}_s) = \ldots = Q_s(\mathbf{y}_1, \ldots, \mathbf{y}_s) = 0$.

The verifier starts by selecting $\mathbf{r} \xleftarrow{\$} \mathbb{F}^s$. Let $\mathbf{r}' \in \mathbb{F}^{s + \binom{s+1}{2}}$ be the vector that is obtained by appending $\binom{s+1}{2}$ zeros to $\mathbf{r}$. In other words, $\mathbf{r}' \coloneqq (\mathbf{r}_1, \ldots, \mathbf{r}_s, 0, \ldots, 0) \in \mathbb{F}^{s + \binom{s+1}{2}}$.

[1] The intuition behind $\mathbf{y} \,(\times)\, \mathbf{y}$ is that its entries consist of each *unique* (quadratic) combination of entries of $\mathbf{y}$ (i.e., $\mathbf{y}_i \mathbf{y}_j$ for $i, j \in [s]$ with $i \le j$). We have s choices for $\mathbf{y}_i \mathbf{y}_i$, and $\binom{s}{2}$ choices for $\mathbf{y}_i \mathbf{y}_j$ $(i \ne j)$. Hence, by Lemma 3.3.4, $\mathbf{y} \,(\times)\, \mathbf{y}$ has $s + \binom{s}{2} = \binom{s+1}{2}$ entries.

[2] Throughout this thesis, we will only use the operator $(\times)$ as a unary operator. That is, we define $\mathbf{y} \,(\times)\, \mathbf{y}$, but not $\mathbf{x} \,(\times)\, \mathbf{y}$ for $\mathbf{x} \ne \mathbf{y}$.

Similarly, let $\widehat{\mathbf{r}}' = (0,\ldots,0)\|\widehat{\mathbf{r}} \in \mathbb{F}^{s+\binom{s+1}{2}}$ be the vector that is obtained by prepending s zeros to $\widehat{\mathbf{r}} \coloneqq (\mathbf{r}_i^2)_{1\le i\le s}\|(2\mathbf{r}_i\mathbf{r}_j)_{1\le i<j\le s} \in \mathbb{F}^{\binom{s+1}{2}}$. For the first fact, the verifier does two linear queries to $\boldsymbol{\pi}$:

$$\text{Query 1: } a_1 \coloneqq \langle \boldsymbol{\pi}, \mathbf{r}' \rangle = \langle \mathbf{y}, \mathbf{r} \rangle \in \mathbb{F},$$

$$\text{Query 2: } a_2 \coloneqq \langle \boldsymbol{\pi}, \widehat{\mathbf{r}}' \rangle = \sum_{i=1}^{s} \widehat{\mathbf{y}}_{i,i}\mathbf{r}_i^2 + \sum_{1\le i<j\le s} \widehat{\mathbf{y}}_{i,j}(2\mathbf{r}_i\mathbf{r}_j) \in \mathbb{F},$$

$$\textbf{Check } a_2 = a_1^2 \text{ (consistency of } \mathbf{y} \text{ and } \widehat{\mathbf{y}}).$$

For the second fact, the verifier wants to query $a_3 \coloneqq \sum_{i=1}^{s} \mathbf{r}_i Q_i(\mathbf{y}_1,\ldots,\mathbf{y}_s)$ and check that $a_3 = 0$. However, the verifier can only query linear combinations of entries of $\boldsymbol{\pi}$. Therefore, we need to prove that we can write a_3 as a linear combination of entries of $\mathbf{y}$ and $\widehat{\mathbf{y}}$.

Lemma 6.1.2. *Given an arithmetic circuit $C_{\mathbf{x}}$ over a finite field $\mathbb{F}$ (where $\mathbf{x} \in \mathbb{F}^n$ is common input), an input $\mathbf{w} \in \mathbb{F}^m$, vectors $\mathbf{r}, \mathbf{y} \in \mathbb{F}^s$ (and $\widehat{\mathbf{y}} \coloneqq \mathbf{y}\,(\times)\,\mathbf{y}$), and the polynomials $Q_1,\ldots,Q_s$ as defined in Definition 6.1.1, we can obtain $a_3 \coloneqq \sum_{i=1}^{s} \mathbf{r}_i Q_i(\mathbf{y}_1,\ldots,\mathbf{y}_s)$ from $\mathbf{r}$, $\mathbf{x}$, and an inner product between $\boldsymbol{\pi} = (\mathbf{y}, \widehat{\mathbf{y}})$ and some $\mathbf{q} \in \mathbb{F}^{s+\binom{s+1}{2}}$.*

Proof. Since each polynomial $Q_1,\ldots,Q_s$ is a quadratic polynomial, we can write each as

$$Q_\ell(Y_1,\ldots,Y_s) = c^{(\ell)} + \sum_{j=1}^{s} \beta_j^{(\ell)} Y_j + \sum_{1\le k\le m\le s} \alpha_{k,m}^{(\ell)} Y_k Y_m,$$

for carefully chosen $c^{(\ell)}, \beta_1^{(\ell)},\ldots,\beta_s^{(\ell)}, \alpha_{1,1}^{(\ell)},\ldots,\alpha_{s,s}^{(\ell)} \in \mathbb{F}$. Note that, by design, only the polynomials corresponding to the common input gates contain constant terms. Hence, $c^{(\ell)} = 0$ for $\ell > n$, and $c^{(\ell)} = -\mathbf{x}_\ell$ for $\ell \le n$ (see Definition 6.1.1). Now,

$$\begin{aligned}
a_3 = \sum_{i=1}^{s} \mathbf{r}_i Q_i(\mathbf{y}_1,\ldots,\mathbf{y}_s) &= \sum_{i=1}^{s} \mathbf{r}_i \left(c^{(i)} + \sum_{j=1}^{s} \beta_j^{(i)} \mathbf{y}_j + \sum_{1\le k\le m\le s} \alpha_{k,m}^{(i)} \mathbf{y}_k \mathbf{y}_m \right) \\
&= \sum_{i=1}^{s} \mathbf{r}_i c^{(i)} + \sum_{i=1}^{s}\sum_{j=1}^{s} \mathbf{r}_i \beta_j^{(i)} \mathbf{y}_j + \sum_{i=1}^{s} \sum_{1\le k\le m\le s} \mathbf{r}_i \alpha_{k,m}^{(i)} \mathbf{y}_k \mathbf{y}_m \\
&= -\sum_{i=1}^{n} \mathbf{r}_i \mathbf{x}_i + \sum_{j=1}^{s} \mathbf{y}_j \left(\sum_{i=1}^{s} \beta_j^{(i)} \mathbf{r}_i \right) + \sum_{1\le k\le m\le s} \mathbf{y}_k \mathbf{y}_m \left(\sum_{i=1}^{s} \alpha_{k,m}^{(i)} \mathbf{r}_i \right).
\end{aligned}$$

Thus, we can obtain $a_3 = \sum_{i=1}^{s} \mathbf{r}_i Q_i(\mathbf{y}_1,\ldots,\mathbf{y}_s)$ from the inner product between $\boldsymbol{\pi} = (\mathbf{y}, \widehat{\mathbf{y}})$ and

$$\mathbf{q} \coloneqq \left(\sum_{i=1}^{s} \beta_1^{(i)} \mathbf{r}_i, \ldots, \sum_{i=1}^{s} \beta_s^{(i)} \mathbf{r}_i \right) \Big\| \left(\sum_{i=1}^{s} \alpha_{k,m}^{(i)} \mathbf{r}_i \right)_{1\le k\le m\le s} \in \mathbb{F}^{s+\binom{s+1}{2}}. \tag{6.1}$$

Namely, $a_3 = \langle \boldsymbol{\pi}, \mathbf{q} \rangle - \sum_{i=1}^{n} \mathbf{r}_i \mathbf{x}_i$. □

Given $\mathbf{q}$ as in Equation (6.1) (which is obtained from the coefficients of the publicly known polynomials $Q_1, \ldots, Q_s$ and the vector $\mathbf{r}$ that is generated by the verifier), the verifier checks that all polynomials are zero by doing the following linear query:

$$\text{Query 3: } a'_3 := \langle \boldsymbol{\pi}, \mathbf{q} \rangle = \sum_{i=1}^{s} \mathbf{r}_i Q_i(\mathbf{y}_1, \ldots, \mathbf{y}_s) + \sum_{i=1}^{n} \mathbf{r}_i \mathbf{x}_i,$$

$$\textbf{Check } a'_3 - \sum_{i=1}^{n} \mathbf{r}_i \mathbf{x}_i = 0 \text{ (check } Q_1(\mathbf{y}_1, \ldots, \mathbf{y}_s) = \cdots = Q_s(\mathbf{y}_1, \ldots, \mathbf{y}_s) = 0).$$

The Hadamard LPCP is presented in Figure 6.2.

We now analyze the completeness and soundness of the Hadamard LPCP.

Proposition 6.1.3. *The Hadamard LPCP achieves perfect completeness.*

Proof. To show that the Hadamard LPCP achieves perfect completeness, we show that if $(\mathbf{x}; \mathbf{w}) \in \mathcal{R}$ is such that $C_{\mathbf{x}}(\mathbf{w}) = 0$ (and if the prover correctly follows the protocol), then the verifier as described in Figure 6.2 accepts with probability 1. If $C_{\mathbf{x}}(\mathbf{w}) = 0$, then $\exists \mathbf{y} \in \mathbb{F}^s$ consisting of the value of each gate in $C_{\mathbf{x}}(\mathbf{w})$. Let $\boldsymbol{\pi} = (\mathbf{y}, \widehat{\mathbf{y}})$, z_1, z_2, and z_3 as in Figure 6.2. We need to show that $z_2 = z_1^2$ and that $z_3 - \sum_{i=1}^{n} \mathbf{r}_i \mathbf{x}_i = 0$. For the first fact,

$$\begin{aligned} z_2 &= \sum_{i=1}^{s} \widehat{\mathbf{y}}_{i,i} \mathbf{r}_i^2 + \sum_{1 \le i < j \le s} \widehat{\mathbf{y}}_{i,j} (2 \mathbf{r}_i \mathbf{r}_j) \\ &= \sum_{i=1}^{s} \mathbf{y}_i^2 \mathbf{r}_i^2 + 2 \sum_{1 \le i < j \le s} \mathbf{y}_i \mathbf{r}_i \mathbf{y}_j \mathbf{r}_j \\ &= \left(\sum_{i=1}^{s} \mathbf{y}_i \mathbf{r}_i \right)^2 = \langle \mathbf{y}, \mathbf{r} \rangle^2 = z_1^2. \end{aligned}$$

For the second fact, we use that, by construction of the quadratic polynomials, we have $Q_i(\mathbf{y}_1, \ldots, \mathbf{y}_s) = 0$ for each $i \in [s]$. Hence,

$$z_3 - \sum_{i=1}^{n} \mathbf{r}_i \mathbf{x}_i = \sum_{i=1}^{s} \mathbf{r}_i Q_i(\mathbf{y}_1, \ldots, \mathbf{y}_s) + \sum_{i=1}^{n} \mathbf{r}_i \mathbf{x}_i - \sum_{i=1}^{n} \mathbf{r}_i \mathbf{x}_i = \sum_{i=1}^{s} \mathbf{r}_i Q_i(\mathbf{y}_1, \ldots, \mathbf{y}_s) = 0.$$

Since the calculations above are independent of the choice of $\mathbf{r}$, this proves that the Hadamard LPCP achieves perfect completeness. □

Prover P	**Verifier V**
$(\mathbf{x}, C_{\mathbf{x}}; \mathbf{w})$	$\mathbf{x}, C_{\mathbf{x}}$
Compute $Q_1, \ldots, Q_s$	Compute $Q_1, \ldots, Q_s$
(following Definition 6.1.1)	(following Definition 6.1.1)
Compute values $\mathbf{y} \in \mathbb{F}^s$ of all gates of $C_{\mathbf{x}}$ such that $C_{\mathbf{x}}(\mathbf{w}) = 0$	
Compute $\widehat{\mathbf{y}} \coloneqq \mathbf{y} \,(\times)\, \mathbf{y} \in \mathbb{F}^{\binom{s+1}{2}}$	
$\boldsymbol{\pi} \coloneqq (\mathbf{y}, \widehat{\mathbf{y}}) \in \mathbb{F}^{s+\binom{s+1}{2}}$ $\longrightarrow$	
	$\mathbf{r} \xleftarrow{\$} \mathbb{F}^s$
	$\mathbf{r}' \coloneqq (\mathbf{r}_1, \ldots, \mathbf{r}_s, 0, \ldots, 0) \in \mathbb{F}^{s+\binom{s+1}{2}}$
	$\widehat{\mathbf{r}} \coloneqq (\mathbf{r}_i^2)_{1 \le i \le s} \| (2\mathbf{r}_i \mathbf{r}_j)_{1 \le i < j \le s} \in \mathbb{F}^{\binom{s+1}{2}}$
	$\widehat{\mathbf{r}}' \coloneqq (0, \ldots, 0) \| \widehat{\mathbf{r}} \in \mathbb{F}^{s+\binom{s+1}{2}}$
	Let $\mathbf{q} \in \mathbb{F}^{s+\binom{s+1}{2}}$ be as in Equation (6.1)
	Query 1. $z_1 \coloneqq \langle \boldsymbol{\pi}, \mathbf{r}' \rangle = \langle \mathbf{y}, \mathbf{r} \rangle$
	Query 2. $z_2 \coloneqq \langle \boldsymbol{\pi}, \widehat{\mathbf{r}}' \rangle$ $= \sum_{i=1}^{s} \widehat{\mathbf{y}}_{i,i} \mathbf{r}_i^2 + \sum_{1 \le i < j \le s} \widehat{\mathbf{y}}_{i,j} (2\mathbf{r}_i \mathbf{r}_j)$
	Query 3. $z_3 \coloneqq \langle \boldsymbol{\pi}, \mathbf{q} \rangle$ $= \sum_{i=1}^{s} \mathbf{r}_i Q_i(\mathbf{y}_1, \ldots, \mathbf{y}_s) + \sum_{i=1}^{n} \mathbf{r}_i \mathbf{x}_i$
	Accept $\iff z_2 = z_1^2 \wedge z_3 - \sum_{i=1}^{n} \mathbf{r}_i \mathbf{x}_i = 0$

Figure 6.2: The Hadamard Linear Probabilistically Checkable Proof. Note that the values of the three queries (i.e., $z_1 = \langle \mathbf{y}, \mathbf{r} \rangle$ etc.) also hold for cheating provers, as they only depend on public parameters.

Proposition 6.1.4. *The Hadamard LPCP has soundness error* $2/|\mathbb{F}| = \mathcal{O}(1/|\mathbb{F}|)$.

Proof. To analyze the soundness error of the Hadamard LPCP, we need to calculate

$$\mathbb{P}\left[z_2 = z_1^2 \wedge z_3 - \sum_{i=1}^{n} \mathbf{r}_i \mathbf{x}_i = 0 \right],$$

assuming $(\mathbf{x}, C_{\mathbf{x}}) \notin L_{\mathcal{R}}$ (i.e., there is no $\mathbf{w}$ for which $C_{\mathbf{x}}(\mathbf{w}) = 0$), and assuming a cheating prover tries to convince an honest verifier to accept. The prover constructs a proof string $\boldsymbol{\pi} = (\mathbf{y}, \widehat{\mathbf{y}}) \in \mathbb{F}^{s+\binom{s+1}{2}}$. We distinguish two cases, namely whether $\widehat{\mathbf{y}} = \mathbf{y} \otimes \mathbf{y}$. The maximum of the two probabilities will be our soundness error.

Case $\widehat{\mathbf{y}} \neq \mathbf{y} \otimes \mathbf{y}$. Note that

$$\mathbb{P}\left[z_2 = z_1^2 \wedge z_3 - \sum_{i=1}^{n} \mathbf{r}_i \mathbf{x}_i = 0\right] \leq \mathbb{P}[z_2 = z_1^2] = \mathbb{P}[z_1^2 - z_2 = 0]$$
$$= \mathbb{P}\left[\langle \mathbf{y}, \mathbf{r} \rangle^2 - \left(\sum_{i=1}^{s} \widehat{\mathbf{y}}_{i,i} \mathbf{r}_i^2 + \sum_{1 \leq i < j \leq s} \widehat{\mathbf{y}}_{i,j} (2\mathbf{r}_i \mathbf{r}_j)\right) = 0\right].$$

Our goal is to bound this probability using the Schwartz-Zippel lemma (Lemma 3.3.9). To this end, we need to write $\langle \mathbf{y}, \mathbf{r} \rangle^2 - \left(\sum_{i=1}^{s} \widehat{\mathbf{y}}_{i,i} \mathbf{r}_i^2 + \sum_{1 \leq i < j \leq s} \widehat{\mathbf{y}}_{i,j} (2\mathbf{r}_i \mathbf{r}_j)\right)$ as a non-zero polynomial in the (uniformly randomly sampled) vector $\mathbf{r}$. We have

$$\begin{aligned}
z_1^2 - z_2 &= \langle \mathbf{y}, \mathbf{r} \rangle^2 - \left(\sum_{i=1}^{s} \widehat{\mathbf{y}}_{i,i} \mathbf{r}_i^2 + \sum_{1 \leq i < j \leq s} \widehat{\mathbf{y}}_{i,j} (2\mathbf{r}_i \mathbf{r}_j)\right) \\
&= \left(\sum_{i=1}^{s} \mathbf{y}_i \mathbf{r}_i\right)^2 - \left(\sum_{i=1}^{s} \widehat{\mathbf{y}}_{i,i} \mathbf{r}_i^2 + 2 \sum_{1 \leq i < j \leq s} \widehat{\mathbf{y}}_{i,j} \mathbf{r}_i \mathbf{r}_j\right) \\
&= \sum_{i=1}^{s} \mathbf{y}_i^2 \mathbf{r}_i^2 + 2 \sum_{1 \leq i < j \leq s} \mathbf{y}_i \mathbf{y}_j \mathbf{r}_i \mathbf{r}_j - \left(\sum_{i=1}^{s} \widehat{\mathbf{y}}_{i,i} \mathbf{r}_i^2 + 2 \sum_{1 \leq i < j \leq s} \widehat{\mathbf{y}}_{i,j} \mathbf{r}_i \mathbf{r}_j\right) \\
&= \sum_{i=1}^{s} (\mathbf{y}_i^2 - \widehat{\mathbf{y}}_{i,i}) \mathbf{r}_i^2 + 2 \sum_{1 \leq i < j \leq s} (\mathbf{y}_i \mathbf{y}_j - \widehat{\mathbf{y}}_{i,j}) \mathbf{r}_i \mathbf{r}_j \ ,
\end{aligned}$$

which we can write as $f(\mathbf{r}_1, \ldots, \mathbf{r}_s)$, where

$$f(X_1, \ldots, X_s) = \sum_{i=1}^{s} (\mathbf{y}_i^2 - \widehat{\mathbf{y}}_{i,i}) X_i^2 + 2 \sum_{1 \leq i < j \leq s} (\mathbf{y}_i \mathbf{y}_j - \widehat{\mathbf{y}}_{i,j}) X_i X_j$$

is an s-variate polynomial of degree 2 over $\mathbb{F}$ with coefficients consisting of (linear combinations of) entries of $\mathbf{y}$ and $\widehat{\mathbf{y}}$. Note that if $\widehat{\mathbf{y}} \neq \mathbf{y} \otimes \mathbf{y} = (\mathbf{y}_i^2)_{1 \leq i \leq s} || (\mathbf{y}_i \mathbf{y}_j)_{1 \leq i < j \leq s}$, then either there exists an $i \in [s]$ such that $\widehat{\mathbf{y}}_{i,i} \neq \mathbf{y}_i^2$, or there exists $i, j \in [s]$ with $i < j$ such that $\widehat{\mathbf{y}}_{i,j} \neq \mathbf{y}_i \mathbf{y}_j$. In both cases, there is some coefficient in the polynomial f that is non-zero[1], implying that $f(X_1, \ldots, X_s)$ is *not* the zero polynomial. Hence, we can apply

[1] Note that we implicitly assume that the field $\mathbb{F}$ does not have characteristic 2, as otherwise $f(X_1, \ldots, X_s) = \sum_{i=1}^{s} (\mathbf{y}_i^2 - \widehat{\mathbf{y}}_{i,i}) X_i^2$, which could be the zero polynomial, even if for all $i, j \in [s]$ with $i < j$ we have $\widehat{\mathbf{y}}_{i,j} \neq \mathbf{y}_i \mathbf{y}_j$. However, as we are mostly interested in the size of the field, this assumption does not affect our analysis.

Schwartz-Zippel as follows

$$\mathbb{P}\left[z_2 = z_1^2 \wedge z_3 - \sum_{i=1}^{n} \mathbf{r}_i \mathbf{x}_i = 0\right] \leq \mathbb{P}\left[z_1^2 - z_2 = 0\right] = \mathbb{P}[f(\mathbf{r}_1, \ldots, \mathbf{r}_s) = 0] \leq \frac{2}{|\mathbb{F}|},$$

where the Schwartz-Zippel lemma (Lemma 3.3.9) is used in the last step, using the facts that $\mathbf{r} \xleftarrow{\$} \mathbb{F}^s$ and f has degree 2.

Case $\widehat{\mathbf{y}} = \mathbf{y}\,(\times)\,\mathbf{y}$. We have

$$\begin{aligned}\mathbb{P}\left[z_2 = z_1^2 \wedge z_3 - \sum_{i=1}^{n} \mathbf{r}_i \mathbf{x}_i = 0\right] &\leq \mathbb{P}\left[z_3 - \sum_{i=1}^{n} \mathbf{r}_i \mathbf{x}_i = 0\right] \\ &= \mathbb{P}\left[\sum_{i=1}^{s} \mathbf{r}_i Q_i(\mathbf{y}_1, \ldots, \mathbf{y}_s) = 0\right] \\ &= \mathbb{P}[\langle \mathbf{r}, \mathbf{Q} \rangle = 0],\end{aligned}$$

where $\mathbf{Q} = (Q_1(\mathbf{y}_1, \ldots, \mathbf{y}_s), \ldots, Q_s(\mathbf{y}_1, \ldots, \mathbf{y}_s)) \in \mathbb{F}^s$. Our goal is to apply Lemma 3.3.8 to prove that $\langle \mathbf{r}, \mathbf{Q} \rangle$ is uniformly distributed over $\mathbb{F}$ (i.e., that $\mathbb{P}[\langle \mathbf{r}, \mathbf{Q} \rangle = 0] = \frac{1}{|\mathbb{F}|}$). To this end, we only need to show that $\mathbf{Q} \neq \mathbf{0}$ (as $\mathbf{r}$ is already uniformly distributed over $\mathbb{F}^s$). Since we assumed that $C_{\mathbf{x}}(\mathbf{w}) \neq 0$ for all $\mathbf{w}$, at least for one $i \in [s]$ we must have $Q_i(\mathbf{y}_1, \ldots, \mathbf{y}_s) \neq 0$, which means that $\mathbf{Q}$ is not the zero-vector. Hence, by Lemma 3.3.8,

$$\mathbb{P}\left[z_2 = z_1^2 \wedge z_3 - \sum_{i=1}^{n} \mathbf{r}_i \mathbf{x}_i = 0\right] \leq \mathbb{P}[\langle \mathbf{r}, \mathbf{Q} \rangle = 0] = \frac{1}{|\mathbb{F}|}.$$

The maximum of the two probabilities is $\frac{2}{|\mathbb{F}|}$, and thus we conclude that the soundness error is $\frac{2}{|\mathbb{F}|} = \mathcal{O}(1/|\mathbb{F}|)$. □

It remains for us to analyze the zero-knowledge property for the Hadamard LPCP. We can prove that a slight variation of the Hadamard LPCP is honest-verifier zero-knowledge. The basic idea is that we augment the relation $\mathcal{R}$ by adding a dummy entry $\mathbf{w}_0$ to the witness $\mathbf{w}$. In the corresponding circuit, we get an additional input gate $\mathbf{w}_0$ that is not attached to the rest of the circuit. In the conversion to quadratic polynomials, we add an additional zero polynomial for the variable corresponding to the $\mathbf{w}_0$-gate. In the protocol, the prover picks the value of $\mathbf{w}_0$ uniformly at random. The verifier then samples some $r_0 \xleftarrow{\$} \mathbb{F}^*$ and augments the vector $\mathbf{r} \xleftarrow{\$} \mathbb{F}^s$ with this r_0; the rest of the protocol remains unchanged. Then, the result of the first query is $z_1 = r_0 \mathbf{w}_0 + \sum_{i=1}^{s} \mathbf{y}_i \mathbf{r}_i$. By Lemma 3.3.6 and Lemma 3.3.7, z_1 is uniformly distributed over $\mathbb{F}$. Hence, we can define a very basic simulator that samples a $z_1 \xleftarrow{\$} \mathbb{F}$ and outputs the correct proof transcript

$$\left(z_1, z_1^2, \sum_{i=1}^{n} \mathbf{r}_i \mathbf{x}_i\right).$$

As z_1 is uniformly distributed over $\mathbb{F}$, so is z_1^2, and thus the proof transcripts that the simulator outputs follow the same distribution as the proof transcripts of the Hadamard LPCP for an honest prover. Hence, this variant of the Hadamard LPCP, which we will call the *augmented Hadamard LPCP*, is perfect honest-verifier zero-knowledge. Note that this augmenting does not affect the soundness and completeness analyses. We summarize the Hadamard LPCP in the following theorem.

