LATTICE-BASED NON-INTERACTIVE ARGUMENT SYSTEMS

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

David J. Wu
August 2018

Abstract

Non-interactive argument systems are an important building block in many cryptographic protocols. In this work, we begin by studying non-interactive zero-knowledge (NIZK) arguments for general NP languages. In a NIZK argument system, a prover can convince a verifier that a statement is true without revealing anything more about the statement. Today, NIZK arguments can be instantiated from random oracles, or, in the common reference string (CRS) model, from trapdoor permutations, pairings, or indistinguishability obfuscation. Notably absent from this list are constructions from lattice assumptions, and realizing NIZKs (for general NP languages) from lattices has been a long-standing open problem. In this work, we make progress on this problem by giving the first construction of a multi-theorem NIZK argument from standard lattice assumptions in a relaxed model called the *preprocessing* model, where we additionally assume the existence of a trusted setup algorithm that generates a proving key (used to construct proofs) and a verification key (used to verify proofs). Moreover, by basing hardness on lattice assumptions, our construction gives the first candidate that plausibly resists quantum attacks.

We then turn our attention to constructing *succinct* non-interactive arguments (SNARGs) for general NP languages. SNARGs enable verifying computations with substantially lower complexity than that required for classic NP verification. Prior to this work, all SNARG constructions relied on random oracles, pairings, or indistinguishability obfuscation. This work gives the first lattice-based SNARG candidates. In fact, we show that one of our new candidates satisfy an appealing property called "quasi-optimality," which means that the SNARG simultaneously minimizes both the prover complexity and the proof size (up to polylogarithmic factors). This is the first quasi-optimal SNARG from any concrete cryptographic assumption. Again, because of our reliance on lattice-based techniques, all of our new candidates resist quantum attacks (in contrast to existing pairing-based constructions).

Acknowledgments

Neither my journey into cryptography nor this thesis would have happened without the support and mentoring of my advisor, Dan Boneh. Six years ago, while I was still a starry-eyed undergrad at Stanford, I had the fortune of taking Dan's Introduction to Cryptography course. Dan's boundless energy and infectious enthusiasm for not only cryptography, but also research and life, was what first drew me into this field. Over the years, I can say without doubt that Dan has shown by example the sheer excitement and joy of research. As I now prepare for the next step of my own academic journey, I will strive to emulate the qualities of the ideal researcher that Dan so brilliantly exemplifies.

I would like to say a special word of thanks to Yuval Ishai and Amit Sahai for all of the insights, advice, and guidance they have provided me in the last few years. I can easily say that my three visits to UCLA in the last two years have been some of the most exciting (and productive!) weeks of my PhD, and I am grateful to both of them for being such wonderful and generous hosts. It has been an absolute pleasure and privilege to collaborate with them on multiple projects (several of which are featured in this thesis), and I look forward to many more. The numerous discussions we had have certainly shaped how I think about and approach research.

Over the years, I have also had the luxury of working with and learning from all of the students in the Stanford Applied Crypto group. I would like to thank all of them for patiently hearing out my half-baked ideas, kindly critiquing my papers, and sharing with me their cool ideas. I consider myself very lucky to be in the company of such a talented group of researchers. I especially would like to thank all of my student co-authors: Henry Corrigan-Gibbs, Sam Kim, Kevin Lewi, Hart Montgomery, and Joe Zimmerman. I thank Joe for teaching me how to reason precisely about abstract concepts and how to communicate my ideas with technical rigor. I thank Kevin for introducing me to the marvelous world of theoretical cryptography and for keeping me motivated and optimistic through all the times we got stuck on problems. I thank Hart for always being willing to share his ideas and for providing a healthy dose of skepticism for my ill-contrived constructions. I thank Henry for not only being an amazing officemate, but also for being so willing to share with me his deep insights into both cryptography and computer security; I am always impressed by the breadth of his knowledge and the depth of his intuition. Finally, I thank Sam for our many fruitful collaborations in the last few years. I cannot count how many hours we have now spent talking about cryptography and life.

And of course, I thank him for patiently bringing me up to speed on lattice-based cryptography. It has truly been an honor and a pleasure.

One of the greatest joys of research is having the opportunity to collaborate with brilliant individuals spanning a wide range of disciplines (and time zones). I would like to thank all of my wonderful collaborators from the last five years: Shashank Agrawal, Gill Bejerano, Bonnie Berger, Johannes Birgmeier, Dan Boneh, Nathan Chenette, Hyunghoon Cho, Henry Corrigan-Gibbs, Tony Feng, Yuval Ishai, Karthik Jagadeesh, Sam Kim, Kristin Lauter, Kevin Lewi, Avradip Mandal, John Mitchell, Hart Montgomery, Michael Naehrig, Alain Passelègue, Jérémy Planul, Arnab Roy, Amit Sahai, Asim Shankar, Ankur Taly, Steve Weis, and Joe Zimmerman. I thank them for making my time in grad school so enjoyable and for always reminding me why I chose to pursue research.

I would also like to thank the members of my thesis committee for all of the helpful feedback they have provided to improve this work: Dan Boneh, Moses Charikar, Brian Conrad, Omer Reingold, and Mary Wootters.

Finally, I would like to thank my friends and family who have stood with me over all these years. I will forever be grateful for their support and encouragement. After all, this quixotic quest for truth and understanding would be a lot less exciting if I could not share the joy with all of them.

This work was supported in part by an NSF Graduate Research Fellowship.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Proof systems are fundamental to modern cryptography and complexity theory. At a high level, a proof system for a language $\mathcal{L} \subseteq \{0,1\}^*$ is a two-party protocol between a prover and a verifier. The goal of the prover is to convince the verifier that some statement $\mathbf{x} \in \{0,1\}^*$ is contained in the language $\mathcal{L}$ (namely, that $\mathbf{x} \in \mathcal{L}$). The two basic properties we expect from a proof system are completeness, which roughly says that an honest prover should be able to convince an honest verifier of any true statement $\mathbf{x} \in \mathcal{L}$, and soundness, which roughly says that a (possibly dishonest) prover should not be able to convince an honest verifier of a false statement $\mathbf{x} \notin \mathcal{L}$.

As a concrete example of a class of languages with a simple proof system, consider the class of NP languages. Recall first that NP is the class of problems with *efficiently checkable* proofs. Namely, a language $\mathcal{L} \subseteq \{0,1\}^*$ is in NP if there exists a polynomial time algorithm $\mathcal{R}$ (often referred to as an NP relation) such that for every $\mathbf{x} \in \{0,1\}^*$,

$$\mathbf{x} \in \mathcal{L} \iff \exists\, \mathbf{w} \in \{0,1\}^{\mathrm{poly}(|\mathbf{x}|)} : \mathcal{R}(\mathbf{x}, \mathbf{w}) = 1.$$

It is easy to see that all NP languages have an efficient *non-interactive* proof system: to convince a verifier that some statement $\mathbf{x}$ is contained in an NP language (for an NP relation $\mathcal{R}$), the prover simply sends the verifier the NP witness $\mathbf{w}$, and the verifier simply checks that $\mathcal{R}(\mathbf{x}, \mathbf{w}) = 1$.

In the last few decades, numerous works have explored different aspects of proof systems, including interactive proof systems [GMR85, LFKN90, Sha90], zero-knowledge proof systems [GMR85], probabilistically checkable proofs [BFLS91, FGL$^+$91, ALM$^+$92], computationally sound proofs (also known as arguments) [Kil92, Mic00],[1] and more. In this work, we focus on two specific properties of NP argument systems: *zero-knowledge arguments* where the proofs do not reveal any additional

[1] The difference between an argument system and a proof system is that in an argument system, the prover is assumed to be computationally bounded (namely, it runs in probabilistic polynomial time), and soundness only needs to hold against computationally-bounded provers [BCC88]. In a proof system, we do not impose any restrictions on the computational resources of the prover.

statement about the statement $\mathbf{x}$ other than the fact that $\mathbf{x}$ is contained in the language, and *succinct arguments* where the length of the proof can be significantly shorter than the NP witness and can be verified much faster than the time needed to check the NP witness (i.e., the time needed to run the NP relation $\mathcal{R}$).

1.1 Non-Interactive Zero-Knowledge Arguments

Introduced in the seminal work of Goldwasser, Micali, and Rackoff [GMR85], a zero-knowledge proof (and argument) system enables a prover to convince a verifier that some statement is true without revealing *anything more* than the truth of the statement. Traditionally, zero-knowledge proof systems for NP are interactive, and in fact, interaction is essential for realizing zero-knowledge (for NP) in the standard model [GO94].

Non-interactive zero-knowledge. Nonetheless, Blum, Feldman, and Micali [BFM88] showed that meaningful notions of zero-knowledge are still realizable in the non-interactive setting, where the proof consists of just a *single* message from the prover to the verifier. In the last three decades, a beautiful line of works has established the existence of non-interactive zero-knowledge (NIZK) proof and argument systems for all of NP in the random oracle model [FS86, PS96] or the common reference string (CRS) model [FLS90, DDO$^+$01, GOS06, Gro10, GOS12, SW14], where the prover and the verifier are assumed to have access to a common string chosen by a trusted third party. Today, we have NIZK candidates in the CRS model from several classes of cryptographic assumptions:[2] (doubly-enhanced) trapdoor permutations [FLS90, DDO$^+$01, Gro10], pairings [GOS06, GOS12], and indistinguishability obfuscation [SW14]. Notably absent from this list are constructions from lattice assumptions [Ajt96, Reg05]. While some partial progress has been made in the case of specific languages [PV08, APSD18], the general case of constructing NIZK proofs (or even arguments) for all of NP from standard lattice assumptions remains a long-standing open problem in cryptography.

NIZKs in a preprocessing model. In this work, we make progress on this problem by giving the first *multi-theorem* NIZK argument (and proof[3]) for NP from standard lattice assumptions in the *preprocessing* model. In the NIZK with preprocessing model [DMP88], there is an initial (trusted) setup phase that generates a proving key k_P and a verification key k_V. The proving key is needed to construct proofs while the verification key is needed to check proofs. In addition, the setup phase is run *before* any statements are proven (and thus, must be statement-independent). In the multi-theorem setting, we require that soundness holds against a prover who has oracle access to

[2]There are also NIZK candidates based on number-theoretic assumptions [BFM88, DMP87, BDMP91] which satisfy weaker properties. We discuss these in greater detail in Section 3.1.1 and Remark 3.33.

[3]A simple variant of our construction gives a lattice-based NIZK *proof* system in the preprocessing model where soundness also holds against computationally-unbounded provers (Remark 3.32). For reasons outlined in Remark 3.32, however, we will focus primarily on our construction of NIZK arguments in the preprocessing model.

the verifier (but does not see k_V), and that zero-knowledge holds against a verifier who has oracle access to the prover (but does not see k_P). The NIZK with preprocessing model generalizes the more traditional settings under which NIZKs have been studied. For instance, the case where k_P is public (but k_V is secret) corresponds to designated-verifier NIZKs [CD04, DFN06, CG15], while the case where both k_P and k_V are public corresponds to the traditional CRS setting, where the CRS is taken to be the pair (k_P, k_V). We describe our construction in detail in Chapter 3.

Why study the preprocessing model? While the preprocessing model is weaker than the more traditional CRS model, constructing multi-theorem NIZK arguments (and proofs) in this model does not appear to be any easier than constructing them in the CRS model. Existing constructions of NIZKs in the preprocessing model from weaker assumptions such as one-way functions [DMP88, LS90, Dam92, IKOS07] or oblivious transfer [KMO89] are only secure in the *single-theorem* setting. As we discuss in greater detail in Remark 3.33, the constructions from [DMP88, LS90, Dam92] only provide single-theorem zero-knowledge, while the constructions in [KMO89, IKOS07] only provide single-theorem soundness. Even in the designated-verifier setting [CD04, DFN06, CG15] (where only the holder of a *secret* verification key can verify the proofs), the existing constructions of NIZKs for NP based on linearly-homomorphic encryption suffer from the so-called "verifier-rejection" problem where soundness holds only against a *logarithmically-bounded* number of statements. Thus, the only candidates of multi-theorem NIZKs where soundness and zero-knowledge hold for an *unbounded* number of theorems are the constructions in the CRS model, which all rely on trapdoor permutations, pairings, or obfuscation. Thus, it remains an interesting problem to realize multi-theorem NIZKs from lattice assumptions even in the preprocessing model.

Moreover, as we show in Section 3.5.1, multi-theorem NIZKs in the preprocessing model suffice to instantiate many of the classic applications of NIZKs for boosting the security of multiparty computation (MPC) protocols. Thus, our new constructions of reusable NIZK arguments from standard lattice assumptions imply new constructions of round-optimal, near-optimal-communication MPC protocols purely from lattice assumptions. Our work also implies a *succinct* version of the classic Goldreich-Micali-Wigderson compiler [GMW86, GMW87] for boosting semi-honest security to malicious security, again purely from standard lattice assumptions. Furthermore, studying NIZKs in the preprocessing model may also serve as a stepping stone towards realizing NIZKs in the CRS model from standard lattice assumptions. For example, the starting point of the first multi-theorem NIZK construction by Feige, Lapidot, and Shamir [FLS90] was a NIZK proof for graph Hamiltonicity in the preprocessing model.

1.2 Succinct Non-Interactive Arguments

We next turn our attention to *succinct* argument systems for NP languages. First, we say an argument system is succinct if the communication complexity (between the prover and the verifier) is *polylogarithmic* in the running time of the NP verifier for the language. Notably, the size of the argument is polylogarithmic in the size of the NP witness.

Computationally sound proofs. In interactive proof systems for NP with statistical soundness, non-trivial savings in communication and verification time are highly unlikely [BHZ87, GH98, GVW01, Wee05]. However, if we relax the requirements and consider proof systems with computational soundness, also known as *argument systems* [BCC88], significant efficiency improvements become possible. Kilian [Kil92] gave the first succinct four-round interactive argument system for NP based on collision-resistant hash functions and probabilistically checkable proofs (PCPs). Subsequently, Micali [Mic00] showed how to convert Kilian's four-round argument into a single-round argument for NP by applying the Fiat-Shamir heuristic [FS86] to Kilian's interactive protocol. Micali's "computationally-sound proofs" (CS proofs) represents the first candidate construction of a *succinct non-interactive argument* (that is, a "SNARG" [GW11]).

SNARGs in the standard model. In the standard model, single-round succinct arguments are highly unlikely for sufficiently hard languages [BP04a, Wee05], so we consider the weaker goal of two-message succinct argument systems where the initial message from the verifier is independent of the statement being verified. We refer to this message as the common reference string (CRS).

Gentry and Wichs [GW11] showed that no SNARG (for a sufficiently difficult language) can be proven secure under any "falsifiable" assumption [Nao03]. Consequently, all existing SNARG constructions for NP in the standard model (with a CRS) have relied on non-falsifiable assumptions such as knowledge-of-exponent assumptions [Dam91, BP04b, Mie08, Gro10, Lip12, GGPR13], extractable collision-resistant hashing [BCCT12, DFH12, BCC^{+}17], homomorphic encryption with a homomorphism extraction property [BC12, GGPR13] and linear-only encryption [BCI^{+}13]. With few exceptions, these existing candidates all rely on number-theoretic or group-theoretic assumptions.

Complexity metrics for SNARGs. In this work, we focus on simultaneously minimizing both the proof size and the prover complexity of succinct non-interactive arguments. For a security parameter λ, we measure the asymptotic cost of achieving soundness against provers (modeled as Boolean circuits) of size 2^{λ} with $2^{-\lambda}$ error (that is, a prover of size 2^{λ} should not be able to convince an honest verifier of a false statement, except with probability $2^{-\lambda}$). We say that a SNARG is *quasi-optimally succinct* if its proof length is $\widetilde{O}(\lambda)$, and that it is *quasi-optimal* if in addition, the prover's runtime is only polylogarithmically greater than the the running time of the classic NP prover. In Section 5.4.1, we show that this notion of quasi-optimal succinctness is tight (up to polylogarithmic factors): assuming NP does not have succinct proofs, no succinct argument system can provide the same soundness guarantees with proofs of size $o(\lambda)$.

Quasi-optimal SNARGs from lattices. In this work, we give two new candidate SNARG constructions: one that provides quasi-optimal succinctness based on standard lattices, and another that satisfies our notion of quasi-optimality based on ideal lattices over polynomial rings. These are the first lattice-based SNARG candidates to achieve these properties. We refer to Table 5.1 for a concrete comparison. Prior to this work, SNARGs with quasi-optimal succinctness were only known from pairing-based assumptions, and no quasi-optimal SNARG candidate from any concrete cryptographic assumption was known.[4]

Similar to previous works [BCI$^+$13], we take a two-step approach to construct our new lattice-based SNARG candidates. First, we construct an information-theoretic proof system that provides soundness against a restricted class of provers (e.g., *linearly-bounded* provers [IKO07]). We then leverage cryptographic tools (e.g., *linear-only* encryption [BCI$^+$13]) to compile the information-theoretic primitive into a succinct argument system. In this work, the core information-theoretic primitives we use are linear probabilistically-checkable proofs (linear PCPs) and linear multi-prover interactive proofs (linear MIPs). We then show how to directly compile linear PCPs and linear MIPs into preprocessing SNARGs using a new cryptographic primitive called linear-only vector encryption. We describe our construction of quasi-optimally succinct SNARGs from lattices in Chapter 4, and our construction of quasi-optimal SNARGs in Chapter 5. Finally, in Chapter 6, we explore some of the implications between *optimally-succinct* SNARGs and powerful forms of encryption.

Applications of succinct argument systems. One of the most direct applications of succinct argument systems is to outsourcing and verifiable delegation of computation. Over the last few years, there has been significant progress in designing and implementing scalable systems for verifiable computation that leverage succinct arguments in both the interactive setting [GKR08, CMT12, TRMP12, SMBW12, SVP$^+$12, Tha13, VSBW13] as well as the non-interactive setting [PHGR13, BCG$^+$13, BFR$^+$13b, BCTV14, WSR$^+$15, CFH$^+$15]. We refer to [WB15] and the references therein for a comprehensive survey of this area. More recently, succinct arguments (in conjunction with zero-knowledge properties) have featured as a core building block for new privacy-preserving cryptocurrencies [BCG$^+$14].

1.3 Why Lattices?

The focus of this thesis is constructing non-interactive argument (and proof) systems satisfying properties like zero-knowledge or succinctness from *lattice-based* assumptions—that is, hardness

[4]We discuss a heuristic approach for achieving quasi-optimality in Remark 5.55.

assumptions for problems defined over point lattices in $\mathbb{R}^n$. We refer to [MR09, Pei16] for surveys on lattice-based cryptography and to Section 2.1 for a technical overview of the hardness assumptions as well as the algebraic tools we leverage in our constructions.