Theorem 6.1.5 (Information-theoretic Hadamard LPCP)**.** *Let $\mathbb{F}$ be a finite field. The Hadamard LPCP (of Figure 6.2) for common input $\mathbf{x}$ and arithmetic circuit $C_{\mathbf{x}}$ of size s over $\mathbb{F}$ proves membership in the relation*

$$\mathcal{R} = \{(\mathbf{x}, C_{\mathbf{x}}; \mathbf{w}) \mid C_{\mathbf{x}}(\mathbf{w}) = 0\},$$

and has the following properties:

- *the proof string has length $s + \binom{s+1}{2}$;*
- *the verifier does 3 queries;*
- *the protocol is perfectly complete;*
- *the protocol has statistical soundness error $2/|\mathbb{F}| = \mathcal{O}(1/|\mathbb{F}|)$;*
- *the augmented Hadamard LPCP is perfect honest-verifier zero-knowledge.*

Remark. Note that the soundness error and zero-knowledge property in Theorem 6.1.5 assume the existence of an oracle that only provides access to linear combinations of proof symbols.

6.2 Compiling LPCPs using linear-only encryption schemes

The best current cryptographic compiler for LPCPs is based on the heavy cryptographic machinery called *linear-only encryption* [10]. For this compiler, we need to work in the *common reference string (CRS) model* (see Section 5.2), where we assume the existence of a trusted third-party that provides a reference string (following a publicly known distribution) to the prover and verifier. Furthermore, *linear-only encryption* is an instance of *public-key (or asymmetric) cryptography*, where messages are encrypted using the public key of the receiver, and are decrypted by the receiver using the private (or secret) key that is only known to them.

Roughly, linear-only encryption says that if $c_1, \ldots, c_m$ are encryptions of plaintexts $x_1, \ldots, x_m$ respectively, then it is possible to derive a ciphertext c that is an encryption

of $a_1x_1 + \ldots + a_mx_m$ (where each a_i is a public scalar) by only using public parameters. Furthermore, if c is obtained from $c_1, \ldots, c_m$ without using the secret key, then in fact it must be an encryption of a linear combination of the plaintexts (i.e., c cannot be an encryption of any other function of the plaintexts).

We give a more formal definition of a linear-only encryption scheme in the common reference string model below. We give this definition in the context of proof systems, so that we can apply it directly when sketching the compiler.

Definition 6.2.1. Let $\mathbb{F}$ be a finite field. Let $(\text{Setup}, \text{Enc}, \text{Dec}, \text{Add})$ denote a linear-only encryption scheme, let $\mathcal{P}$ be a prover, and let $\mathcal{V}$ be a verifier. A trusted third-party runs the algorithm Setup and provides a reference string σ (containing the public key pk) to both $\mathcal{P}$ and $\mathcal{V}$, while providing a secret key sk to $\mathcal{V}$ only.

- Given $x \in \mathbb{F}$, we write $c \coloneqq \text{Enc}_{\text{pk}}(x) \in \mathbb{F}$ for the ciphertext obtained by encrypting x using the public key pk.

- We write $x = \text{Dec}_{\text{sk}}(c)$ for the message x obtained by decrypting the ciphertext c using the secret key sk.

- Given encryptions $c_1 = \text{Enc}_{\text{pk}}(x_1), \ldots, c_s = \text{Enc}_{\text{pk}}(x_s)$ of messages $x_1, \ldots, x_s \in \mathbb{F}$, and given a $\boldsymbol{\pi} \in \mathbb{F}^s$, we write

$$c = \text{Add}_{\text{pk}}(\boldsymbol{\pi}, (c_1, \ldots, c_s)) \coloneqq \text{Enc}_{\text{pk}}(\boldsymbol{\pi}_1 x_1 + \ldots + \boldsymbol{\pi}_s x_s)$$

  for the ciphertext c obtained by homomorphic addition of the ciphertexts $c_1, \ldots, c_s$.

We can now sketch the cryptographic compiler for LPCPs that is based on linear-only encryption. Suppose we have an LPCP, where the prover constructs a proof string $\boldsymbol{\pi} \in \mathbb{F}^s$, the verifier does *input-oblivious*[1] queries $\mathbf{q}^{(1)}, \ldots, \mathbf{q}^{(\xi)} \in \mathbb{F}^s$, and the verifier obtains responses $z_i = \langle \boldsymbol{\pi}, \mathbf{q}^{(i)} \rangle$ for each $i \in [\xi]$. Suppose further that we have a linear-only encryption scheme $(\text{Setup}, \text{Enc}, \text{Dec}, \text{Add})$. A trusted third-party runs the algorithm Setup to generate a secret key sk, a public key pk, and the common reference string

[1] Queries are *input-oblivious* if they do not depend on the input. For linear-only encryption based compilers, the common reference string contains encryptions of the queries, which is why the queries need to be input-oblivious. In (arithmetic) circuit satisfiability problems (e.g., the Hadamard LPCP), the circuit can be viewed as a public parameter and not part of the input, so the input-oblivious requirement does not pose a problem in such cases.

$\sigma = (\mathrm{pk}, c_1^{(1)}, \ldots, c_s^{(1)}, \ldots, c_1^{(\xi)}, \ldots, c_s^{(\xi)})$, where[1]

$$\begin{aligned}
(c_1^{(1)}, \ldots, c_s^{(1)}) &= \left(\mathrm{Enc}_{\mathrm{pk}}(\mathbf{q}_1^{(1)}), \ldots, \mathrm{Enc}_{\mathrm{pk}}(\mathbf{q}_s^{(1)})\right) \approx \mathrm{Enc}_{\mathrm{pk}}(\mathbf{q}^{(1)}),\\
(c_1^{(2)}, \ldots, c_s^{(2)}) &= \left(\mathrm{Enc}_{\mathrm{pk}}(\mathbf{q}_1^{(2)}), \ldots, \mathrm{Enc}_{\mathrm{pk}}(\mathbf{q}_s^{(2)})\right) \approx \mathrm{Enc}_{\mathrm{pk}}(\mathbf{q}^{(2)}),\\
&\vdots\\
(c_1^{(\xi)}, \ldots, c_s^{(\xi)}) &= \left(\mathrm{Enc}_{\mathrm{pk}}(\mathbf{q}_1^{(\xi)}), \ldots, \mathrm{Enc}_{\mathrm{pk}}(\mathbf{q}_s^{(\xi)})\right) \approx \mathrm{Enc}_{\mathrm{pk}}(\mathbf{q}^{(\xi)}).
\end{aligned}$$

Now, when the prover wants to prove a statement, they use homomorphic addition of the ciphertexts in σ to produce encryptions of the answers to the queries. This addition is done as follows:

$$\begin{aligned}
a_1 &\coloneqq \mathrm{Add}_{\mathrm{pk}}(\boldsymbol{\pi}, (c_1^{(1)}, \ldots, c_s^{(1)})) = \mathrm{Enc}_{\mathrm{pk}}\left(\boldsymbol{\pi}_1 \mathbf{q}_1^{(1)} + \ldots + \boldsymbol{\pi}_s \mathbf{q}_s^{(1)}\right) = \mathrm{Enc}_{\mathrm{pk}}(\langle \boldsymbol{\pi}, \mathbf{q}^{(1)} \rangle),\\
&\vdots\\
a_\xi &\coloneqq \mathrm{Add}_{\mathrm{pk}}(\boldsymbol{\pi}, (c_1^{(\xi)}, \ldots, c_s^{(\xi)})) = \mathrm{Enc}_{\mathrm{pk}}\left(\boldsymbol{\pi}_1 \mathbf{q}_1^{(\xi)} + \ldots + \boldsymbol{\pi}_s \mathbf{q}_s^{(\xi)}\right) = \mathrm{Enc}_{\mathrm{pk}}(\langle \boldsymbol{\pi}, \mathbf{q}^{(\xi)} \rangle).
\end{aligned}$$

The verifier decrypts these answers using their secret key (i.e., $z_i \coloneqq \mathrm{Dec}_{\mathrm{sk}}(a_i) = \langle \boldsymbol{\pi}, \mathbf{q}^{(i)} \rangle$ for each $i \in [\xi]$) and checks whether $z_1, \ldots, z_\xi$ convince the information-theoretic LPCP verifier[2]. The LOE-compiled LPCP is presented graphically in Figure 6.3.

Since the linear-only encryption based compiler proves knowledge soundness of the resulting cryptographic proof [10], we will need the following lemma.

Lemma 6.2.2 ([44]). *If an interactive protocol is knowledge sound with knowledge error κ, then it is also statistically sound with soundness error κ. Furthermore, a computational knowledge error of κ implies a computational soundness error of κ.*

Now, an LPCP that is compiled with *linear-only encryption (LOE)* schemes has the following properties [10, Lemma 3.2, Lemma 6.2].

Theorem 6.2.3 (General LOE-compiled LPCP). *Let $\mathbb{F}$ be a finite field of size q, where q scales with a security parameter λ, and let $s \in \mathbb{N}$. Let Π be a perfectly complete and perfect honest-verifier zero-knowledge information-theoretic LPCP for a relation $\mathcal{R} = \{(\mathbf{x}; \mathbf{w})\} \subseteq$*

[1] We use the symbol '$\approx$' to emphasize the conceptual meaning of an object. For example, we write $(c_1^{(1)}, \ldots, c_s^{(1)}) = \left(\mathrm{Enc}_{\mathrm{pk}}(\mathbf{q}_1^{(1)}), \ldots, \mathrm{Enc}_{\mathrm{pk}}(\mathbf{q}_s^{(1)})\right) \approx \mathrm{Enc}_{\mathrm{pk}}(\mathbf{q}^{(1)})$ to emphasize the fact that $(c_1^{(1)}, \ldots, c_s^{(1)})$ should be interpreted as the encryption of the vector $\mathbf{q}^{(1)} \in \mathbb{F}^s$, whereas in reality we have to individually encrypt each entry of $\mathbf{q}^{(1)}$.

[2] Technically, an LPCP is first transformed into a *linear interactive proof (LIP)* [10], which is an interactive proof where each prover message must be a linear function of the previously sent verifier challenges [27]. The resulting LIP is compiled in the manner described in this section. However, these details are outside the scope of this thesis.

$\mathbb{F}^n \times \mathbb{F}^m$. *Suppose* Π *has a proof string* $\boldsymbol{\pi} \in \mathbb{F}^s$ *and has statistical soundness error* $\varepsilon_{\text{LPCP}}$. *Further, suppose* Π *consists of* ξ *input-oblivious queries* $\mathbf{q}^{(1)}, \ldots, \mathbf{q}^{(\xi)} \in \mathbb{F}^s$ *to the proof string oracle, to which the responses are* $z_i = \langle \mathbf{q}^{(i)}, \boldsymbol{\pi} \rangle$ *for* $i \in [\xi]$.

Then, Π *can be compiled to a cryptographic proof using a linear-only encryption scheme* $(\text{Setup}, \text{Enc}, \text{Dec}, \text{Add})$. *The resulting linear-only-compiled LPCP has the following parameters:*

- *in the plain model*[1]*, the protocol is non-interactive and respectively has a proof string and common reference string of lengths*

$$|\boldsymbol{\pi}| = \text{poly}(\lambda) \cdot (\xi + 1),$$
$$|\sigma| = \text{poly}(\lambda) \cdot (\xi + 1)s.$$

- *the protocol is perfectly complete;*
- *the protocol is perfect honest-verifier zero-knowledge;*
- *the protocol has computational soundness error* $\varepsilon_{\text{LPCP}} + \frac{1}{|\mathbb{F}|} + \text{negl}(\lambda)$, *assuming the existence of linear-only encryption schemes.*

Proof. An LPCP with soundness error $\varepsilon_{\text{LPCP}}$ that consists of ξ queries $\mathbf{q}^{(1)}, \ldots, \mathbf{q}^{(\xi)} \in \mathbb{F}^s$ can be transformed into a 2-message LIP with computational knowledge error $\varepsilon_{\text{LPCP}} + \frac{1}{|\mathbb{F}|}$, where the prover message is in $\mathbb{F}^{\xi+1}$ and the verifier message is in $\mathbb{F}^{(\xi+1)s}$ [10, Lemma 3.2]. Such an LIP can then be transformed into a *succinct non-interactive argument of knowledge (SNARK)* (i.e., a non-interactive cryptographic proof that is perfectly complete and perfect honest-verifier zero-knowledge) with proof string length $\text{poly}(\lambda) \cdot (\xi + 1)$ and common reference string length $\text{poly}(\lambda) \cdot (\xi + 1)s$ [10, Lemma 6.2]. Furthermore, the SNARK has computational knowledge error $\varepsilon_{\text{LPCP}} + \frac{1}{|\mathbb{F}|} + \text{negl}(\lambda)$. By Lemma 6.2.2, the SNARK has computational soundness error $\varepsilon_{\text{LPCP}} + \frac{1}{|\mathbb{F}|} + \text{negl}(\lambda)$. □

Remark. Note that, in Theorem 6.2.3, we are actually overloading the symbol $\boldsymbol{\pi}$ for two different meanings of *proof string* in this context. We have the information-theoretic proof string $\boldsymbol{\pi}$ that is used to construct the query responses $z_i = \langle \mathbf{q}^{(i)}, \boldsymbol{\pi} \rangle$, and we have the proof string $\boldsymbol{\pi}$ in the compiled LPCP, which consists of $a_1, \ldots, a_\xi$ (the ciphertexts of the query answers, obtained by homomorphic addition). This second meaning of $\boldsymbol{\pi}$ has length $\text{poly}(\lambda) \cdot (\xi + 1)$ and appears on the arrow in Figure 6.3.

[1] In the *plain model* we do not make any special assumptions, as opposed to the *random oracle model* where we assume the existence of random oracles (see Definition 4.5.2).

Prover $\mathcal{P}$		**Verifier** $\mathcal{V}$
$(\mathbf{x}; \mathbf{w}), \sigma$	$\sigma = (\mathrm{pk}, c_1^{(1)}, \ldots, c_s^{(\xi)}), \mathrm{sk}$ generated by trusted third-party, where $c_i^{(j)} = \mathrm{Enc}_{\mathrm{pk}}(\mathbf{q}_i^{(j)})$ $(i \in [s],\, j \in [\xi])$	$\mathbf{x}, \sigma, \mathrm{sk}$
Proof string $\boldsymbol{\pi} \in \mathbb{F}^s$		Queries $\mathbf{q}^{(1)}, \ldots, \mathbf{q}^{(\xi)} \in \mathbb{F}^s$
$a_i = \mathrm{Add}_{\mathrm{pk}}(\boldsymbol{\pi}, (c_1^{(i)}, \ldots, c_s^{(i)}))$ $= \mathrm{Enc}_{\mathrm{pk}}(\langle \boldsymbol{\pi}, \mathbf{q}^{(i)} \rangle)$ for $i \in [\xi]$		
	$a_1, \ldots, a_\xi \longrightarrow$	
		$z_i = \mathrm{Dec}_{\mathrm{sk}}(a_i) = \langle \boldsymbol{\pi}, \mathbf{q}^{(i)} \rangle$ for $i \in [\xi]$
		Accept $\iff z_1, \ldots, z_\xi$ convince information-theoretic verifier $\mathbf{V}$

Figure 6.3: The general linear-only-encryption-compiled LPCP.

We can now apply Theorem 6.2.3 to the Hadamard LPCP of Theorem 6.1.5, where we have $\xi = 3$ queries, soundness error $\varepsilon_{\mathrm{LPCP}} = \mathcal{O}(1/|\mathbb{F}|)$, and proof string length $s + \binom{s+1}{2}$. To make the queries of the Hadamard LPCP input-oblivious, we simply make the circuit $C_{\mathbf{x}}$ a public parameter. Since the queries only depend on $C_{\mathbf{x}}$, they are now input-oblivious.

Theorem 6.2.4 (LOE-compiled Hadamard LPCP)**.** *Let $\mathbb{F}$ be a finite field of size q, where q scales with a security parameter λ. Let $C_{\mathbf{x}}$ be a public arithmetic circuit of size s over $\mathbb{F}$. The linear-only-compiled Hadamard LPCP for the relation*

$$\mathcal{R} = \{(\mathbf{x}, C_{\mathbf{x}}; \mathbf{w}) \mid C_{\mathbf{x}}(\mathbf{w}) = 0\}$$

has the following properties:

- *in the plain model, the protocol is non-interactive and respectively has a proof string and common reference string of lengths*

$$|\boldsymbol{\pi}| = \mathrm{poly}(\lambda),$$
$$|\sigma| = \mathrm{poly}(\lambda) \cdot (s + \tbinom{s+1}{2}).$$

- *the protocol is perfectly complete;*
- *the protocol is perfect honest-verifier zero-knowledge;*
- *the protocol has computational soundness error* $\mathcal{O}(1/|\mathbb{F}|) + \mathrm{negl}(\lambda)$*, assuming the existence of linear-only encryption schemes.*

Proof. Most properties follow directly from plugging in $\xi = 3$ and $s = s + \binom{s+1}{2}$ in Theorem 6.2.3. The soundness error becomes

$$\varepsilon_{\mathrm{LPCP}} + \frac{1}{|\mathbb{F}|} + \mathrm{negl}(\lambda) = \mathcal{O}(1/|\mathbb{F}|) + \frac{1}{|\mathbb{F}|} + \mathrm{negl}(\lambda) = \mathcal{O}(1/|\mathbb{F}|) + \mathrm{negl}(\lambda),$$

and the proof string and common reference string lengths become

$$|\boldsymbol{\pi}| = 4 \cdot \mathrm{poly}(\lambda) = \mathrm{poly}(\lambda),$$
$$|\sigma| = \mathrm{poly}(\lambda) \cdot 4(s + \tbinom{s+1}{2}) = \mathrm{poly}(\lambda) \cdot (s + \tbinom{s+1}{2}).$$

□

7

Cryptographic compiler for linear queries

In this chapter, we introduce our first cryptographic compiler for LPCPs. We realize the linear queries by running a *compressed* Σ*-protocol* as a black-box for each query. The main contribution of this compiler is that its security depends on the discrete logarithm assumption only (Assumption 1), which is a much simpler and more widely-studied assumption than the heavy machinery needed for compiling LPCPs with linear-only encryption schemes.

Note that both in this chapter and in Chapter 8, we focus on compressed Σ-protocols that rely on the discrete logarithm assumption [5]. However, we could also replace these compressed Σ-protocols by ones that rely on other cryptographic assumptions, such as:

- *RSA assumption* [5], where we assume that it is hard to compute an n-th root of an arbitrary number modulo N;
- *short integer solutions (SIS)* [6], which is a lattice-based assumption that roughly states that it is hard to find a vector (adhering to some norm bound) in the null space of a random matrix $\mathbf{A} \xleftarrow{\$} \mathbb{Z}_q^{n \times m}$.

This second assumption can for example be useful for achieving *post-quantum security*[1]. However, post-quantum secure proof systems are outside the scope of this thesis, which is why we decided to focus on the simplest of these assumptions (DLOG).

[1] A proof system is *post-quantum secure* if its security properties (i.e., soundness, and zero-knowledge) also hold against adversaries that can run computations on a quantum computer.

Although our compiled LPCPs become interactive protocols, we can make them non-interactive again by applying the Fiat-Shamir transform (Section 4.5). To apply this transform, we must assume the Random Oracle Model (see Definition 4.5.2).

In Section 7.1, we introduce the compressed Σ-protocol that is at the core of our compiler. Then, in Section 7.2, we compile the Hadamard LPCP using our compiler. In Section 7.3, we apply a typical *amortization* trick to reduce the amount of communication in the compiled Hadamard LPCP. Finally, in Section 7.4, we compile general LPCPs using our compiler.

7.1 The compiler: a compressed Σ-protocol for linear forms

For our first cryptographic compiler, we will interpret each linear query of the verifier in an LPCP as an evaluation of a *linear form*. We then use a *compressed Σ-protocol* as a black-box to give a proof of validity of each linear form evaluation (i.e., each query) [5]. First, we define linear forms and introduce some general notation.

Definition 7.1.1. Let q be a prime power. A map $L\colon \mathbb{Z}_q^n \to \mathbb{Z}_q$ is called a *linear form* if L is linear (i.e., for any $\mathbf{x}, \mathbf{y} \in \mathbb{Z}_q^n$ and $c \in \mathbb{Z}_q$, we have $L(\mathbf{x}+\mathbf{y}) = L(\mathbf{x}) + L(\mathbf{y})$ and $L(c\mathbf{x}) = cL(\mathbf{x})$). We write $\mathcal{L}(\mathbb{Z}_q^n) \coloneqq \{(L\colon \mathbb{Z}_q^n \to \mathbb{Z}_q) \mid L \text{ is a linear form}\}$.

Remark 7.1.2. Note that for a fixed $\mathbf{a} \in \mathbb{Z}_q^n$, we can write $L(\mathbf{x})$ as $L(\mathbf{x}) = \mathbf{a}_1\mathbf{x}_1 + \ldots + \mathbf{a}_n\mathbf{x}_n = \langle \mathbf{a}, \mathbf{x} \rangle$, as any linear map from $\mathbb{Z}_q^n$ to $\mathbb{Z}_q$ can be written as an $n \times 1$ matrix (i.e., a vector $\mathbf{a}$ of length n).

Notation. Let $\mathbb{G}$ be an Abelian group of prime order q, let $n \in \mathbb{N}$, $\mathbf{g} \in \mathbb{G}^n$, and $\mathbf{x} \in \mathbb{Z}_q^n$. We write $\mathbf{g}^{\mathbf{x}} \coloneqq \prod_{i=1}^n \mathbf{g}_i^{\mathbf{x}_i}$.

Our first cryptographic compiler is mostly based on the following proposition [5, Theorem 3], which we will use as a black-box for the linear queries.

Proposition 7.1.3. *Let $\mathbb{G}$ be a cyclic Abelian group of prime order q for which the discrete logarithm assumption holds, and let $n \in \mathbb{N}$. Let $\mathbf{g} = (\mathbf{g}_1, \ldots, \mathbf{g}_n) \xleftarrow{\$} \mathbb{G}^n$ and $h, k \xleftarrow{\$} \mathbb{G}$ be $n+2$ generators of $\mathbb{G}$. There exists a protocol Π_c for the relation*

$$\mathcal{R} = \left\{ \left(P \in \mathbb{G}, L \in \mathcal{L}(\mathbb{Z}_q^n), y \in \mathbb{Z}_q; \mathbf{x} \in \mathbb{Z}_q^n, \gamma \in \mathbb{Z}_q \right) \mid P = \mathbf{g}^{\mathbf{x}} h^{\gamma}, y = L(\mathbf{x}) \right\}$$

with the following parameters, where $\mu = \lceil \log_2(n+1) \rceil - 1$:

- *Π_c consists of $2\mu + 5$ moves;*
- *Π_c is perfectly complete;*

- Π_c *is perfect honest-verifier zero-knowledge;*
- Π_c *is computationally* $(2, 2, k_1, \ldots, k_\mu)$*-special sound, under the discrete logarithm assumption (Assumption 1), where each* $k_i = 3$*;*
- Π_c *communicates:*
  - $2\lceil \log_2(n+1) \rceil - 1$ *elements of* $\mathbb{G}$ *and* 3 *elements of* $\mathbb{Z}_q$ *from prover to verifier;*
  - $\lceil \log_2(n+1) \rceil + 1$ *elements of* $\mathbb{Z}_q$ *from verifier to prover.*

The idea is that the prover reveals an evaluation of a linear form $L(\mathbf{x})$ with a proof of validity without revealing information on $\mathbf{x}$. The proof of validity relies on a Pedersen commitment[1] P to an input $\mathbf{x}$ for which $L(\mathbf{x}) = y$ (i.e., $\langle \mathbf{a}, \mathbf{x} \rangle = y$ for the $\mathbf{a}$ that defines L).

For the Hadamard LPCP, we want to reveal $\langle \mathbf{r}', \boldsymbol{\pi} \rangle$, $\langle \widehat{\mathbf{r}}', \boldsymbol{\pi} \rangle$, and $\langle \mathbf{q}, \boldsymbol{\pi} \rangle$ without revealing information on the proof string $\boldsymbol{\pi}$, which is exactly what Π_c does in Proposition 7.1.3. To aid the reader, we reformulate this theorem to focus on the special case of the Hadamard LPCP.

Proposition 7.1.4 (Properties of Π_c)**.** *Let* $\mathbb{G}$ *be a cyclic Abelian group of prime order* q *for which the discrete logarithm assumption holds*[2]*. Let* $\mathbb{F}$ *be a finite field of order* q *and let* $s \in \mathbb{N}$*. Let* $\mathbf{g} = (\mathbf{g}_1, \ldots, \mathbf{g}_{s+\binom{s+1}{2}}) \xleftarrow{\$} \mathbb{G}^{s+\binom{s+1}{2}}$ *and* $h, k \xleftarrow{\$} \mathbb{G}$ *be* $s + \binom{s+1}{2} + 2$ *generators of* $\mathbb{G}$*. There exists a protocol* Π_c *for the relation*

$$\mathcal{R} = \left\{ \left(P \in \mathbb{G}, L \in \mathcal{L}(\mathbb{F}^{s+\binom{s+1}{2}}), z \in \mathbb{F}; \boldsymbol{\pi} \in \mathbb{F}^{s+\binom{s+1}{2}}, \gamma \in \mathbb{F} \right) \mid P = \mathbf{g}^{\boldsymbol{\pi}} h^{\gamma}, z = L(\boldsymbol{\pi}) \right\}$$

with the following parameters, where $\mu = \lceil \log_2(s + \binom{s+1}{2} + 1) \rceil - 1$*:*

- Π_c *consists of* $2\mu + 5$ *moves;*
- Π_c *is perfectly complete;*
- Π_c *is perfect honest-verifier zero-knowledge;*
- Π_c *is computationally* $(2, 2, k_1, \ldots, k_\mu)$*-special sound, under the discrete logarithm assumption, where each* $k_i = 3$*;*
- Π_c *communicates:*

[1] Recall that a Pedersen (vector) commitment is computationally binding and perfectly hiding (Section 4.6).

[2] Note that, without loss of generality, we implicitly use multiplicative notation for the group operation of $\mathbb{G}$.

- $2\lceil \log_2(s + \binom{s+1}{2} + 1)\rceil - 1$ *elements of* $\mathbb{G}$ *and* 3 *elements of* $\mathbb{F}$ *from prover to verifier;*
- $\lceil \log_2(s + \binom{s+1}{2} + 1)\rceil + 1$ *elements of* $\mathbb{F}$ *from verifier to prover.*

Proof. We apply Proposition 7.1.3, where $\mathbb{Z}_q$ in the theorem is identified with $\mathbb{F}$, $n = s + \binom{s+1}{2}$, $y = z$, and $\mathbf{x} = \boldsymbol{\pi}$. Note that we use $\mathbb{F}$ and $\mathbb{Z}_q$ interchangeably, as $\mathbb{F}$ is isomorphic to $\mathbb{Z}_q$ (because $\mathbb{F}$ and $\mathbb{Z}_q$ are finite fields of order q, and finite fields of prime order are unique up to isomorphism [35, Theorem 2.5]). □

Remark. In practice, the cyclic group $\mathbb{G}$ in Proposition 7.1.3 and Proposition 7.1.4 is typically a group coming from an *elliptic curve*, as these groups seem to be most suitable for the discrete logarithm assumption, while still allowing for efficient group operations.

We will determine the soundness error of Π_c as follows. First, we apply the fact that computational $(2, 2, k_1, \ldots, k_\mu)$-special soundness implies computational knowledge soundness with a certain knowledge error [6, Theorem 1]. Then, we use that the knowledge error can serve as an upper bound for the soundness error (Lemma 6.2.2). These two facts give us the soundness error of Π_c. To aid the reader, we state the lemma that shows that special soundness implies knowledge soundness [6, Theorem 1].

Lemma 7.1.5 ([6])**.** *Given constants* $\mu, k_1, \ldots, k_\mu \in \mathbb{N}$ *and a (computationally)* $(k_1, \ldots, k_\mu)$*-special sound* $(2\mu + 1)$*-move interactive protocol for some relation* $\mathcal{R}$*, where the verifier challenges are sampled uniformly at random from a challenge set of size* $N \geq \max_i(k_i)$*, then the protocol is (computationally) knowledge sound with knowledge error*

$$\kappa \leq \frac{\sum_{i=1}^{\mu}(k_i - 1)}{N}.$$

Proposition 7.1.6. *Let* $\mathbb{G}$ *be a cyclic Abelian group of prime order* q *for which the discrete logarithm assumption holds. Let* $\mathbb{F}$ *be a finite field of order* q*, let* $s \in \mathbb{N}$*, and let* $\mu = \lceil \log_2(s + \binom{s+1}{2} + 1)\rceil - 1$*. Under the discrete logarithm assumption (Assumption 1), the protocol* Π_c *is computationally sound with soundness error* $\frac{2+2\mu}{|\mathbb{F}|}$*.*

Proof. By Proposition 7.1.4, the protocol Π_c is computationally $(2, 2, k_1, \ldots, k_\mu)$-special sound, where each $k_i = 3$, under the discrete logarithm assumption. By Lemma 7.1.5, that means that Π_c is computationally knowledge sound with knowledge error

$$\kappa \leq \frac{(2-1) + (2-1) + \sum_{i=1}^{\mu}(k_i - 1)}{|\mathbb{F}|} = \frac{2 + 2\mu}{|\mathbb{F}|}.$$

Lemma 6.2.2 then implies that Π_c is computationally sound with soundness error $\frac{2+2\mu}{|\mathbb{F}|}$. □

7.2 Compiling the Hadamard LPCP

To compile the Hadamard LPCP (Figure 6.2) with our newly defined compiler, we apply Proposition 7.1.4 three times, namely once for each query:

1. For the first query, we apply Proposition 7.1.4 to the linear form $L_1 \colon \mathbb{F}^{s+\binom{s+1}{2}} \to \mathbb{F}$ defined by $\mathbf{x} \mapsto \langle \mathbf{r}', \mathbf{x} \rangle$; the prover reveals $z_1 = L_1(\boldsymbol{\pi}) = \langle \mathbf{r}', \boldsymbol{\pi} \rangle$.

2. For the second query, the linear form is defined by $L_2(\mathbf{x}) = \langle \widehat{\mathbf{r}}', \mathbf{x} \rangle$; the prover reveals $z_2 = L_2(\boldsymbol{\pi}) = \langle \widehat{\mathbf{r}}', \boldsymbol{\pi} \rangle$.

3. For the third query, the linear form is defined by $L_3(\mathbf{x}) = \langle \mathbf{q}, \mathbf{x} \rangle$; the prover reveals $z_3 = L_3(\boldsymbol{\pi}) = \langle \mathbf{q}, \boldsymbol{\pi} \rangle$.

The resulting compiled Hadamard LPCP, called the Π_c-*compiled Hadamard LPCP*, is presented in Figure 7.1.

We are now ready to analyze the security (i.e., completeness, zero-knowledge, and soundness) of the Π_c-compiled Hadamard LPCP.

Proposition 7.2.1. *The Π_c-compiled Hadamard LPCP (of Figure 7.1) is perfectly complete.*

Proof. If $(\mathbf{x}; \mathbf{w}) \in \mathcal{R}$ is such that $C_{\mathbf{x}}(\mathbf{w}) = 0$, then by perfect completeness of the information-theoretic Hadamard LPCP (Proposition 6.1.3), an honest prover can construct a proof string $\boldsymbol{\pi}$, such that for all verifier queries $\mathbf{r}'$, $\widehat{\mathbf{r}}'$, and $\mathbf{q}$, the prover can respond with answers $z_1 = \langle \boldsymbol{\pi}, \mathbf{r}' \rangle$, $z_2 = \langle \boldsymbol{\pi}, \widehat{\mathbf{r}}' \rangle$, and $z_3 = \langle \boldsymbol{\pi}, \mathbf{q} \rangle$ that convince the verifier to accept. Furthermore, the prover can convince the verifier that z_1, z_2, and z_3 are obtained from inner products with the proof string $\boldsymbol{\pi}$ by perfect completeness of Π_c (Proposition 7.1.4). □

Intuitively, the Π_c-compiled Hadamard LPCP is perfect honest-verifier zero-knowledge, as Π_c is perfect honest-verifier zero-knowledge (Proposition 7.1.4), meaning the verifier learns nothing but the answers z_1, z_2, z_3 to their queries from the three runs of Π_c. Assuming the witness is augmented with a dummy uniform randomly sampled entry w_0 (see Theorem 6.1.5), the verifier learns nothing from z_1, z_2, and z_3, implying that the Π_c-compiled Hadamard LPCP is perfect honest-verifier zero-knowledge. We formalize this intuition in Proposition 7.2.2.

Prover $\mathcal{P}$		**Verifier** $\mathcal{V}$
$(\mathbf{x}, C_{\mathbf{x}}; \mathbf{w}, \gamma \xleftarrow{\$} \mathbb{F})$		$\mathbf{x}, C_{\mathbf{x}}$
$\mathbf{g} \in \mathbb{G}^{s+\binom{s+1}{2}}, h, k \in \mathbb{G}$ generators of $\mathbb{G}$		$\mathbf{g}, h, k$
Compute $Q_1, \ldots, Q_s$		Compute $Q_1, \ldots, Q_s$
(following Definition 6.1.1)		(following Definition 6.1.1)
Compute values $\mathbf{y} \in \mathbb{F}^s$ of all gates		$\mathbf{r} \xleftarrow{\$} \mathbb{F}^s$
of $C_{\mathbf{x}}$ such that $C_{\mathbf{x}}(\mathbf{w}) = 0$		$\mathbf{r}' \coloneqq (\mathbf{r}_1, \ldots, \mathbf{r}_s, 0, \ldots, 0) \in \mathbb{F}^{s+\binom{s+1}{2}}$
		$\widehat{\mathbf{r}} \coloneqq (\mathbf{r}_i^2)_{1 \le i \le s} \| (2\mathbf{r}_i \mathbf{r}_j)_{1 \le i < j \le s} \in \mathbb{F}^{\binom{s+1}{2}}$
Compute $\widehat{\mathbf{y}} \coloneqq \mathbf{y} \,(\times)\, \mathbf{y} \in \mathbb{F}^{\binom{s+1}{2}}$		$\widehat{\mathbf{r}}' \coloneqq (0, \ldots, 0) \| \widehat{\mathbf{r}} \in \mathbb{F}^{s+\binom{s+1}{2}}$
Compute $\boldsymbol{\pi} \coloneqq (\mathbf{y}, \widehat{\mathbf{y}}) \in \mathbb{F}^{s+\binom{s+1}{2}}$		Let $\mathbf{q} \in \mathbb{F}^{s+\binom{s+1}{2}}$ be as in Equation (6.1)
	$P \coloneqq \mathbf{g}^{\boldsymbol{\pi}} h^{\gamma}$ $\longrightarrow$	
	$\longleftarrow$ $\mathbf{r}', \widehat{\mathbf{r}}', \mathbf{q}$	
	$z_1 \coloneqq \langle \boldsymbol{\pi}, \mathbf{r}' \rangle, z_2 \coloneqq \langle \boldsymbol{\pi}, \widehat{\mathbf{r}}' \rangle, z_3 \coloneqq \langle \boldsymbol{\pi}, \mathbf{q} \rangle$ $\longrightarrow$	
$\longrightarrow$	Run Π_c (Proposition 7.1.4) The linear form is $L_1 \colon \mathbf{p} \mapsto \langle \mathbf{p}, \mathbf{r}' \rangle$ Parameters: $(P, L_1, z_1, \mathbf{g}, h, k; \boldsymbol{\pi}, \gamma)$ **Verifies that** $z_1 = L_1(\boldsymbol{\pi})$	$\longleftarrow$
$\longrightarrow$	Run Π_c (Proposition 7.1.4) The linear form is $L_2 \colon \mathbf{p} \mapsto \langle \mathbf{p}, \widehat{\mathbf{r}}' \rangle$ Parameters: $(P, L_2, z_2, \mathbf{g}, h, k; \boldsymbol{\pi}, \gamma)$ **Verifies that** $z_2 = L_2(\boldsymbol{\pi})$	$\longleftarrow$
$\longrightarrow$	Run Π_c (Proposition 7.1.4) The linear form is $L_3 \colon \mathbf{p} \mapsto \langle \mathbf{p}, \mathbf{q} \rangle$ Parameters: $(P, L_3, z_3, \mathbf{g}, h, k; \boldsymbol{\pi}, \gamma)$ **Verifies that** $z_3 = L_3(\boldsymbol{\pi})$	$\longleftarrow$
		Accept $\iff z_2 = z_1^2 \wedge z_3 - \sum_{i=1}^{n} \mathbf{r}_i \mathbf{x}_i = 0$ and each run of Π_c succeeds

Figure 7.1: The Π_c-compiled Hadamard Linear Probabilistically Checkable Proof.

Proposition 7.2.2. *The Π_c-compiled Hadamard LPCP of Figure 7.1 is perfect honest-verifier zero-knowledge.*

Proof. By Proposition 7.1.4, there exist simulators $\mathtt{Sim}_1, \mathtt{Sim}_2, \mathtt{Sim}_3$ that output proof transcripts to the three runs of Π_c with the same probability distributions as the correct proof transcripts of the three runs respectively. Now, we can define a simulator $\mathtt{Sim}$ that runs the three simulators, producing three proof transcripts Π_1, Π_2, Π_3. The simulator outputs the concatenation of the three proof transcripts. As each simulator $\mathtt{Sim}_i$ has the same probability distribution as the corresponding proof transcript Π_i, the concatenation has the same distribution as the correct proof transcripts for the Π_c-compiled Hadamard LPCP. □

We will bound the soundness error for the Π_c-compiled Hadamard LPCP by the sum of the soundness errors from the runs of Π_c and the soundness error of the information-theoretic Hadamard LPCP. First, we prove a lemma that we will need in the soundness analysis of the Π_c-compiled Hadamard LPCP.

Lemma 7.2.3. *Let $\mu(s) = \lceil \log_2(\binom{s+1}{2} + s + 1) \rceil - 1$ be a function defined on domain $[0, \infty)$. Then,*

$$\mu(s) = \mathcal{O}(\log_2(s)).$$

Proof. Since $\binom{s+1}{2} + s + 1 = \frac{(s+1)!}{2!(s-1)!} + s + 1 = \frac{1}{2}(s+1)s + s + 1 = \mathcal{O}(s^2)$, there exist $C, s_0 > 0$ such that for all $s \geq s_0$ it holds that $\binom{s+1}{2} + s + 1 \leq Cs^2$. For $s \geq s_0$, we have

$$\begin{aligned}\mu(s) = \lceil \log_2(\tbinom{s+1}{2} + s + 1) \rceil - 1 &\leq \log_2(\tbinom{s+1}{2} + s + 1) \\ &\leq \log_2(Cs^2) = \log_2(C) + 2\log_2(s) = \mathcal{O}(\log_2(s)).\end{aligned}$$

□

Now, we can prove the following soundness error for the Π_c-compiled Hadamard LPCP.

Proposition 7.2.4. *Let $\varepsilon_{\text{LPCP}}$ be the statistical soundness error of the information-theoretic Hadamard LPCP (Theorem 6.1.5), and let ε_{Π_c} be the computational soundness error of Π_c (Proposition 7.1.6). Under the discrete logarithm assumption with respect to the group $\mathbb{G}$ (Assumption 1), the Π_c-compiled Hadamard LPCP (of Figure 7.1) for an arithmetic circuit of size s over a finite field $\mathbb{F}$ is computationally sound with soundness error $\varepsilon_{\text{LPCP}} + 3 \cdot \varepsilon_{\Pi_c} = \mathcal{O}(\log_2(s)/|\mathbb{F}|)$.*

Proof. Assume that $\forall \mathbf{w} \in \mathbb{F}^m, C_{\mathbf{x}}(\mathbf{w}) \neq 0$, and that a malicious prover $\widehat{\mathcal{P}}$ tries to convince an honest verifier $\mathcal{V}$ to accept. The probability that $\widehat{\mathcal{P}}$ succeeds in convincing $\mathcal{V}$ to accept is the soundness error. We condition this probability on the event that $\widehat{\mathcal{P}}$ convinces $\mathbf{V}$ in the information-theoretic Hadamard LPCP of Figure 6.2 (i.e., the event that the prover is able

to construct a proof string $\boldsymbol{\pi}^*$ such that the verifier accepts on $z_1 = \langle \boldsymbol{\pi}^*, \mathbf{r}' \rangle$, $z_2 = \langle \boldsymbol{\pi}^*, \widehat{\mathbf{r}}' \rangle$, and $z_3 = \langle \boldsymbol{\pi}^*, \mathbf{q} \rangle$). We denote this event by

$$\mathrm{E}_{\mathrm{LPCP}} \coloneqq \widehat{\mathcal{P}} \text{ convinces } \mathbf{V} \text{ in the information-theoretic Hadamard LPCP.}$$

We get

$$\begin{aligned}
&\mathbb{P}[\widehat{\mathcal{P}} \text{ convinces } \mathcal{V}] \\
&\quad = \mathbb{P}[\widehat{\mathcal{P}} \text{ convinces } \mathcal{V} \mid \mathrm{E}_{\mathrm{LPCP}}] \cdot \mathbb{P}[\mathrm{E}_{\mathrm{LPCP}}] + \mathbb{P}[\widehat{\mathcal{P}} \text{ convinces } \mathcal{V} \mid \neg \mathrm{E}_{\mathrm{LPCP}}] \cdot \mathbb{P}[\neg \mathrm{E}_{\mathrm{LPCP}}] \\
&\quad \leq \mathbb{P}[\mathrm{E}_{\mathrm{LPCP}}] + \mathbb{P}[\widehat{\mathcal{P}} \text{ convinces } \mathcal{V} \mid \neg \mathrm{E}_{\mathrm{LPCP}}].
\end{aligned}$$

Note that, by definition, $\mathbb{P}[\mathrm{E}_{\mathrm{LPCP}}] = \varepsilon_{\mathrm{LPCP}}$. We will argue that

$$\mathbb{P}[\widehat{\mathcal{P}} \text{ convinces } \mathcal{V} \mid \neg \mathrm{E}_{\mathrm{LPCP}}] \leq 3 \cdot \varepsilon_{\Pi_c},$$

where the 3 emerges from the fact that Π_c is run three times as a result of the three queries in the information-theoretic Hadamard LPCP.

Given $\neg \mathrm{E}_{\mathrm{LPCP}}$, the verifier can only accept if $\widehat{\mathcal{P}}$ convinces $\mathcal{V}$ of a wrong answer in at least one of the runs of Π_c. Suppose towards a contradiction that $\widehat{\mathcal{P}}$ is honest in all three runs of Π_c (i.e., $\widehat{\mathcal{P}}$ commits to some proof string $\boldsymbol{\pi}^*$ and honestly convinces $\mathcal{V}$ that z_1, z_2, and z_3 are inner products between $\boldsymbol{\pi}^*$ and the queries of $\mathcal{V}$). If $\mathcal{V}$ accepts, then $\mathbf{V}$ has accepted in the information-theoretic Hadamard LPCP, which contradicts the event $\neg \mathrm{E}_{\mathrm{LPCP}}$. Hence, $\mathbb{P}[\widehat{\mathcal{P}} \text{ convinces } \mathcal{V} \mid \neg \mathrm{E}_{\mathrm{LPCP}}]$ is bounded by the probability that $\widehat{\mathcal{P}}$ convinces $\mathcal{V}$ of a wrong answer in at least one of the runs of Π_c. We write

$$\mathrm{E}_i \coloneqq \widehat{\mathcal{P}} \text{ convinces } \mathcal{V} \text{ of a wrong answer in the } i\text{-th run of } \Pi_c,$$

for $i = 1, 2, 3$. Then, by the union bound (Lemma 3.3.5),

$$\mathbb{P}[\widehat{\mathcal{P}} \text{ convinces } \mathcal{V} \mid \neg \mathrm{E}_{\mathrm{LPCP}}] \leq \mathbb{P}[\mathrm{E}_1 \cup \mathrm{E}_2 \cup \mathrm{E}_3] \leq \mathbb{P}[\mathrm{E}_1] + \mathbb{P}[\mathrm{E}_2] + \mathbb{P}[\mathrm{E}_3] \leq 3 \cdot \varepsilon_{\Pi_c}.$$

By Proposition 7.1.6, the protocol Π_c is computationally sound with soundness error $\varepsilon_{\Pi_c} = \frac{2+2\mu}{|\mathbb{F}|}$, where $\mu = \lceil \log_2(s + \binom{s+1}{2} + 1) \rceil - 1$. Furthermore, by Theorem 6.1.5, the soundness error $\varepsilon_{\mathrm{LPCP}}$ of the information-theoretic Hadamard LPCP is bounded by $\mathcal{O}(1/|\mathbb{F}|)$. Hence,

$$\begin{aligned}
\mathbb{P}[\widehat{\mathcal{P}} \text{ convinces } \mathcal{V}] &\leq \mathbb{P}[\mathrm{E}_{\mathrm{LPCP}}] + \mathbb{P}[\widehat{\mathcal{P}} \text{ convinces } \mathcal{V} \mid \neg \mathrm{E}_{\mathrm{LPCP}}] \\
&\leq \varepsilon_{\mathrm{LPCP}} + 3 \cdot \varepsilon_{\Pi_c} \\
&\leq \mathcal{O}(1/|\mathbb{F}|) + 3 \cdot \frac{2 + 2\mu}{|\mathbb{F}|} \overset{\text{Lemma 7.2.3}}{\leq} \mathcal{O}(\log_2(s)/|\mathbb{F}|).
\end{aligned}$$

□

Note that with this computational soundness error, the Π_c-compiled Hadamard LPCP is only useful if the circuit size s is much smaller than $2^{|\mathbb{F}|}$. Say, for example, we want to have a soundness error of $\frac{1}{2}$, then we need $\log_2(s)/|\mathbb{F}| \leq \frac{1}{2}$, which is the case when $s \leq \sqrt{2^{|\mathbb{F}|}}$. Generally, the field size in arithmetic circuit satisfiability problems is of order 2^λ for some security parameter λ, and the circuit size is of order $\text{poly}(\lambda)$ [14], which means this soundness error is usable in practice.

To fully compare the Π_c-compiled Hadamard LPCP of Figure 7.1 to the LOE-compiled Hadamard LPCP of Theorem 6.2.4, we need to calculate the amount of communication between the prover and verifier. We do this in the proof of the following theorem, where we summarize all properties of the Π_c-compiled Hadamard LPCP of Figure 7.1.

Theorem 7.2.5 (Π_c-compiled Hadamard LPCP)**.** *Let $\mathbb{G}$ be a cyclic Abelian group of prime order q for which the discrete logarithm assumption holds. Let $\mathbb{F}$ be a finite field of order q. Let $\mathbf{x} \in \mathbb{F}^n$ be a common input and let $C_\mathbf{x}$ be an arithmetic circuit of size s over $\mathbb{F}$. Let $\mathbf{g} \xleftarrow{\$} \mathbb{G}^{s+\binom{s+1}{2}}$ and $h, k \xleftarrow{\$} \mathbb{G}$ be $s + \binom{s+1}{2} + 2$ generators of $\mathbb{G}$. The Π_c-compiled Hadamard LPCP (of Figure 7.1) for the relation*

$$\mathcal{R} = \{(\mathbf{x}, C_\mathbf{x}; \mathbf{w}) \mid C_\mathbf{x}(\mathbf{w}) = 0\}$$

has the following properties, where $\mu = \lceil \log_2(s + \binom{s+1}{2} + 1) \rceil - 1$, $\varepsilon_{\text{LPCP}}$ is the soundness error of the information-theoretic Hadamard LPCP, and ε_{Π_c} is the soundness error of Π_c:

- *the protocol consists of $6\mu + 18$ moves;*
- *the protocol is perfectly complete;*
- *under the discrete logarithm assumption (Assumption 1), the protocol has computational soundness error $\varepsilon_{\text{LPCP}} + 3 \cdot \varepsilon_{\Pi_c} = \mathcal{O}(\log_2(s)/|\mathbb{F}|)$;*
- *the (augmented) protocol is perfect honest-verifier zero-knowledge;*
- *the protocol communicates:*
  - *$6\lceil \log_2(s + \binom{s+1}{2} + 1) \rceil - 2$ elements of $\mathbb{G}$ and 12 elements of $\mathbb{F}$ from prover to verifier;*
  - *$3\lceil \log_2(s + \binom{s+1}{2} + 1) \rceil + 3\binom{s+1}{2} + 3s + 3$ elements[1] of $\mathbb{F}$ from verifier to prover.*

[1] Note that instead of sending the three query vectors $\mathbf{r}', \widehat{\mathbf{r}}', \mathbf{q} \in \mathbb{F}^{s+\binom{s+1}{2}}$, it suffices for the verifier to only send the vector $\mathbf{r} \in \mathbb{F}^s$, as the prover then has all information needed to construct the three queries. However, we decided to send over the three queries in full, as this more accurately resembles the general LPCP compiler that we will introduce in Section 7.4. Furthermore, we will show that by slightly altering the Hadamard LPCP, we can even reduce this communication to a single element $r \in \mathbb{F}$.

Proof. Perfect completeness, soundness, and zero-knowledge were proved in Proposition 7.2.1, Proposition 7.2.4, and Proposition 7.2.2, respectively.

The Π_c-compiled Hadamard LPCP starts with three moves to communicate the public parameters between prover and verifier. That is, the prover sends the Pedersen commitment $P \in \mathbb{G}$ to the verifier, the verifier sends the vectors $\mathbf{r'}, \widehat{\mathbf{r}}', \mathbf{q} \in \mathbb{F}^{s+\binom{s+1}{2}}$ (based on the uniform randomly sampled vector $\mathbf{r} \in \mathbb{F}^s$ and Equation (6.1)) to the prover, and the prover sends the query answers $z_1, z_2, z_3 \in \mathbb{F}$ to the verifier. Then, we have three runs of Π_c, where (by Proposition 7.1.4) each run consists of $2\mu + 5$ moves and communicates:

- $2\lceil \log_2(s + \binom{s+1}{2} + 1)\rceil - 1$ elements of $\mathbb{G}$ and 3 elements of $\mathbb{F}$ from prover to verifier;
- $\lceil \log_2(s + \binom{s+1}{2} + 1)\rceil + 1$ elements of $\mathbb{F}$ from verifier to prover.