As discussed in Sections 1.1 and 1.2, constructions of both non-interactive zero-knowledge arguments as well as succinct non-interactive arguments are already known in the random oracle model and from number-theoretic and group-theoretic assumptions in the CRS model. Realizing these primitives from standard lattice-assumptions (with similar properties) has remained open. Below, we outline several motivating reasons for studying lattice-based instantiations of these fundamental cryptographic primitives:

- **Conjectured post-quantum security.** A key appeal of lattice based cryptography is their conjectured security against *quantum* attacks. Traditional number-theoretic and group-theoretic problems such as computing discrete logarithms or factoring are all solvable in polynomial time on a quantum computer [Sho94]. In contrast, no efficient quantum algorithms are known for the typical problems used in lattice-based cryptography (e.g., the short integer solutions (SIS) or the learning with errors (LWE) problems). In light of the potential threat to classical cryptosystems posed by quantum computers, it is an important challenge to realize existing cryptographic primitives from post-quantum assumptions. In this work, we give new constructions of NIZK arguments (in a preprocessing model) as well as SNARGs from lattice-based assumptions which plausibly resist quantum attacks. Previous constructions of these primitives based on factoring or pairing-based assumptions do not provide security against quantum adversaries.

- **Worst-case hardness guarantees.** From a more theoretical perspective, one of the appeals of lattice-based cryptography is that we can base security on the hardness of solving certain problems in the *worst-case* (i.e., there exists at least one intractable instance of the problem). Falsifying a worst-case hardness assumption requires exhibiting an algorithm that solves *every* instance of the underlying problem. In contrast, typical cryptographic assumptions (e.g., factoring, discrete logarithm, etc.) are all formulated in the language of *average-case* hardness: namely, that there is some distribution over problem instances such that a *random* instance is intractable for a computationally-bounded algorithm. When formulating an average-case assumption, we must carefully specify the *hard* distribution (for instance, there are many "easy" distributions for factoring or discrete log over $\mathbb{Z}_p^*$). Identifying a hard distribution is unnecessary when working with worst-case hardness assumptions.

  The seminal work of Ajtai [Ajt96] gave the first worst-case to average-case reduction for lattice problems. Specifically, Ajtai's work showed that solving the short integers solutions (SIS) problem in the *average* case is as hard as solving the decisional approximate shortest vector problem (GapSVP) in the *worst* case. Ajtai's work thus gave the first construction of a cryptographic primitive with security from a worst-case complexity assumption. Subsequently, a long sequence of works have strengthened the worst-case to average-case reductions for both the SIS as well as the LWE problems [Mic04, Reg05, MR07, GPV08, Pei09, ACPS09, MM11, MP12, BLP$^+$13, MP13]. Thus, constructing new cryptosystems from lattice-based assumptions allows us to base security on *worst-case* assumptions.

- **Understanding the power (and limitations) of lattices.** Starting with Gentry's groundbreaking work in realizing fully homomorphic encryption from lattices [Gen09a, Gen09b], the last decade has seen a multitude of works leveraging lattice-based techniques to realize a number of powerful cryptographic notions such as identity-based encryption [ABB10a, ABB10b], attribute-based encryption [GVW13, BGG$^+$14], predicate encryption [GVW15a], homomorphic signatures [BF11a, GVW15b], constrained PRFs [BV15, BKM17, CC17a, BTVW17, BKW17, PS18], cryptographic watermarking [KW17], and traitor tracing [LPSS14, GKW18]. For many of these primitives, the only known instantiation from standard assumptions is based on lattice assumptions. In spite of the tremendous successes of lattice-based techniques, there are still numerous cryptographic primitives that still elude our best efforts to instantiate from lattices (a major one being NIZKs from lattices). In this thesis, we fill in some of these gaps and show how to realize several types of non-interactive argument (and proof) systems from lattices.

1.4 Works Contained in this Thesis

The results in this thesis are based on material that originally appeared in the following three publications:

- **Chapter 3:** *Multi-Theorem Preprocessing NIZKs from Lattices* with Sam Kim (Crypto, 2018) [KW18].
- **Chapter 4:** *Lattice-Based SNARGs and Their Application to More Efficient Obfuscation* with Dan Boneh, Yuval Ishai, and Amit Sahai (Eurocrypt, 2017) [BISW17].
- **Chapters 5-6:** *Quasi-Optimal SNARGs via Linear Multi-Prover Interactive Proofs* with Dan Boneh, Yuval Ishai, and Amit Sahai (Eurocrypt, 2018) [BISW18].

Chapter 2

Preliminaries

We begin by introducing the basic notation that we use throughout this thesis. Then, in Section 2.1, we provide some background on lattice-based cryptography. For an integer $n \geq 1$, we write $[n]$ to denote the set of integers $\{1, \ldots, n\}$. For positive integers $p, q > 1$, we write $\mathbb{Z}_p$ and $\mathbb{Z}_q$ to denote the ring of integers modulo p and q, respectively. For a finite set S, we write $x \xleftarrow{\text{R}} S$ to denote that x is sampled uniformly at random from S. For a distribution $\mathcal{D}$, we write $x \leftarrow \mathcal{D}$ to denote that x is sampled from $\mathcal{D}$.

Throughout this work, we use λ to denote a computational security parameter and κ to denote a statistical security parameter. We say that a function f is negligible in λ, denoted $\text{negl}(\lambda)$, if $f(\lambda) = o(1/\lambda^c)$ for all constants $c \in \mathbb{N}$. We say that an event happens with negligible probability if the probability of the event occurring is bounded by a negligible function, and we say that an event happens with overwhelming probability if its complement occurs with negligible probability. We say an algorithm is efficient if it runs in probabilistic polynomial time in the length of its input. We write $\text{poly}(\lambda)$ to denote a quantity whose value is upper-bounded by a fixed polynomial in λ. We say that two families of distributions $\mathcal{D}_1 = \{\mathcal{D}_{1,\lambda}\}_{\lambda \in \mathbb{N}}$ and $\mathcal{D}_2 = \{\mathcal{D}_{2,\lambda}\}_{\lambda \in \mathbb{N}}$ are computationally indistinguishable if no efficient algorithm can distinguish samples from either $\mathcal{D}_1$ or $\mathcal{D}_2$, except with negligible probability. We denote this by writing $\mathcal{D}_1 \overset{c}{\approx} \mathcal{D}_2$. We write $\mathcal{D}_1 \overset{s}{\approx} \mathcal{D}_2$ to denote that $\mathcal{D}_1$ and $\mathcal{D}_2$ are statistically indistinguishable (i.e., the statistical distance between $\mathcal{D}_1$ and $\mathcal{D}_2$ is bounded by a negligible function).

Vectors and matrices. We typically use bold uppercase letters (e.g., $\mathbf{A}$, $\mathbf{B}$) to denote matrices and bold lowercase letters (e.g., $\mathbf{u}, \mathbf{v}$) to denote vectors. Given two vectors $\mathbf{u} \in \mathbb{Z}^m$ and $\mathbf{v} \in \mathbb{Z}^n$, we write $\mathbf{u} \otimes \mathbf{v} \in \mathbb{Z}^{mn}$ to denote the tensor product of $\mathbf{u}$ with $\mathbf{v}$, or equivalently, the vector of pairwise products $u_i v_j$ for $i \in [m]$ and $j \in [n]$ of the entries in $\mathbf{u}$ and $\mathbf{v}$, respectively. For a matrix $\mathbf{A} \in \mathbb{F}^{t \times q}$ over a finite field $\mathbb{F}$, we write $\mathbf{A}_{[i_1, i_2]}$ (where $i_1, i_2 \in [t]$) to denote the sub-matrix of $\mathbf{A}$ containing rows i_1 through i_2 of $\mathbf{A}$ (inclusive). For $i \in [t]$ and $j \in [q]$, we write $\mathbf{A}_{i,j}$ and $\mathbf{A}[i, j]$ to refer to the entry in row i and column j of $\mathbf{A}$.

Boolean circuit satisfiability. For a Boolean circuit $C \colon \{0,1\}^n \times \{0,1\}^m \to \{0,1\}$, the Boolean circuit satisfaction problem is defined by the relation $\mathcal{R}_C = \{(\mathbf{x}, \mathbf{w}) \in \{0,1\}^n \times \{0,1\}^m : C(\mathbf{x}, \mathbf{w}) = 1\}$. We refer to $\mathbf{x} \in \{0,1\}^n$ as the statement and $\mathbf{w} \in \{0,1\}^m$ as the witness. We write $\mathcal{L}_C$ to denote the language associated with $\mathcal{R}_C$: namely, the set of statements $\mathbf{x} \in \{0,1\}^n$ for which there exists a witness $\mathbf{w} \in \{0,1\}^m$ such that $C(\mathbf{x}, \mathbf{w}) = 1$. For a family of Boolean circuits $\mathcal{C} = \left\{C_\ell \colon \{0,1\}^{n(\ell)} \times \{0,1\}^{m(\ell)} \to \{0,1\}\right\}_{\ell \in \mathbb{N}}$ indexed by a parameter ℓ, we write $\mathcal{R}_{\mathcal{C}} = \bigcup_{\ell \in \mathbb{N}} \mathcal{R}_{C_\ell}$ and $\mathcal{L}_{\mathcal{C}} = \bigcup_{\ell \in \mathbb{N}} \mathcal{L}_{C_\ell}$ for the corresponding (infinite) relation and language, respectively.

Arithmetic circuit satisfiability. In several cases in this work, it will be more natural to work with arithmetic circuits. For an arithmetic circuit $C \colon \mathbb{F}^n \times \mathbb{F}^m \to \mathbb{F}^h$ over a finite field $\mathbb{F}$, we say that C is satisfied if on an input $(\mathbf{x}, \mathbf{w}) \in \mathbb{F}^n \times \mathbb{F}^m$, all of the outputs are 0. Specifically, we define the relation for arithmetic circuit satisfiability to be $\mathcal{R}_C = \left\{(\mathbf{x}, \mathbf{w}) \in \mathbb{F}^n \times \mathbb{F}^m : C(\mathbf{x}, \mathbf{w}) = 0^h\right\}$.

Chosen-plaintext security. We also review the definition of chosen-plaintext security (CPA-security) for a symmetric encryption scheme.

Definition 2.1 (CPA-Secure Symmetric Encryption)**.** A (secret-key) encryption scheme with message space $\mathcal{M}$ is a tuple of efficient algorithms $\Pi_{\mathsf{enc}} = (\mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt})$ with the following properties:

- $\mathsf{KeyGen}(1^\lambda) \to \mathsf{sk}$: On input the security parameter λ, the key-generation algorithm outputs a secret key sk.
- $\mathsf{Encrypt}(\mathsf{sk}, m) \to \mathsf{ct}$: On input a secret key sk and a message $m \in \mathcal{M}$, the encryption algorithm outputs a ciphertext ct.
- $\mathsf{Decrypt}(\mathsf{sk}, \mathsf{ct}) \to m$: On input a secret key sk and a ciphertext ct, the decryption algorithm either outputs a message $m \in \mathcal{M}$ or a special symbol $\perp$ (to denote that decryption failed).

A CPA-secure symmetric encryption scheme should satisfy the following properties:

- **Correctness:** For all messages $m \in \mathcal{M}$, if we take $\mathsf{sk} \leftarrow \mathsf{KeyGen}(1^\lambda)$, then
$$\Pr[\mathsf{Decrypt}(\mathsf{sk}, \mathsf{Encrypt}(\mathsf{sk}, m)) = m] = 1.$$
- **CPA-Security:** For all efficient adversaries $\mathcal{A}$, if we take $\mathsf{sk} \leftarrow \mathsf{KeyGen}(1^\lambda)$, then
$$\left| \Pr\left[\mathcal{A}^{\mathcal{O}_0(\mathsf{sk}, \cdot, \cdot)}(1^\lambda) = 1\right] - \Pr\left[\mathcal{A}^{\mathcal{O}_1(\mathsf{sk}, \cdot, \cdot)}(1^\lambda) = 1\right] \right| = \operatorname{negl}(\lambda),$$
where $\mathcal{O}_b(\mathsf{sk}, m_0, m_1)$ outputs $\mathsf{Encrypt}(\mathsf{sk}, m_b)$ for $b \in \{0,1\}$.

The Schwartz-Zippel lemma. Finally, we recall the statement of the Schwartz-Zippel lemma [Sch80, Zip79], which we will use throughout this work.

Lemma 2.2 (Schwartz-Zippel Lemma [Sch80, Zip79])**.** *Let p be a prime and let $f \in \mathbb{Z}_p[x_1, \ldots, x_n]$ be a multivariate polynomial of total degree d, not identically zero. Then,*

$$\Pr[\alpha_1, \ldots, \alpha_n \xleftarrow{\text{R}} \mathbb{Z}_p : f(\alpha_1, \ldots, \alpha_n) = 0] \leq \frac{d}{p}.$$

2.1 Background on Lattice-Based Cryptography

In this section, we describe several known results for lattice-based cryptography that we use in this work.

Norms for vectors and matrices. Throughout this work, we will always use the infinity norm for vectors and matrices. This means that for a vector $\mathbf{x}$, the norm $\|\mathbf{x}\|$ is the maximal absolute value of an element in $\mathbf{x}$. Similarly, for a matrix $\mathbf{A}$, $\|\mathbf{A}\|$ is the maximal absolute value of any of its entries. If $\mathbf{x} \in \mathbb{Z}_q^n$ and $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$, then $\left\|\mathbf{x}^T \mathbf{A}\right\| \leq n \cdot \|\mathbf{x}\| \cdot \|\mathbf{A}\|$.

Learning with errors. We first review the learning with errors (LWE) assumption [Reg05]. Let $n, m, q \in \mathbb{N}$ be positive integers and χ be a noise (or error) distribution over $\mathbb{Z}_q$. In the $\mathsf{LWE}(n, m, q, \chi)$ problem, the adversary's goal is to distinguish between the two distributions

$$(\mathbf{A}, \mathbf{A}^T \mathbf{s} + \mathbf{e}) \quad \text{and} \quad (\mathbf{A}, \mathbf{u})$$

where $\mathbf{A} \xleftarrow{\text{R}} \mathbb{Z}_q^{n \times m}$, $\mathbf{s} \xleftarrow{\text{R}} \mathbb{Z}_q^n$, $\mathbf{e} \leftarrow \chi^m$, and $\mathbf{u} \xleftarrow{\text{R}} \mathbb{Z}_q^m$. When the error distribution χ is β-bounded[1], and under mild assumptions on the modulus q, the $\mathsf{LWE}(n, m, q, \chi)$ problem is as hard as approximating certain worst-case lattice problems such as GapSVP and SIVP on n-dimensional lattices to within a $\tilde{O}(n \cdot q/\beta)$ factor [Reg05, Pei09, ACPS09, MM11, MP12, BLP+13].

Short integer solutions. We also review the short integers solution (SIS) assumption [Ajt96]. Let $n, m, q, \beta \in \mathbb{N}$ be positive integers. In the $\mathsf{SIS}(n, m, q, \beta)$ problem, the adversary is given a uniformly random matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ and its goal is to find a vector $\mathbf{u} \in \mathbb{Z}_q^m$ with $\mathbf{u} \neq \mathbf{0}$ and $\|\mathbf{u}\| \leq \beta$ such that $\mathbf{A}\mathbf{u} = \mathbf{0}$. For any $m = \text{poly}(n)$, $\beta > 0$, and any sufficiently large $q \geq \beta \cdot \text{poly}(n)$, solving the $\mathsf{SIS}(n, m, q, \beta)$ problem is as hard as approximating certain worst-case lattice problems such as GapSVP and SIVP on n-dimensional lattices to within a $\beta \cdot \text{poly}(n)$ factor [Ajt96, Mic04, MR07, MP13]. It is also implied by the hardness of the LWE problem.

[1]We say that a distribution $\mathcal{D}$ is β-bounded if the support of $\mathcal{D}$ is $\{-\beta, \ldots, \beta - 1, \beta\}$ with probability 1.

The gadget matrix. We define the "gadget matrix" $\mathbf{G} = \mathbf{g} \otimes \mathbf{I}_n \in \mathbb{Z}_q^{n \times n \cdot \lceil \log q \rceil}$ where $\mathbf{g} = (1, 2, 4, \ldots, 2^{\lceil \log q \rceil - 1})$. We define the inverse function $\mathbf{G}^{-1} \colon \mathbb{Z}_q^{n \times m} \to \mathbb{Z}_q^{n \cdot \lceil \log q \rceil \times m}$ which expands each entry $x \in \mathbb{Z}_q$ in the input matrix into a column of size $\lceil \log q \rceil$ consisting of the bits of the binary representation of x. To simplify the notation, we always assume that $\mathbf{G}$ has width m (in our construction, $m = \Theta(n \log q)$). Note that this is without loss of generality since we can always extend $\mathbf{G}$ by appending zero columns. For any matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$, we have that $\mathbf{G} \cdot \mathbf{G}^{-1}(\mathbf{A}) = \mathbf{A}$.

Lattice trapdoors. Although solving the SIS problem for a uniformly random matrix $\mathbf{A}$ is believed to be hard, with some auxiliary trapdoor information (e.g., a set of short generating vectors for the lattice induced by $\mathbf{A}$), the problem becomes easy. Lattice trapdoors have featured in many applications and have been extensively studied [Ajt99, GPV08, AP09, MP12, LW15]. Since the specific details of the constructions are not essential for understanding this work, we just recall the main properties that we require in the following theorem.

Theorem 2.3 (Lattice Trapdoors [Ajt99, GPV08, AP09, MP12, LW15])**.** *Fix a security parameter λ and lattice parameters n, q, m and a norm bound β where $m = O(n \log q)$ and $\beta = O(n\sqrt{\log q})$. Then, there exists a tuple of efficient algorithms* $(\mathsf{TrapGen}, \mathsf{Sample}, \mathsf{SamplePre})$ *with the following properties:*

- $\mathsf{TrapGen}(1^\lambda) \to (\mathbf{A}, \mathsf{td})$*: On input the security parameter λ, the trapdoor generation algorithm outputs a rank-n matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ and a trapdoor* td.
- $\mathsf{Sample}(\mathbf{A}) \to \mathbf{U}$*: On input a matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$, the sampling algorithm returns a matrix $\mathbf{U} \in \mathbb{Z}_q^{m \times m}$.*
- $\mathsf{SamplePre}(\mathbf{A}, \mathbf{V}, \mathsf{td}) \to \mathbf{U}$*: On input a matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$, a target matrix $\mathbf{V} \in \mathbb{Z}_q^{n \times m}$, and a trapdoor* td*, the preimage-sampling algorithm outputs a matrix $\mathbf{U} \in \mathbb{Z}_q^{m \times m}$.*
- *The above algorithms satisfy the following properties. Take $(\mathbf{A}, \mathsf{td}) \leftarrow \mathsf{TrapGen}(1^\lambda)$. Then,*
    1. *For $\mathbf{U} \leftarrow \mathsf{Sample}(\mathbf{A})$, we have $\|\mathbf{U}\| \leq \beta$.*
    2. *For any $\mathbf{V} \in \mathbb{Z}_q^{n \times m}$ and $\mathbf{U} \leftarrow \mathsf{SamplePre}(\mathbf{A}, \mathbf{V}, \mathsf{td})$, we have $\mathbf{AU} = \mathbf{V}$ and $\|\mathbf{U}\| \leq \beta$.*
    3. *For $(\mathbf{A}, \mathsf{td}) \leftarrow \mathsf{TrapGen}(1^\lambda)$, $\mathbf{A}' \xleftarrow{\text{R}} \mathbb{Z}_q^{n \times m}$, $\mathbf{U} \leftarrow \mathsf{Sample}(\mathbf{A})$, $\mathbf{V} = \mathbf{AU}$, $\mathbf{V}' \xleftarrow{\text{R}} \mathbb{Z}_q^{n \times m}$, and $\mathbf{U}' \leftarrow \mathsf{SamplePre}(\mathbf{A}, \mathbf{V}', \mathsf{td})$, we have*

$$\mathbf{A} \overset{s}{\approx} \mathbf{A}' \qquad \text{and} \qquad (\mathbf{A}, \mathsf{td}, \mathbf{U}, \mathbf{V}) \overset{s}{\approx} (\mathbf{A}, \mathsf{td}, \mathbf{U}', \mathbf{V}').$$

Traditionally, lattice trapdoors consist of a set of short generating vectors of the lattice that is induced by a public SIS matrix $\mathbf{A}$. In this work, we make use of an alternative form of lattice trapdoors called a $\mathbf{G}$-trapdoor formalized in [MP12]. A $\mathbf{G}$-trapdoor of a matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ consists of a full-rank, low-norm matrix $\mathbf{R} \in \mathbb{Z}_q^{m \times m}$ satisfying the relation $\mathbf{AR} = \mathbf{G}$. These types of trapdoor matrices have additional statistical properties that we use in Sections 3.2 and 3.4. We summarize these properties in the following theorem.

Theorem 2.4 (Lattice Sampling [CHKP10, ABB10a, MP12, BGG$^+$14, LW15])**.** *Fix a security parameter λ and lattice parameters n, q, m, and a norm bound β, where $m = O(n \log q)$ and $\beta = O(n\sqrt{\log q})$. Then, in addition to the algorithms* $(\mathsf{TrapGen}, \mathsf{Sample}, \mathsf{SamplePre})$ *from Theorem 2.3, there exists a pair of algorithms* $(\mathsf{SampleLeft}, \mathsf{SampleRight})$ *with the following properties:*

- $\mathsf{SampleLeft}(\mathbf{A}, \mathbf{B}, \mathbf{R}, \mathbf{v}, \beta^*) \to \mathbf{u}$*: On input matrices $\mathbf{A}, \mathbf{B} \in \mathbb{Z}_q^{n\times m}$, a matrix $\mathbf{R} \in \mathbb{Z}_q^{m\times m}$ (trapdoor of $\mathbf{A}$), a target vector $\mathbf{v} \in \mathbb{Z}_q^n$, and a norm bound β^*,* $\mathsf{SampleLeft}$ *returns a vector $\mathbf{u} \in \mathbb{Z}_q^{2m}$.*

- $\mathsf{SampleRight}(\mathbf{A}, \mathbf{B}, \mathbf{U}, \mathbf{v}, \beta^*) \to \mathbf{u}$*: On input matrices $\mathbf{A}, \mathbf{B} \in \mathbb{Z}_q^{n\times m}$, a matrix $\mathbf{U} \in \mathbb{Z}_q^{m\times m}$, a target vector $\mathbf{v} \in \mathbb{Z}_q^n$, and a norm bound β^*,* $\mathsf{SampleRight}$ *returns a vector $\mathbf{u} \in \mathbb{Z}_q^{2m}$.*

- *The algorithms above satisfies the following properties. For* any *rank-n matrices $\mathbf{A}, \mathbf{B} \in \mathbb{Z}_q^{n\times m}$ and a target vector $\mathbf{v} \in \mathbb{Z}_q^n$, we have:*

    1. *Let $\mathbf{R} \in \mathbb{Z}_q^{m\times m}$ be any matrix satisfying $\mathbf{AR} = \mathbf{G}$ and $\|\mathbf{R}\| \cdot \omega(m\sqrt{\log m}) \le \beta^* \le q$. Then, for $\mathbf{u}_0 \leftarrow$* $\mathsf{SampleLeft}(\mathbf{A}, \mathbf{B}, \mathbf{R}, \mathbf{v}, \beta^*)$*, we have that $[\mathbf{A} \mid \mathbf{B}] \cdot \mathbf{u}_0 = \mathbf{v}$ and $\|\mathbf{u}_0\| \le \beta^*$.*
    2. *Let $\mathbf{U} \in \mathbb{Z}_q^{m\times m}$ be any matrix satisfying $\mathbf{AU} + y\mathbf{G} = \mathbf{B}$ for some $y \neq 0$ where $y \in \mathbb{Z}_q$ and $\|\mathbf{U}\| \cdot \omega(m\sqrt{\log m}) \le \beta^* \le q$. Then, for $\mathbf{u}_1 \leftarrow$* $\mathsf{SampleRight}(\mathbf{A}, \mathbf{B}, \mathbf{U}, \mathbf{v}, \beta^*)$*, we have that $[\mathbf{A} \mid \mathbf{B}] \cdot \mathbf{u}_1 = \mathbf{v}$ and $\|\mathbf{u}_1\| \le \beta^*$.*
    3. *The distributions of $\mathbf{u}_0, \mathbf{u}_1$ above are statistically indistinguishable.*

GSW homomorphic operations. In this work, we use the homomorphic structure from the fully homomorphic encryption (FHE) scheme by Gentry, Sahai, and Waters [GSW13]. Since we do not require the specific details of the homomorphic operations, we summarize the properties we need in the theorem below. In the language of FHE, the algorithm EvalPK corresponds to homomorphic evaluation over ciphertexts, while EvalU corresponds to homomorphic evaluation over the encryption randomness.

Theorem 2.5 (GSW Homomorphic Operations [GSW13, BV14, AP14, GV15])**.** *Fix a security parameter λ, lattice parameters n, q, m, a norm bound β, a depth bound d, and a message length ℓ, where $m = O(n \log q)$ and $\beta \cdot 2^{\widetilde{O}(d)} < q$. Then, there exists a pair of efficient deterministic algorithms* $(\mathsf{EvalPK}, \mathsf{EvalU})$ *with the following properties:*

- $\mathsf{EvalPK}(\mathbf{V}_1, \dots, \mathbf{V}_\ell, C) \to \mathbf{V}_C$*: On input matrices $\mathbf{V}_1, \dots, \mathbf{V}_\ell \in \mathbb{Z}_q^{n\times m}$ and a circuit $C\colon \{0,1\}^\ell \to \{0,1\}$ of depth at most d,* EvalPK *returns an evaluated matrix $\mathbf{V}_C \in \mathbb{Z}_q^{n\times m}$.*

- $\mathsf{EvalU}\big((\mathbf{V}_1, x_1, \mathbf{U}_1), \dots, (\mathbf{V}_\ell, x_\ell, \mathbf{U}_\ell), C\big) \to \mathbf{U}_C$*: On input tuples $(\mathbf{V}_i, x_i, \mathbf{U}_i)$, where $\mathbf{V}_i \in \mathbb{Z}_q^{n\times m}$, $x_i \in \{0,1\}$, and $\mathbf{U}_i \in \mathbb{Z}_q^{m\times m}$ for all $i \in [\ell]$, and a circuit $C\colon \{0,1\}^\ell \to \{0,1\}$,* EvalU *returns an evaluated matrix $\mathbf{U}_C \in \mathbb{Z}_q^{m\times m}$.*

- *For all circuits $C \colon \{0,1\}^\ell \to \{0,1\}$ of depth at most d, and all matrices $\mathbf{A}, \mathbf{V}_1, \ldots, \mathbf{V}_\ell \in \mathbb{Z}_q^{n \times m}$, inputs $x_1, \ldots, x_\ell \in \{0,1\}$, and matrices $\mathbf{U}_1, \ldots, \mathbf{U}_\ell \in \mathbb{Z}_q^{m \times m}$ where*

$$\mathbf{A}\mathbf{U}_i + x_i \cdot \mathbf{G} = \mathbf{V}_i \quad \forall i \in [\ell],$$

and $\|\mathbf{U}_i\| \le \beta$, the EvalPK and EvalU algorithms satisfy the following property. For $\mathbf{V}_C \leftarrow \mathsf{EvalPK}(\mathbf{V}_1, \ldots, \mathbf{V}_\ell, C)$, and $\mathbf{U}_C \leftarrow \mathsf{EvalU}((\mathbf{V}_1, x_1, \mathbf{U}_1), \ldots, (\mathbf{V}_\ell, x_\ell, \mathbf{U}_\ell), C)$, we have that

$$\mathbf{A}\mathbf{U}_C + C(x) \cdot \mathbf{G} = \mathbf{V}_C \quad \textit{and} \quad \|\mathbf{U}_C\| \le \beta \cdot 2^{\widetilde{O}(d)} < q.$$

CPA-secure encryption from lattices. Finally, we note that CPA-secure symmetric encryption (Definition 2.1) can be built from any one-way function. Here, we state one candidate that follows from any lattice-based PRF that can be computed by a circuit of depth independent of its output length (c.f., [GGM84, Ajt96]).

Fact 2.6 (CPA-Secure Encryption from LWE)**.** Let λ be a security parameter. Under the LWE assumption (see Section 2.1), there exists a CPA-secure encryption scheme $\Pi_{\mathsf{enc}} = (\mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt})$ over a message space $\mathcal{M}$ with the following properties:

- For all $m \in \mathcal{M}$, $\mathsf{sk} \leftarrow \mathsf{KeyGen}(1^\lambda)$, and $\mathsf{ct} \leftarrow \mathsf{Encrypt}(\mathsf{sk}, m)$, we have that $|\mathsf{ct}| = |m| + \mathrm{poly}(\lambda)$.
- The decryption algorithm $\mathsf{Decrypt}$ can be computed by a circuit of depth $\mathrm{poly}(\lambda)$.

Chapter 3

Non-Interactive Zero-Knowledge Arguments

In this chapter, we show how to construct non-interactive zero-knowledge (NIZK) arguments (and proofs) in the preprocessing model from standard lattice assumptions. We refer to NIZKs in the preprocessing model as a "preprocessing NIZK." As we discuss in Section 3.1.1 and in Remark 3.33, our works gives the *first* candidate of a reusable (i.e., multi-theorem) NIZK argument (and proof) from a standard lattice assumption.

3.1 Construction Overview

We begin by providing a high-level overview of our construction. Then, in Section 3.1.1, we discuss our results in the context of other relevant works.

Homomorphic signatures. A *homomorphic signature* scheme [BF11a, BF11b, GVW15b, ABC$^+$15] enables computations on *signed* data. Specifically, a user can sign a message $\mathbf{x} \in \{0,1\}^\ell$ using her private signing key to obtain a signature $\boldsymbol{\sigma}$. Later on, she can delegate the pair $(\mathbf{x}, \boldsymbol{\sigma})$ to an untrusted data processor. The data processor can then compute an arbitrary function g on the signed data to obtain a value $y = g(\mathbf{x})$ along with a signature $\sigma_{g,y}$. The computed signature $\sigma_{g,y}$ should certify that the value y corresponds to a *correct* evaluation of the function g on the original input $\mathbf{x}$. In a *context-hiding* homomorphic signature scheme [BFF$^+$09, BF11a], the computed signature $\sigma_{g,y}$ also *hides* the input message $\mathbf{x}$. Namely, the pair $(y, \sigma_{g,y})$ reveals no information about $\mathbf{x}$ other than what could be inferred from the output $y = g(\mathbf{x})$. Gorbunov et al. [GVW15b] gave the first construction of a context-hiding homomorphic signature scheme for general Boolean circuits (with bounded depth) from standard lattice assumptions. We refer to Section 3.1.1 for a more comprehensive survey on homomorphic signature schemes.

From homomorphic signatures to zero-knowledge. The notion of context-hiding in a homomorphic signature scheme already bears a strong resemblance to zero-knowledge. Namely, a context-hiding homomorphic signature scheme allows a user (e.g., a prover) to certify the result of a computation (e.g., the output of an NP relation) without revealing any additional information about the input (e.g., the NP witness) to the computation. Consider the following scenario. Suppose the prover has a statement-witness pair $(\mathbf{x}, \mathbf{w})$ for some NP relation $\mathcal{R}$ and wants to convince the verifier that $\mathcal{R}(\mathbf{x}, \mathbf{w}) = 1$ without revealing $\mathbf{w}$. For sake of argument, suppose the prover has obtained a signature $\sigma_{\mathbf{w}}$ on the witness $\mathbf{w}$ (but does not have the signing key for the signature scheme), and the verifier holds the verification key for the signature scheme. In this case, the prover can construct a zero-knowledge proof for $\mathbf{x}$ by evaluating the relation $\mathcal{R}_{\mathbf{x}}(\mathbf{w}) := \mathcal{R}(\mathbf{x}, \mathbf{w})$ on $(\mathbf{w}, \sigma_{\mathbf{w}})$. If $\mathcal{R}(\mathbf{x}, \mathbf{w}) = 1$, then this yields a new signature $\sigma_{\mathcal{R},\mathbf{x}}$ on the bit 1. The proof for $\mathbf{x}$ is just the signature $\sigma_{\mathcal{R},\mathbf{x}}$. Context-hiding of the homomorphic signature scheme says that the signature $\sigma_{\mathcal{R},\mathbf{x}}$ reveals no information about the input to the computation (the witness $\mathbf{w}$) other than what is revealed by the output of the computation (namely, that $\mathcal{R}(\mathbf{x}, \mathbf{w}) = 1$). This is precisely the zero-knowledge property. Soundness of the proof system follows by unforgeability of the homomorphic signature scheme (if there is no $\mathbf{w}$ such that $\mathcal{R}_{\mathbf{x}}(\mathbf{w}) = 1$, the prover would not be able to produce a signature on the value 1 that verifies according to the function $\mathcal{R}_{\mathbf{x}}$).

While this basic observation suggests a connection between homomorphic signatures and zero-knowledge, it does not directly give a NIZK argument. A key problem is that to construct the proof, the prover must already possess a signature on its witness $\mathbf{w}$. But since the prover does not have the signing key (if it did, then the proof system is no longer sound), it is unclear how the prover obtains this signature on $\mathbf{w}$ without interacting with the verifier (who could hold the signing key). This is the case even in the preprocessing model, because we require that the preprocessing be statement-independent (and in fact, reusable for arbitrarily many adaptively-chosen statements).

Preprocessing NIZKs from homomorphic signatures. Nonetheless, the basic observation shows that if we knew ahead of time which witness $\mathbf{w}$ the prover would use to construct its proofs, then the setup algorithm can simply give the prover a homomorphic signature $\sigma_{\mathbf{w}}$ on $\mathbf{w}$. To support this, we add a layer of indirection. Instead of proving that it knows a witness $\mathbf{w}$ where $\mathcal{R}(\mathbf{x}, \mathbf{w}) = 1$, the prover instead demonstrates that it has an encryption $\mathsf{ct}_{\mathbf{w}}$ of $\mathbf{w}$ (under some key sk), and that it knows some secret key sk such that $\mathsf{ct}_{\mathbf{w}}$ decrypts to a valid witness $\mathbf{w}$ where $\mathcal{R}(\mathbf{x}, \mathbf{w}) = 1$.[1] A proof of the statement $\mathbf{x}$ then consists of the encrypted witness $\mathsf{ct}_{\mathbf{w}}$ and a proof $\pi_{\mathcal{R},\mathbf{x},\mathsf{ct}_{\mathbf{w}}}$ that $\mathsf{ct}_{\mathbf{w}}$ is an encryption of a satisfying witness (under *some* key). First, if the encryption scheme is semantically-secure and the proof is zero-knowledge, then the resulting construction satisfies (computational)

[1]This is a classic technique in the construction of non-interactive proof systems and has featured in many contexts (e.g., [SP92, GGI$^+$15]).

zero-knowledge. Moreover, the witness the prover uses to construct $\pi_{\mathcal{R},\mathbf{x},\mathsf{ct}_\mathbf{w}}$ is always the same: the secret key sk. Notably, the witness is statement-independent and can be reused to prove arbitrarily many statements (provided the encryption scheme is CPA-secure).

This means we can combine context-hiding homomorphic signatures (for general circuits) with any CPA-secure symmetric encryption scheme to obtain NIZKs in the preprocessing model as follows:

- **Setup:** The setup algorithm generates a secret key sk for the encryption scheme as well as parameters for a homomorphic signature scheme. Both the proving and verification keys include the public parameters for the signature scheme. The proving key k_P additionally contains the secret key sk and a signature σ_{sk} on sk.

- **Prove:** To generate a proof that an NP statement $\mathbf{x}$ is true, the prover takes a witness $\mathbf{w}$ where $\mathcal{R}(\mathbf{x}, \mathbf{w}) = 1$ and encrypts $\mathbf{w}$ under sk to obtain a ciphertext $\mathsf{ct}_\mathbf{w}$. Next, we define the witness-checking function $\mathsf{CheckWitness}[\mathcal{R}, \mathbf{x}, \mathsf{ct}_\mathbf{w}]$ (parameterized by $\mathcal{R}$, $\mathbf{x}$, and $\mathsf{ct}_\mathbf{w}$) that takes as input a secret key sk and outputs 1 if $\mathcal{R}(\mathbf{x}, \mathsf{Decrypt}(\mathsf{sk}, \mathsf{ct}_\mathbf{w})) = 1$, and 0 otherwise. The prover homomorphically evaluates $\mathsf{CheckWitness}[\mathcal{R}, \mathbf{x}, \mathsf{ct}_\mathbf{w}]$ on $(\mathsf{sk}, \sigma_{\mathsf{sk}})$ to obtain a new signature σ^* on the value 1. The proof consists of the ciphertext $\mathsf{ct}_\mathbf{w}$ and the signature σ^*.

- **Verify:** Given a statement $\mathbf{x}$ for an NP relation $\mathcal{R}$ and a proof $\pi = (\mathsf{ct}, \sigma^*)$, the verifier checks that σ^* is a valid signature on the bit 1 according to the function $\mathsf{CheckWitness}[\mathcal{R}, \mathbf{x}, \mathsf{ct}]$. Notice that the description on the function only depends on the relation $\mathcal{R}$, the statement $\mathbf{x}$, and the ciphertext ct, all of which are known to the verifier.