Hence, in total, the Π_c-compiled Hadamard LPCP consists of

$$3 + 3(2\mu + 5) = 6\mu + 18$$

moves. Furthermore, as we have three runs of Π_c, the total communication from prover to verifier consists of:

$$1 + 3(2\lceil \log_2(s + \tbinom{s+1}{2} + 1)\rceil - 1) = 6\lceil \log_2(s + \tbinom{s+1}{2} + 1)\rceil - 2 \text{ elements of } \mathbb{G} \text{ and}$$
$$3 + 3 \cdot 3 = 12 \text{ elements of } \mathbb{F}.$$

Similarly, the total communication from verifier to prover consists of:

$$3(s + \tbinom{s+1}{2}) + 3(\lceil \log_2(s + \tbinom{s+1}{2} + 1)\rceil + 1) = 3\lceil \log_2(s + \tbinom{s+1}{2} + 1)\rceil + 3\tbinom{s+1}{2} + 3s + 3$$

elements of $\mathbb{F}$. □

7.3 Reducing communication with an amortization trick

At the moment, the Π_c-compiled Hadamard LPCP contains much communication from the verifier to the prover. In particular, sending over the three vectors $\mathbf{r}', \widehat{\mathbf{r}}', \mathbf{q} \in \mathbb{F}^{s+\binom{s+1}{2}}$ is quite costly. We will introduce a common trick, often called an *amortization trick*, to send only one element $r \in \mathbb{F}$, at the cost of a slightly worse soundness error. The idea is that instead of sampling $\mathbf{r} \xleftarrow{\$} \mathbb{F}^s$, the verifier samples $r \xleftarrow{\$} \mathbb{F}$ and uses the vector $\mathbf{r} \coloneqq (r, r^2, \ldots, r^s) \in \mathbb{F}^s$ in the rest of the protocol. The amortized Hadamard LPCP is presented in Figure 7.2, where the changes with regard to the non-amortized Hadamard LPCP (Figure 6.2) are emphasized in red.

Note that since the compiler Π_c only realizes the linear queries, the amortization only affects the information-theoretic part of the protocol. Intuitively, completeness and zero-knowledge are not affected, as these properties hold for the non-amortized Hadamard LPCP

for any $\mathbf{r} \xleftarrow{\$} \mathbb{F}^s$, so they also hold for a specific choice of $\mathbf{r}$ (namely $\mathbf{r} = (r, r^2, \ldots, r^s)$ for $r \xleftarrow{\$} \mathbb{F}$). Only the soundness is affected, as shown in Proposition 7.3.1.

Prover P	**Verifier V**
$(\mathbf{x}, C_{\mathbf{x}}; \mathbf{w})$	$\mathbf{x}, C_{\mathbf{x}}$
Compute $Q_1, \ldots, Q_s$	Compute $Q_1, \ldots, Q_s$
(following Definition 6.1.1)	(following Definition 6.1.1)
Compute values $\mathbf{y} \in \mathbb{F}^s$ of all gates of $C_{\mathbf{x}}$ such that $C_{\mathbf{x}}(\mathbf{w}) = 0$	
Compute $\widehat{\mathbf{y}} := \mathbf{y} \,(\times)\, \mathbf{y} \in \mathbb{F}^{\binom{s+1}{2}}$	
$\boldsymbol{\pi} := (\mathbf{y}, \widehat{\mathbf{y}}) \in \mathbb{F}^{s+\binom{s+1}{2}}$ $\longrightarrow$	
	$r \xleftarrow{\$} \mathbb{F}$, $\mathbf{r} := (r, r^2, \ldots, r^s) \in \mathbb{F}^s$
	$\mathbf{r}' := (r, r^2, \ldots, r^s, 0, \ldots, 0) \in \mathbb{F}^{s+\binom{s+1}{2}}$
	$\widehat{\mathbf{r}} := (r^{2i})_{1 \le i \le s} \| (2r^{i+j})_{1 \le i < j \le s} \in \mathbb{F}^{\binom{s+1}{2}}$
	$\widehat{\mathbf{r}}' := (0, \ldots, 0) \| \widehat{\mathbf{r}} \in \mathbb{F}^{s+\binom{s+1}{2}}$
	Let $\mathbf{q} \in \mathbb{F}^{s+\binom{s+1}{2}}$ be as in Equation (6.1)
	Query 1. $z_1 := \langle \boldsymbol{\pi}, \mathbf{r}' \rangle = \langle \mathbf{y}, \mathbf{r} \rangle = \sum_{i=1}^{s} \mathbf{y}_i r^i$
	Query 2. $z_2 := \langle \boldsymbol{\pi}, \widehat{\mathbf{r}}' \rangle = \sum_{i=1}^{s} \widehat{\mathbf{y}}_{i,i} r^{2i} + \sum_{1 \le i < j \le s} \widehat{\mathbf{y}}_{i,j} (2r^{i+j})$
	Query 3. $z_3 := \langle \boldsymbol{\pi}, \mathbf{q} \rangle = \sum_{i=1}^{s} r^i Q_i(\mathbf{y}_1, \ldots, \mathbf{y}_s) + \sum_{i=1}^{n} r^i \mathbf{x}_i$
	Accept $\iff z_2 = z_1^2 \wedge z_3 - \sum_{i=1}^{n} r^i \mathbf{x}_i = 0$

Figure 7.2: The amortized Hadamard Linear Probabilistically Checkable Proof.

Proposition 7.3.1. *The amortized Hadamard LPCP has soundness error* $2s/|\mathbb{F}| = \mathcal{O}(s/|\mathbb{F}|)$.

Proof. To analyze the soundness error of the amortized Hadamard LPCP, we need to calculate

$$\mathbb{P}\left[z_2 = z_1^2 \wedge z_3 - \sum_{i=1}^{n} r^i \mathbf{x}_i = 0\right],$$

assuming $(\mathbf{x}, C_{\mathbf{x}}) \notin L_{\mathcal{R}}$ (i.e., there is no $\mathbf{w}$ for which $C_{\mathbf{x}}(\mathbf{w}) = 0$), and assuming a cheating prover tries to convince an honest verifier to accept. Note that the following proof is almost identical to that of Proposition 6.1.4, where the only difference is that we use the Schwartz-Zippel lemma (Lemma 3.3.9) in both cases with slightly different parameters. To avoid repetition, we omit some details and focus on the essence of the proof.

We again distinguish two cases, namely whether $\widehat{\mathbf{y}} = \mathbf{y} \,(\times)\, \mathbf{y}$. The maximum of the two probabilities will be our soundness error.

Case $\widehat{\mathbf{y}} \neq \mathbf{y} \,(\times)\, \mathbf{y}$. Note that

$$\begin{aligned} z_1^2 - z_2 &= \left(\sum_{i=1}^{s} \mathbf{y}_i r^i\right)^2 - \left(\sum_{i=1}^{s} \widehat{\mathbf{y}}_{i,i} r^{2i} + \sum_{1 \le i < j \le s} \widehat{\mathbf{y}}_{i,j} (2r^{i+j})\right) \\ &= \sum_{i=1}^{s} \mathbf{y}_i^2 r^{2i} + 2 \sum_{1 \le i < j \le s} \mathbf{y}_i r^i \mathbf{y}_j r^j - \left(\sum_{i=1}^{s} \widehat{\mathbf{y}}_{i,i} r^{2i} + 2 \sum_{1 \le i < j \le s} \widehat{\mathbf{y}}_{i,j} r^{i+j}\right) \\ &= \sum_{i=1}^{s} (\mathbf{y}_i^2 - \widehat{\mathbf{y}}_{i,i}) r^{2i} + 2 \sum_{1 \le i < j \le s} (\mathbf{y}_i \mathbf{y}_j - \widehat{\mathbf{y}}_{i,j}) r^{i+j} = f(r), \end{aligned}$$

where

$$f(X) = \sum_{i=1}^{s} (\mathbf{y}_i^2 - \widehat{\mathbf{y}}_{i,i}) X^{2i} + 2 \sum_{1 \le i < j \le s} (\mathbf{y}_i \mathbf{y}_j - \widehat{\mathbf{y}}_{i,j}) X^{i+j}$$

is a univariate polynomial of degree at most $2s$. If $\widehat{\mathbf{y}} \neq \mathbf{y} \,(\times)\, \mathbf{y}$, then for at least one pair $(i, j) \in [s] \times [s]$ we have $\widehat{\mathbf{y}}_{i,j} \neq \mathbf{y}_i \mathbf{y}_j$, which means f is not the zero-polynomial. Hence, Schwartz-Zippel (Lemma 3.3.9) gives us

$$\mathbb{P}\left[z_2 = z_1^2 \wedge z_3 - \sum_{i=1}^{n} \mathbf{r}_i \mathbf{x}_i = 0\right] \le \mathbb{P}[z_1^2 - z_2 = 0] = \mathbb{P}[f(r) = 0] \overset{\text{Lemma 3.3.9}}{\le} \frac{2s}{|\mathbb{F}|}.$$

Case $\widehat{\mathbf{y}} = \mathbf{y} \,(\times)\, \mathbf{y}$. If $\widehat{\mathbf{y}} = \mathbf{y} \,(\times)\, \mathbf{y}$, then

$$z_3 - \sum_{i=1}^{n} r^i \mathbf{x}_i = \sum_{i=1}^{s} r^i Q_i(\mathbf{y}_1, \ldots, \mathbf{y}_s) = g(r),$$

where $g(X) = \sum_{i=1}^{s} Q_i(\mathbf{y}_1, \ldots, \mathbf{y}_s) X^i$ is a univariate polynomial of degree at most s. Since $(\mathbf{x}, C_{\mathbf{x}}) \notin L_{\mathcal{R}}$, we must have $Q_i(\mathbf{y}_1, \ldots, \mathbf{y}_s) \neq 0$ for at least one $i \in [s]$. Hence, g is not the

zero-polynomial, and Schwartz-Zippel (Lemma 3.3.9) gives us

$$\mathbb{P}\left[z_2 = z_1^2 \wedge z_3 - \sum_{i=1}^{n} r^i \mathbf{x}_i = 0\right] \leq \mathbb{P}\left[z_3 - \sum_{i=1}^{n} r^i \mathbf{x}_i = 0\right] = \mathbb{P}[g(r) = 0] \overset{\text{Lemma 3.3.9}}{\leq} \frac{s}{|\mathbb{F}|}.$$

The maximum of the two probabilities is $\frac{2s}{|\mathbb{F}|}$, and thus we conclude that the soundness error is $\frac{2s}{|\mathbb{F}|} = \mathcal{O}(s/|\mathbb{F}|)$. □

We can now plug the soundness error of Proposition 7.3.1 into Theorem 7.2.5; we emphasize the changes with regard to Theorem 7.2.5 in red.

Theorem 7.3.2 (Amortized Π_c-compiled Hadamard LPCP)**.** *Let $\mathbb{G}$ be a cyclic Abelian group of prime order q for which the discrete logarithm assumption holds. Let $\mathbb{F}$ be a finite field of order q. Let $\mathbf{x} \in \mathbb{F}^n$ be common input and let $C_{\mathbf{x}}$ be an arithmetic circuit of size s over $\mathbb{F}$. Let $\mathbf{g} \xleftarrow{\$} \mathbb{G}^{s+\binom{s+1}{2}}$ and $h, k \xleftarrow{\$} \mathbb{G}$ be $s + \binom{s+1}{2} + 2$ generators of $\mathbb{G}$. The amortized Π_c-compiled Hadamard LPCP for the relation*

$$\mathcal{R} = \{(\mathbf{x}, C_{\mathbf{x}}; \mathbf{w}) \mid C_{\mathbf{x}}(\mathbf{w}) = 0\}$$

has the following properties, where $\mu = \lceil \log_2(s + \binom{s+1}{2} + 1)\rceil - 1$, $\varepsilon_{\text{LPCP}}$ is the soundness error of the amortized information-theoretic Hadamard LPCP, and ε_{Π_c} is the soundness error of Π_c:

- $6\mu + 18$ *moves;*
- *the protocol is perfectly complete;*
- *under the discrete logarithm assumption, the protocol has computational soundness error $\varepsilon_{\text{LPCP}} + 3 \cdot \varepsilon_{\Pi_c} = \mathcal{O}(s/|\mathbb{F}|)$;*
- *the (augmented) protocol is perfect honest-verifier zero-knowledge;*
- *the protocol communicates:*
  - $6\lceil \log_2(s + \binom{s+1}{2} + 1)\rceil - 2$ *elements of $\mathbb{G}$ and 12 elements of $\mathbb{F}$ from prover to verifier;*
  - $3\lceil \log_2(s + \binom{s+1}{2} + 1)\rceil + 4$ *elements of $\mathbb{F}$ from verifier to prover.*

Proof. As mentioned before, the amortization only affects the information-theoretic part of the protocol. Hence, the proof mostly follows from Theorem 7.2.5. The only difference

is that the computational soundness error of the full protocol now becomes

$$\begin{aligned}\varepsilon_{\text{LPCP}} + 3 \cdot \varepsilon_{\Pi_c} &\overset{\text{Proposition 7.3.1}}{\leq} \mathcal{O}(s/|\mathbb{F}|) + 3 \cdot \varepsilon_{\Pi_c} \\ &\overset{\text{Proposition 7.1.6}}{\leq} \mathcal{O}(s/|\mathbb{F}|) + 3 \cdot \frac{2+2\mu}{|\mathbb{F}|} \\ &\overset{\text{Lemma 7.2.3}}{\leq} \mathcal{O}(s/|\mathbb{F}|) + 3 \cdot \mathcal{O}(\log_2(s)/|\mathbb{F}|) = \mathcal{O}(s/|\mathbb{F}|).\end{aligned}$$

Furthermore, the verifier now only sends 1 element of $\mathbb{F}$ to the prover (namely $r \xleftarrow{\$} \mathbb{F}$) instead of $3(s + \binom{s+1}{2})$). □

7.4 Compiling general LPCPs

The attentive reader might have noticed that all proofs related to compiling the Hadamard LPCP rarely rely on the specifics of the Hadamard LPCP, which shows that we can quite directly generalize our compiler to general LPCPs.

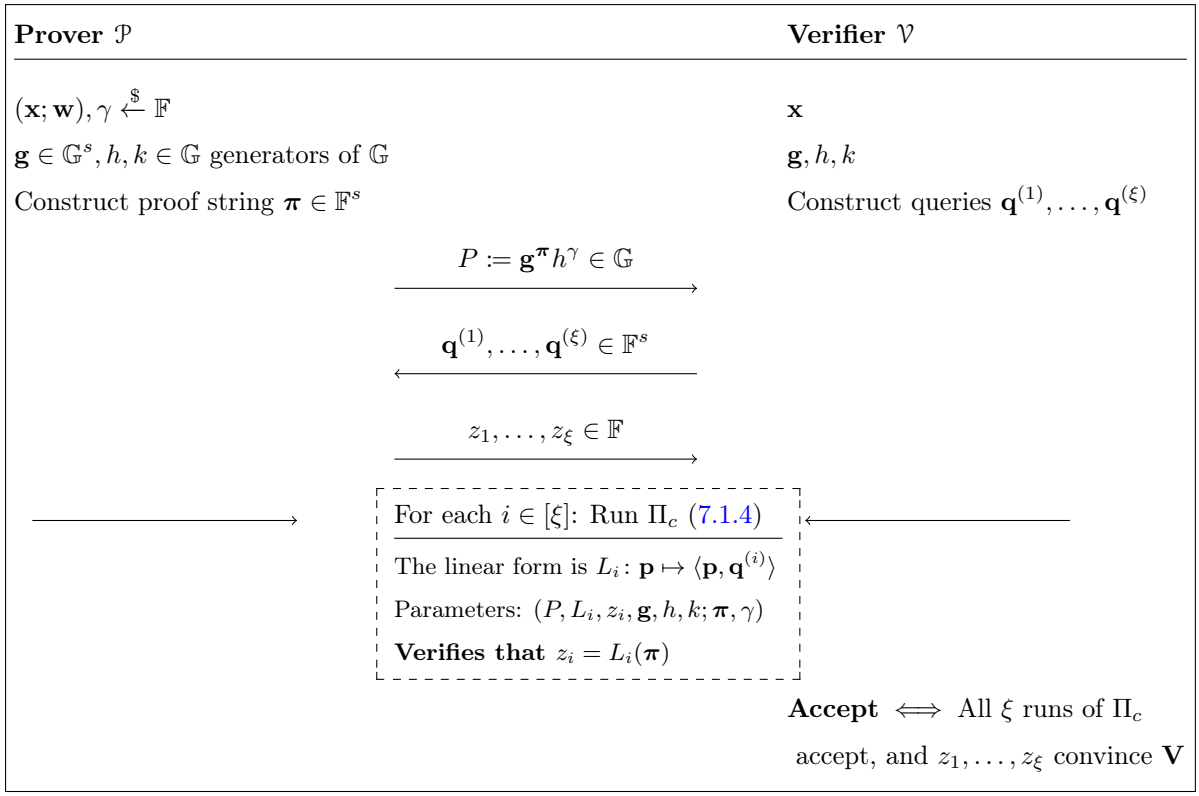

Figure 7.3: A general Π_c-compiled Linear Probabilistically Checkable Proof, where the corresponding information-theoretic LPCP consists of a proof string $\boldsymbol{\pi} \in \mathbb{F}^s$, ξ queries $\mathbf{q}^{(1)}, \ldots, \mathbf{q}^{(\xi)} \in \mathbb{F}^s$, and responses $z_i = \langle \mathbf{q}^{(i)}, \boldsymbol{\pi} \rangle$ for $i \in [\xi]$.

The main idea is that we interpret the linear query $\mathbf{q}^{(i)}$ of an LPCP as an evaluation of the linear form

$$L_i \colon \mathbf{p} \mapsto \langle \mathbf{p}, \mathbf{q}^{(i)} \rangle$$

in the proof string $\boldsymbol{\pi}$ (i.e., $z_i = L_i(\boldsymbol{\pi})$). These queries can then be compiled using the protocol Π_c. A general Π_c-compiled LPCP is presented in Figure 7.3. To avoid repetition, in the proof of Theorem 7.4.1, we omit the details that are almost identical to the proofs of Proposition 7.2.4 and Theorem 7.2.5.

Theorem 7.4.1 (General Π_c-compiled LPCP)**.** *Let $\mathbb{G}$ be a cyclic Abelian group of prime order q for which the discrete logarithm assumption holds. Let $\mathbb{F}$ be a finite field of order q and let $s \in \mathbb{N}$. Let $\mathbf{g} \xleftarrow{\$} \mathbb{G}^s$ and $h, k \xleftarrow{\$} \mathbb{G}$ be $s+2$ generators of $\mathbb{G}$. Let Π be a perfectly complete and perfect honest-verifier zero-knowledge information-theoretic LPCP for a relation $\mathcal{R} = \{(\mathbf{x}; \mathbf{w})\} \subseteq \mathbb{F}^n \times \mathbb{F}^m$. Suppose Π has a proof string $\boldsymbol{\pi} \in \mathbb{F}^s$ and has statistical soundness error $\varepsilon_{\text{LPCP}}$, and that Π consists of ξ queries $\mathbf{q}^{(1)}, \ldots, \mathbf{q}^{(\xi)} \in \mathbb{F}^s$ to the proof string oracle $\boldsymbol{\pi}$, to which the responses are $z_i = \langle \mathbf{q}^{(i)}, \boldsymbol{\pi} \rangle$ for $i \in [\xi]$.*

Then, Π can be compiled to a cryptographic proof using the protocol Π_c (which has computational soundness error ε_{Π_c}). The resulting Π_c-compiled LPCP has the following parameters, where $\mu = \lceil \log_2(s+1) \rceil - 1$:

- *the protocol consists of $3 + \xi(2\mu + 5)$ moves;*
- *the protocol is perfectly complete;*
- *the protocol is perfect honest-verifier zero-knowledge;*
- *the protocol has computational soundness error $\varepsilon_{\text{LPCP}} + \xi \cdot \varepsilon_{\Pi_c} = \varepsilon_{\text{LPCP}} + \xi \cdot \mathcal{O}(\log_2(s)/|\mathbb{F}|)$ under the discrete logarithm assumption;*
- *the protocol communicates:*
  - *$2\xi\lceil \log_2(s+1) \rceil - \xi + 1$ elements of $\mathbb{G}$ and 4ξ elements of $\mathbb{F}$ from prover to verifier;*
  - *$\xi\lceil \log_2(s+1) \rceil + (s+1)\xi$ elements of $\mathbb{F}$ from verifier to prover.*

Proof. **Number of moves and communication.** The protocol consists of 3 moves at the start, followed by ξ runs of Π_c, each consisting of $2\mu + 5$ moves (Proposition 7.1.3). Hence, in total, the protocol consists of $3 + \xi(2\mu + 5)$ moves. Furthermore, in the first three moves the communication consists of:

- 1 element of $\mathbb{G}$ and ξ elements of $\mathbb{F}$ from prover to verifier (Pedersen commitment and query responses);
- ξ elements of $\mathbb{F}^s$ (i.e., $s\xi$ elements of $\mathbb{F}$) from verifier to prover (the queries).

Recall that each run of Π_c communicates (Proposition 7.1.3):

- $2\lceil \log_2(s+1) \rceil - 1$ elements of $\mathbb{G}$ and 3 elements of $\mathbb{F}$ from prover to verifier;
- $\lceil \log_2(s+1) \rceil + 1$ elements of $\mathbb{F}$ from verifier to prover.

Since we have ξ runs of Π_c, in total, we have

$$1 + \xi(2\lceil \log_2(s+1) \rceil - 1) = 2\xi\lceil \log_2(s+1) \rceil - \xi + 1$$

elements of $\mathbb{G}$ and $\xi + 3\xi = 4\xi$ elements of $\mathbb{F}$ that are communicated from prover to verifier. Similarly, we have

$$s\xi + \xi(\lceil \log_2(s+1) \rceil + 1) = \xi\lceil \log_2(s+1) \rceil + (s+1)\xi$$

elements of $\mathbb{F}$ that are communicated from verifier to prover.

Completeness *(sketch).* Since the information-theoretic LPCP is perfectly complete, and since Π_c is perfectly complete, the Π_c-compiled LPCP is also perfectly complete.

Zero-knowledge *(sketch).* Since the information-theoretic LPCP is perfect honest-verifier zero-knowledge, the verifier learns nothing from the query answers. Similarly, since Π_c is perfect honest-verifier zero-knowledge, the verifier learns nothing from the runs of Π_c. Hence, the Π_c-compiled LPCP is perfect honest-verifier zero-knowledge.

Soundness. Assume that $\mathbf{x} \notin L_{\mathcal{R}}$, and that a malicious prover $\widehat{\mathcal{P}}$ tries to convince an honest verifier $\mathcal{V}$ to accept. We condition the probability that $\widehat{\mathcal{P}}$ succeeds in convincing $\mathcal{V}$ to accept on the event that $\widehat{\mathcal{P}}$ convinces $\mathbf{V}$ in the information-theoretic LPCP (i.e., the event that the prover is able to construct a proof string $\boldsymbol{\pi}^*$ such that the verifier accepts on the query answers z_i for $i \in [\xi]$). We denote this event by

$$\mathrm{E}_{\mathrm{LPCP}} \coloneqq \widehat{\mathcal{P}} \text{ convinces } \mathbf{V} \text{ in the information-theoretic LPCP.}$$

We get

$$\begin{aligned}
&\mathbb{P}[\widehat{\mathcal{P}} \text{ convinces } \mathcal{V}] \\
&\quad = \mathbb{P}[\widehat{\mathcal{P}} \text{ convinces } \mathcal{V} \mid \mathrm{E}_{\mathrm{LPCP}}] \cdot \mathbb{P}[\mathrm{E}_{\mathrm{LPCP}}] + \mathbb{P}[\widehat{\mathcal{P}} \text{ convinces } \mathcal{V} \mid \neg\mathrm{E}_{\mathrm{LPCP}}] \cdot \mathbb{P}[\neg\mathrm{E}_{\mathrm{LPCP}}] \\
&\quad \leq \mathbb{P}[\mathrm{E}_{\mathrm{LPCP}}] + \mathbb{P}[\widehat{\mathcal{P}} \text{ convinces } \mathcal{V} \mid \neg\mathrm{E}_{\mathrm{LPCP}}].
\end{aligned}$$

Note that, by definition, $\mathbb{P}[\mathrm{E}_{\mathrm{LPCP}}] = \varepsilon_{\mathrm{LPCP}}$. Given $\neg\mathrm{E}_{\mathrm{LPCP}}$, the verifier can only accept if $\widehat{\mathcal{P}}$ convinces $\mathcal{V}$ of a wrong answer in at least one of the runs of Π_c (otherwise $x \in L_{\mathcal{R}}$). Hence, $\mathbb{P}[\widehat{\mathcal{P}} \text{ convinces } \mathcal{V} \mid \neg\mathrm{E}_{\mathrm{LPCP}}]$ is bounded by the probability that $\widehat{\mathcal{P}}$ convinces $\mathcal{V}$ of a wrong answer in at least one of the runs of Π_c. We write

$$\mathrm{E}_i \coloneqq \widehat{\mathcal{P}} \text{ convinces } \mathcal{V} \text{ of a wrong answer in the } i\text{-th run of } \Pi_c,$$

for $i \in [\xi]$. Then, by the union bound (Lemma 3.3.5),

$$\mathbb{P}[\widehat{\mathcal{P}} \text{ convinces } \mathcal{V} \mid \neg \mathrm{E}_{\mathrm{LPCP}}] \leq \mathbb{P}\left[\bigcup_{i \in [\xi]} \mathrm{E}_i\right] \overset{\text{Lemma 3.3.5}}{\leq} \sum_{i=1}^{\xi} \mathbb{P}[\mathrm{E}_i] \leq \xi \cdot \varepsilon_{\Pi_c}.$$

By Proposition 7.1.6 and Lemma 7.2.3, we know $\varepsilon_{\Pi_c} = \mathcal{O}(\log_2(s)/|\mathbb{F}|)$. □

Remark. Note that, although the $\mathcal{O}(s\xi)$ communication from verifier to prover in Theorem 7.4.1 seems quite large, this communication can typically be reduced via an appropriate adaptation of the amortization trick that we saw in Section 7.3. However, the effectiveness of this amortization trick depends on the specific distribution that the query vectors are sampled from, so the effectiveness must be analyzed on a case-by-case basis.