Since the homomorphic signature scheme is context-hiding, the signature σ^* hides the input to $\mathsf{CheckWitness}[\mathcal{R}, \mathbf{x}, \mathsf{ct}_\mathbf{w}]$, which in this case, is the secret key sk. By CPA-security of the encryption scheme, the ciphertext hides the witness $\mathbf{w}$, so the scheme provides zero-knowledge. Soundness again follows from unforgeability of the signature scheme. Thus, by combining a lattice-based homomorphic signature scheme for general circuits [GVW15b] with any lattice-based CPA-secure symmetric encryption scheme, we obtain a (multi-theorem) preprocessing NIZK from lattices.

An appealing property of our preprocessing NIZKs is that the proofs are short: the length of a NIZK argument for an NP relation $\mathcal{R}$ is $|\mathbf{w}| + \mathrm{poly}(\lambda, d)$ bits, where $|\mathbf{w}|$ is the length of a witness for $\mathcal{R}$ and d is the depth of the circuit computing $\mathcal{R}$. The proof size in NIZK constructions from trapdoor permutations or pairings [FLS90, DDO$^+$01, GOS06, Gro10, GOS12] typically scale with the *size* of the circuit computing $\mathcal{R}$ and *multiplicatively* with the security parameter. Previously, Gentry et al. [GGI$^+$15] gave a generic approach using fully homomorphic encryption (FHE) to reduce the proof size in any NIZK construction. The advantage of our approach is that we naturally satisfy this succinctness property, and the entire construction can be based only on lattice assumptions (without needing to mix assumptions). We discuss this in greater detail in Remark 3.29. We also give the complete description of our preprocessing NIZK and security analysis in Section 3.3.

In the above construction, if the homomorphic signature scheme is unforgeable even against computationally-unbounded adversaries, then the construction gives a NIZK *proof* in the preprocessing model. In Remark 3.32, we describe how to realize this using lattice-based context-hiding statistically-binding *homomorphic commitments* [GVW15b]. One of the advantages of using homomorphic signatures instead of homomorphic commitments is that they enable an efficient two-party protocol for implementing the preprocessing, which we discuss below. For this reason, we focus primarily on constructing preprocessing NIZK arguments from homomorphic signatures.

Blind homomorphic signatures for efficient preprocessing. A limitation of preprocessing NIZKs is we require a trusted setup to generate the proving and verification keys. One solution is to have the prover and verifier run a (malicious-secure) two-party computation protocol (e.g., [LP07]) to generate the proving and verification keys. However, generic MPC protocols are often costly and require making *non-black-box* use of the underlying homomorphic signature scheme.

In this work, we describe a conceptually simple and more efficient way of implementing the preprocessing without relying on general MPC. We do so by introducing a new cryptographic notion called *blind homomorphic signatures*. First, we observe that we can view the two-party computation in the setup phase as essentially implementing a "blind signing" protocol where the verifier holds the signing key for the homomorphic signature scheme and the prover holds the secret key sk. At the end of the blind signing protocol, the prover should learn σ_{sk} while the verifier should not learn anything about sk. This is precisely the properties guaranteed by a blind signature protocol [Cha82, Fis06]. In this work, we introduce the notion of a blind homomorphic signature scheme which combines the blind signing protocol of traditional blind signature schemes while retaining the ability to homomorphically operate on ciphertexts. Since the notion of a blind homomorphic signatures is inherently a two-party functionality, we formalize it in the model of universal composability [Can01]. We provide the formal definition of the ideal blind homomorphic signature functionality in Section 3.4.

In Section 3.4.3, we show how to securely realize our ideal blind homomorphic signature functionality in the presence of *malicious* adversaries by combining homomorphic signatures with any UC-secure oblivious transfer (OT) protocol [CLOS02]. Note that security against malicious adversaries is critical for our primary application of leveraging blind homomorphic signatures to implement the setup algorithm of our preprocessing NIZK candidate. At a high-level, we show how to construct a blind homomorphic signature scheme from any "bitwise" homomorphic signature scheme—namely, a homomorphic signature scheme where the signature on an ℓ-bit message consists of ℓ signatures, one for each bit of the message. Moreover, we assume that the signature on each bit position only depends on the value of that particular bit (and not the value of any of the other bits of the message); of course, the ℓ signatures can still be generated using common or correlated randomness. Given a bitwise homomorphic signature scheme, we can implement the blind signing protocol (on ℓ-bit messages) using ℓ independent 1-out-of-2 OTs. Specifically, the signer plays the role of the sender in the OT protocol and for each index $i \in [\ell]$, the signer signs both the bit 0 as well as the bit 1. Then, to obtain a signature on an ℓ-bit message, the receiver requests the signatures corresponding to the bits of its message.

While the high-level schema is simple, there are a few additional details that we have to handle to achieve robustness against a malicious signer. For instance, a malicious signer can craft the parameters of the homomorphic signature scheme so that when an evaluator computes on a signature, the resulting signatures no longer provide context-hiding. Alternatively, a malicious signer might mount a "selective-failure" attack during the blind-signing protocol to learn information about the receiver's message. We discuss how to address these problems by giving strong definitions of malicious context-hiding for homomorphic signatures in Section 3.2, and give the full construction of blind homomorphic signatures from oblivious transfer in Section 3.4.3. In particular, we show that the Gorbunov et al. [GVW15b] homomorphic signature construction satisfies our stronger security notions, and so, coupled with the UC-secure lattice-based OT protocol of Peikert et al. [PVW08], we obtain a UC-secure blind homomorphic signature scheme from standard lattice assumptions. Moreover, the blind signing protocol is a two-round protocol, and only makes black-box use of the underlying homomorphic signature scheme.

UC-secure preprocessing NIZKs. Finally, we show that using our UC-secure blind homomorphic signature candidate, we can in fact realize the stronger notion of UC-secure NIZK arguments in a preprocessing model from standard lattice assumptions. This means that our NIZKs can be arbitrarily composed with other cryptographic protocols. Our new candidates are thus suitable to instantiate many of the classic applications of NIZKs for boosting the security of general MPC protocols. As we show in Section 3.5, combining our preprocessing UC-NIZKs with existing lattice-based semi-malicious MPC protocols such as [MW16] yields malicious-secure protocols purely from standard lattice assumptions (in a reusable preprocessing model). We also show that our constructions imply a *succinct* version of the classic GMW [GMW86, GMW87] protocol compiler (where the total communication overhead of the compiled protocol depends only on the *depth*, rather than the *size* of the computation).

3.1.1 Additional Related Work

In this section, we survey some additional related work on NIZK constructions, blind signatures, and homomorphic signatures.

Other NIZK proof systems. In the CRS model, there are several NIZK constructions based on specific number-theoretic assumptions such as quadratic residuosity [BFM88, DMP87, BDMP91]. These candidates are also secure in the *bounded-theorem* setting where the CRS can only be used for an *a priori* bounded number of proofs. Exceeding this bound compromises soundness or zero-knowledge. In the preprocessing model, Kalai and Raz [KR06] gave a single-theorem *succinct* NIZK proof system for the class LOGSNP from polylogarithmic private information retrieval (PIR) and *exponentially-hard* OT. In this work, we focus on constructing multi-theorem NIZKs, where an *arbitrary* number of proofs can be constructed after an initial setup phase.

NIZKs have also been constructed for specific algebraic languages in both the publicly-verifiable setting [Gro06, GS08] as well as the designated-verifier setting [CC17b]. In the specific case of lattice-based constructions, there are several works on building hash-proof systems, (also known as smooth projective hash functions [CS02]) [KV09, ZY17, BBDQ18], which are designated-verifier NIZK proofs for a *specific* language (typically, this is the language of ciphertexts associated with a particular message). In the random oracle model, there are also constructions of lattice-based NIZK arguments from Σ-protocols [LNSW13, XXW13]. Recently, there has also been work on instantiating the random oracle in Σ-protocols with lattice-based correlation-intractable hash functions [CCRR18]. However, realizing the necessary correlation-intractable hash functions from lattices requires making the non-standard assumption that Regev's encryption scheme [Reg05] is *exponentially KDM-secure* against all polynomial-time adversaries. In our work, we focus on NIZK constructions for general NP languages in the plain model (without random oracles) from the *standard* LWE assumption (i.e., polynomial hardness of LWE with a subexponential approximation factor).

Very recently, Rothblum et al. [RSS18] showed that a NIZK proof system for a decisional variant of the bounded distance decoding (BDD) problem suffices for building NIZK proof systems for NP.

Blind signatures. The notion of blind signatures was first introduced by Chaum [Cha82]. There are many constructions of blind signatures from a wide range of assumptions in the random oracle model [Sch89, Bra00, PS00, Abe01, Bol03, BNPS03, Rüc10, BL13], the CRS model [CKW04, KZ06, Fis06, AO09, Fuc09, AFG$^+$10, AHO10, GS12], as well as the standard model [GRS$^+$11, FHS15, FHKS16, HK16].

Homomorphic signatures. There are numerous constructions of linearly homomorphic signatures [ABC$^+$07, SW08, DVW09, AKK09, BFKW09, GKKR10, BF11a, AL11, BF11b, CFW12, Fre12, ABC$^+$15]. Beyond linear homomorphisms, a number of works [BF11a, BFR13a, CFW14] have constructed homomorphic signatures for polynomial functions from lattices or multilinear maps. Subsequently, Gorbunov et al. [GVW15b] gave the first homomorphic signature scheme for general circuits from lattices, and Fiore et al. [FMNP16] gave the first "multi-key" homomorphic signature scheme for general circuits from lattices. In a multi-key scheme, homomorphic operations can be performed on signatures signed under *different* keys.

3.2 Homomorphic Signatures

A homomorphic signature scheme enables computations on signed data. Given a Boolean function $C \colon \{0,1\}^\ell \to \{0,1\}$ (modeled as a Boolean circuit) and a signature $\boldsymbol{\sigma}_{\mathbf{x}}$ that certifies a message

$\mathbf{x} \in \{0,1\}^\ell$, one can homomorphic derive a signature $\boldsymbol{\sigma}_{C(\mathbf{x})}$ that certifies the value $C(\mathbf{x})$ with respect to the circuit C. The two main security notions that we are interested in are unforgeability and context-hiding. We first provide a high-level description of the properties:

- **Unforgeability:** We say a signature scheme is unforgeable if an adversary who has a signature $\boldsymbol{\sigma}_{\mathbf{x}}$ on a message $\mathbf{x}$ cannot produce a valid signature on any message $y \neq C(\mathbf{x})$ that verifies with respect to the function C.

- **Context-hiding:** Context-hiding says that when one evaluates a function C on a message-signature pair $(\mathbf{x}, \boldsymbol{\sigma}_{\mathbf{x}})$, the resulting signature $\sigma_{C(\mathbf{x})}$ on $C(\mathbf{x})$ should not reveal any information about the original message $\mathbf{x}$ other than the circuit C and the value $C(\mathbf{x})$. In our definition, the homomorphic signature scheme contains an explicit "hide" function that implements this transformation.

Syntax and notation. Our construction of blind homomorphic signatures from standard homomorphic signatures (Section 3.4.3) will impose some additional structural requirements on the underlying scheme. Suppose the message space for the homomorphic signature scheme consists of ℓ-tuples of elements over a set $\mathcal{X}$ (e.g., the case where $\mathcal{X} = \{0,1\}$ corresponds to the setting where the message space consists of ℓ-bit strings). Then, we require that the public parameters pp of the scheme can be split into a vector of public keys $\mathsf{pp} = (\mathsf{pk}_1, \ldots, \mathsf{pk}_\ell)$. In addition, a (fresh) signature on a vector $\mathbf{x} \in \mathcal{X}^\ell$ can also be written as a tuple of ℓ signatures $\boldsymbol{\sigma} = (\sigma_1, \ldots, \sigma_\ell)$ where σ_i can be verified with respect to the verification key vk and the i^{th} public key pk_i for all $i \in [\ell]$. In our description below, we often use vector notation to simplify the presentation.

Definition 3.1 (Homomorphic Signatures [BF11b, GVW15b])**.** A *homomorphic signature scheme* with message space $\mathcal{X}$, message length $\ell \in \mathbb{N}$, and function class $\mathcal{C} = \{\mathcal{C}_\lambda\}_{\lambda \in \mathbb{N}}$, where each $\mathcal{C}_\lambda$ is a collection of functions from $\mathcal{X}^\ell$ to $\mathcal{X}$, is defined by a tuple of algorithms $\Pi_{\mathsf{HS}} = (\mathsf{PrmsGen}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{PrmsEval}, \mathsf{SigEval}, \mathsf{Hide}, \mathsf{Verify}, \mathsf{VerifyFresh}, \mathsf{VerifyHide})$ with the following properties:

- $\mathsf{PrmsGen}(1^\lambda, 1^\ell) \to \mathsf{pp}$: On input the security parameter λ and message length ℓ, the parameter-generation algorithm returns a set of ℓ public keys $\mathsf{pp} = (\mathsf{pk}_1, \ldots, \mathsf{pk}_\ell)$.

- $\mathsf{KeyGen}(1^\lambda) \to (\mathsf{vk}, \mathsf{sk})$: On input the security parameter λ, the key-generation algorithm returns a verification key vk, and a signing key sk.

- $\mathsf{Sign}(\mathsf{pk}_i, \mathsf{sk}, x_i) \to \sigma_i$: On input a public key pk_i, a signing key sk, and a message $x_i \in \mathcal{X}$, the signing algorithm returns a signature σ_i.

  Vector variant: For $\mathsf{pp} = (\mathsf{pk}_1, \ldots, \mathsf{pk}_\ell)$, and $\mathbf{x} = (x_1, \ldots, x_\ell) \in \mathcal{X}^\ell$, we write $\mathsf{Sign}(\mathsf{pp}, \mathsf{sk}, \mathbf{x})$ to denote component-wise signing of each message. Namely, $\mathsf{Sign}(\mathsf{pp}, \mathsf{sk}, \mathbf{x})$ outputs signatures $\boldsymbol{\sigma} = (\sigma_1, \ldots, \sigma_\ell)$ where $\sigma_i \leftarrow \mathsf{Sign}(\mathsf{pk}_i, \mathsf{sk}, x_i)$ for all $i \in [\ell]$.

- $\mathsf{PrmsEval}(C, \mathsf{pp}') \to \mathsf{pk}_C$: On input a function $C \colon \mathcal{X}^\ell \to \mathcal{X}$ and a collection of public keys $\mathsf{pp}' = (\mathsf{pk}'_1, \ldots, \mathsf{pk}'_\ell)$, the parameter-evaluation algorithm returns an evaluated public key pk_C.

  Vector variant: For a circuit $C \colon \mathcal{X}^\ell \to \mathcal{X}^k$, we write $\mathsf{PrmsEval}(C, \mathsf{pp}')$ to denote component-wise parameter evaluation. Namely, let $C_1, \ldots, C_k$ be functions such that $C(x_1, \ldots, x_\ell) = \big(C_1(x_1, \ldots, x_\ell), \ldots, C_k(x_1, \ldots, x_\ell)\big)$. Then, $\mathsf{PrmsEval}(C, \mathsf{pp}')$ evaluates $\mathsf{pk}_{C_i} \leftarrow \mathsf{PrmsEval}(C_i, \mathsf{pp}')$ for $i \in [k]$, and outputs $\mathsf{pk}_C = (\mathsf{pk}_{C_1}, \ldots, \mathsf{pk}_{C_k})$.

- $\mathsf{SigEval}(C, \mathsf{pp}', \mathbf{x}, \boldsymbol{\sigma}) \to \sigma$: On input a function $C \colon \mathcal{X}^\ell \to \mathcal{X}$, public keys $\mathsf{pp}' = (\mathsf{pk}'_1, \ldots, \mathsf{pk}'_\ell)$, messages $\mathbf{x} \in \mathcal{X}^\ell$, and signatures $\boldsymbol{\sigma} = (\sigma_1, \ldots, \sigma_\ell)$, the signature-evaluation algorithm returns an evaluated signature σ.

  Vector variant: We can define a vector variant of $\mathsf{SigEval}$ analogously to that of $\mathsf{PrmsEval}$.

- $\mathsf{Hide}(\mathsf{vk}, x, \sigma) \to \sigma^*$: On input a verification key vk, a message $x \in \mathcal{X}$, and a signature σ, the hide algorithm returns a signature σ^*.

  Vector variant: For $\mathbf{x} = (x_1, \ldots, x_k)$ and $\boldsymbol{\sigma} = (\sigma_1, \ldots, \sigma_k)$, we write $\mathsf{Hide}(\mathsf{vk}, \mathbf{x}, \boldsymbol{\sigma})$ to denote component-wise evaluation of the hide algorithm. Namely, $\mathsf{Hide}(\mathsf{vk}, \mathbf{x}, \boldsymbol{\sigma})$ returns $(\sigma^*_1, \ldots, \sigma^*_k)$ where $\sigma^*_i \leftarrow \mathsf{Hide}(\mathsf{vk}, x_i, \sigma_i)$ for all $i \in [k]$.

- $\mathsf{Verify}(\mathsf{pk}, \mathsf{vk}, x, \sigma) \to \{0, 1\}$: On input a public key pk, a verification key vk, a message $x \in \mathcal{X}$, and a signature σ, the verification algorithm either accepts (returns 1) or rejects (returns 0).

  Vector variant: For a collection of public keys $\mathsf{pp}' = (\mathsf{pk}'_1, \ldots, \mathsf{pk}'_k)$, messages $\mathbf{x} = (x_1, \ldots, x_k)$, and signatures $\boldsymbol{\sigma} = (\sigma_1, \ldots, \sigma_k)$, we write $\mathsf{Verify}(\mathsf{pp}', \mathsf{vk}, \mathbf{x}, \boldsymbol{\sigma})$ to denote applying the verification algorithm to each signature component-wise. In other words, $\mathsf{Verify}(\mathsf{pp}', \mathsf{vk}, \mathbf{x}, \boldsymbol{\sigma})$ accepts if and only if $\mathsf{Verify}(\mathsf{pk}'_i, \mathsf{vk}, x_i, \sigma_i)$ accepts for all $i \in [k]$.

- $\mathsf{VerifyFresh}(\mathsf{pk}, \mathsf{vk}, x, \sigma) \to \{0, 1\}$: On input a public key pk, a verification key vk, a message $x \in \mathcal{X}$, and a signature σ, the fresh verification algorithm either accepts (returns 1) or rejects (returns 0).

  Vector variant: We can define a vector variant of $\mathsf{VerifyFresh}$ analogously to that of Verify.

- $\mathsf{VerifyHide}(\mathsf{pk}, \mathsf{vk}, x, \sigma^*) \to \{0, 1\}$: On input a public key pk, a verification key vk, a message $x \in \mathcal{X}$, and a signature σ^*, the hide verification algorithm either accepts (returns 1) or rejects (returns 0).

  Vector variant: We can define a vector variant of $\mathsf{VerifyHide}$ analogously to that of Verify.

Correctness. We now state the correctness requirements for a homomorphic signature scheme. Our definitions are adapted from the corresponding ones in [GVW15b]. Our homomorphic signature syntax has three different verification algorithms. The standard verification algorithm Verify can be used to verify fresh signatures (output by Sign) as well as homomorphically-evaluated signatures (output by $\mathsf{SigEval}$). The hide verification algorithm $\mathsf{VerifyHide}$ is used for verifying signatures output by the context-hiding transformation Hide, which may be structurally different from the signatures output by Sign or $\mathsf{SigEval}$. Finally, we have a special verification algorithm $\mathsf{VerifyFresh}$ that can be used to verify signatures output by Sign (before any homomorphic evaluation has taken place). While Verify subsumes $\mathsf{VerifyFresh}$, having a separate $\mathsf{VerifyFresh}$ algorithm is useful for formulating a strong version of evaluation correctness. We now state our correctness definitions. First, we have the standard correctness requirement of any signature scheme. Specifically, signatures output by the honest signing algorithm should verify according to both Verify and $\mathsf{VerifyFresh}$.

Definition 3.2 (Signing Correctness)**.** A homomorphic signature scheme $\Pi_{\mathsf{HS}} = (\mathsf{PrmsGen}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{PrmsEval}, \mathsf{SigEval}, \mathsf{Hide}, \mathsf{Verify}, \mathsf{VerifyFresh}, \mathsf{VerifyHide})$ with message space $\mathcal{X}$, message length ℓ, and function class $\mathcal{C}$ satisfies *signing correctness* if for all $\lambda \in \mathbb{N}$, messages $\mathbf{x} \in \mathcal{X}^\ell$, and setting $\mathsf{pp} \leftarrow \mathsf{PrmsGen}(1^\lambda, 1^\ell)$, $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$, $\boldsymbol{\sigma} \leftarrow \mathsf{Sign}(\mathsf{pp}, \mathsf{sk}, \mathbf{x})$, we have

$$\Pr[\mathsf{Verify}(\mathsf{pp}, \mathsf{vk}, \mathbf{x}, \boldsymbol{\sigma}) = 1] = 1 \quad \text{and} \quad \Pr[\mathsf{VerifyFresh}(\mathsf{pp}, \mathsf{vk}, \mathbf{x}, \boldsymbol{\sigma}) = 1] = 1.$$

Evaluation correctness. Next, we require that if one applies the honest signature-evaluation algorithm $\mathsf{SigEval}$ to valid *fresh* signatures (namely, signatures that are accepted by $\mathsf{VerifyFresh}$), then the resulting signature verifies according to Verify (with respect to the corresponding evaluated public key). This is a *stronger* definition than the usual notion of evaluation correctness from [GVW15b], which only requires correctness to holds when $\mathsf{SigEval}$ is applied to signatures output by the *honest* signing algorithm Sign. In our definition, correctness must hold against *all* signatures deemed valid by $\mathsf{VerifyFresh}$ (with respect to an arbitrary public key pk and verification key vk), which may be a larger set of signatures than those that could be output by Sign. This notion of correctness will be useful in our construction of (malicious-secure) blind homomorphic signatures in Section 3.4.

Definition 3.3 (Evaluation Correctness)**.** A homomorphic signature scheme $\Pi_{\mathsf{HS}} = (\mathsf{PrmsGen}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{PrmsEval}, \mathsf{SigEval}, \mathsf{Hide}, \mathsf{Verify}, \mathsf{VerifyFresh}, \mathsf{VerifyHide})$ with message space $\mathcal{X}$, message length ℓ, and function class $\mathcal{C} = \{\mathcal{C}_\lambda\}_{\lambda \in \mathbb{N}}$ (where each $\mathcal{C}_\lambda$ is a collection of functions from $\mathcal{X}^\ell$ to $\mathcal{X}$) satisfies *evaluation correctness* if for all $\lambda \in \mathbb{N}$, all public keys pp, all verification keys vk, and all messages $\mathbf{x} \in \mathcal{X}^\ell$, the following properties hold:

- **Single-Hop Correctness:** For all $C \in \mathcal{C}_\lambda$ and all signatures $\boldsymbol{\sigma}$ where $\mathsf{VerifyFresh}(\mathsf{pp}, \mathsf{vk}, \mathbf{x}, \boldsymbol{\sigma}) = 1$, if we set $\mathsf{pk}_C \leftarrow \mathsf{PrmsEval}(C, \mathsf{pp})$ and $\sigma \leftarrow \mathsf{SigEval}(C, \mathsf{pp}, \mathbf{x}, \boldsymbol{\sigma})$, then

$$\Pr[\mathsf{Verify}(\mathsf{pk}_C, \mathsf{vk}, C(\mathbf{x}), \sigma) = 1] = 1.$$

- **Multi-Hop Correctness:** For any collection of functions $C_1, \ldots, C_\ell \in \mathcal{C}_\lambda$ and $C' \colon \mathcal{X}^\ell \to \mathcal{X}$, define the composition $(C' \circ \vec{C}) \colon \mathcal{X}^\ell \to \mathcal{X}$ to be the mapping $\mathbf{x} \mapsto C'(C_1(\mathbf{x}), \ldots, C_\ell(\mathbf{x}))$. If

$(C' \circ \vec{C}) \in \mathcal{C}_\lambda$, then for any set of signatures $\boldsymbol{\sigma} = (\sigma_1, \ldots, \sigma_\ell)$ where $\mathsf{Verify}(\mathsf{pk}_{C_i}, \mathsf{vk}, x_i, \sigma_i) = 1$ and $\mathsf{pk}_{C_i} \leftarrow \mathsf{PrmsEval}(C_i, \mathsf{pp})$ for all $i \in [\ell]$, we have that

$$\Pr[\mathsf{Verify}(\mathsf{PrmsEval}(C', (\mathsf{pk}_{C_1}, \ldots, \mathsf{pk}_{C_\ell})), \mathsf{vk}, \mathbf{x}, \mathsf{SigEval}(C', (\mathsf{pk}_{C_1}, \ldots, \mathsf{pk}_{C_\ell}), \mathbf{x}, \boldsymbol{\sigma})) = 1] = 1.$$

Hiding correctness. Finally, we require that the hide algorithm also produces valid signatures. Similar to the case of evaluation correctness, we require that correctness holds whenever Hide is applied to any valid signature accepted by Verify (which need not coincide with the set of signatures output by an honest execution of Sign or $\mathsf{SigEval}$). This is essentially the definition in [GVW15b].

Definition 3.4 (Hiding Correctness)**.** A homomorphic signature scheme $\Pi_{\mathsf{HS}} = (\mathsf{PrmsGen}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{PrmsEval}, \mathsf{SigEval}, \mathsf{Hide}, \mathsf{Verify}, \mathsf{VerifyFresh}, \mathsf{VerifyHide})$ with message space $\mathcal{X}$, message length ℓ, and function class $\mathcal{C}$ satisfies *hiding correctness* if for all $\lambda \in \mathbb{N}$, all verification keys vk, messages $x \in \mathcal{X}$, and all signatures σ where $\mathsf{Verify}(\mathsf{pk}, \mathsf{vk}, x, \sigma) = 1$, we have that

$$\Pr[\mathsf{VerifyHide}\big(\mathsf{pk}, \mathsf{vk}, x, \mathsf{Hide}(\mathsf{vk}, x, \sigma)\big) = 1] = 1.$$

Unforgeability. We now formally define unforgeability for a homomorphic signature scheme. Intuitively, a homomorphic signature scheme is unforgeable if no efficient adversary who only possesses signatures $\sigma_1, \ldots, \sigma_\ell$ on messages $x_1, \ldots, x_\ell$ can produce a signature σ_y that is valid with respect to a function C where $y \neq C(x_1, \ldots, x_\ell)$.

Definition 3.5 (Unforgeability)**.** Fix a security parameter λ. Let $\Pi_{\mathsf{HS}} = (\mathsf{PrmsGen}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{PrmsEval}, \mathsf{SigEval}, \mathsf{Hide}, \mathsf{Verify}, \mathsf{VerifyFresh}, \mathsf{VerifyHide})$ be a homomorphic signature scheme with message space $\mathcal{X}$, message length ℓ, and function class $\mathcal{C} = \{\mathcal{C}_\lambda\}_{\lambda \in \mathbb{N}}$, where each $\mathcal{C}_\lambda$ is a collection of functions from $\mathcal{X}^\ell$ to $\mathcal{X}$. Then, for an adversary $\mathcal{A}$, we define the unforgeability security experiment $\mathsf{Expt}^{\mathsf{uf}}_{\mathcal{A}, \Pi_{\mathsf{HS}}}(\lambda, \ell)$ as follows:

1. The challenger begins by generating public keys $\mathsf{pp} \leftarrow \mathsf{PrmsGen}(1^\lambda, 1^\ell)$, and a signing-verification key $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$. It gives pp and vk to $\mathcal{A}$.
2. The adversary $\mathcal{A}$ submits a message $\mathbf{x} \in \mathcal{X}^\ell$ to be signed.
3. The challenger signs the messages $\boldsymbol{\sigma} \leftarrow \mathsf{Sign}(\mathsf{pp}, \mathsf{sk}, \mathbf{x})$ and sends the signatures $\boldsymbol{\sigma}$ to $\mathcal{A}$.
4. The adversary $\mathcal{A}$ outputs a circuit $C \in \mathcal{C}_\lambda$, a message $y \in \mathcal{X}$, and a signature σ_y.
5. The output of the experiment is 1 if $C \in \mathcal{C}_\lambda$, $y \neq C(\mathbf{x})$, and $\mathsf{VerifyHide}(\mathsf{pk}_C, \mathsf{vk}, y, \sigma_y) = 1$, where $\mathsf{pk}_C \leftarrow \mathsf{PrmsEval}(C, \mathsf{pp})$. Otherwise, the output of the experiment is 0.

We say that a homomorphic signature scheme Π_{HS} satisfies unforgeability if for all efficient adversaries $\mathcal{A}$,

$$\Pr[\mathsf{Expt}^{\mathsf{uf}}_{\mathcal{A}, \Pi_{\mathsf{HS}}}(\lambda, \ell) = 1] = \operatorname{negl}(\lambda).$$

Remark 3.6 (Selective Unforgeability)**.** We can also define a weaker notion of unforgeability called *selective unforgeability* where the adversary commits to the messages $\mathbf{x} \in \mathcal{X}^\ell$ at the start of the experiment *before* it sees the public keys pp and the verification key vk. In Section 3.2.1, we describe a simplified variant of the [GVW15b] construction that satisfies this weaker notion of selective unforgeability. In Section 3.2.2, we give the full construction from [GVW15b] that satisfies the definition of adaptive unforgeability from Definition 3.5.

Context-hiding. The second security requirement on a homomorphic signature scheme is *context-hiding*, which roughly says that if a user evaluates a function C on a message-signature pair $(\mathbf{x}, \boldsymbol{\sigma})$ to obtain a signature $\sigma_{C(\mathbf{x})}$, and then runs the hide algorithm on $\sigma_{C(\mathbf{x})}$, the resulting signature $\sigma^*_{C(\mathbf{x})}$ does not contain any information about $\mathbf{x}$ other than what is revealed by C and $C(\mathbf{x})$. Previous works such as [GVW15b] captured this notion by requiring that there exists an efficient simulator that can simulate the signature $\sigma^*_{C(\mathbf{x})}$ given just the signing key sk,[2] the function C, and the value $C(\mathbf{x})$. Notably, the simulator does not see the original message $\mathbf{x}$ or the signature $\sigma_{C(\mathbf{x})}$

While this is a very natural notion of context-hiding, it can be difficult to satisfy. The homomorphic signature candidate by Gorbunov et al. [GVW15b] satisfies *selective unforgeability* (Remark 3.6) and context-hiding. Gorbunov et al. also give a variant of their construction that achieves adaptive unforgeability; however, this scheme does *not* simultaneously satisfy the notion of context-hiding. Nonetheless, the adaptively-secure scheme from [GVW15b] can be shown to satisfy a weaker notion of context-hiding that suffices for all of our applications (and still captures all of the intuitive properties we expect from context-hiding). Specifically, in our weaker notion of context-hiding, we allow the simulator to also take in some components of the original signatures $\sigma_{C(\mathbf{x})}$, provided that those components are *independent* of the value that is signed.[3]

To formalize this notion, we first define the notion of a *decomposable* homomorphic signature scheme. In a decomposable homomorphic signature scheme, any valid signature $\sigma = (\sigma^{\mathsf{pk}}, \sigma^{\mathsf{m}})$ can be decomposed into a message-independent component σ^{pk} that contains no information about the signed message, and a message-dependent component σ^{m}.

Definition 3.7 (Decomposable Homomorphic Signatures)**.** Let $\Pi_{\mathsf{HS}} = (\mathsf{PrmsGen}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{PrmsEval}, \mathsf{SigEval}, \mathsf{Hide}, \mathsf{Verify}, \mathsf{VerifyFresh}, \mathsf{VerifyHide})$ be a homomorphic signature scheme with message space $\mathcal{X}$, message length ℓ, and function class $\mathcal{C} = \{\mathcal{C}_\lambda\}_{\lambda \in \mathbb{N}}$. We say that Π_{HS} is *decomposable* if the signing and evaluation algorithms can be decomposed into a message-independent and a message-dependent algorithm as follows:

- The signing algorithm Sign splits into a pair of algorithms $(\mathsf{SignPK}, \mathsf{SignM})$:

[2]Note that the simulator must take in some secret value (not known to the evaluator). Otherwise, the existence of such a simulator breaks unforgeability of the signature scheme.

[3]The construction in Section 3.2.2 combines the homomorphic signature scheme that satisfies (full) unforgeability but not context-hiding and the selectively unforgeable homomorphic signature scheme that satisfies context-hiding in [GVW15b].

  - $\mathsf{SignPK}(\mathsf{pk}_i, \mathsf{sk}) \to \sigma_i^{\mathsf{pk}}$: On input a public key pk_i and a signing key sk, the SignPK algorithm outputs a message-independent component σ_i^{pk}.
  - $\mathsf{SignM}(\mathsf{pk}_i, \mathsf{sk}, x_i, \sigma_i^{\mathsf{pk}}) \to \sigma_i^{\mathsf{m}}$: On input a public key pk_i, a signing key sk, a message $x_i \in \mathcal{X}$, and a message-independent component σ_i^{pk}, the SignM algorithm outputs a message-dependent component σ_i^{m}.

  The actual signing algorithm $\mathsf{Sign}(\mathsf{pk}_i, \mathsf{sk}, x_i)$ then computes $\sigma_i^{\mathsf{pk}} \leftarrow \mathsf{SignPK}(\mathsf{pk}_i, \mathsf{sk})$ and $\sigma_i^{\mathsf{m}} \leftarrow \mathsf{SignM}(\mathsf{pk}_i, \mathsf{sk}, x_i, \sigma_i^{\mathsf{pk}})$. The final signature is the pair $\sigma_i = (\sigma_i^{\mathsf{pk}}, \sigma_i^{\mathsf{m}})$.

- The evaluation algorithm $\mathsf{SigEval}$ splits into a pair of algorithms: $(\mathsf{SigEvalPK}, \mathsf{SigEvalM})$:
  - $\mathsf{SigEvalPK}(C, \mathsf{pp}', \boldsymbol{\sigma}^{\mathsf{pk}}) \to \sigma^{\mathsf{pk}}$: On input a circuit $C \in \mathcal{C}_\lambda$, public keys $\mathsf{pp}' = (\mathsf{pk}_1', \dots, \mathsf{pk}_\ell')$, and message-independent signature components $\boldsymbol{\sigma}^{\mathsf{pk}} = (\sigma_1^{\mathsf{pk}}, \dots, \sigma_\ell^{\mathsf{pk}})$, the $\mathsf{SigEvalPK}$ algorithm outputs a message-independent component σ^{pk}.
  - $\mathsf{SigEvalM}(C, \mathsf{pp}', \mathbf{x}, \boldsymbol{\sigma}) \to \sigma^{\mathsf{m}}$: On input a circuit $C \in \mathcal{C}_\lambda$, public keys $\mathsf{pp}' = (\mathsf{pk}_1', \dots, \mathsf{pk}_\ell')$, messages $\mathbf{x} \in \mathcal{X}^\ell$, and signatures $\boldsymbol{\sigma}$, the $\mathsf{SigEvalM}$ algorithm outputs a message-dependent component σ^{m}.

  The signature evaluation algorithm $\mathsf{SigEval}(C, \mathsf{pp}', \mathbf{x}, \boldsymbol{\sigma})$ first parses $\boldsymbol{\sigma} = (\boldsymbol{\sigma}^{\mathsf{pk}}, \boldsymbol{\sigma}^{\mathsf{m}})$, computes $\sigma^{\mathsf{pk}} \leftarrow \mathsf{SigEvalPK}(C, \mathsf{pp}', \boldsymbol{\sigma}^{\mathsf{pk}})$, $\sigma^{\mathsf{m}} \leftarrow \mathsf{SigEvalM}(C, \mathsf{pp}', \mathbf{x}, \boldsymbol{\sigma})$, and returns $\sigma = (\sigma^{\mathsf{pk}}, \sigma^{\mathsf{m}})$.

To formalize context-hiding, we require that there exists a simulator that can simulate the output of the hide algorithm given only the secret signing key sk, the function C, the output $C(\mathbf{x})$, and the message-independent component of the signature $\sigma_{C(\mathbf{x})}^{\mathsf{pk}}$. We give the formal definition below:

Definition 3.8 (Context-Hiding Against Honest Signers)**.** Fix a security parameter λ. Let $\Pi_{\mathsf{HS}} = (\mathsf{PrmsGen}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{PrmsEval}, \mathsf{SigEval}, \mathsf{Hide}, \mathsf{Verify}, \mathsf{VerifyFresh}, \mathsf{VerifyHide})$ be a decomposable homomorphic signature scheme (Definition 3.7) with message space $\mathcal{X}$, message length ℓ, and function class $\mathcal{C} = \{\mathcal{C}_\lambda\}_{\lambda \in \mathbb{N}}$, where each $\mathcal{C}_\lambda$ is a collection of functions from $\mathcal{X}^\ell$ to $\mathcal{X}$. For a bit $b \in \{0, 1\}$, a simulator $\mathcal{S}$ and an adversary $\mathcal{A}$, we define the *weak context-hiding* security experiment against an honest signer $\mathsf{Expt}_{\mathcal{A},\mathcal{S},\Pi_{\mathsf{HS}}}^{\mathsf{ch\text{-}honest}}(\lambda, b)$ as follows:

1. The challenger begins by generating a signing and verification key $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$ and sends $(\mathsf{vk}, \mathsf{sk})$ to $\mathcal{A}$.
2. The adversary $\mathcal{A}$ can then submit (adaptive) queries to the challenger where each query consists of a public key pk, a message $x \in \mathcal{X}$, and a signature $\sigma = (\sigma^{\mathsf{pk}}, \sigma^{\mathsf{m}})$. On each query, the challenger first checks that $\mathsf{Verify}(\mathsf{pk}, \mathsf{vk}, x, \sigma) = 1$. If this is not the case, then the challenger ignores the query and replies with $\perp$. Otherwise, the challenger proceeds as follows:
   - If $b = 0$, the challenger evaluates $\sigma^* \leftarrow \mathsf{Hide}(\mathsf{vk}, x, \sigma)$, and sends σ^* to $\mathcal{A}$.
   - If $b = 1$, the challenger computes $\sigma^* \leftarrow \mathcal{S}(\mathsf{pk}, \mathsf{vk}, \mathsf{sk}, x, \sigma^{\mathsf{pk}})$. It sends σ^* to $\mathcal{A}$.