8

Cryptographic compiler for quadratic queries

In this chapter, we introduce the concept of a *quadratic probabilistically checkable proof (QPCP)*. A QPCP is a strengthening of the LPCP system, where quadratic queries are allowed instead of linear queries. That is, given a finite field $\mathbb{F}$, the prover presents an oracle of the proof string $\boldsymbol{\pi} \in \mathbb{F}^s$ to the verifier, and the verifier can do queries of the form

$$q(X_1, \ldots, X_s) = c + \sum_{i=1}^{s} \beta_i X_i + \sum_{1 \le k \le m \le s} \alpha_{k,m} X_k X_m \in \mathbb{F}[X_1, \ldots, X_s]_{\le 2},$$

for constants $c, \beta_1, \ldots, \beta_s, \alpha_{1,1}, \ldots, \alpha_{s,s} \in \mathbb{F}$, to which the oracle responds with $q(\boldsymbol{\pi}_1, \ldots, \boldsymbol{\pi}_s)$. Such an oracle is represented in purple in Figure 4.4. The QPCP is shown in Figure 8.1.

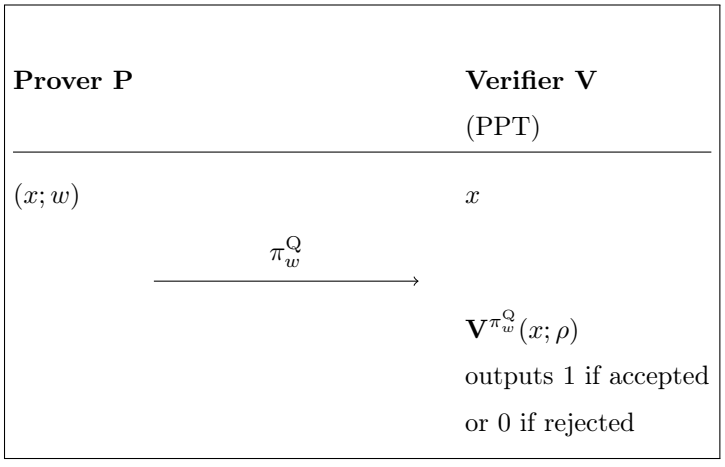

Figure 8.1: A quadratic probabilistically checkable proof with common input x and purported witness w.

We transform the Hadamard LPCP into a QPCP with a shorter proof and only a single query, after which we compile the resulting *Hadamard QPCP* using a *compressed* Σ*-protocol* [5]. The Hadamard QPCP is motivated by an important cryptographic task, namely arithmetic circuit satisfiability.

Note that, as was the case in Chapter 7, the compressed Σ-protocol in this chapter can be based on other standard cryptographic assumptions, such as the *RSA assumption* and the lattice-based *short integer solution (SIS)* assumption. For simplicity, the version in this thesis relies on the discrete logarithm assumption (Assumption 1). Also, our compiled proofs are interactive, but can be made non-interactive by applying the Fiat-Shamir transform (Section 4.5), assuming the random oracle model (Definition 4.5.2).

In Section 8.1, we introduce the information-theoretic Hadamard QPCP. Then, in Section 8.2, we introduce the compressed Σ-protocol that compiles quadratic queries. We apply this compiler in Section 8.3 to compile the Hadamard QPCP, where we also apply a typical *amortization* trick to reduce the amount of communication in the compiled Hadamard QPCP. For the sake of completeness, in Appendix A, we also compile general LPCPs using the compiler of this chapter by interpreting the linear queries of an LPCP as quadratic queries without quadratic terms. We do not expect this general compiler to be more efficient than the general compiler in Section 7.4, as using a more powerful compiler for the same information-theoretic proof system seems to be wasteful. Indeed, the analysis in Chapter 9 confirms this expectation.

8.1 The information-theoretic Hadamard QPCP

By strengthening the LPCP model to allow quadratic queries, we can replace the Hadamard LPCP by a QPCP where only one query is needed; we call this new system the *Hadamard QPCP*. The reason that this replacement is possible is that the Hadamard LPCP essentially proves that there is a $\mathbf{y} \in \mathbb{F}^s$ such that the quadratic polynomials $Q_1, \ldots, Q_s$ in Definition 6.1.1 are zero when evaluated in $\mathbf{y}$. Since quadratic queries are possible in a QPCP, this check can be done by a single query: the prover commits to the proof string $\boldsymbol{\pi} = \mathbf{y} \in \mathbb{F}^s$, and the verifier queries

$$q(X_1, \ldots, X_s) = \sum_{i=1}^{s} \mathbf{r}_i Q_i(X_1, \ldots, X_s)$$

for a sampled vector $\mathbf{r} \xleftarrow{\$} \mathbb{F}^s$ (which is a valid query for a QPCP, as each Q_i is a public quadratic polynomial). The verifier then accepts if $q(\boldsymbol{\pi}_1, \ldots, \boldsymbol{\pi}_s) = 0$. The Hadamard QPCP is presented graphically in Figure 8.2, and we prove its properties in Theorem 8.1.1.

Prover P	**Verifier V**
$(\mathbf{x}, C_{\mathbf{x}}; \mathbf{w})$	$\mathbf{x}, C_{\mathbf{x}}$
Compute $Q_1, \ldots, Q_s$	Compute $Q_1, \ldots, Q_s$
(following Definition 6.1.1)	(following Definition 6.1.1)
Compute values $\mathbf{y} \in \mathbb{F}^s$ of all gates of $C_{\mathbf{x}}$ such that $C_{\mathbf{x}}(\mathbf{w}) = 0$	
$\boldsymbol{\pi} \coloneqq \mathbf{y} \in \mathbb{F}^s \longrightarrow$	
	$\mathbf{r} \xleftarrow{\$} \mathbb{F}^s$
	$q(X_1, \ldots, X_s) \coloneqq \sum_{i=1}^{s} \mathbf{r}_i Q_i(X_1, \ldots, X_s)$
	Query 1. $z \coloneqq q(\boldsymbol{\pi}_1, \ldots, \boldsymbol{\pi}_s)$
	Accept $\iff z = 0$

Figure 8.2: The Hadamard Quadratic Probabilistically Checkable Proof.

Theorem 8.1.1 (Information-theoretic Hadamard QPCP)**.** *Let $\mathbb{F}$ be a finite field. The Hadamard QPCP (of Figure 8.2) for common input $\mathbf{x}$ and arithmetic circuit $C_{\mathbf{x}}$ of size s over $\mathbb{F}$ proves membership in the relation*

$$\mathcal{R} = \{(\mathbf{x}, C_{\mathbf{x}}; \mathbf{w}) \mid C_{\mathbf{x}}(\mathbf{w}) = 0\},$$

and has the following properties:

- *the proof string has length s;*
- *the verifier does 1 query;*
- *the protocol is perfectly complete;*
- *the protocol has statistical soundness error $1/|\mathbb{F}|$;*
- *the protocol is perfect honest-verifier zero-knowledge.*

Proof. **Completeness.** If $(\mathbf{x}; \mathbf{w}) \in \mathcal{R}$ is such that $C_{\mathbf{x}}(\mathbf{w}) = 0$, then $\exists \mathbf{y} \in \mathbb{F}^s$ consisting of the value of each gate in $C_{\mathbf{x}}(\mathbf{w})$. Let $\boldsymbol{\pi} = \mathbf{y}$, $\mathbf{r} \xleftarrow{\$} \mathbb{F}^s$, and

$$z = q(\boldsymbol{\pi}_1, \ldots, \boldsymbol{\pi}_s) = \sum_{i=1}^{s} \mathbf{r}_i Q_i(\boldsymbol{\pi}_1, \ldots, \boldsymbol{\pi}_s)$$

according to the protocol (Figure 8.2). By construction of the quadratic polynomials Q_i, we have $Q_i(\mathbf{y}_1, \ldots, \mathbf{y}_s) = 0$ for each $i \in [s]$. Hence,

$$z = \sum_{i=1}^{s} \mathbf{r}_i Q_i(\boldsymbol{\pi}_1, \ldots, \boldsymbol{\pi}_s) = \sum_{i=1}^{s} \mathbf{r}_i Q_i(\mathbf{y}_1, \ldots, \mathbf{y}_s) = 0,$$

which means the verifier will accept, and the protocol is perfectly complete.

Soundness. Assume $(\mathbf{x}, C_{\mathbf{x}}) \notin L_{\mathcal{R}}$ (i.e., there is no $\mathbf{w}$ for which $C_{\mathbf{x}}(\mathbf{w}) = 0$), and assume a cheating prover tries to convince an honest verifier to accept for this circuit. The prover needs to commit to a proof string $\boldsymbol{\pi}$. We need to calculate $\mathbb{P}[z = 0]$. Define $\mathbf{Q} \coloneqq (Q_1(\boldsymbol{\pi}_1, \ldots, \boldsymbol{\pi}_s), \ldots, Q_s(\boldsymbol{\pi}_1, \ldots, \boldsymbol{\pi}_s)) \in \mathbb{F}^s$. We know that $\mathbf{Q} \neq \mathbf{0}$ (if $\mathbf{Q} = \mathbf{0}$, then $\exists \mathbf{w}, C_{\mathbf{x}}(\mathbf{w}) = 0$ by definition of the polynomials Q_i, which contradicts our assumption). Given a sampled vector $\mathbf{r} \xleftarrow{\$} \mathbb{F}^s$ by the verifier, we now have

$$\mathbb{P}[z = 0] = \mathbb{P}\left[\sum_{i=1}^{s} \mathbf{r}_i Q_i(\boldsymbol{\pi}_1, \ldots, \boldsymbol{\pi}_s) = 0\right] = \mathbb{P}[\langle \mathbf{r}, \mathbf{Q} \rangle = 0] = \frac{1}{|\mathbb{F}|},$$

where in the last step we use Lemma 3.3.8 as $\mathbf{Q} \neq \mathbf{0}$.

Zero-knowledge. The protocol is perfect honest-verifier zero-knowledge, as the only information that the verifier gets is $q(\boldsymbol{\pi}_1, \ldots, \boldsymbol{\pi}_s) = 0$, which means a trivial simulator that always outputs 0 has the same distribution as the correct proof transcripts of the Hadamard QPCP. □

8.2 The compiler: a compressed Σ-protocol for arithmetic circuit satisfiability

In this section, we introduce the compiler for quadratic queries, which is based on a compressed Σ-protocol for arithmetic circuit satisfiability for vector commitments [5, Theorem 5].

Proposition 8.2.1 (Properties of $\Pi_{\text{cs}}^{(1)}$)**.** *Let $\mathbb{G}$ be a cyclic Abelian group of prime order q for which the discrete logarithm assumption holds. Let $\mathbb{F}$ be a finite field of order q and let $s \in \mathbb{N}$. Let $\mathbf{g} = (\mathbf{g}_1, \ldots, \mathbf{g}_s) \xleftarrow{\$} \mathbb{G}^s$ and $h \xleftarrow{\$} \mathbb{G}$ be $s+1$ generators of $\mathbb{G}$. There exists a protocol $\Pi_{\text{cs}}^{(1)}$ for the relation*

$$\mathcal{R} = \{(P \in \mathbb{G}, q \in \mathbb{F}[X_1, \ldots, X_s]_{\leq 2}; \boldsymbol{\pi} \in \mathbb{F}^s, \gamma \in \mathbb{F}) \mid P = \mathbf{g}^{\boldsymbol{\pi}} h^{\gamma}, q(\boldsymbol{\pi}) = 0\}, \text{ where}$$

$$q(X_1, \ldots, X_s) = c + \sum_{i=1}^{s} \beta_i X_i + \sum_{1 \leq k \leq m \leq s} \alpha_{k,m} X_k X_m$$

$$\text{with } c, \beta_1, \ldots, \beta_s, \alpha_{1,1}, \ldots, \alpha_{s,s} \in \mathbb{F}$$

with the following parameters, where t is the number of $\alpha_{k,m} \neq 0$ in $q(X_1, \ldots, X_s)$, and $\mu = \lceil \log_2(s + 2t + 6) \rceil - 1 \leq \lceil \log_2(s + 2\binom{s+1}{2} + 6) \rceil - 1$:

- $\Pi_{\mathrm{cs}}^{(1)}$ *consists of* $2\mu + 13$ *moves;*
- $\Pi_{\mathrm{cs}}^{(1)}$ *is perfectly complete;*
- $\Pi_{\mathrm{cs}}^{(1)}$ *is perfect honest-verifier zero-knowledge;*
- $\Pi_{\mathrm{cs}}^{(1)}$ *is computationally* $(2t+1, \max(s,4)+1, 2, 2, 3, 2, k_1, \ldots, k_\mu)$*-special sound, under the discrete logarithm assumption, where each* $k_i = 3$*;*
- $\Pi_{\mathrm{cs}}^{(1)}$ *communicates:*
  - $2\lceil \log_2(s + 2t + 6) \rceil + 4 \leq 2\lceil \log_2(s + 2\binom{s+1}{2} + 6) \rceil + 4$ *elements of* $\mathbb{G}$ *and* 12 *elements of* $\mathbb{F}$ *from prover to verifier;*
  - $\lceil \log_2(s + 2t + 6) \rceil + 5 \leq \lceil \log_2(s + 2\binom{s+1}{2} + 6) \rceil + 5$ *elements of* $\mathbb{F}$ *from verifier to prover.*

Proof. In the original formulation of this theorem [5, Theorem 5], instead of polynomials q, the relation contains general arithmetic circuits C with n inputs and s outputs, consisting of m multiplication gates (where scalar multiplications are not counted towards m). The expression inside the logarithms is $n + 2m + 6$, and the protocol is computationally $(2m + 1, \max(n, s+3) + 1, 2, 2, 3, 2, k_1, \ldots, k_\mu)$-special sound, where each $k_i = 3$. In Theorem 8.1.1, we only consider quadratic polynomials in s variables

$$q(X_1, \ldots, X_s) = c + \sum_{i=1}^{s} \beta_i X_i + \sum_{1 \leq k \leq m \leq s} \alpha_{k,m} X_k X_m \quad (c, \beta_1, \ldots, \beta_s, \alpha_{1,1}, \ldots, \alpha_{s,s} \in \mathbb{F}),$$

which can be written as arithmetic circuits with s inputs, 1 output, and $t \leq \binom{s+1}{2}$ (non-scalar) multiplication gates[1] as follows:

- for each $\alpha_{k,m} \neq 0$, we get a multiplication gate $X_k X_m$ and a scalar multiplication gate with its output and $\alpha_{k,m}$;
- for each $\beta_i \neq 0$, we get a scalar multiplication gate $\beta_i X_i$;
- we get addition gates for c and the outputs of the scalar multiplication gates.

By filling in $n = s$ (number of inputs), $s = 1$ (number of outputs), and $m = t \leq \binom{s+1}{2}$ (number of non-scalar multiplication gates) in the original theorem formulation [5, Theorem 5], we get Theorem 8.1.1, as

$$n + 2m + 6 = s + 2t + 6 \leq s + 2\binom{s+1}{2} + 6,$$

[1] For a polynomial $q(X_1, \ldots, X_s) = c + \sum_{i=1}^{s} \beta_i X_i + \sum_{1 \leq k \leq m \leq s} \alpha_{k,m} X_k X_m$, we have $\binom{s}{2}$ possible combinations $X_k X_m$ with $k \neq m$ and s possibilities for $X_k X_k$, which means we have at most $\binom{s}{2} + s$ non-scalar multiplications, so (by Lemma 3.3.4) $t \leq \binom{s}{2} + s = \binom{s+1}{2}$.

and

$$2m + 1 = 2t + 1,$$
$$\max(n, s + 3) + 1 = \max(s, 4) + 1.$$

□

Now, we determine the soundness error of the $\Pi_{\mathrm{cs}}^{(1)}$-protocol. We use that special soundness implies knowledge soundness (Lemma 7.1.5), and that knowledge soundness implies statistical soundness (Lemma 6.2.2). We prove an auxiliary lemma first.

Lemma 8.2.2. *Let $t\colon \mathbb{N} \to \mathbb{N}$ be such that $t(s) \leq \binom{s+1}{2}$ and let $\mu\colon \mathbb{N} \to [0, \infty)$ be defined by $\mu(s) = \lceil \log_2(s + 2t(s) + 6) \rceil - 1$. Then,*

$$2t(s) + \max(s, 4) + 2\mu(s) + 5 = \mathcal{O}(s^2).$$

Proof. Note that $\binom{s}{2} = \frac{s!}{2! \cdot (s-2)!} = \frac{1}{2}s(s-1) = \mathcal{O}(s^2)$. Thus, for s large enough and some constant $C \in \mathbb{R}_{>0}$, we have $\log_2(2\binom{s}{2} + 3s + 6) \leq 2\binom{s}{2} + 3s + 6 \leq 2Cs^2 + 3s + 6 = \mathcal{O}(s^2)$. Hence, for s large enough and constants $C_1, C_2 \in \mathbb{R}_{>0}$ we have

$$\begin{aligned} 2t(s) + \max(s, 4) + 2\mu(s) + 5 &\leq 2\tbinom{s}{2} + 2s + s + 4 + 2\lceil \log_2(s + 2t(s) + 6) \rceil - 2 + 5 \\ &\leq 2\tbinom{s}{2} + 3s + 2\log_2(2\tbinom{s}{2} + 3s + 6) + 2 + 7 \\ &\leq 2C_1 s^2 + 3s + 2C_2 s^2 + 9 = \mathcal{O}(s^2). \end{aligned}$$

□

Proposition 8.2.3. *Let $\mathbb{G}$ be a cyclic Abelian group of prime order q for which the discrete logarithm assumption holds. Let $\mathbb{F}$ be a finite field of order q, let $s, t \in \mathbb{N}$, and let $\mu = \lceil \log_2(s + 2t + 6) \rceil - 1$. Under the discrete logarithm assumption with respect to $\mathbb{G}$ (Assumption 1), the protocol $\Pi_{\mathrm{cs}}^{(1)}$ has computational soundness error $\mathcal{O}(s^2/|\mathbb{F}|)$.*

Proof. By Proposition 8.2.1, the protocol $\Pi_{\mathrm{cs}}^{(1)}$ is computationally $(2t + 1, \max(s, 4) + 1, 2, 2, 3, 2, k_1, \ldots, k_\mu)$-special sound, where each $k_i = 3$, under the discrete logarithm assumption. By Lemma 7.1.5, that means that $\Pi_{\mathrm{cs}}^{(1)}$ is computationally knowledge sound with knowledge error

$$\kappa \leq \frac{2t + \max(s, 4) + 1 + 1 + 2 + 1 + 2\mu}{|\mathbb{F}|} = \frac{2t + \max(s, 4) + 2\mu + 5}{|\mathbb{F}|} \overset{\text{Lemma 8.2.2}}{=} \mathcal{O}(s^2/|\mathbb{F}|).$$

Lemma 6.2.2 then implies that $\Pi_{\mathrm{cs}}^{(1)}$ is computationally sound with soundness error $\mathcal{O}(s^2/|\mathbb{F}|)$.

□

8.3 Compiling the Hadamard QPCP and reducing communication

In this section, we compile the Hadamard QPCP by writing the query $q(X_1, \ldots, X_s)$ as an arithmetic circuit, to which we can apply the $\Pi_{\text{cs}}^{(1)}$-compiler (Proposition 8.2.1). The resulting protocol, which we call the $\Pi_{\text{cs}}^{(1)}$-*compiled Hadamard QPCP*, is presented in Figure 8.3. We will also apply the amortization trick to reduce the communication in exchange for a slightly worse soundness error.

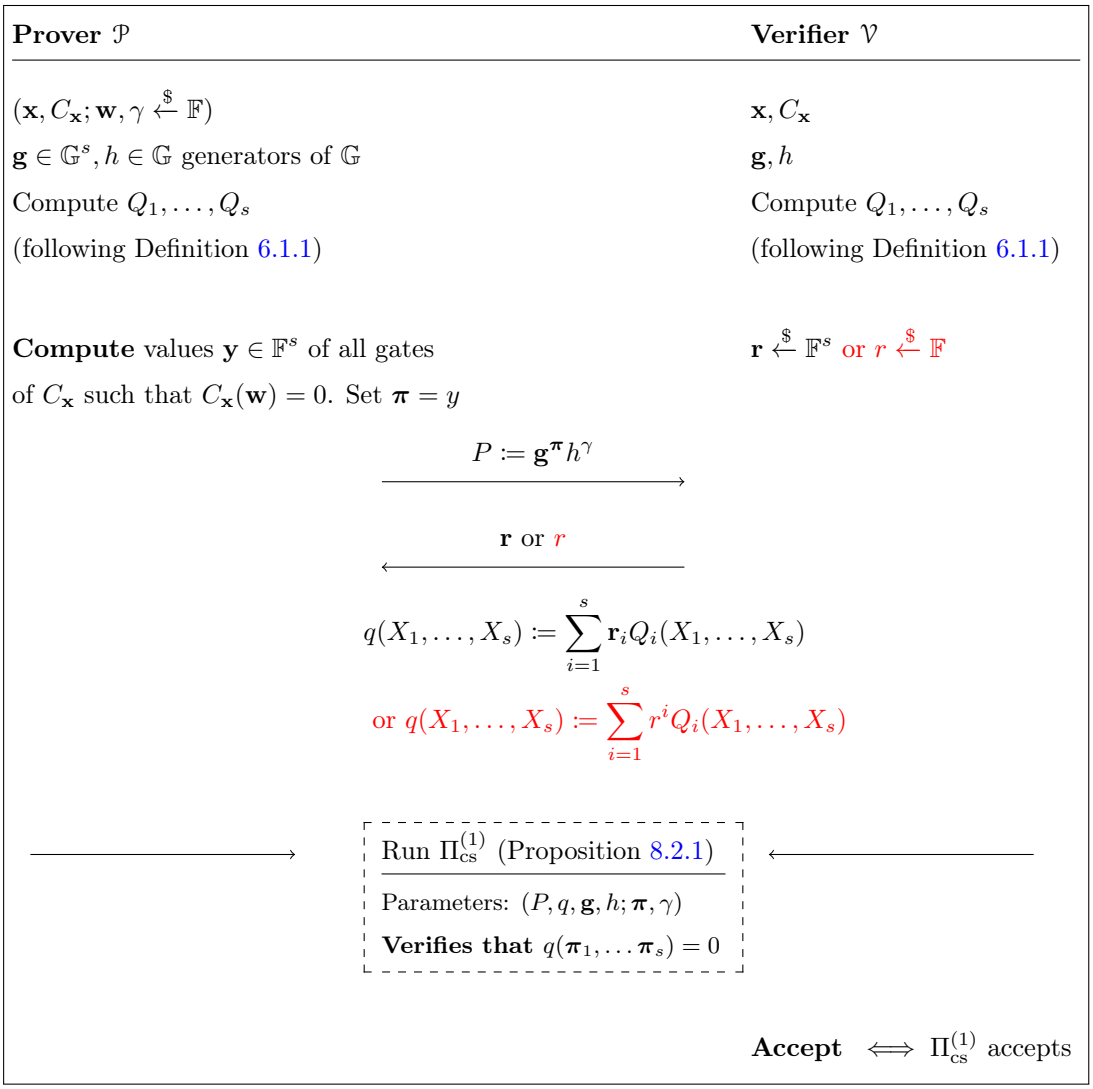

Figure 8.3: The $\Pi_{\text{cs}}^{(1)}$-compiled Hadamard Quadratic Probabilistically Checkable Proof, where the amortized version is represented by the changes in red.

We start with the soundness error for the non-amortized version of the $\Pi_{\text{cs}}^{(1)}$-compiled

Hadamard QPCP.