3. Finally, $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$, which is also the output of the experiment.

We say that a homomorphic signature scheme Π_{HS} satisfies *statistical context-hiding against an honest signer* if there exists an efficient simulator $\mathcal{S}$ such that for all (computationally-unbounded) adversaries $\mathcal{A}$,

$$\left| \Pr[\mathsf{Expt}^{\mathsf{ch\text{-}honest}}_{\mathcal{A},\mathcal{S},\Pi_{\mathsf{HS}}}(\lambda,0) = 1] - \Pr[\mathsf{Expt}^{\mathsf{ch\text{-}honest}}_{\mathcal{A},\mathcal{S},\Pi_{\mathsf{HS}}}(\lambda,1) = 1] \right| = \mathrm{negl}(\lambda).$$

Context-hiding against malicious signers. Typically, context-hiding is defined with respect to an *honest* signer that generates the signing and verification keys using the honest key-generation algorithm KeyGen. However, when constructing *blind homomorphic signatures* (Section 3.4) with security against *malicious* signers, the assumption that the keys are correctly generated no longer makes sense. Hence, we need a stronger security property that context-hiding holds even if the signing and verification keys for the homomorphic signature scheme are maliciously constructed.

Definition 3.8 does not satisfy this stronger notion of context-hiding because the challenger samples $(\mathsf{vk}, \mathsf{sk})$ using the honest KeyGen algorithm, and the simulator is provided the (honestly-generated) signing key sk. The natural way to extend Definition 3.8 to achieve security against malicious signers is to allow the adversary to choose the verification key vk and signing key sk. However, this is too restrictive because the adversary could potentially cook up signatures that verify under vk, and yet, there is no natural notion of a signing key. To circumvent this issue, we introduce a stronger notion of context-hiding that holds against any party with the *capability* to sign messages. More concretely, we require the existence of a simulator that can extract a *simulation trapdoor* td from any *admissible* set of valid message-signature pairs. This trapdoor information td replaces the signing key sk as input to the simulator.

In our construction of homomorphic signatures (Construction 3.11), we say that a pair of messages $(\tilde{x}_0, \tilde{\sigma}_0)$ and $(\tilde{x}_1, \tilde{\sigma}_1)$ is admissible if $\tilde{x}_0 \neq \tilde{x}_1$ and $\tilde{\sigma}_0$ and $\tilde{\sigma}_1$ are valid signatures of $\tilde{x}_0$ and $\tilde{x}_1$, respectively (with the *same* public components). Intuitively, our definition captures the fact that context-hiding holds against any signer, as long as they are able to produce or *forge* valid signatures on distinct messages $\tilde{x}_0$ and $\tilde{x}_1$ under some verification key vk. Note that this definition subsumes Definition 3.8, since any signer with an honestly-generated signing key can sign arbitrary messages of its choosing.

Definition 3.9 (Context-Hiding)**.** Fix a security parameter λ. Let $\Pi_{\mathsf{HS}} = (\mathsf{PrmsGen}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{PrmsEval}, \mathsf{SigEval}, \mathsf{Hide}, \mathsf{Verify}, \mathsf{VerifyFresh}, \mathsf{VerifyHide})$ be a decomposable homomorphic signature scheme (Definition 3.7) with message space $\mathcal{X}$, message length ℓ, and function class $\mathcal{C} = \{\mathcal{C}_\lambda\}_{\lambda \in \mathbb{N}}$, where each $\mathcal{C}_\lambda$ is a collection of functions from $\mathcal{X}^\ell$ to $\mathcal{X}$. For a bit $b \in \{0,1\}$, a simulator $\mathcal{S} = (\mathcal{S}^{\mathsf{Ext}}, \mathcal{S}^{\mathsf{Gen}})$, and an adversary $\mathcal{A}$, we define the *context-hiding* security experiment $\mathsf{Expt}^{\mathsf{ch}}_{\mathcal{A},\mathcal{S},\Pi_{\mathsf{HS}}}(\lambda, b)$ as follows:

1. At the start of the experiment, $\mathcal{A}$ submits a public key pk, a verification key vk, and two message-signature pairs $(\tilde{x}_0, \tilde{\sigma}_0)$, $(\tilde{x}_1, \tilde{\sigma}_1)$ where $\tilde{x}_0, \tilde{x}_1 \in \mathcal{X}$ and $\tilde{x}_0 \neq \tilde{x}_1$ to the challenger.

2. The challenger parses the signatures as $\tilde{\sigma}_0 = (\tilde{\sigma}_0^{\mathsf{pk}}, \tilde{\sigma}_0^{\mathsf{m}})$ and $\tilde{\sigma}_1 = (\tilde{\sigma}_1^{\mathsf{pk}}, \tilde{\sigma}_1^{\mathsf{m}})$, and checks that $\tilde{x}_0 \neq \tilde{x}_1$, $\tilde{\sigma}_0^{\mathsf{pk}} = \tilde{\sigma}_1^{\mathsf{pk}}$, and that $\mathsf{Verify}(\mathsf{pk}, \mathsf{vk}, \tilde{x}_0, \tilde{\sigma}_0) = 1 = \mathsf{Verify}(\mathsf{pk}, \mathsf{vk}, \tilde{x}_1, \tilde{\sigma}_1)$. If this is not the case, then the experiment halts with output 0. Otherwise, the challenger invokes the simulator $\mathsf{td} \leftarrow \mathcal{S}^{\mathsf{Ext}}(\mathsf{pk}, \mathsf{vk}, (\tilde{x}_0, \tilde{\sigma}_0), (\tilde{x}_1, \tilde{\sigma}_1))$.

3. The adversary $\mathcal{A}$ can then submit (adaptive) queries to the challenger where each query consists of a public key pk', a message $x \in \mathcal{X}$, and a signature $\sigma = (\sigma^{\mathsf{pk}}, \sigma^{\mathsf{m}})$. For each query, the challenger checks that $\mathsf{Verify}(\mathsf{pk}', \mathsf{vk}, x, \sigma) = 1$. If this is not the case, then the challenger ignores the query and replies with $\perp$. Otherwise, it proceeds as follows:

   - If $b = 0$, the challenger evaluates $\sigma^* \leftarrow \mathsf{Hide}\big(\mathsf{vk}, x, (\sigma^{\mathsf{pk}}, \sigma^{\mathsf{m}})\big)$, and sends σ^* to $\mathcal{A}$.
   - If $b = 1$, the challenger computes $\sigma^* \leftarrow \mathcal{S}^{\mathsf{Gen}}(\mathsf{pk}', \mathsf{vk}, \mathsf{td}, x, \sigma^{\mathsf{pk}})$. It provides σ^* to $\mathcal{A}$.

4. Finally, $\mathcal{A}$ outputs a bit $b' \in \{0, 1\}$, which is also the output of the experiment.

We say that a homomorphic signature scheme Π_{HS} satisfies *statistical context-hiding* if there exists an efficient simulator $\mathcal{S} = (\mathcal{S}^{\mathsf{Ext}}, \mathcal{S}^{\mathsf{Gen}})$ such that for all (computationally-unbounded) adversaries $\mathcal{A}$,

$$\big| \Pr[\mathsf{Expt}^{\mathsf{ch}}_{\mathcal{A},\mathcal{S},\Pi_{\mathsf{HS}}}(\lambda, 0) = 1] - \Pr[\mathsf{Expt}^{\mathsf{ch}}_{\mathcal{A},\mathcal{S},\Pi_{\mathsf{HS}}}(\lambda, 1) = 1] \big| = \text{negl}(\lambda).$$

Compactness. The final property that we require from a homomorphic signature scheme is compactness. Roughly speaking, compactness requires that given a message-signature pair $(\mathbf{x}, \boldsymbol{\sigma})$, the size of the signature obtained from homomorphically evaluating a function C on $\boldsymbol{\sigma}$ depends only on the size of the output message $|C(\mathbf{x})|$ (and the security parameter) and is *independent* of the size of the original message $|\mathbf{x}|$.

Definition 3.10 (Compactness)**.** Fix a security parameter λ. Let $\Pi_{\mathsf{HS}} = (\mathsf{PrmsGen}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{PrmsEval}, \mathsf{SigEval}, \mathsf{Hide}, \mathsf{Verify}, \mathsf{VerifyFresh}, \mathsf{VerifyHide})$ be a homomorphic signature scheme with message space $\mathcal{X}$, message length ℓ, and function class $\mathcal{C} = \{\mathcal{C}_\lambda\}_{\lambda \in \mathbb{N}}$, where each $\mathcal{C}_\lambda$ is a collection of Boolean circuits from $\mathcal{X}^\ell$ to $\mathcal{X}$ of depth at most $d = d(\lambda)$. We say that Π_{HS} is *compact* if there exists a universal polynomial poly$(\cdot)$ such that for all $\lambda \in \mathbb{N}$, messages $\mathbf{x} \in \mathcal{X}^\ell$, and functions $C \in \mathcal{C}_\lambda$, and setting $\mathsf{pp} \leftarrow \mathsf{PrmsGen}(1^\lambda, 1^\ell)$, $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$, $\boldsymbol{\sigma} \leftarrow \mathsf{Sign}(\mathsf{pp}, \mathsf{sk}, \mathbf{x})$, and $\sigma \leftarrow \mathsf{SigEval}(C, \mathsf{pp}, \mathbf{x}, \boldsymbol{\sigma})$, we have that $|\sigma| \leq \text{poly}(\lambda, d)$. In particular, the size of the evaluated signature $|\sigma|$ depends only on the depth of the circuit C, and *not* on the message length ℓ.

3.2.1 Selectively-Secure Homomorphic Signatures

In this section, we show that the [GVW15b] homomorphic signature construction is decomposable in the sense of Definition 3.7 and in addition, satisfies our stronger notion of context-hiding (Definition 3.9). We start with a description of a simpler variant of the [GVW15b] construction that satisfies *selective unforgeability* (Remark 3.6), and show that it satisfies context-hiding (against malicious signers). Although it is possible to directly construct a homomorphic signature scheme that satisfies adaptive security, the simpler variant better demonstrates the main ideas of the construction. In Section 3.2.2, we show how to generically modify Construction 3.11 and show that it satisfies both adaptive unforgeability and strong context-hiding. (Corollary 3.22).

Construction 3.11 (Selectively-Secure Homomorphic Signature [GVW15b, adapted])**.** Fix a security parameter λ and a message length $\ell = \text{poly}(\lambda)$. Let $\mathcal{C} = \{\mathcal{C}_\lambda\}_{\lambda \in \mathbb{N}}$ be a function class where each $\mathcal{C}_\lambda$ is a collection of Boolean circuits of depth at most $d = d(\lambda)$ from $\{0,1\}^\ell$ to $\{0,1\}$. In our description, we use lattice trapdoors and the GSW homomorphic operations described in Section 2.1. For lattice parameters n, m, q, and norm bounds β_{ini}, β_{eval}, β_{hide} we construct a decomposable homomorphic signature scheme $\Pi_{\mathsf{HS}} = (\mathsf{PrmsGen}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{PrmsEval}, \mathsf{SigEval}, \mathsf{Hide}, \mathsf{Verify}, \mathsf{VerifyFresh}, \mathsf{VerifyHide})$ with message space $\mathcal{X} = \{0,1\}$, message length ℓ, and function class $\mathcal{C}$ as follows:

- $\mathsf{PrmsGen}(1^\lambda, 1^\ell) \to \mathsf{pp}$: On input the security parameter λ and the message length ℓ, the parameter-generation algorithm samples matrices $\mathbf{V}_1, \ldots, \mathbf{V}_\ell \xleftarrow{\text{R}} \mathbb{Z}_q^{n \times m}$. It sets $\mathsf{pk}_i = \mathbf{V}_i$ for $i \in [\ell]$ and returns the public keys $\mathsf{pp} = (\mathsf{pk}_1, \ldots, \mathsf{pk}_\ell)$.

- $\mathsf{KeyGen}(1^\lambda) \to (\mathsf{vk}, \mathsf{sk})$: On input the security parameter λ, the key-generation algorithm samples a lattice trapdoor $(\mathbf{A}, \mathsf{td}) \leftarrow \mathsf{TrapGen}(1^\lambda)$. It sets $\mathsf{vk} = \mathbf{A}$ and $\mathsf{sk} = (\mathbf{A}, \mathsf{td})$.

- $\mathsf{Sign}(\mathsf{pk}_i, \mathsf{sk}, x_i) \to \sigma_i$: The signing algorithm computes $\sigma_i^{\mathsf{pk}} \leftarrow \mathsf{SignPK}(\mathsf{pk}_i, \mathsf{sk})$ and $\sigma_i^{\mathsf{m}} \leftarrow \mathsf{SignM}(\mathsf{pk}_i, \mathsf{sk}, x_i, \sigma_i^{\mathsf{pk}})$ where the algorithms SignPK and SignM are defined as follows:

  - $\mathsf{SignPK}(\mathsf{pk}_i, \mathsf{sk}) \to \sigma_i^{\mathsf{pk}}$: The SignPK algorithm outputs the empty string $\sigma_i^{\mathsf{pk}} = \varepsilon$.
  - $\mathsf{SignM}(\mathsf{pk}_i, \mathsf{sk}, x_i, \sigma_i^{\mathsf{pk}}) \to \sigma_i^{\mathsf{m}}$: On input a public key $\mathsf{pk}_i = \mathbf{V}_i$, a signing key $\mathsf{sk} = (\mathbf{A}, \mathsf{td})$, a message $x \in \{0,1\}$, and the public signature component σ_i^{pk}, the SignM algorithm samples a preimage $\mathbf{U}_i \leftarrow \mathsf{SamplePre}(\mathbf{A}, \mathbf{V}_i - x_i \cdot \mathbf{G}, \mathsf{td})$ and outputs $\sigma_i^{\mathsf{m}} = \mathbf{U}_i$.

  Finally, the signing algorithm outputs the signature $\sigma_i = (\sigma_i^{\mathsf{pk}}, \sigma_i^{\mathsf{m}})$.

- $\mathsf{PrmsEval}(C, \mathsf{pp}')) \to \mathsf{pk}_C$: On input a Boolean circuit $C\colon \{0,1\}^\ell \to \{0,1\}$ and a collection of public keys $\mathsf{pp}' = (\mathsf{pk}'_1, \ldots, \mathsf{pk}'_i)$ where $\mathsf{pk}'_i = \mathbf{V}'_i$ for $i \in [\ell]$, the parameter-evaluation algorithm outputs the evaluated public key $\mathsf{pk}_C = \mathbf{V}_C \leftarrow \mathsf{EvalPK}(\mathbf{V}'_1, \ldots, \mathbf{V}'_\ell, C)$.

- $\mathsf{SigEval}(C, \mathsf{pp}', \mathbf{x}, \boldsymbol{\sigma}) \to \sigma$: The signature-evaluation algorithm first parses $\boldsymbol{\sigma} = (\boldsymbol{\sigma}^{\mathsf{pk}}, \boldsymbol{\sigma}^{\mathsf{m}})$. Then, it computes $\sigma^{\mathsf{pk}} \leftarrow \mathsf{SigEvalPK}(C, \mathsf{pp}', \boldsymbol{\sigma}^{\mathsf{pk}})$ and $\sigma^{\mathsf{m}} \leftarrow \mathsf{SigEvalM}(C, \mathsf{pp}', \mathbf{x}, \boldsymbol{\sigma})$, where the algorithms $\mathsf{SigEvalPK}$ and $\mathsf{SigEvalM}$ are defined as follows:
  - $\mathsf{SigEvalPK}(C, \mathsf{pp}', \boldsymbol{\sigma}^{\mathsf{pk}}) \to \sigma^{\mathsf{pk}}$: The $\mathsf{SigEvalPK}$ algorithm outputs the empty string $\sigma^{\mathsf{pk}} = \varepsilon$.
  - $\mathsf{SigEvalM}(C, \mathsf{pp}', \mathbf{x}, \boldsymbol{\sigma}) \to \sigma^{\mathsf{m}}$: On input a Boolean circuit $C \colon \{0,1\}^\ell \to \{0,1\}$, a set of public keys $\mathsf{pp}' = (\mathsf{pk}'_1, \ldots, \mathsf{pk}'_\ell)$, messages $\mathbf{x} = (x_1, \ldots, x_\ell)$, and signatures $\boldsymbol{\sigma} = (\sigma_1, \ldots, \sigma_\ell)$, the $\mathsf{SigEvalM}$ algorithm first parses $\mathsf{pk}'_i = \mathbf{V}'_i$ and $\sigma_i = (\sigma_i^{\mathsf{pk}}, \sigma_i^{\mathsf{m}}) = (\varepsilon, \mathbf{U}_i)$ for all $i \in [\ell]$. Then, it outputs $\sigma^{\mathsf{m}} = \mathbf{U}_C \leftarrow \mathsf{EvalU}\big((\mathbf{V}'_1, x_1, \mathbf{U}_1), \ldots, (\mathbf{V}'_\ell, x_\ell, \mathbf{U}_\ell), C\big)$.

  Finally, it outputs the signature $\sigma = (\sigma^{\mathsf{pk}}, \sigma^{\mathsf{m}})$.

- $\mathsf{Hide}(\mathsf{vk}, x, \sigma) \to \sigma^*$: On input a verification key $\mathsf{vk} = \mathbf{A}$, a message $x \in \{0,1\}$, and a signature $\sigma = (\varepsilon, \mathbf{U})$, the hide algorithm samples and outputs a signature

$$\sigma^* = \mathbf{u} \leftarrow \mathsf{SampleRight}(\mathbf{A}, \mathbf{AU} + (2x - 1) \cdot \mathbf{G}, \mathbf{U}, \mathbf{0}, \beta_{\mathsf{hide}}).$$

- $\mathsf{Verify}(\mathsf{pk}, \mathsf{vk}, x, \sigma) \to \{0,1\}$: On input a public key $\mathsf{pk} = \mathbf{V}$, a verification key $\mathsf{vk} = \mathbf{A}$, a message $x \in \mathcal{X}$, and a signature $\sigma = (\varepsilon, \mathbf{U})$, the verification algorithm first checks if $\mathbf{A}$ is a rank-n matrix and outputs 0 if this is the case. Then, it outputs 1 if $\|\mathbf{U}\| \le \beta_{\mathsf{eval}}$ and $\mathbf{AU} + x \cdot \mathbf{G} = \mathbf{V}$ and 0 otherwise.

- $\mathsf{VerifyFresh}(\mathsf{pk}, \mathsf{vk}, x, \sigma) \to \{0,1\}$: On input a public key $\mathsf{pk} = \mathbf{V}$, a verification key $\mathsf{vk} = \mathbf{A}$, a message $x \in \mathcal{X}$ and a signature $\sigma = (\varepsilon, \mathbf{U})$, the fresh verification algorithm first checks if $\mathbf{A}$ is a rank-n matrix and outputs 0 if this is the case. Then, it outputs 1 if $\|\mathbf{U}\| \le \beta_{\mathsf{ini}}$ and $\mathbf{AU} + x \cdot \mathbf{G} = \mathbf{V}$ and 0 otherwise.

- $\mathsf{VerifyHide}(\mathsf{pk}_C, \mathsf{vk}, x, \sigma^*) \to \{0,1\}$: On input a public key $\mathsf{pk}_C = \mathbf{V}$, a verification key $\mathsf{vk} = \mathbf{A}$, a message $x \in \{0,1\}$, and a signature $\sigma^* = \mathbf{u}$, the hide-verification algorithm first checks if $\mathbf{A}$ is a rank-n matrix and outputs 0 if this is the case. Then, it checks that $\|\mathbf{u}\| \le \beta_{\mathsf{hide}}$ and that $[\mathbf{A} \mid \mathbf{V} + (x-1) \cdot \mathbf{G}] \cdot \mathbf{u} = \mathbf{0}$, and accepts if both of these conditions hold. Otherwise, it rejects.

We now state and prove the correctness and security theorems for Construction 3.11.

Theorem 3.12 (Correctness)**.** *Fix a security parameter λ, lattice parameters n, m, q, norm bounds $\beta_{\mathsf{ini}}, \beta_{\mathsf{eval}}, \beta_{\mathsf{hide}}$, and a depth bound d. Suppose $m = O(n \log q)$, $\beta_{\mathsf{ini}} \ge O(n\sqrt{\log q})$, $\beta_{\mathsf{eval}} \ge \beta_{\mathsf{ini}} \cdot 2^{\widetilde{O}(d)}$, $\beta_{\mathsf{hide}} \ge \beta_{\mathsf{eval}} \cdot \omega(m\sqrt{\log m})$, and $q \ge \beta_{\mathsf{hide}}$. Then, Π_{HS} from Construction 3.11 satisfies signing correctness (Definition 3.2), evaluation correctness (Definition 3.3), and hiding correctness (Definition 3.4).*

Proof. Signing correctness follows from Theorem 2.3, evaluation correctness follows from Theorem 2.5, and hiding correctness follows from Theorem 2.4. $\square$

Theorem 3.13 (Unforgeability). *Fix a security parameter λ, lattice parameters n, m, q, norm bounds $\beta_{\mathsf{ini}}, \beta_{\mathsf{eval}}, \beta_{\mathsf{hide}}$, and a depth bound d. Suppose $m = O(n \log q)$. Then, under the $\mathsf{SIS}(n, m, q, \beta_{\mathsf{eval}})$ assumption, Π_{HS} in Construction 3.11 satisfies selective unforgeability (Definition 3.5, Remark 3.6).*

Proof. Follows from [GVW15b, §6]. □

Theorem 3.14 (Context-Hiding). *Fix a security parameter λ, lattice parameters n, m, q, norm bounds $\beta_{\mathsf{ini}}, \beta_{\mathsf{eval}}, \beta_{\mathsf{hide}}$, and a depth bound d. Suppose $m = O(n \log q)$, $\beta_{\mathsf{hide}} \geq 2 \cdot \beta_{\mathsf{eval}} \cdot \omega(m\sqrt{\log m})$, and $q \geq \beta_{\mathsf{hide}}$. Then, Π_{HS} in Construction 3.11 satisfies context-hiding security (Definition 3.9).*

Proof of Theorem 3.14. We construct a simulator $\mathcal{S} = (\mathcal{S}^{\mathsf{Ext}}, \mathcal{S}^{\mathsf{Gen}})$ as follows:

- $\mathcal{S}^{\mathsf{Ext}}(\mathsf{pk}, \mathsf{vk}, (\tilde{x}_0, \tilde{\sigma}_0), (\tilde{x}_1, \tilde{\sigma}_1))$: On input a public key pk, a verification key vk, and two message-signature pairs $(\tilde{x}_0, \tilde{\sigma}_0)$, $(\tilde{x}_1, \tilde{\sigma}_1)$, the simulator first parses $\tilde{\sigma}_0 = (\varepsilon, \tilde{\mathbf{U}}_0)$, $\tilde{\sigma}_1 = (\varepsilon, \tilde{\mathbf{U}}_1)$, and then outputs the simulation trapdoor $\mathsf{td} = \tilde{\mathbf{U}}_0 - \tilde{\mathbf{U}}_1$.
- $\mathcal{S}^{\mathsf{Gen}}(\mathsf{pk}, \mathsf{vk}, \mathsf{td}, x, \sigma^{\mathsf{pk}})$: On input a public key $\mathsf{pk} = \mathbf{V}$, a verification key $\mathsf{vk} = \mathbf{A}$, a trapdoor $\mathsf{td} = \tilde{\mathbf{U}}$, a message $x \in \{0,1\}$, and a message-independent component σ^{pk}, the simulator computes $\mathbf{u} \leftarrow \mathsf{SampleLeft}(\mathbf{A}, \mathbf{V} + (x-1) \cdot \mathbf{G}, \tilde{\mathbf{U}}, \mathbf{0}, \beta^*)$, and returns $\mathbf{u}$.

We now show that for any adversary $\mathcal{A}$, the experiments $\mathsf{Expt}^{\mathsf{ch}}_{\mathcal{A},\mathcal{S},\Pi_{\mathsf{HS}}}(\lambda, 0)$ and $\mathsf{Expt}^{\mathsf{ch}}_{\mathcal{A},\mathcal{S},\Pi_{\mathsf{HS}}}(\lambda, 1)$ are statistically indistinguishable. Consider the context-hiding experiment:

- Let $\mathsf{pk} = \mathbf{V}$, $\mathsf{vk} = \mathbf{A}$, and $(\tilde{x}_0, \tilde{\sigma}_0)$, $(\tilde{x}_1, \tilde{\sigma}_1)$ be the values that $\mathcal{A}$ sends to the challenger. Write $\tilde{\sigma}_0 = (\varepsilon, \tilde{\mathbf{U}}_0)$ and $\tilde{\sigma}_1 = (\varepsilon, \tilde{\mathbf{U}}_1)$. Without loss of generality, we can assume that $\mathbf{A}$ is a rank-n matrix, $\tilde{x}_0 \neq \tilde{x}_1$, and $\mathsf{Verify}(\mathsf{pk}, \mathsf{vk}, \tilde{x}_0, \tilde{\sigma}_0) = 1 = \mathsf{Verify}(\mathsf{pk}, \mathsf{vk}, \tilde{x}_1, \tilde{\sigma}_1)$. Otherwise, the output is always 0 in both experiments. Since $\tilde{x}_0, \tilde{x}_1 \in \{0,1\}$ and $\tilde{x}_0 \neq \tilde{x}_1$, we can assume without loss of generality that $\tilde{x}_0 = 0$ and $\tilde{x}_1 = 1$. Moreover, since $\tilde{\sigma}_0$ and $\tilde{\sigma}_1$ are valid signatures, $\|\tilde{\mathbf{U}}_0\|, \|\tilde{\mathbf{U}}_1\| \leq \beta_{\mathsf{eval}}$. This means that $\tilde{\mathbf{U}} = \tilde{\mathbf{U}}_0 - \tilde{\mathbf{U}}_1$ has bounded norm $\|\tilde{\mathbf{U}}\| \leq 2 \cdot \beta_{\mathsf{eval}}$, and moreover, that $\mathbf{A}\tilde{\mathbf{U}} = \mathbf{G}$, so $\mathsf{td} = \tilde{\mathbf{U}}$ is a $\mathbf{G}$-trapdoor for $\mathbf{A}$ (Theorem 2.4).
- Let $\mathsf{pk}' = \mathbf{V}'$, $x \in \{0,1\}$, $\sigma = (\varepsilon, \mathbf{U})$ be a query that $\mathcal{A}$ makes to the challenger. If $\mathbf{AU} + x \cdot \mathbf{G} \neq \mathbf{V}'$ or $\|\mathbf{U}\| > \beta_{\mathsf{eval}}$, then the challenger ignores the query (and replies with $\perp$) in both experiments. Therefore, assume that $\mathbf{AU} + x \cdot \mathbf{G} = \mathbf{V}'$ and $\|\mathbf{U}\| \leq \beta_{\mathsf{eval}}$. Then the challenger proceeds as follows:
  - In $\mathsf{Expt}^{\mathsf{ch}}_{\mathcal{A},\mathcal{S},\Pi_{\mathsf{HS}}}(\lambda, 0)$, the challenger's response is $\mathbf{u} \leftarrow \mathsf{SampleRight}(\mathbf{A}, \mathbf{AU} + (2x-1) \cdot \mathbf{G}, \mathbf{U}, \mathbf{0}, \beta_{\mathsf{hide}})$.
  - In $\mathsf{Expt}^{\mathsf{ch}}_{\mathcal{A},\mathcal{S},\Pi_{\mathsf{HS}}}(\lambda, 1)$, the challenger responds with $\mathbf{u} \leftarrow \mathcal{S}^{\mathsf{Gen}}(\mathsf{pk}', \mathsf{vk}, \mathsf{td}, x, \sigma^{\mathsf{pk}})$, which is equivalent to $\mathbf{u} \leftarrow \mathsf{SampleLeft}(\mathbf{A}, \mathbf{V}' + (x-1) \cdot \mathbf{G}, \tilde{\mathbf{U}}, \mathbf{0}, \beta_{\mathsf{hide}})$.

Since $\mathbf{V}' + (x-1) \cdot \mathbf{G} = \mathbf{AU} + (2x-1) \cdot \mathbf{G}$, by Theorem 2.4, as long as $\max(\|\mathbf{U}\|, \|\tilde{\mathbf{U}}\|) \cdot \omega(m\sqrt{\log m}) \leq \beta_{\mathsf{hide}} \leq q$ the challenger's responses to all of the queries in the two experiments are statistically indistinguishable. From above, $\|\tilde{\mathbf{U}}\| \leq 2 \cdot \beta_{\mathsf{eval}}$ and $\|\mathbf{U}\| \leq \beta_{\mathsf{eval}}$, and the claim follows. □

Remark 3.15 (Weak Context-Hiding)**.** Theorem 3.14 implies that the homomorphic signature scheme Π_{HS} in Construction 3.11 also satisfies context-hiding security against honest signers (Definition 3.8). Specifically, Theorem 3.14 guarantees the existence of a simulator $\mathcal{S} = (\mathcal{S}^{\mathsf{Ext}}, \mathcal{S}^{\mathsf{Gen}})$ that can be used to simulate the signatures generated by the Hide algorithm. In the context-hiding security game against honest signers, the signing key sk and verification key vk are generated honestly, and the context-hiding simulator $\mathcal{S}_{\mathsf{hon}}$ is given both vk and sk. Given sk, the simulator $\mathcal{S}_{\mathsf{hon}}$ can choose an arbitrary public key $\mathsf{pk} = \mathbf{V} \xleftarrow{\text{R}} \mathbb{Z}_q^{n \times m}$, and construct honest signatures $\tilde{\sigma}_0 \leftarrow \mathsf{Sign}(\mathsf{pk}, \mathsf{sk}, 0)$ and $\tilde{\sigma}_1 \leftarrow \mathsf{Sign}(\mathsf{pk}, \mathsf{sk}, 1)$. Simulator $\mathcal{S}_{\mathsf{hon}}$ can then invoke $\mathcal{S}^{\mathsf{Ext}}$ on $(\mathsf{pk}, \mathsf{vk}, (0, \tilde{\sigma}_0), (1, \tilde{\sigma}_1))$ to obtain the simulation trapdoor td, and then use $\mathcal{S}^{\mathsf{Gen}}$ to simulate the Hide algorithm. Thus, we can construct a simulator $\mathcal{S}_{\mathsf{hon}}$ for the weak context-hiding security game using the simulator $\mathcal{S}$ guaranteed by Theorem 3.14.

Theorem 3.16 (Compactness)**.** *Fix a security parameter λ, lattice parameters n, m, q, norm bounds $\beta_{\mathsf{ini}}, \beta_{\mathsf{eval}}, \beta_{\mathsf{hide}}$, and a depth bound d. Suppose $n = \operatorname{poly}(\lambda)$, $m = O(n \log q)$, and $q = 2^{\operatorname{poly}(\lambda, d)}$. Then, Π_{HS} in Construction 3.11 satisfies compactness (Definition 3.10).*

Proof. Follows from Theorem 2.5. Specifically, the signature output by $\mathsf{SigEval}$ is a matrix $\mathbf{U} \in \mathbb{Z}_q^{m \times m}$ which has size $m^2 \log q = \operatorname{poly}(\lambda, d)$. □

3.2.2 From Selective Security to Adaptive Security

In this section, we show how to transform a homomorphic signature scheme that satisfies only *selective unforgeability* to full unforgeability (Definition 3.5). Although the transformation follows the construction of [GVW15b], we give the full construction to show that the resulting construction still satisfies our strengthened notion of context-hiding (Definition 3.9).

Construction 3.17 (Adaptively-Secure Homomorphic Signature [GVW15b, adapted])**.** Fix a security parameter λ and a message length $\ell \in \mathbb{N}$. Let $\mathcal{C} = \{\mathcal{C}_\lambda\}_{\lambda \in \mathbb{N}}$ be a function class where each $\mathcal{C}_\lambda$ is a collection of Boolean circuits (on ℓ-bit inputs). Then, define the following quantities:

- First, let $\Pi_{\mathsf{HS,in}} = (\mathsf{PrmsGen_{in}}, \mathsf{KeyGen_{in}}, \mathsf{Sign_{in}}, \mathsf{PrmsEval_{in}}, \mathsf{SigEval_{in}}, \mathsf{Hide_{in}}, \mathsf{Verify_{in}}, \mathsf{VerifyFresh_{in}}, \mathsf{VerifyHide_{in}})$ be a *selectively-secure* decomposable homomorphic signature scheme with message space $\{0,1\}$, message length $\ell \in \mathbb{N}$, and function class $\mathcal{C} = \{\mathcal{C}_\lambda\}_{\lambda \in \mathbb{N}}$. This is the "inner" homomorphic signature scheme that will be used to sign messages $\mathbf{x} \in \{0,1\}^\ell$. For simplicity, assume also that the signatures in $\Pi_{\mathsf{HS,in}}$ have an "empty" message-independent component.[4]
- Let ρ be the length of the public keys in $\Pi_{\mathsf{HS,in}}$. For a circuit $C \in \mathcal{C}_\lambda$, let F_C be the function that maps $\mathsf{pp_{in}} \mapsto \mathsf{PrmsEval_{in}}(C, \mathsf{pp_{in}})$, where $\mathsf{pp_{in}}$ are the public parameters output by $\mathsf{PrmsGen_{in}}$.

[4]This restriction simplifies the presentation of our construction, and is satisfied by Construction 3.11. It is straightforward (but notationally cumbersome) to modify this generic construction to apply to the setting where the signatures in $\Pi_{\mathsf{HS,in}}$ have a non-empty message-independent component.

Let $\mathcal{C}' = \{\mathcal{C}'_\lambda\}_{\lambda \in \mathbb{N}}$ be a function class where each function class $\mathcal{C}'_\lambda$ contains all functions F_C for $C \in \mathcal{C}_\lambda$.

- Finally, let $\Pi_{\mathsf{HS,out}} = (\mathsf{PrmsGen}_{\mathsf{out}}, \mathsf{KeyGen}_{\mathsf{out}}, \mathsf{Sign}_{\mathsf{out}}, \mathsf{PrmsEval}_{\mathsf{out}}, \mathsf{SigEval}_{\mathsf{out}}, \mathsf{Hide}_{\mathsf{out}}, \mathsf{Verify}_{\mathsf{out}}, \mathsf{VerifyFresh}_{\mathsf{out}}, \mathsf{VerifyHide}_{\mathsf{out}})$ be a *selectively-secure* homomorphic signature scheme with message space $\{0,1\}$, message length $\rho \in \mathbb{N}$, and function class $\mathcal{C}' = \{\mathcal{C}'_\lambda\}_{\lambda \in \mathbb{N}}$. This is the "outer" homomorphic signature scheme that will be used to sign the public keys of $\Pi_{\mathsf{HS,in}}$.

We construct a homomorphic signature scheme $\Pi_{\mathsf{HS}} = (\mathsf{PrmsGen}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{PrmsEval}, \mathsf{SigEval}, \mathsf{Hide}, \mathsf{Verify}, \mathsf{VerifyFresh}, \mathsf{VerifyHide})$ for message space $\{0,1\}$, message length ℓ, and function class $\mathcal{C}$ as follows:

- $\mathsf{PrmsGen}(1^\lambda, 1^\ell) \to \mathsf{pp}$: On input the security parameter λ and message length ℓ, the parameter-generation algorithm generates independent public parameters $\mathsf{pp}_{\mathsf{out},i} \leftarrow \mathsf{PrmsGen}_{\mathsf{out}}(1^\lambda, 1^\rho)$ for $i \in [\ell]$. Then, it sets $\mathsf{pk}_i = \mathsf{pp}_{\mathsf{out},i}$ for $i \in [\ell]$ and returns $\mathsf{pp} = (\mathsf{pk}_1, \ldots, \mathsf{pk}_\ell)$.

- $\mathsf{KeyGen}(1^\lambda) \to (\mathsf{vk}, \mathsf{sk})$: On input the security parameter λ, the key-generation algorithm generates two pairs of keys $(\mathsf{vk}_{\mathsf{in}}, \mathsf{sk}_{\mathsf{in}}) \leftarrow \mathsf{KeyGen}_{\mathsf{in}}(1^\lambda)$, $(\mathsf{vk}_{\mathsf{out}}, \mathsf{sk}_{\mathsf{out}}) \leftarrow \mathsf{KeyGen}_{\mathsf{out}}(1^\lambda)$, and sets $\mathsf{vk} = (\mathsf{vk}_{\mathsf{in}}, \mathsf{vk}_{\mathsf{out}})$ and $\mathsf{sk} = (\mathsf{sk}_{\mathsf{in}}, \mathsf{sk}_{\mathsf{out}})$.