Proposition 8.3.1. *Let $\varepsilon_{\text{QPCP}}$ be the statistical soundness error of the information-theoretic Hadamard QPCP (Theorem 8.1.1), and let $\varepsilon_{\Pi_{\text{cs}}^{(1)}}$ be the computational soundness error of $\Pi_{\text{cs}}^{(1)}$ (Proposition 8.2.3). Under the discrete logarithm assumption (Assumption 1) with respect to the group $\mathbb{G}$, the $\Pi_{\text{cs}}^{(1)}$-compiled Hadamard QPCP (of Figure 8.3) for an arithmetic circuit of size s over a finite field $\mathbb{F}$ has computational soundness error $\varepsilon_{\text{QPCP}} + \varepsilon_{\Pi_{\text{cs}}^{(1)}} = \mathcal{O}(s^2/|\mathbb{F}|)$.*

Proof. Assume that $\forall \mathbf{w} \in \mathbb{F}^m, C_{\mathbf{x}}(\mathbf{w}) \neq 0$, and that a malicious prover $\widehat{\mathcal{P}}$ tries to convince an honest verifier $\mathcal{V}$ to accept. We condition the soundness error (i.e., the probability that $\widehat{\mathcal{P}}$ succeeds in convincing $\mathcal{V}$ to accept) on the event that $\widehat{\mathcal{P}}$ convinces $\mathbf{V}$ in the information-theoretic Hadamard QPCP of Figure 8.2 (i.e., the event that the prover is able to construct a proof string $\boldsymbol{\pi}^*$ such that the verifier accepts on $z = q(\boldsymbol{\pi}^*)$, where $q(X_1, \ldots, X_s) = \sum_{i=1}^{s} \mathbf{r}_i Q_i(X_1, \ldots, X_s)$). We denote this event by

$$\text{E}_{\text{QPCP}} \coloneqq \widehat{\mathcal{P}} \text{ convinces } \mathbf{V} \text{ in the information-theoretic Hadamard QPCP.}$$

We get

$$\begin{aligned}
&\mathbb{P}[\widehat{\mathcal{P}} \text{ convinces } \mathcal{V}] \\
&\quad = \mathbb{P}[\widehat{\mathcal{P}} \text{ convinces } \mathcal{V} \mid \text{E}_{\text{QPCP}}] \cdot \mathbb{P}[\text{E}_{\text{QPCP}}] + \mathbb{P}[\widehat{\mathcal{P}} \text{ convinces } \mathcal{V} \mid \neg\text{E}_{\text{QPCP}}] \cdot \mathbb{P}[\neg\text{E}_{\text{QPCP}}] \\
&\quad \leq \mathbb{P}[\text{E}_{\text{QPCP}}] + \mathbb{P}[\widehat{\mathcal{P}} \text{ convinces } \mathcal{V} \mid \neg\text{E}_{\text{QPCP}}].
\end{aligned}$$

Note that $\mathbb{P}[\text{E}_{\text{QPCP}}] = \varepsilon_{\text{QPCP}} = 1/|\mathbb{F}|$ (Theorem 8.1.1). Similarly,

$$\mathbb{P}[\widehat{\mathcal{P}} \text{ convinces } \mathcal{V} \mid \neg\text{E}_{\text{QPCP}}] = \varepsilon_{\Pi_{\text{cs}}^{(1)}},$$

as the event $\neg\text{E}_{\text{QPCP}}$ implies that the verifier can only accept if $\widehat{\mathcal{P}}$ convinces $\mathcal{V}$ that $q(\boldsymbol{\pi}) = 0$ in $\Pi_{\text{cs}}^{(1)}$, even though in reality $q(\boldsymbol{\pi}) \neq 0$ (if $q(\boldsymbol{\pi}) = 0$ in reality, then E_{QPCP} holds). By Proposition 8.2.3,

$$\mathbb{P}[\widehat{\mathcal{P}} \text{ convinces } \mathcal{V} \mid \neg\text{E}_{\text{QPCP}}] = \varepsilon_{\Pi_{\text{cs}}^{(1)}} = \mathcal{O}(s^2/|\mathbb{F}|).$$

Hence,

$$\begin{aligned}
\mathbb{P}[\widehat{\mathcal{P}} \text{ convinces } \mathcal{V}] &\leq \mathbb{P}[\text{E}_{\text{QPCP}}] + \mathbb{P}[\widehat{\mathcal{P}} \text{ convinces } \mathcal{V} \mid \neg\text{E}_{\text{QPCP}}] \\
&= \varepsilon_{\text{QPCP}} + \varepsilon_{\Pi_{\text{cs}}^{(1)}} \leq 1/|\mathbb{F}| + \mathcal{O}(s^2/|\mathbb{F}|) = \mathcal{O}(s^2/|\mathbb{F}|).
\end{aligned}$$

□

Recall that in arithmetic circuit satisfiability problems, we generally work with finite fields of order $|\mathbb{F}| \approx 2^\lambda$ and circuit sizes of size $s \approx \text{poly}(\lambda)$ for some security parameter λ [14]. Hence, the soundness error of the $\Pi_{\text{cs}}^{(1)}$-compiled Hadamard QPCP is relatively small.

Note that the verifier sends $\mathbf{r} \in \mathbb{F}^s$ to the prover. Using the amortization trick that we used in Section 7.3, we can reduce this communication to a single element $r \in \mathbb{F}$ at the cost of a slightly worse soundness error. The amortized version of the $\Pi_{\text{cs}}^{(1)}$-compiled Hadamard QPCP is presented in Figure 8.3 in red. The query of the verifier now becomes $q(X_1, \ldots, X_s) = \sum_{i=1}^{s} r^i Q_i(X_1, \ldots, X_s)$. The amortization does not affect completeness of the information-theoretic protocol, as we still have $Q_i(\mathbf{y}_1, \ldots, \mathbf{y}_s) = 0$ for all $i \in [s]$, which means $q(\mathbf{y}_1, \ldots, \mathbf{y}_s) = 0$. Furthermore, zero-knowledge is not affected, as the simulator that outputs zero is still valid. The amortization only affects soundness as follows.

Proposition 8.3.2. *Let $\mathbb{F}$ be a finite field. The amortized Hadamard QPCP for common input $\mathbf{x}$ and arithmetic circuit $C_{\mathbf{x}}$ of size s over $\mathbb{F}$ proves membership in the relation*

$$\mathcal{R} = \{(\mathbf{x}, C_{\mathbf{x}}; \mathbf{w}) \mid C_{\mathbf{x}}(\mathbf{w}) = 0\},$$

and has statistical soundness error $s/|\mathbb{F}|$.

Furthermore, the amortized $\Pi_{\text{cs}}^{(1)}$-compiled Hadamard QPCP (of Figure 8.3 in red) has computational soundness error $\mathcal{O}(s^2/|\mathbb{F}|)$ under the discrete logarithm assumption with respect to the group $\mathbb{G}$.

Proof. Assume $(\mathbf{x}, C_{\mathbf{x}}) \notin L_{\mathcal{R}}$. A cheating prover constructs a proof string $\boldsymbol{\pi}^*$ and tries to convince an honest verifier to accept. The verifier samples $r \xleftarrow{\$} \mathbb{F}$, queries $q(X_1, \ldots, X_s) = \sum_{i=1}^{s} r^i Q_i(X_1, \ldots, X_s)$, and receives the oracle response

$$z = q(\boldsymbol{\pi}_1^*, \ldots, \boldsymbol{\pi}_s^*) = \sum_{i=1}^{s} r^i Q_i(\boldsymbol{\pi}_1^*, \ldots, \boldsymbol{\pi}_s^*).$$

The probability that the cheating prover succeeds is

$$\mathbb{P}[z = 0] = \mathbb{P}\left[\sum_{i=1}^{s} r^i Q_i(\boldsymbol{\pi}_1^*, \ldots, \boldsymbol{\pi}_s^*) = 0\right] = \mathbb{P}[g(r) = 0],$$

where $g(X) = \sum_{i=1}^{s} X^i Q_i(\boldsymbol{\pi}_1^*, \ldots, \boldsymbol{\pi}_s^*)$. Note that we must have $Q_i(\boldsymbol{\pi}_1^*, \ldots, \boldsymbol{\pi}_s^*) \neq 0$ for at least one $i \in [s]$ (otherwise $(\mathbf{x}, C_{\mathbf{x}}) \in L_{\mathcal{R}}$). Hence, g is not the zero-polynomial, which means we can apply the Schwartz-Zippel lemma (Lemma 3.3.9). Since g has degree s, and r is sampled from $\mathbb{F}$, Schwartz-Zippel says that $\mathbb{P}[g(r) = 0] \leq s/|\mathbb{F}|$. Thus, the soundness error of the information-theoretic amortized Hadamard QPCP is $s/|\mathbb{F}|$.

The proof of the computational soundness error of the amortized $\Pi_{\text{cs}}^{(1)}$-compiled Hadamard QPCP is completely analogous to the proof of the non-amortized computational soundness error (Proposition 8.3.1), with the only difference being that $\varepsilon_{\text{QPCP}} = s/|\mathbb{F}|$ instead of $1/|\mathbb{F}|$. The computational soundness error of the amortized compiled version then becomes

$$\varepsilon_{\text{QPCP}} + \varepsilon_{\Pi_{\text{cs}}^{(1)}} = s/|\mathbb{F}| + \mathcal{O}(s^2/|\mathbb{F}|) = \mathcal{O}(s^2/|\mathbb{F}|).$$

□

We see that the amortization trick is very useful for the Hadamard QPCP: we reduce the amount of communication, while (asymptotically) achieving the same soundness error. We will now prove the remaining properties of the $\Pi_{\mathrm{cs}}^{(1)}$-compiled Hadamard QPCP, where we indicate the changes for the amortized version in red.

Theorem 8.3.3 ($\Pi_{\mathrm{cs}}^{(1)}$-compiled Hadamard QPCP)**.** *Let $\mathbb{G}$ be a cyclic Abelian group of prime order q for which the discrete logarithm assumption holds. Let $\mathbb{F}$ be a finite field of order q. Let $\mathbf{x} \in \mathbb{F}^n$ be common input and let $C_{\mathbf{x}}$ be an arithmetic circuit of size s over $\mathbb{F}$. Let $\mathbf{g} \xleftarrow{\$} \mathbb{G}^s$ and $h \xleftarrow{\$} \mathbb{G}$ be $s+1$ generators of $\mathbb{G}$. The (amortized) $\Pi_{\mathrm{cs}}^{(1)}$-compiled Hadamard QPCP (of Figure 8.3) for the relation*

$$\mathcal{R} = \{(\mathbf{x}, C_{\mathbf{x}}; \mathbf{w}) \mid C_{\mathbf{x}}(\mathbf{w}) = 0\}$$

has the following properties, where t is the number of non-zero quadratic terms in the query $q(X_1, \ldots, X_s)$, and $\mu = \lceil \log_2(s + 2t + 6) \rceil - 1 \leq \lceil \log_2(s + 2\binom{s+1}{2} + 6) \rceil - 1$:

- *the protocol consists of $2\mu + 15$ moves;*
- *the protocol is perfectly complete;*
- *the protocol is perfect honest-verifier zero-knowledge;*
- *the protocol has computational soundness error $\mathcal{O}(s^2/|\mathbb{F}|)$ under the discrete logarithm assumption;*
- *the protocol communicates:*
  - *$2\lceil \log_2(s + 2t + 6) \rceil + 5 \leq 2\lceil \log_2(s + 2\binom{s+1}{2} + 6) \rceil + 5$ elements of $\mathbb{G}$ and 12 elements of $\mathbb{F}$ from prover to verifier;*
  - *$\lceil \log_2(s+2t+6) \rceil + s + 5 \leq \lceil \log_2(s+2\binom{s+1}{2}+6) \rceil + s + 5$ or $\lceil \log_2(s+2t+6) \rceil + 6$ elements of $\mathbb{F}$ from verifier to prover.*

Proof. **Number of moves and communication.** The $\Pi_{\mathrm{cs}}^{(1)}$-compiled Hadamard QPCP starts with *two moves*, where the prover sends the Pedersen commitment $P \in \mathbb{G}$ to the verifier, and the verifier sends $\mathbf{r} \in \mathbb{F}^s$ or $r \in \mathbb{F}$ to the prover. After that, the protocol $\Pi_{\mathrm{cs}}^{(1)}$ is run, which consists of $2\mu + 13$ moves, and which communicates (Proposition 8.2.1):

- $2\lceil \log_2(s + 2t + 6) \rceil + 4 \leq 2\lceil \log_2(s + 2\binom{s+1}{2} + 6) \rceil + 4$ elements of $\mathbb{G}$ and 12 elements of $\mathbb{F}$ from prover to verifier;
- $\lceil \log_2(s + 2t + 6) \rceil + 5 \leq \lceil \log_2(s + 2\binom{s+1}{2} + 6) \rceil + 5$ elements of $\mathbb{F}$ from verifier to prover.

Hence, the $\Pi_{\mathrm{cs}}^{(1)}$-compiled Hadamard QPCP consists of $2+2\mu+13 = 2\mu+15$ moves, and communicates

- $1+2\lceil\log_2(s+2t+6)\rceil+4 \leq 2\lceil\log_2(s+2\binom{s+1}{2}+6)\rceil+5$ elements of $\mathbb{G}$ and 12 elements of $\mathbb{F}$ from prover to verifier;
- s or 1 element(s) of $\mathbb{F}$ plus $\lceil\log_2(s+2t+6)\rceil+5 \leq \lceil\log_2(s+2\binom{s+1}{2}+6)\rceil+5$ elements of $\mathbb{F}$ from verifier to prover.

Completeness. If $C_{\mathbf{x}}(\mathbf{w}) = 0$, then by perfect completeness of the (amortized) information-theoretic Hadamard QPCP (Theorem 8.1.1), an honest prover can construct a proof string $\boldsymbol{\pi} \in \mathbb{F}^s$, such that for all verifier queries $q(X_1, \ldots, X_s)$, the prover can respond with 0. By perfect completeness of $\Pi_{\mathrm{cs}}^{(1)}$ (Proposition 8.2.1), the prover can furthermore prove that this 0 is obtained by evaluating $q(\boldsymbol{\pi}_1, \ldots, \boldsymbol{\pi}_s)$.

Zero-knowledge. Since the (amortized) information-theoretic Hadamard QPCP is perfect honest-verifier zero-knowledge (Theorem 8.1.1), and since $\Pi_{\mathrm{cs}}^{(1)}$ is perfect honest-verifier zero-knowledge (Proposition 8.2.1), the compiled version is also perfect honest-verifier zero-knowledge. Note that a more formal proof would be fully analogous to the proof of Proposition 7.2.2; we omit the details here.

Soundness was proven in Proposition 8.3.2. □

9

Comparing the compilers

In this chapter, we discuss our results by comparing the cryptographic compilers for LPCPs that we developed in this thesis. For realizing LPCPs, the general aim is for the prover to provide (to the verifier) inner products between the queries and the proof string, without revealing any other information about the proof string. In other words, the prover needs to be able to give a *zero-knowledge proof* that some value is the result of the inner product between the proof string and a query of the verifier. A cryptographic compiler is essentially such a zero-knowledge proof system that preferably relies on a standard cryptographic assumption.

The cryptographic compilers discussed in this thesis can be classified into three groups, of which the latter two constitute our own contributions:

- the compilers relying on *linear-only encryption* (LOE), which is the heavy cryptographic machinery that the currently best LPCP compiler relies upon;
- the compilers relying on the compressed Σ-protocol Π_c: the linear queries of an LPCP are interpreted as linear form evaluations, and Π_c provides a zero-knowledge proof of the validity of these evaluations;
- the compilers relying on the compressed Σ-protocol $\Pi_{\text{cs}}^{(1)}$: an LPCP is transformed into a QPCP first, after which the quadratic queries are interpreted as evaluations of arithmetic circuits, for which $\Pi_{\text{cs}}^{(1)}$ provides a zero-knowledge proof of validity.

We will compare these compilers in the context of the Hadamard LPCP (Theorem 6.1.5) and general LPCPs.

For the Hadamard LPCP, we will compare the linear-only encryption (LOE) compiled Hadamard LPCP (Theorem 6.2.4), the (amortized) Π_c-compiled Hadamard LPCP

(Theorem 7.2.5, Theorem 7.3.2), the (amortized) $\Pi_{\text{cs}}^{(1)}$-compiled Hadamard QPCP (Theorem 8.3.3), and the $\Pi_{\text{cs}}^{(1)}$-compiled Hadamard LPCP (Theorem A.2.1) in Section 9.1. Note that all these proof systems are perfectly complete and perfect honest-verifier zero-knowledge, so we will not consider those properties in the comparison.

For general LPCPs, we will compare the general LOE-compiled LPCP (Theorem 6.2.3), the general Π_c-compiled LPCP (Theorem 7.4.1), and the general $\Pi_{\text{cs}}^{(1)}$-compiled LPCP (Theorem A.1.1) in Section 9.2. These proof systems are also perfectly complete and perfect honest-verifier zero-knowledge, assuming the underlying information-theoretic proof systems have these properties as well.

Finally, in Section 9.3, we will analyze how our best compiled Hadamard LPCP performs against the direct usage of a compressed Σ-protocol for arithmetic circuit satisfiability.

9.1 Comparison for the Hadamard LPCP

To aid the reader, we present the parameters of the information-theoretic Hadamard LPCP and Hadamard QPCP for arithmetic circuits of size s over a finite field $\mathbb{F}$ in Table 9.1 (where $|\boldsymbol{\pi}|$ is the length of the proof string and ε is the soundness error). The main difference is that the Hadamard QPCP only needs a single query and a shorter proof string of length s, whereas the Hadamard LPCP needs three queries and a proof string of length $s + \binom{s+1}{2}$. Below, we will see how the simpler and more-powerful QPCP positively affects the performance of the cryptographically compiled proof.

Table 9.1: The information-theoretic Hadamard LPCP and Hadamard QPCP for arithmetic circuits of size s over the finite field $\mathbb{F}$.

	Hadamard LPCP (Theorem 6.1.5)	**Hadamard QPCP (Theorem 8.1.1)**
#queries	3	1
$\lvert\boldsymbol{\pi}\rvert$	$s + \binom{s+1}{2}$	s
ε	$\mathcal{O}(1/\lvert\mathbb{F}\rvert)$	$\mathcal{O}(1/\lvert\mathbb{F}\rvert)$
soundness assumptions	Existence of linear-only oracles	Existence of quadratic-only oracles
non-interactive?	YES	YES

Before comparing the three compiler classes when applied to the Hadamard LPCP, recall that we applied the protocol $\Pi_{\text{cs}}^{(1)}$ in two ways to compile the Hadamard LPCP: we interpreted the three linear queries of the information-theoretic LPCP as quadratic queries

(without quadratic terms) and directly applied $\Pi_{\text{cs}}^{(1)}$ to this LPCP (Appendix A.2), and we directly compiled the single query in the Hadamard QPCP using $\Pi_{\text{cs}}^{(1)}$ (Section 8.3). The resulting parameters of these two proofs are presented in Table 9.2, where the changes for the amortized $\Pi_{\text{cs}}^{(1)}$-compiled Hadamard QPCP are presented in red[1].

Table 9.2: The $\Pi_{\text{cs}}^{(1)}$-compiled Hadamard LPCP and the (amortized) $\Pi_{\text{cs}}^{(1)}$-compiled Hadamard QPCP for arithmetic circuits of size s over a finite field $\mathbb{F}$ of prime order q.

	$\Pi_{\text{cs}}^{(1)}$-compiled Hadamard LPCP (Theorem A.2.1)	**$\Pi_{\text{cs}}^{(1)}$-compiled Hadamard QPCP (Theorem 8.3.3)**
#moves	$6\lceil\log_2(s+\binom{s+1}{2}+6)\rceil+36$	$2\lceil\log_2(s+2t+6)\rceil+13$
communication $\mathcal{P}\to\mathcal{V}$	$6\lceil\log_2(s+\binom{s+1}{2}+6)\rceil+13$ in $\mathbb{G}$ 39 in $\mathbb{F}$	$2\lceil\log_2(s+2t+6)\rceil+5$ in $\mathbb{G}$ 12 in $\mathbb{F}$
communication $\mathcal{V}\to\mathcal{P}$	$3\lceil\log_2(s+\binom{s+1}{2}+6)\rceil+3\binom{s+1}{2}+3s+15$ in $\mathbb{F}$	$\lceil\log_2(s+2t+6)\rceil+s+5$ (or $\lceil\log_2(s+2t+6)\rceil+6$) in $\mathbb{F}$
ε	$\mathcal{O}(s^2/\lvert\mathbb{F}\rvert)$	$O(s^2/\lvert\mathbb{F}\rvert)$
soundness assumptions	DLOG, RSA, or SIS	DLOG, RSA, or SIS
non-interactive	NO (Fiat-Shamir in ROM)	NO (Fiat-Shamir in ROM)

When we compare the $\Pi_{\text{cs}}^{(1)}$-compiled Hadamard LPCP to the $\Pi_{\text{cs}}^{(1)}$-compiled Hadamard QPCP, we see that the QPCP consists of fewer moves (as $t \leq \binom{s+1}{2}$), less communication, and the same soundness error as the LPCP version. Roughly, the number of moves and the amount of communication for the LPCP are three times as high as those for the QPCP. Hence, ad hoc replacement of linear queries in LPCPs for more clever quadratic queries seems to be useful. Most of the parameters, however, are asymptotically similar, with the only exception being that the LPCP verifier needs to communicate $\mathcal{O}(s^2)$ elements of $\mathbb{F}$ to the prover (since $\binom{s+1}{2} = \frac{1}{2}(s+1)s = \mathcal{O}(s^2)$), whereas the (non-amortized) QPCP verifier needs to communicate only $\mathcal{O}(s)$ elements of $\mathbb{F}$ to the prover. This difference in communication is also the result of the clever quadratic query in the Hadamard QPCP.

Since the $\Pi_{\text{cs}}^{(1)}$-compiled Hadamard QPCP performs better than the $\Pi_{\text{cs}}^{(1)}$-compiled Hadamard LPCP in every aspect, we omit the $\Pi_{\text{cs}}^{(1)}$-compiled Hadamard LPCP from our further analysis, and instead we compare the other two compiler classes to the $\Pi_{\text{cs}}^{(1)}$-compiled Hadamard QPCP in Table 9.3.

First, we will compare the $\Pi_{\text{cs}}^{(1)}$-compiled Hadamard QPCP to the Π_c-compiled Hadamard LPCP. We notice that, on first sight, both the non-amortized and amortized version of the

[1] As the $\Pi_{\text{cs}}^{(1)}$-compiled Hadamard QPCP has better parameters in all aspects, and since amortization only affects communication from verifier to prover and soundness, we decided to omit the amortization trick for the $\Pi_{\text{cs}}^{(1)}$-compiled Hadamard LPCP in this thesis.

Table 9.3: Three cryptographically compiled (amortized) Hadamard LPCPs for arithmetic circuits of size s over a finite field $\mathbb{F}$ of prime order q, where q scales with a security parameter λ. For the QPCP, $t \leq \binom{s+1}{2}$ is the number of non-zero quadratic terms in the query $q(X_1, \ldots, X_s)$.

Hadamard	**LOE-compiled LPCP (Theorem 6.2.4)**	**Π_c-compiled LPCP (Theorem 7.2.5, Theorem 7.3.2)**	**$\Pi_{cs}^{(1)}$-compiled QPCP (Theorem 8.3.3)**
$\lvert\boldsymbol{\pi}\rvert$	$\text{poly}(\lambda)$	-	-
$\lvert\sigma\rvert$	$\text{poly}(\lambda) \cdot (s + \binom{s+1}{2})$	-	-
#moves	1	$6\lceil\log_2(s + \binom{s+1}{2} + 1)\rceil + 12$	$2\lceil\log_2(s + 2t + 6)\rceil + 13$
communication $\mathcal{P} \to \mathcal{V}$	-	$6\lceil\log_2(s + \binom{s+1}{2} + 1)\rceil - 2$ in $\mathbb{G}$ 12 in $\mathbb{F}$	$2\lceil\log_2(s + 2t + 6)\rceil + 5$ in $\mathbb{G}$ 12 in $\mathbb{F}$
communication $\mathcal{V} \to \mathcal{P}$	-	$3\lceil\log_2(s + \binom{s+1}{2} + 1)\rceil + 3\binom{s+1}{2} + 3s + 3$ (or $3\lceil\log_2(s + \binom{s+1}{2} + 1)\rceil + 4$) in $\mathbb{F}$	$\lceil\log_2(s + 2t + 6)\rceil + s + 5$ (or $\lceil\log_2(s + 2t + 6)\rceil + 6$) in $\mathbb{F}$
ε	$\mathcal{O}(1/\lvert\mathbb{F}\rvert) + \text{negl}(\lambda)$	$\mathcal{O}(\log_2(s)/\lvert\mathbb{F}\rvert)$ (or $\mathcal{O}(s/\lvert\mathbb{F}\rvert)$)	$\mathcal{O}(s^2/\lvert\mathbb{F}\rvert)$
soundness assumptions	LOE schemes, CRS model	DLOG, RSA, or SIS	DLOG, RSA, or SIS
non-interactive	YES	NO (Fiat-Shamir in ROM)	NO (Fiat-Shamir in ROM)

Π_c-compiled LPCP seems to achieve a much better soundness error (namely $\mathcal{O}(\log_2(s)/|\mathbb{F}|)$ and $\mathcal{O}(s/|\mathbb{F}|)$ respectively) than the $\Pi_{cs}^{(1)}$-compiled Hadamard QPCP (which achieves a soundness error of $\mathcal{O}(s^2/|\mathbb{F}|)$ for both versions). However, recall that in arithmetic circuit related problems, we usually work with circuit sizes of order $s = \text{poly}(\lambda)$ over finite fields of order 2^λ for some security parameter λ [14]. This means that all three soundness errors are small enough to be workable[1], and therefore we can afford the slightly worse soundness error of $\mathcal{O}(s^2/|\mathbb{F}|)$ if, in exchange, we reduce the amount of communication. Since for the Π_c-compiled Hadamard LPCP the amortization trick affects the soundness error only slightly, and since the trick reduces the amount of communication for both the Π_c and $\Pi_{cs}^{(1)}$ compiled versions, we will only compare the amortized versions below.

When comparing the amount of communication and the number of moves of the amortized $\Pi_{cs}^{(1)}$-compiled Hadamard QPCP to those of the amortized Π_c-compiled Hadamard LPCP, we find that the QPCP performs better, although the exact analysis mostly depends on the number of quadratic terms t in the query of the Hadamard QPCP: recall that the Hadamard QPCP consists of a single query $q(X_1, \ldots, X_s) = \sum_{i=1}^{s} r^i Q_i(X_1, \ldots, X_s)$ (where s is the size of the arithmetic circuit, r is sampled uniformly at random from $\mathbb{F}$, and $Q_1, \ldots, Q_s$ are the quadratic polynomials for which we want to prove that there exists a $\mathbf{y} \in \mathbb{F}^s$ such that $Q_1(\mathbf{y}) = \ldots = Q_s(\mathbf{y}) = 0$). As $q(X_1, \ldots, X_s)$ is a quadratic polynomial,

[1] Say $\lambda = 128$ (which is a common choice for the security parameter) and $\text{poly}(\lambda) = \lambda^2$, then our circuit size is $s = 128^2 = 16384$, and our field size is 2^{128}. Thus, our soundness errors are of the orders $\log_2(16384)/2^{128} \approx 4.1 \times 10^{-38}$, $16384/2^{128} \approx 4.8 \times 10^{-35}$, and $16384^2/2^{128} \approx 7.9 \times 10^{-31}$. These probabilities are so small, that their differences are insignificant, and that in practice any soundness error of order $\mathcal{O}(\text{poly}(s)/|\mathbb{F}|)$ is sufficient.

we have $t \leq s + \binom{s}{2} = \binom{s+1}{2}$ (Lemma 3.3.4) non-zero quadratic terms (we have s choices for X_iX_i and $\binom{s}{2}$ choices for X_iX_j with $i \neq j$). The worst-case scenario for the QPCP is when $t = \binom{s+1}{2}$, in which case both compiled proofs have very similar communication parameters. In fact, asymptotically they require the same amount of communication. The best-case scenario for the QPCP is when $t = 0$, in which case they still have the same amount of communication asymptotically, but the $\Pi_{\text{cs}}^{(1)}$-compiled Hadamard QPCP involves less communication (as the logarithms become $\lceil \log_2(s+6) \rceil$ which is generally less than the $\lceil \log_2(s + \binom{s+1}{2} + 1) \rceil$ that appears in the Π_c-compiled LPCP parameters).

Both compilers achieve the same asymptotic performance. However, the $\Pi_{\text{cs}}^{(1)}$-compiled Hadamard QPCP outperforms, especially when t is small. This analysis confirms the idea that translating LPCPs into QPCPs by using clever quadratic queries is very powerful, especially when combined with our $\Pi_{\text{cs}}^{(1)}$-compiler.

Our two compilers have a significant advantage over the LOE-compiled Hadamard LPCP: they rely on much simpler assumptions (in particular, the DLOG assumption), instead of on the existence of linear-only encryption schemes. There are some downsides to our compilers as well, but we will discuss them in Section 9.2 (where we compare the three compilers for general LPCPs) to avoid repetition.

9.2 Comparison for general LPCPs

In this section, we compare the three cryptographic compilers for general LPCPs. We present the parameters in Table 9.4, where ξ is the number of queries and s is the length of the proof string in the information-theoretic LPCP. We assume the information-theoretic LPCP is defined over a finite field $\mathbb{F}$ of order q (where q scales with a security parameter λ) and has soundness error $\varepsilon_{\text{LPCP}}$.

We immediately see that if we interpret the information-theoretic LPCP as a QPCP without applying clever tricks to reduce the number of queries or the length of the proof string, then the Π_c-compiler is the clear winner: it consists of fewer moves, less communication, and it has a better soundness error than the $\Pi_{\text{cs}}^{(1)}$-compiler (although, as we saw in Section 9.1, the difference in soundness errors is insignificant). Hence, the choice for the $\Pi_{\text{cs}}^{(1)}$-compiler only makes sense for concrete LPCPs that can be transformed into a different QPCP that has a shorter proof string and/or uses fewer queries. In the future, researchers could explore the possibility of developing a general method to reduce the number of queries or shorten the proof string when interpreting an LPCP as a QPCP. However, for now such reductions need to be done on a case-by-case basis.

Table 9.4: The parameters of three cryptographic compilers applied to a general information-theoretic LPCP over a finite field $\mathbb{F}$ of prime order q, where q scales with a security parameter λ, that has statistical soundness error $\varepsilon_{\text{LPCP}}$. We assume that the verifier in the information-theoretic LPCP does ξ queries to a proof string oracle $\boldsymbol{\pi} \in \mathbb{F}^s$.

	General LOE (Theorem 6.2.3)	**General Π_c (Theorem 7.4.1)**	**General $\Pi_{\text{cs}}^{(1)}$ (Theorem A.1.1)**
$\lvert\boldsymbol{\pi}\rvert$	$\text{poly}(\lambda) \cdot (\xi + 1)$	-	-
$\lvert\sigma\rvert$	$\text{poly}(\lambda) \cdot (\xi + 1) \cdot s$	-	-
#moves	1	$2\xi\lceil\log_2(s+1)\rceil + 3\xi + 3$	$2\xi\lceil\log_2(s+6)\rceil + 11\xi + 3$
communication $\mathcal{P} \to \mathcal{V}$	-	$2\xi\lceil\log_2(s+1)\rceil - \xi + 1$ in $\mathbb{G}$ 4ξ in $\mathbb{F}$	$2\xi\lceil\log_2(s+6)\rceil + 4\xi + 1$ in $\mathbb{G}$ 13ξ in $\mathbb{F}$
communication $\mathcal{V} \to \mathcal{P}$	-	$\xi\lceil\log_2(s+1)\rceil + (s+1)\xi$ in $\mathbb{F}$	$\xi\lceil\log_2(s+6)\rceil + (5+s)\xi$ in $\mathbb{F}$
ε	$\varepsilon_{\text{LPCP}} + \frac{1}{\lvert\mathbb{F}\rvert} + \text{negl}(\lambda)$	$\varepsilon_{\text{LPCP}} + \xi \cdot \mathcal{O}(\log_2(s)/\lvert\mathbb{F}\rvert)$	$\varepsilon_{\text{LPCP}} + \xi \cdot \mathcal{O}(s^2/\lvert\mathbb{F}\rvert)$
soundness assumptions	LOE schemes, CRS model	DLOG, RSA, or SIS	DLOG, RSA, or SIS
non-interactive?	YES	NO (Fiat-Shamir in ROM)	NO (Fiat-Shamir in ROM)

It remains for us to compare our compilers to the linear-only encryption (LOE) based compiler. Note that, because the field sizes for LPCPs are generally very large, our compilers asymptotically achieve the same soundness error as the LOE-compiler. The LOE-compiler does, however, offer some trade-offs when compared to our compilers. The first advantage of an LOE-compiled LPCP, is that it is non-interactive in the plain model[1], whereas for our compilers, we need to apply the Fiat-Shamir transform (Section 4.5) to make our compiled proofs non-interactive, which requires us to assume the *random oracle model* (Definition 4.5.2). However, the LOE-compiler does need a common reference string (CRS), which our compilers do not need. Another advantage of the LOE-compiler is that it has a proof length of only $\text{poly}(\lambda) \cdot (\xi + 1)$ for some security parameter λ, whereas our amount of communication depends linearly on the proof length s of the information-theoretic LPCP (although amortization tricks may reduce the amount of communication). However, for the LOE-compiler, the CRS has length $\text{poly}(\lambda) \cdot (\xi + 1) \cdot s$, which means that the length of the CRS depends linearly on s. Hence, there is a trade-off between the CRS length and the amount of communication in our compilers. Asymptotically, the CRS length is similar to the communication in our compilers, but a direct comparison is not straightforward and depends on the specific LPCP at hand.

[1]Recall from Section 4.5 that interactivity can be costly in terms of added communication complexity, and that interactivity can be problematic in terms of latency, as both the prover and the verifier need to be online during the entire proof. Therefore, we generally prefer non-interactive proof systems.

As mentioned before, the main drawback of LOE-compilers is that the existence of linear-only encryption schemes is based on heuristics (although their existence is believable), as their existence lacks a foundation in more standard cryptographic assumptions. Our compilers can, however, be based on the DLOG assumption, the RSA assumption, and on lattice-based assumptions (in particular, short integer solutions – SIS) [5, 6], which are much simpler assumptions. The SIS assumption even implies post-quantum security.

A final subtle advantage of our compilers is that LOE-compiler requires the queries of the underlying LPCP to be input-oblivious (i.e., to not depend on the input). Note that for the Hadamard LPCP the queries are input-oblivious, as the circuit they depend on is viewed as public, and only the input values are input. We could in principle allow the circuit to be part of the input, which would be preferable if we are not computing the same circuit over and over again. In such cases, the LOE-compiler would not work as the queries are not input-oblivious anymore, whereas our compilers can still be applied.

In conclusion, our compilers are very valuable for compiling LPCPs into cryptographic proofs. Currently, our $\Pi_{\text{cs}}^{(1)}$-compiler is only useful for concrete LPCPs that can be transformed into QPCPs with shorter proof strings and fewer queries than the LPCPs. In the future, the possibility could be explored for developing a general method that translates LPCPs into equivalent QPCPs with shorter proof strings and fewer queries.

9.3 Comparison for arithmetic circuit satisfiability

Recall that one of the motivations for the study of the Hadamard LPCP (and thus, also the Hadamard QPCP) was a desire to obtain an efficient proof system for the arithmetic circuit satisfiability problem (see Section 3.2). However, as pointed out in Section 8.2, the compressed Σ-protocol framework already addresses this problem directly. Therefore, in this section we compare the direct application of the compressed Σ-protocol Π_{cs} (which solves arithmetic circuit satisfiability) [5, Theorem 4] to our best compiled Hadamard LPCP (which is the amortized $\Pi_{\text{cs}}^{(1)}$-compiled Hadamard QPCP, as we saw in Section 9.1). We give the parameters of the compressed Σ-protocol Π_{cs} [5, Theorem 4] in our context.

Proposition 9.3.1 (Properties of Π_{cs})**.** *Let $\mathbb{G}$ be a cyclic Abelian group of prime order q for which the discrete logarithm assumption holds. Let $\mathcal{R} = \{(C; \mathbf{w}) \mid C(\mathbf{w}) = 0\}$ be the arithmetic circuit relation for circuits C taking s inputs over a finite field $\mathbb{F}$ of order q. There exists a protocol Π_{cs} for the relation $\mathcal{R}$ with the following parameters, where t is the number of multiplication gates in the circuit, and $\mu = \lceil \log_2(s + 2t + 4) \rceil - 1$:*

- Π_{cs} *consists of* $2\mu + 9$ *moves;*

- Π_{cs} *is perfectly complete;*
- Π_{cs} *is perfect honest-verifier zero-knowledge;*
- Π_{cs} *has computational soundness error* $O((\log_2(s)+t)/|\mathbb{F}|)$*, under the discrete logarithm assumption;*
- Π_{cs} *communicates:*
  - $2\lceil\log_2(s+2t+4)\rceil$ *elements of* $\mathbb{G}$ *and* 6 *elements of* $\mathbb{F}$ *from prover to verifier;*
  - $\lceil\log_2(s+2t+4)\rceil+3$ *elements of* $\mathbb{F}$ *from verifier to prover.*

Proof. In the original formulation of this theorem [5, Theorem 4], the relation contains general arithmetic circuits C with n inputs and s outputs, consisting of m multiplication gates (where scalar multiplications are not counted towards m). The expression inside the logarithms is $n+2m+4$, and the protocol is computationally $(2m+1, s+3, 2, 2, k_1, \ldots, k_\mu)$-special sound, where each $k_i = 3$. In this thesis, we only consider arithmetic circuits with s inputs, 1 output, and t (non-scalar) multiplication gates. By filling in $n = s$ (number of inputs), $s = 1$ (number of outputs), and $m = t$ (number of non-scalar multiplication gates) in the original theorem formulation [5, Theorem 4], we get the following inside the logarithms:

$$n + 2m + 4 = s + 2t + 4,$$

and for the special soundness we get

$$\begin{aligned} 2m + 1 &= 2t + 1, \\ s + 3 &= 4. \end{aligned}$$

Thus, the protocol is computationally $(2t+1, 4, 2, 2, k_1, \ldots, k_\mu)$-special sound (where each $k_i = 3$), and thus by Lemma 7.1.5, Π_{cs} is computationally knowledge sound with knowledge error

$$\kappa \leq \frac{2t+3+1+1+2\mu}{|\mathbb{F}|} = \frac{2\mu+2t+5}{|\mathbb{F}|}.$$

By Lemma 6.2.2, Π_{cs} has computational soundness error $\frac{2\mu+2t+5}{|\mathbb{F}|}$. Now, following the proofs of Lemma 7.2.3 and Lemma 8.2.2, we get a computational soundness error of

$$\frac{2\mu+2t+5}{|\mathbb{F}|} = \mathcal{O}((\log_2(s+t)+t)/|\mathbb{F}|) = \mathcal{O}((\log_2(s)+t)/|\mathbb{F}|).$$

□

The comparison between the direct application of Π_{cs} and the $\Pi_{\text{cs}}^{(1)}$-compiled Hadamard QPCP is given in Table 9.5.

Table 9.5: The direct application of a compressed Σ-protocol to arithmetic circuit satisfiability and the amortized $\Pi_{\text{cs}}^{(1)}$-compiled Hadamard QPCP for arithmetic circuits taking s inputs over a finite field $\mathbb{F}$ of prime order q.

	Compressed Σ-protocol for arithmetic circuit SAT [5, Theorem 4]	**$\Pi_{\text{cs}}^{(1)}$-compiled Hadamard QPCP (amortized, Theorem 8.3.3)**
#moves	$2\lceil\log_2(s+2t+4)\rceil+7$	$2\lceil\log_2(s+2t+6)\rceil+13$
communication $\mathcal{P}\to\mathcal{V}$	$2\lceil\log_2(s+2t+4)\rceil$ in $\mathbb{G}$ 6 in $\mathbb{F}$	$2\lceil\log_2(s+2t+6)\rceil+5$ in $\mathbb{G}$ 12 in $\mathbb{F}$
communication $\mathcal{V}\to\mathcal{P}$	$\lceil\log_2(s+2t+4)\rceil+3$ in $\mathbb{F}$	$\lceil\log_2(s+2t+6)\rceil+6$ in $\mathbb{F}$
ε	$\mathcal{O}((\log_2(s)+t)/\lvert\mathbb{F}\rvert)$	$O(s^2/\lvert\mathbb{F}\rvert)$
soundness assumptions	DLOG, RSA, or SIS	DLOG, RSA, or SIS
non-interactive	NO (Fiat-Shamir in ROM)	NO (Fiat-Shamir in ROM)

Admittedly, it seems that our compiler does not perform better than applying the compressed Σ-protocol framework directly, if one is only concerned with circuit-satisfiability. However, for other computational tasks, it could certainly be the case that there is a fast LPCP that we could directly compile with our framework, whereas first reducing the computational task to a circuit-satisfiability problem can introduce large computational overheads. Furthermore, as discussed in Chapter 1, LPCPs — being so closely connected to PCPs — are an interesting object of study from a complexity-theoretic standpoint, and thus are certainly worthy of study. For this reason, we still believe that it is valuable to work in a modular fashion, and to give a cryptographic compiler for the LPCP framework.

In addition, LPCPs (or, more specifically, *fully-linear* PCPs, where the verifier has linear-only access to both the *input* and the proof string) find applications in secure multi-party computation and other areas of cryptography [12]. Therefore, it is valuable to be able to work directly with the LPCP model and to have a cryptographic compiler for this model. We leave a thorough investigation of the applications of our ideas in these domains to future work.

10

Conclusion and recommendation

In this thesis, we studied *linear probabilistically checkable proofs* (LPCPs). LPCPs are non-interactive proof systems where the verifier's access to the proof is limited to linear combinations of the proof symbols. This limited access is made available via an oracle provided by the prover. LPCPs are a strengthening of *probabilistically checkable proofs (PCPs)*, inducing simpler proofs, such as the Hadamard LPCP for NP-relations (Section 6.1). PCPs, in turn, provide very efficient proofs for NP-relations, as demonstrated by the renowned *PCP theorem* (see Theorem 4.2.2) [3, 4]. Therefore, we were driven to develop and improve LPCP constructions.

In this thesis, we adopted the paradigm of separating the information-theoretic proof system from its cryptographic compilation into a cryptographic proof [12, 30]. This separation generally allows us to independently study and develop these two components of a proof system. We saw that this separation introduces a trade-off, where adding power to the information-theoretic proof system results in simpler and stronger information-theoretic proofs, but requires stronger cryptography. In particular, current cryptographic compilers for LPCPs rely on *linear-only encryption (LOE)*, whose existence is believable, but heuristic in the sense that the existence does not rely on standard cryptographic assumptions. Therefore, the main goal of this thesis was to develop cryptographic compilers for LPCPs that do rely on standard cryptographic assumptions.

The main challenge for a cryptographic compiler for LPCPs is to realize the *linear-only oracle*, which reveals inner products between the verifier's queries and the proof string without revealing any other information about the proof string. We pursued two strategies, resulting in two cryptographic compilers for dealing with such oracles. Firstly, we interpreted the inner products as evaluations of linear forms, which allowed us to compile

LPCPs using a compressed Σ-protocol Π_c that relies on standard cryptographic assumptions (in particular, on the discrete logarithm assumption – Assumption 1). The resulting compiler was named the Π_c-*compiler*.

Secondly, we interpreted the linear queries in an LPCP as quadratic queries without quadratic terms, thereby interpreting an LPCP as a *quadratic probabilistically checkable proof (QPCP)*. We then applied a different compressed Σ-protocol $\Pi_{\text{cs}}^{(1)}$ to compile these quadratic queries. The $\Pi_{\text{cs}}^{(1)}$-*compiler* also relies on standard cryptographic assumptions.

In Chapter 9, we extensively compared the cryptographic compilers for general LPCPs and for the Hadamard LPCP. Our analysis revealed the significant value of our compilers, as they rely on standard cryptographic assumptions. Additionally, they achieve the same asymptotic soundness error and communication complexity as the LOE-based compiler. One drawback is that the compiled proofs are interactive. They can be made non-interactive using the Fiat-Shamir transform (Section 4.5), but this transformation requires the *random oracle model* (Definition 4.5.2).

Between our two compilers, we found that the Π_c-compiler performs best as a general compiler for LPCPs in terms of communication complexity. However, we observed that in cases where an LPCP can cleverly be transformed into a QPCP with a shorter proof string and/or fewer queries than the LPCP, the $\Pi_{\text{cs}}^{(1)}$-compiler can yield better results. In particular, we transformed the Hadamard LPCP (which has three queries) into an equivalent QPCP with a shorter proof length and only a single query. In that case, the $\Pi_{\text{cs}}^{(1)}$-compiler outperformed the Π_c-compiler.

In conclusion, we achieved all of our research goals: we cryptographically compiled LPCPs using standard cryptographic assumptions, by using the compressed Σ-protocol for linear forms, and for arithmetic circuits. We compared our new compilers to the linear-only encryption based compiler, and concluded that both of our new compilers are highly valuable for compiling LPCPs into cryptographic proofs. Given an information-theoretic LPCP, one should first try to transform it into a QPCP with a shorter proof string and/or fewer queries. Then, this QPCP can be compiled using our $\Pi_{\text{cs}}^{(1)}$-compiler. The original LPCP can also be compiled using our Π_c-compiler. With the aid of Table 9.4, the best of these two options should be picked as the final cryptographic proof.

Further research

This thesis aimed to highlight the value of studying the information-theoretic part of a proof system separately from the cryptographic compiler that translates the system into a zero-knowledge cryptographic proof. In particular, we replaced the current linear-only

encryption based compiler for LPCPs by our newly developed cryptographic compilers, without changing the underlying information-theoretic LPCPs.

Our thesis raises several open questions that require further research. Firstly, the possibility could be explored of developing a general method that translates LPCPs into equivalent QPCPs with shorter proof strings and fewer queries than the LPCPs. Combining such a method with our $\Pi_{\mathrm{cs}}^{(1)}$-compiler could be very powerful for the development of efficient cryptographic proofs.

Secondly, researchers could investigate the possibility of compiling other proof systems using compressed Σ-protocols (e.g., PCPs or *interactive oracle proofs*, which are an interactive variant of PCPs, where the verifier can only read a few locations of each prover message through oracles [9]).

Thirdly, instead of focusing on the improvement of linear-only encryption schemes for the LOE-compilers, researchers could explore other compiling methods, similar to how we explored the compressed Σ-protocol as an alternative compiling method in this thesis.

Fourthly, researchers could attempt to prove better soundness bounds or investigate different types of soundness errors for our compilers. One possibility for a different type of soundness error would be the application of the recently developed Γ-*out-of*-$\mathcal{C}$ *special-soundness* techniques [7], which generalize special soundness and imply improved bounds on the knowledge soundness error for a broader class of multi-round protocols. By applying these techniques to our compiled proof systems, it could be possible to prove that they are knowledge sound, surpassing the computational soundness which we currently have.

Lastly, as discussed in Chapter 9, (fully)-linear PCPs find applications in secure multi-party computation and other areas of cryptography [12]. Researchers could investigate the applications of our cryptographic compilers for LPCPs in these domains.

Addressing all these questions will deepen our understanding of the theory of non-interactive and interactive proof systems, their relation to cryptographic proofs, and their wide-ranging applications in the field of cryptography.

Bibliography

[1] Amir Abboud, Aviad Rubinstein, and Ryan Williams. 2017. Distributed PCP Theorems for Hardness of Approximation in P. In *IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*. 25–36.

[2] Sanjeev Arora and Boaz Barak. 2009. *Computational complexity: a modern approach*. Cambridge University Press.

[3] Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan, and Mario Szegedy. 1998. Proof Verification and the Hardness of Approximation Problems. *J. ACM* 45 (1998), 501–555.

[4] Sanjeev Arora and Shmuel Safra. 1998. Probabilistic Checking of Proofs: A New Characterization of NP. *J. ACM* 45 (1998), 70–122.

[5] Thomas Attema and Ronald Cramer. 2020. Compressed-Protocol Theory and Practical Application to Plug & Play Secure Algorithmics. In *Proceedings of the 40th Annual International Cryptology Conference (CRYPTO 2020)*. 513–543.

[6] Thomas Attema, Ronald Cramer, and Lisa Kohl. 2021. A Compressed Σ-Protocol Theory for Lattices. In *Advances in Cryptology – CRYPTO 2021*. 549–579.

[7] Thomas Attema, Serge Fehr, and Nicolas Resch. 2023. A Generalized Special-Soundness Notion and its Knowledge Extractors. *Cryptology ePrint Archive* (2023).

[8] László Babai. 1985. Trading group theory for randomness. In *Proceedings of the seventeenth annual ACM symposium on Theory of computing*. 421–429.

[9] Eli Ben-Sasson, Alessandro Chiesa, and Nicholas Spooner. 2016. Interactive Oracle Proofs. In *Theory of Cryptography*, Martin Hirt and Adam Smith (Eds.). Springer Berlin Heidelberg, 31–60.

[10] Nir Bitansky, Alessandro Chiesa, Yuval Ishai, Omer Paneth, and Rafail Ostrovsky. 2013. Succinct non-interactive arguments via linear interactive proofs. In *Theory of Cryptography: 10th Theory of Cryptography Conference, TCC 2013*. 315–333.

[11] Manuel Blum, Paul Feldman, and Silvio Micali. 1988. Non-Interactive Zero-Knowledge and its Applications. In *20th ACM Symposium on the Theory of Computing*. 103–112.

[12] Dan Boneh, Elette Boyle, Henry Corrigan-Gibbs, Niv Gilboa, and Yuval Ishai. 2019. Zero-knowledge proofs on secret-shared data via fully linear PCPs. In *Advances in Cryptology – CRYPTO 2019*. Springer, 67–97.

[13] Dan Boneh, Shafi Goldwasser, Dawn Song, Justin Thaler, and Yupeng Zhang. 2023. *Zero-Knowledge Proofs (MOOC)*. https://zk-learning.org/

[14] Jonathan Bootle, Andrea Cerulli, Essam Ghadafi, Jens Groth, Mohammad Hajiabadi, and Sune K. Jakobsen. 2017. Linear-time zero-knowledge proofs for arithmetic circuit satisfiability. In *Advances in Cryptology – ASIACRYPT 2017*. Springer, 336–365.

[15] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. 2018. Bulletproofs: Short proofs for confidential transactions and more. In *2018 IEEE symposium on security and privacy (SP)*. IEEE, 315–334.

[16] Ronald Cramer, Ivan Damgård, and Berry Schoenmakers. 2001. Proofs of partial knowledge and simplified design of witness hiding protocols. In *Advances in Cryptology – CRYPTO '94*. Springer, 174–187.

[17] Ivan Damgård. 1999. Commitment Schemes and Zero-Knowledge Protocols. In *Lectures on Data Security: Modern Cryptology in Theory and Practice*. Springer, 63–86.

[18] Irit Dinur and Shmuel Safra. 2004. On the hardness of approximating label-cover. *Inform. Process. Lett.* 89, 5 (2004), 247–254.

[19] Irit Dinur and Samuel Safra. 2005. On the hardness of approximating minimum vertex cover. *Annals of mathematics* (2005), 439–485.

[20] Amos Fiat and Adi Shamir. 1986. How to Prove Yourself: Practical Solutions to Identification and Signature Problems. In *Crypto*, Vol. 86. Springer, 186–194.

[21] Uriel Fiege, Amos Fiat, and Adi Shamir. 1987. Zero knowledge proofs of identity. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*. 210–217.

[22] Alexander Glaser, Boaz Barak, and Robert J Goldston. 2014. A zero-knowledge protocol for nuclear warhead verification. *Nature* 510 (2014), 497–502.

[23] Oded Goldreich. 2002. Zero-Knowledge twenty years after its invention. *IACR Cryptol. ePrint Arch.* (2002), 186.

[24] Oded Goldreich. 2004. *Foundations of Cryptography*. Cambridge University Press.

[25] Oded Goldreich. 2008. Probabilistic Proof Systems: A Primer. *Foundations and Trends in Theoretical Computer Science* (2008), 1–91.

[26] Oded Goldreich and Ariel Kahan. 1996. How to construct constant-round zero-knowledge proof systems for NP. *Journal of Cryptology* 9, 3 (1996), 167–189.

[27] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. 1985. The Knowledge Complexity of Interactive Proof-Systems. In *Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing*. 291–304.

[28] Shafi Goldwasser and Michael Sipser. 1986. Private coins versus public coins in interactive proof systems. In *Proceedings of the eighteenth annual ACM symposium on Theory of computing*. 59–68.

[29] Carmit Hazay, Yehuda Lindell, Carmit Hazay, and Yehuda Lindell. 2010. Sigma protocols and efficient zero-knowledge. *Efficient Secure Two-Party Protocols: Techniques and Constructions* (2010), 147–175.

[30] Yuval Ishai. 2020. *Blog: Zero-Knowledge Proofs from Information-Theoretic Proof Systems*. https://zkproof.org/2020/10/15/information-theoretic-proof-systems-part-ii/ [Accessed: March 10, 2023].

[31] Yuval Ishai, Eyal Kushilevitz, and Rafail Ostrovsky. 2007. Efficient arguments without short PCPs. In *Twenty-Second Annual IEEE Conference on Computational Complexity*. IEEE, 278–291.

[32] Yuval Ishai, Mohammad Mahmoody, Amit Sahai, and David Xiao. 2015. On Zero-Knowledge PCPs: Limitations, Simplifications, and Applications. University of Virginia.

[33] Susumu Kiyoshima. 2018. No-signaling linear PCPs. In *Theory of Cryptography: 16th International Conference (TCC 2018)*. Springer, 67–97.

[34] Peeter Laud and Alisa Pankova. 2014. Verifiable computation in multiparty protocols with honest majority. In *Provable Security: 8th International Conference (ProvSec 2014)*. Springer, 146–161.

[35] Rudolf Lidl and Harald Niederreiter. 1986. *Introduction to finite fields and their applications*. Cambridge University Press.

[36] Ian Miers, Christina Garman, Matthew Green, and Aviel D. Rubin. 2013. Zerocoin: Anonymous Distributed E-Cash from Bitcoin. In *2013 IEEE Symposium on Security and Privacy*. 397–411.

[37] Ore Øystein. 1922. Über höhere Kongruenzen (German) [About higher congruences]. *Norsk Mat. Forenings Skrifter Ser. I* 7, 15 (1922).

[38] Torben Pryds Pedersen. 1992. Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing. In *Advances in Cryptology – CRYPTO '91*. 129–140.

[39] David Pointcheval and Jacques Stern. 1996. Security Proofs for Signature Schemes. In *Advances in Cryptology – EUROCRYPT '96*. 387–398.

[40] Aviad Rubinstein. 2018. Hardness of approximate nearest neighbor search. In *Proceedings of the 50th annual ACM SIGACT symposium on theory of computing*. 1260–1268.

[41] Jacob T. Schwartz. 1980. Fast Probabilistic Algorithms for Verification of Polynomial Identities. *J. ACM* 27, 4 (1980), 701–717.

[42] Daniel R Simon. 1998. Finding collisions on a one-way street: Can secure hash functions be based on general assumptions?. In *Eurocrypt*, Vol. 98. 334–345.

[43] Rajeev Sobti and Ganesan Geetha. 2012. Cryptographic hash functions: a review. *International Journal of Computer Science Issues (IJCSI)* (2012).

[44] Douglas Wikström. 2021. Special soundness in the random oracle model. *Cryptology ePrint Archive* (2021).

[45] Richard Zippel. 1979. Probabilistic algorithms for sparse polynomials. In *Symbolic and Algebraic Computation (EUROSAM)*, Edward W. Ng (Ed.). Springer, 216–226.

Appendix A

Cryptographically compiling QPCPs

In this appendix, we define a general LPCP compiler by using the QPCP compiler from Chapter 8. That is, we interpret the linear queries of an LPCP as quadratic queries without quadratic terms, and apply the powerful $\Pi_{\mathrm{cs}}^{(1)}$-compiler to the quadratic queries. In Appendix A.1, we define a general compiler. Then, in Appendix A.2, we use that compiler to compile the Hadamard LPCP.

A.1 Compiling general QPCPs and LPCPs

In this section, we define a cryptographic compiler for QPCPs, which will also serve as a cryptographic compiler for LPCPs by interpreting the linear queries as quadratic queries without quadratic terms. First, notice that the protocol $\Pi_{\mathrm{cs}}^{(1)}$ as defined in Proposition 8.2.1 only proves statements of the form $q(\boldsymbol{\pi}) = 0$ for quadratic s-variate polynomials (queries) q, whereas for general QPCPs, we would want to be able to prove that $q(\boldsymbol{\pi}) = z$ for any $z \in \mathbb{F}$. We can, however, still use $\Pi_{\mathrm{cs}}^{(1)}$ by slightly altering the QPCP: instead of the query q, the verifier queries $\widehat{q}(X_1, \ldots, X_s) = q(X_1, \ldots, X_s) - z$. Then, $q(\boldsymbol{\pi}) = z \iff \widehat{q}(\boldsymbol{\pi}) = 0$, which means we can run $\Pi_{\mathrm{cs}}^{(1)}$ for the query $\widehat{q}$ to verify that $q(\boldsymbol{\pi}) = z$.

Since an LPCP only supports linear queries (which can be interpreted as quadratic queries without quadratic terms), we can directly interpret any LPCP as a QPCP, where the number of non-zero quadratic terms in all queries $t = 0$. Hence, we can compile an LPCP analogously to how we compiled the Hadamard QPCP in Section 8.3. The basic idea is as follows. We start from an information-theoretic LPCP, consisting of ξ queries $\mathbf{q}^{(1)}, \ldots, \mathbf{q}^{(\xi)} \in \mathbb{F}^s$, to which the responses are $z_i = \langle \mathbf{q}^{(i)}, \boldsymbol{\pi} \rangle$ (for $i \in [\xi]$), where $\boldsymbol{\pi} \in \mathbb{F}^s$ is the proof string. We can interpret these linear queries as evaluations in $\boldsymbol{\pi}$ of quadratic

s-variate polynomials:

$$q_i(X_1, \ldots, X_s) \coloneqq \left(\sum_{j=1}^{s} \mathbf{q}_j^{(i)} X_j \right) - z_i,$$

where instead of proving that $z_i = \langle \mathbf{q}^{(i)}, \boldsymbol{\pi} \rangle$, the prover now (equivalently) proves that $q_i(\boldsymbol{\pi}_1, \ldots, \boldsymbol{\pi}_s) = 0$. As the queries are now quadratic polynomials, we have created an information-theoretic QPCP, which we can compile with $\Pi_{\text{cs}}^{(1)}$: the prover first sends a Pedersen commitment P of the proof string $\boldsymbol{\pi}$ to the verifier, then the verifier sends its queries $\mathbf{q}^{(1)}, \ldots, \mathbf{q}^{(\xi)} \in \mathbb{F}^s$ to the prover, next the prover sends the responses $z_1, \ldots, z_\xi$ to the verifier, after which the prover and verifier engage in ξ runs of $\Pi_{\text{cs}}^{(1)}$ to verify that $q_i(\boldsymbol{\pi}) = 0$ for each $i \in [\xi]$. The resulting $\Pi_{\text{cs}}^{(1)}$-compiled LPCP is presented in Figure A.1.

Note that all proofs in Section 8.3 that are related to compiling the Hadamard QPCP rarely rely on the specifics of the Hadamard QPCP, which shows that we can quite directly generalize our compiler to general QPCPs (and therefore general LPCPs). To avoid repetition, in the proof of the following theorem, we omit the details that are almost identical to the proofs of Proposition 8.3.1 and Theorem 8.3.3.

Theorem A.1.1 (General $\Pi_{\text{cs}}^{(1)}$-compiled LPCP)**.** *Let $\mathbb{G}$ be a cyclic Abelian group of prime order q for which the discrete logarithm assumption holds. Let $\mathbb{F}$ be a finite field of order q and let $s \in \mathbb{N}$. Let $\mathbf{g} \xleftarrow{\$} \mathbb{G}^s$ and $h \xleftarrow{\$} \mathbb{G}$ be $s+1$ generators of $\mathbb{G}$. Let Π be a perfectly complete and perfect honest-verifier zero-knowledge information-theoretic LPCP for a relation $\mathcal{R} = \{(\mathbf{x}; \mathbf{w})\} \subseteq \mathbb{F}^n \times \mathbb{F}^m$. Suppose Π has a proof string $\boldsymbol{\pi} \in \mathbb{F}^s$ and has statistical soundness error $\varepsilon_{\text{LPCP}}$. Suppose Π consists of ξ queries $\mathbf{q}^{(1)}, \ldots, \mathbf{q}^{(\xi)} \in \mathbb{F}^s$ to the proof string oracle $\boldsymbol{\pi}$, to which the responses are $z_i = \langle \mathbf{q}^{(i)}, \boldsymbol{\pi} \rangle$ for $i \in [\xi]$.*

Then, Π can be interpreted as a QPCP where each query response is zero. Furthermore, Π can be compiled to a cryptographic proof using the protocol $\Pi_{\text{cs}}^{(1)}$ (which has computational soundness error $\varepsilon_{\Pi_{\text{cs}}^{(1)}}$). The resulting $\Pi_{\text{cs}}^{(1)}$-compiled LPCP has the following parameters, where $\mu = \lceil \log_2(s+6) \rceil - 1$:

- *the protocol consists of $3 + \xi(2\mu + 13)$ moves;*
- *the protocol is perfectly complete;*
- *the protocol is perfect honest-verifier zero-knowledge;*
- *the protocol has computational soundness error $\varepsilon_{\text{LPCP}} + \xi \cdot \varepsilon_{\Pi_{\text{cs}}^{(1)}} = \varepsilon_{\text{LPCP}} + \xi \cdot \mathcal{O}(s^2/|\mathbb{F}|)$ under the discrete logarithm assumption;*
- *the protocol communicates:*

– $2\xi\lceil\log_2(s+6)\rceil + 4\xi + 1$ *elements of* $\mathbb{G}$ *and* 13ξ *elements of* $\mathbb{F}$ *from prover to verifier;*

– $\xi\lceil\log_2(s+6)\rceil + (5+s)\xi$ *elements of* $\mathbb{F}$ *from verifier to prover.*

Proof. **Number of moves and communication.** The protocol consists of 3 moves at the start, followed by ξ runs of $\Pi_{\mathrm{cs}}^{(1)}$, each consisting of $2\mu+13$ moves (Proposition 8.2.1). Hence, in total, the protocol consists of $3+\xi(2\mu+13)$ moves. Since each query q_i does not contain quadratic terms (see Figure A.1), we have $t=0$ in Proposition 8.2.1. Thus, each run of $\Pi_{\mathrm{cs}}^{(1)}$ communicates:

- $2\lceil\log_2(s+6)\rceil + 4$ elements of $\mathbb{G}$ and 12 elements of $\mathbb{F}$ from prover to verifier;
- $\lceil\log_2(s+6)\rceil + 5$ elements of $\mathbb{F}$ from verifier to prover.

Since we have ξ runs of $\Pi_{\mathrm{cs}}^{(1)}$, we multiply these numbers by ξ. The communication in the first three rounds of the protocol consists of 1 element of $\mathbb{G}$ and ξ elements of $\mathbb{F}$ from prover to verifier (Pedersen commitment and query answers), and ξ elements of $\mathbb{F}^s$ (i.e., $s\xi$ elements of $\mathbb{F}$) from verifier to prover (queries). Adding all these numbers together yields the total numbers in the theorem statement.

Completeness *(sketch).* Since the information-theoretic LPCP is perfectly complete, and since $\Pi_{\mathrm{cs}}^{(1)}$ is perfectly complete, the $\Pi_{\mathrm{cs}}^{(1)}$-compiled LPCP is also perfectly complete.

Zero-knowledge *(sketch).* Since the information-theoretic LPCP is perfect honest-verifier zero-knowledge, and since $\Pi_{\mathrm{cs}}^{(1)}$ is perfect honest-verifier zero-knowledge, the $\Pi_{\mathrm{cs}}^{(1)}$-compiled LPCP is also perfect honest-verifier zero-knowledge.

Soundness. Assume that $\mathbf{x} \notin L_{\mathcal{R}}$, and that a malicious prover $\widehat{\mathcal{P}}$ tries to convince an honest verifier $\mathcal{V}$ to accept. We condition the soundness error on the event $\mathrm{E}_{\mathrm{LPCP}}$ that $\widehat{\mathcal{P}}$ can construct a proof string $\boldsymbol{\pi}^*$ that convinces an honest verifier in the information-theoretic LPCP. We get

$$\begin{aligned}
&\mathbb{P}[\widehat{\mathcal{P}} \text{ convinces } \mathcal{V}] \\
&\quad = \mathbb{P}[\widehat{\mathcal{P}} \text{ convinces } \mathcal{V} \mid \mathrm{E}_{\mathrm{LPCP}}] \cdot \mathbb{P}[\mathrm{E}_{\mathrm{LPCP}}] + \mathbb{P}[\widehat{\mathcal{P}} \text{ convinces } \mathcal{V} \mid \neg\mathrm{E}_{\mathrm{LPCP}}] \cdot \mathbb{P}[\neg\mathrm{E}_{\mathrm{LPCP}}] \\
&\quad \leq \mathbb{P}[\mathrm{E}_{\mathrm{LPCP}}] + \mathbb{P}[\widehat{\mathcal{P}} \text{ convinces } \mathcal{V} \mid \neg\mathrm{E}_{\mathrm{LPCP}}].
\end{aligned}$$

Now, $\mathbb{P}[\mathrm{E}_{\mathrm{LPCP}}] = \varepsilon_{\mathrm{LPCP}}$. If $\neg\mathrm{E}_{\mathrm{LPCP}}$ holds, then $\widehat{\mathcal{P}}$ must cheat in (at least) one of the runs of $\Pi_{\mathrm{cs}}^{(1)}$ to have a chance at convincing the verifier to accept (if $\widehat{\mathcal{P}}$ convinces the verifier while being honest in each run of $\Pi_{\mathrm{cs}}^{(1)}$, then $\mathrm{E}_{\mathrm{LPCP}}$ holds). Denote

$$\mathrm{E}_i \coloneqq \ \widehat{\mathcal{P}} \text{ cheats and convinces } \mathcal{V} \text{ in the } i\text{-th run of } \Pi_{\mathrm{cs}}^{(1)},$$

for $i \in [\xi]$. Then, using the union bound (Lemma 3.3.5), we have

$$\mathbb{P}[\widehat{\mathcal{P}} \text{ convinces } \mathcal{V} \mid \neg\mathrm{E}_{\mathrm{LPCP}}] \leq \mathbb{P}\left[\bigcup_{i\in[\xi]} \mathrm{E}_i\right] \overset{\text{Lemma 3.3.5}}{\leq} \sum_{i=1}^{\xi} \mathbb{P}[\mathrm{E}_i] \leq \xi \cdot \varepsilon_{\Pi_{\mathrm{cs}}^{(1)}}.$$

Hence,

$$\mathbb{P}[\widehat{\mathcal{P}} \text{ convinces } \mathcal{V}] \leq \mathbb{P}[\mathrm{E}_{\mathrm{LPCP}}] + \mathbb{P}[\widehat{\mathcal{P}} \text{ convinces } \mathcal{V} \mid \neg \mathrm{E}_{\mathrm{LPCP}}] \leq \varepsilon_{\mathrm{LPCP}} + \xi \cdot \varepsilon_{\Pi_{\mathrm{cs}}^{(1)}}.$$

By Proposition 8.2.3, we know $\varepsilon_{\Pi_{\mathrm{cs}}^{(1)}} = \mathcal{O}(s^2/|\mathbb{F}|)$. □

Remark. As was the case for the general Π_c-compiled LPCP (Theorem 7.4.1), the $\mathcal{O}(s\xi)$ communication from verifier to prover can typically be reduced via an amortization trick. The effectiveness of the trick depends on the specific distribution that the query vectors are sampled from, and thus must be analyzed on a case-by-case basis.

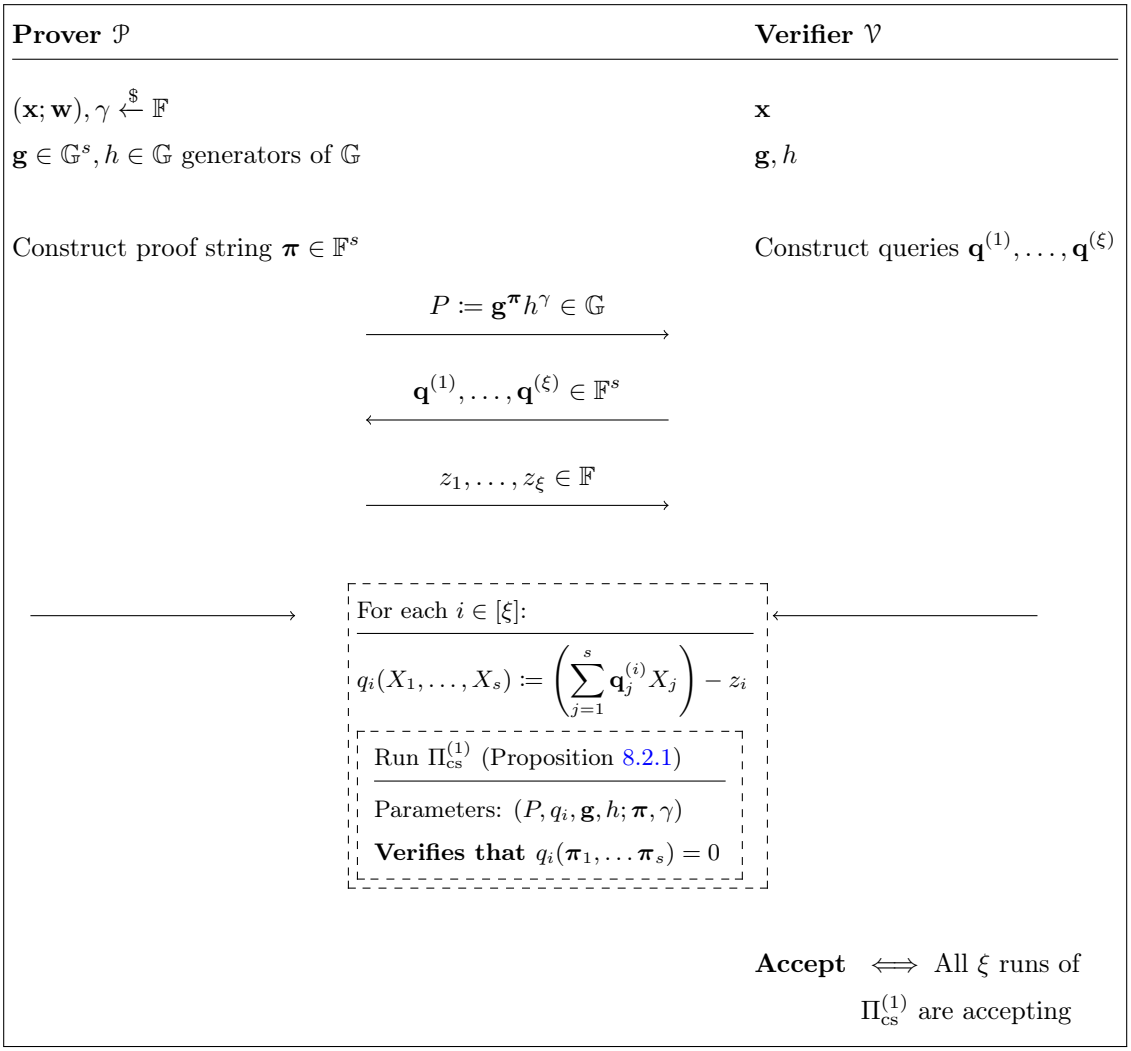

Figure A.1: A general $\Pi_{\mathrm{cs}}^{(1)}$-compiled Linear Probabilistically Checkable Proof, where the corresponding information-theoretic LPCP consists of a proof string $\boldsymbol{\pi} \in \mathbb{F}^s$, ξ queries $\mathbf{q}^{(1)}, \ldots, \mathbf{q}^{(\xi)} \in \mathbb{F}^s$, and responses $z_i = \langle \mathbf{q}^{(i)}, \boldsymbol{\pi} \rangle$ for $i \in [\xi]$.

A.2 Compiling the Hadamard LPCP

In this section, we directly compile the Hadamard LPCP using our general $\Pi_{\mathrm{cs}}^{(1)}$-compiler from Theorem A.1.1 (i.e., by interpreting the three linear queries as quadratic queries). To this end, we need to plug in $s = s + \binom{s+1}{2}$ (length of proof string) and $\xi = 3$ (number of queries) in Theorem A.1.1. We name the resulting protocol the $\Pi_{\mathrm{cs}}^{(1)}$-*compiled Hadamard LPCP*. We get the following result.

Theorem A.2.1 ($\Pi_{\mathrm{cs}}^{(1)}$-compiled Hadamard LPCP)**.** *Let $\mathbb{G}$ be a cyclic Abelian group of prime order q for which the discrete logarithm assumption holds. Let $\mathbb{F}$ be a finite field of order q. Let $\mathbf{x} \in \mathbb{F}^n$ be common input and let $C_{\mathbf{x}}$ be an arithmetic circuit of size s over $\mathbb{F}$. The $\Pi_{\mathrm{cs}}^{(1)}$-compiled Hadamard LPCP for*

$$\mathcal{R} = \{(\mathbf{x}, C_{\mathbf{x}}; \mathbf{w}) \mid C_{\mathbf{x}}(\mathbf{w}) = 0\}$$

has the following properties, where $\mu = \lceil \log_2(s + \binom{s+1}{2} + 6) \rceil - 1$:

- *the protocol consists of $3 + 3(2\mu + 13) = 6\mu + 42$ moves;*
- *the protocol is perfectly complete;*
- *the protocol is perfect honest-verifier zero-knowledge;*
- *the protocol has computational soundness error $\mathcal{O}(s^2/|\mathbb{F}|)$ under the discrete logarithm assumption;*
- *the protocol communicates:*
  - *$6\lceil \log_2(s + \binom{s+1}{2} + 6) \rceil + 13$ elements of $\mathbb{G}$ and 39 elements of $\mathbb{F}$ from prover to verifier;*
  - *$3\lceil \log_2(s + \binom{s+1}{2} + 6) \rceil + 3\binom{s+1}{2} + 3s + 15$ elements of $\mathbb{F}$ from verifier to prover.*

Proof. Most properties follow directly from plugging in $s = s + \binom{s+1}{2}$ and $\xi = 3$ in Theorem A.1.1. For the soundness, we have

$$\varepsilon_{\mathrm{LPCP}} + 3\varepsilon_{\Pi_{\mathrm{cs}}^{(1)}} \overset{\text{Theorem 6.1.5}}{\leq} \mathcal{O}(1/|\mathbb{F}|) + 3\varepsilon_{\Pi_{\mathrm{cs}}^{(1)}} \overset{\text{Proposition 8.2.3}}{\leq} \mathcal{O}(1/|\mathbb{F}|) + \mathcal{O}(s^2/|\mathbb{F}|) = \mathcal{O}(s^2/|\mathbb{F}|).$$

□