- $\mathsf{Sign}(\mathsf{pk}_i, \mathsf{sk}, x_i) \to \sigma_i$: On input a public key $\mathsf{pk}_i = \mathsf{pp}_{\mathsf{out},i}$, a signing key $\mathsf{sk} = (\mathsf{sk}_{\mathsf{in}}, \mathsf{sk}_{\mathsf{out}})$, and a message $x_i \in \{0,1\}$, the signing algorithm computes $\sigma_i^{\mathsf{pk}} \leftarrow \mathsf{SignPK}(\mathsf{pk}_i, \mathsf{sk})$ and $\sigma_i^{\mathsf{m}} \leftarrow \mathsf{SignM}(\mathsf{pk}_i, \mathsf{sk}, x_i, \sigma_i^{\mathsf{pk}})$ where the algorithms SignPK and SignM are defined as follows:
  - $\mathsf{SignPK}(\mathsf{pk}_i, \mathsf{sk})$: The message-independent signing algorithm first samples a fresh public key $\mathsf{pk}_{\mathsf{in},i} \leftarrow \mathsf{PrmsGen}_{\mathsf{in}}(1^\lambda, 1^1)$ for the inner homomorphic signature scheme.[5] By assumption, $\mathsf{pk}_{\mathsf{in},i}$ is a bit-string of length ρ. Then, the algorithm signs the public key $\mathsf{pk}_{\mathsf{in},i}$ using the outer signature scheme: $\boldsymbol{\sigma}_{\mathsf{out},i} \leftarrow \mathsf{Sign}_{\mathsf{out}}(\mathsf{pp}_{\mathsf{out},i}, \mathsf{sk}_{\mathsf{out}}, \mathsf{pk}_{\mathsf{in},i})$. It returns $\sigma_i^{\mathsf{pk}} = (\boldsymbol{\sigma}_{\mathsf{out},i}, \mathsf{pk}_{\mathsf{in},i})$.
  - $\mathsf{SignM}(\mathsf{pk}_i, \mathsf{sk}, x_i, \sigma^{\mathsf{pk}})$: The message-dependent signing algorithm parses $\sigma^{\mathsf{pk}} = (\boldsymbol{\sigma}_{\mathsf{out},i}, \mathsf{pk}_{\mathsf{in},i})$, and signs the message using the inner signature scheme: $\sigma_{\mathsf{in},i} \leftarrow \mathsf{Sign}_{\mathsf{in}}(\mathsf{pk}_{\mathsf{in},i}, \mathsf{sk}_{\mathsf{in}}, x_i)$. It outputs $\sigma_i^{\mathsf{m}} = \sigma_{\mathsf{in},i}$.

  Finally, the signing algorithm outputs $\sigma_i = (\sigma_i^{\mathsf{pk}}, \sigma_i^{\mathsf{m}})$.

- $\mathsf{PrmsEval}(C, \mathsf{pp}) \to \mathsf{pk}_C$: On input a circuit $C \in \mathcal{C}$ and public parameters $\mathsf{pp} = (\mathsf{pk}_1, \ldots, \mathsf{pk}_\ell)$, the parameter-evaluation algorithm parses $\mathsf{pk}_i = \mathsf{pp}_{\mathsf{out},i}$ for each $i \in [\ell]$. It outputs $\mathsf{pk}_C \leftarrow \mathsf{PrmsEval}_{\mathsf{out}}(F_C, (\mathsf{pp}_{\mathsf{out},1}, \ldots, \mathsf{pp}_{\mathsf{out},\ell}))$.

[5]Note that we are implicitly assuming here that the public keys $\mathsf{pk}_{\mathsf{in},i}$ for each $i \in [\ell]$ can be generated independently of one another: namely, that the output distribution of $\mathsf{PrmsGen}_{\mathsf{in}}(1^\lambda, 1^\ell)$ is identical to ℓ independent invocations of $\mathsf{PrmsGen}_{\mathsf{in}}(1^\lambda, 1^1)$. This property is satisfied by the homomorphic signature scheme in Construction 3.11.

- $\mathsf{SigEval}(C, \mathsf{pp}', \mathbf{x}, \boldsymbol{\sigma}) \to \sigma$: On input a circuit $C \in \mathcal{C}$, public parameters $\mathsf{pp}' = (\mathsf{pk}'_1, \ldots, \mathsf{pk}'_\ell)$, a message $\mathbf{x} \in \{0,1\}^\ell$ and a signature $\boldsymbol{\sigma} = (\boldsymbol{\sigma}^{\mathsf{pk}}, \boldsymbol{\sigma}^{\mathsf{m}})$, the signature-evaluation algorithm parses $\mathsf{pk}'_i = \mathsf{pp}'_{\mathsf{out},i}$ for all $i \in [\ell]$. Then, it computes $\sigma^{\mathsf{pk}} \leftarrow \mathsf{SigEvalPK}(C, \mathsf{pp}', \boldsymbol{\sigma}^{\mathsf{pk}})$ and $\sigma^{\mathsf{m}} \leftarrow \mathsf{SigEvalM}(C, \mathsf{pp}', \mathbf{x}, \boldsymbol{\sigma})$, where $\mathsf{SigEvalPK}$ and $\mathsf{SigEvalM}$ are defined as follows:
  - $\mathsf{SigEvalPK}(C, \mathsf{pp}, \boldsymbol{\sigma}^{\mathsf{pk}})$: The message-independent signature-evaluation algorithm first parses $\boldsymbol{\sigma}^{\mathsf{pk}} = ((\boldsymbol{\sigma}_{\mathsf{out},1}, \mathsf{pk}_{\mathsf{in},1}), \ldots, (\boldsymbol{\sigma}_{\mathsf{out},\ell}, \mathsf{pk}_{\mathsf{in},\ell}))$. It then computes
  $$\boldsymbol{\sigma}_{\mathsf{out},C} \leftarrow \mathsf{SigEval}_{\mathsf{out}}(F_C, (\mathsf{pp}'_{\mathsf{out},1}, \ldots, \mathsf{pp}'_{\mathsf{out},\ell}), (\mathsf{pk}_{\mathsf{in},1}, \ldots, \mathsf{pk}_{\mathsf{in},\ell}), (\boldsymbol{\sigma}_{\mathsf{out},1}, \ldots, \boldsymbol{\sigma}_{\mathsf{out},\ell})),$$
  and $\mathsf{pk}_{\mathsf{in},C} \leftarrow \mathsf{PrmsEval}_{\mathsf{in}}(C, (\mathsf{pk}_{\mathsf{in},1}, \ldots, \mathsf{pk}_{\mathsf{in},\ell}))$. Finally, it returns $\sigma^{\mathsf{pk}} = (\boldsymbol{\sigma}_{\mathsf{out},C}, \mathsf{pk}_{\mathsf{in},C})$.
  - $\mathsf{SigEvalM}(C, \mathsf{pp}', \mathbf{x}, \boldsymbol{\sigma})$: The message-dependent signature-evaluation algorithm writes $\boldsymbol{\sigma}$ as $(\boldsymbol{\sigma}^{\mathsf{pk}}, \boldsymbol{\sigma}^{\mathsf{m}})$, where $\boldsymbol{\sigma}^{\mathsf{pk}} = ((\boldsymbol{\sigma}_{\mathsf{out},1}, \mathsf{pk}_{\mathsf{in},1}), \ldots, (\boldsymbol{\sigma}_{\mathsf{out},\ell}, \mathsf{pk}_{\mathsf{in},\ell}))$, and $\boldsymbol{\sigma}^{\mathsf{m}} = (\sigma_{\mathsf{in},1}, \ldots, \sigma_{\mathsf{in},\ell})$. It outputs the signature $\sigma^{\mathsf{m}} \leftarrow \mathsf{SigEval}_{\mathsf{in}}(C, (\mathsf{pk}_{\mathsf{in},1}, \ldots, \mathsf{pk}_{\mathsf{in},\ell}), \mathbf{x}, (\sigma_{\mathsf{in},1}, \ldots, \sigma_{\mathsf{in},\ell}))$.

  Finally, the signature-evaluation algorithm outputs $\sigma = (\sigma^{\mathsf{pk}}, \sigma^{\mathsf{m}})$.

- $\mathsf{Hide}(\mathsf{vk}, x, \sigma) \to \sigma^*$: On input a verification key $\mathsf{vk} = (\mathsf{vk}_{\mathsf{in}}, \mathsf{vk}_{\mathsf{out}})$, a message $x \in \{0,1\}$, and a signature $\sigma = (\sigma^{\mathsf{pk}}, \sigma^{\mathsf{m}})$, the hide algorithm parses $\sigma^{\mathsf{m}} = \sigma_{\mathsf{in}}$. It computes $\sigma^*_{\mathsf{in}} \leftarrow \mathsf{Hide}_{\mathsf{in}}(\mathsf{vk}_{\mathsf{in}}, x, \sigma_{\mathsf{in}})$, and returns $\sigma^* = (\sigma^{\mathsf{pk}}, \sigma^*_{\mathsf{in}})$.

- $\mathsf{Verify}(\mathsf{pk}, \mathsf{vk}, x, \sigma) \to \{0,1\}$: On input a public key $\mathsf{pk} = \mathsf{pp}_{\mathsf{out}}$, a verification key $\mathsf{vk} = (\mathsf{vk}_{\mathsf{in}}, \mathsf{vk}_{\mathsf{out}})$, a message $x \in \{0,1\}$, and a signature $\sigma = (\sigma^{\mathsf{pk}}, \sigma^{\mathsf{m}})$, the verification algorithm parses $\sigma^{\mathsf{pk}} = (\boldsymbol{\sigma}_{\mathsf{out}}, \mathsf{pk}_{\mathsf{in}})$, $\sigma^{\mathsf{m}} = \sigma_{\mathsf{in}}$, and accepts if
$$\mathsf{Verify}_{\mathsf{out}}(\mathsf{pp}_{\mathsf{out}}, \mathsf{vk}_{\mathsf{out}}, \mathsf{pk}_{\mathsf{in}}, \boldsymbol{\sigma}_{\mathsf{out}}) = 1 \quad \text{and} \quad \mathsf{Verify}_{\mathsf{in}}(\mathsf{pk}_{\mathsf{in}}, \mathsf{vk}_{\mathsf{in}}, x, \sigma_{\mathsf{in}}) = 1.$$
  Otherwise, it rejects.

- $\mathsf{VerifyFresh}(\mathsf{pk}, \mathsf{vk}, x, \sigma) \to \{0,1\}$: On input a public key $\mathsf{pk} = \mathsf{pp}_{\mathsf{out}}$, a verification key $\mathsf{vk} = (\mathsf{vk}_{\mathsf{in}}, \mathsf{vk}_{\mathsf{out}})$, a message $x \in \{0,1\}$, and signature $\sigma = (\sigma^{\mathsf{pk}}, \sigma^{\mathsf{m}})$, the fresh verification algorithm parses $\sigma^{\mathsf{pk}} = (\boldsymbol{\sigma}_{\mathsf{out}}, \mathsf{pk}_{\mathsf{in}})$, $\sigma^{\mathsf{m}} = \sigma_{\mathsf{in}}$, and accepts if
$$\mathsf{VerifyFresh}_{\mathsf{out}}(\mathsf{pp}_{\mathsf{out}}, \mathsf{vk}_{\mathsf{out}}, \mathsf{pk}_{\mathsf{in}}, \boldsymbol{\sigma}_{\mathsf{out}}) = 1 \quad \text{and} \quad \mathsf{VerifyFresh}_{\mathsf{in}}(\mathsf{pk}_{\mathsf{in}}, \mathsf{vk}_{\mathsf{in}}, x, \sigma_{\mathsf{in}}) = 1.$$
  Otherwise, it rejects.

- $\mathsf{VerifyHide}(\mathsf{pk}, \mathsf{vk}, x, \sigma^*) \to \{0,1\}$: On input a public key $\mathsf{pk} = \mathsf{pp}_{\mathsf{out}}$, a verification key $\mathsf{vk} = (\mathsf{vk}_{\mathsf{in}}, \mathsf{vk}_{\mathsf{out}})$, a message $x \in \{0,1\}$, and signature $\sigma^* = (\sigma^{\mathsf{pk}}, \sigma^{\mathsf{m}})$, the hide verification algorithm parses $\sigma^{\mathsf{pk}} = (\boldsymbol{\sigma}_{\mathsf{out}}, \mathsf{pk}_{\mathsf{in}})$, $\sigma^{\mathsf{m}} = \sigma_{\mathsf{in}}$, and accepts if
$$\mathsf{Verify}_{\mathsf{out}}(\mathsf{pp}_{\mathsf{out}}, \mathsf{vk}_{\mathsf{out}}, \mathsf{pk}_{\mathsf{in}}, \boldsymbol{\sigma}_{\mathsf{out}}) = 1 \quad \text{and} \quad \mathsf{VerifyHide}_{\mathsf{in}}(\mathsf{pk}_{\mathsf{in}}, \mathsf{vk}_{\mathsf{in}}, x, \sigma_{\mathsf{in}}) = 1.$$

Otherwise, it rejects.

Theorem 3.18 (Correctness). *Suppose $\Pi_{\mathsf{HS,in}}$ and $\Pi_{\mathsf{HS,out}}$ satisfy signing correctness (Definition 3.2), evaluation correctness (Definition 3.3), and hiding correctness (Definition 3.4). Then, Construction 3.17 satisfies signing correctness, evaluation correctness, and hiding correctness.*

Proof. Follows by construction. ☐

Theorem 3.19 (Unforgeability). *Suppose $\Pi_{\mathsf{HS,in}}$ and $\Pi_{\mathsf{HS,out}}$ satisfy selective-unforgeability (Definition 3.5). Then, Construction 3.17 satisfies unforgeability (Definition 3.3).*

Proof. Follows from [GVW15b, §4]. ☐

Theorem 3.20 (Context-Hiding). *Suppose $\Pi_{\mathsf{HS,in}}$ satisfies context-hiding (Definition 3.9). Then, Construction 3.17 satisfies context-hiding.*

Proof. Let $\mathcal{S}_{\mathsf{in}} = (\mathcal{S}_{\mathsf{in}}^{\mathsf{Ext}}, \mathcal{S}_{\mathsf{in}}^{\mathsf{Gen}})$ be the context-hiding simulator for $\Pi_{\mathsf{HS,in}}$. We construct a context-hiding simulator $\mathcal{S} = (\mathcal{S}^{\mathsf{Ext}}, \mathcal{S}^{\mathsf{Gen}})$ for Π_{HS} as follows:

- $\mathcal{S}^{\mathsf{Ext}}(\mathsf{pk}, \mathsf{vk}, (\tilde{x}_0, \tilde{\sigma}_0), (\tilde{x}_1, \tilde{\sigma}_1))$: On input a public key pk, a verification key $\mathsf{vk} = (\mathsf{vk}_{\mathsf{in}}, \mathsf{vk}_{\mathsf{out}})$, and two message-signature pairs $(\tilde{x}_0, \tilde{\sigma}_0)$, $(\tilde{x}_1, \tilde{\sigma}_1)$, the simulator first parses $\tilde{\sigma}_0 = (\tilde{\sigma}_0^{\mathsf{pk}}, \tilde{\sigma}_0^{\mathsf{m}})$, and $\tilde{\sigma}_1 = (\tilde{\sigma}_1^{\mathsf{pk}}, \tilde{\sigma}_1^{\mathsf{m}})$. Then, it parses $\tilde{\sigma}_0^{\mathsf{pk}} = (\boldsymbol{\sigma}_{\mathsf{out}}, \mathsf{pk}_{\mathsf{in}}) = \tilde{\sigma}_1^{\mathsf{pk}}$, $\tilde{\sigma}_0^{\mathsf{m}} = \tilde{\sigma}_{\mathsf{in},0}$, and $\tilde{\sigma}_1^{\mathsf{m}} = \tilde{\sigma}_{\mathsf{in},1}$. Finally, it outputs the trapdoor $\mathsf{td} \leftarrow \mathcal{S}_{\mathsf{in}}^{\mathsf{Ext}}(\mathsf{pk}_{\mathsf{in}}, \mathsf{vk}_{\mathsf{in}}, (\tilde{x}_0, \tilde{\sigma}_{\mathsf{in},0}), (\tilde{x}_1, \tilde{\sigma}_{\mathsf{in},1}))$.
- $\mathcal{S}^{\mathsf{Gen}}(\mathsf{pk}, \mathsf{vk}, \mathsf{td}, x, \sigma^{\mathsf{pk}})$: On input a public key pk, a verification key $\mathsf{vk} = (\mathsf{vk}_{\mathsf{in}}, \mathsf{vk}_{\mathsf{out}})$, a trapdoor td, a message $x \in \{0,1\}$, and a message-independent signature component $\sigma^{\mathsf{pk}} = (\boldsymbol{\sigma}_{\mathsf{out}}, \mathsf{pk}_{\mathsf{in}})$, the simulator computes $\sigma_{\mathsf{in}}^* \leftarrow \mathcal{S}_{\mathsf{in}}^{\mathsf{Gen}}(\mathsf{pk}_{\mathsf{in}}, \mathsf{vk}_{\mathsf{in}}, \mathsf{td}, x, \varepsilon)$ and outputs $\sigma^* \leftarrow (\sigma^{\mathsf{pk}}, \sigma_{\mathsf{in}}^*)$. Here, we rely on the assumption that the signatures in $\Pi_{\mathsf{HS,in}}$ have an empty message-independent component.

We now show that if $\Pi_{\mathsf{HS,in}}$ is context-hiding, then experiments $\mathsf{Expt}^{\mathsf{ch}}_{\mathcal{A},\mathcal{S},\Pi_{\mathsf{HS}}}(\lambda, 0)$ and $\mathsf{Expt}^{\mathsf{ch}}_{\mathcal{A},\mathcal{S},\Pi_{\mathsf{HS}}}(\lambda, 1)$ are indistinguishable for any unbounded adversary $\mathcal{A}$.

- Let pk be the public key, $\mathsf{vk} = (\mathsf{vk}_{\mathsf{in}}, \mathsf{vk}_{\mathsf{out}})$ be the verification key, and $(\tilde{x}_0, \tilde{\sigma}_0)$, $(\tilde{x}_1, \tilde{\sigma}_1)$ be the message-signature pairs that $\mathcal{A}$ submits to the context-hiding challenger at the beginning of the experiment. Write $\tilde{\sigma}_0 = (\tilde{\sigma}_0^{\mathsf{pk}}, \tilde{\sigma}_0^{\mathsf{m}})$ and $\tilde{\sigma}_1 = (\tilde{\sigma}_1^{\mathsf{pk}}, \tilde{\sigma}_1^{\mathsf{m}})$, where $\tilde{\sigma}_0^{\mathsf{m}} = \tilde{\sigma}_{\mathsf{in},0}$ and $\tilde{\sigma}_1^{\mathsf{m}} = \tilde{\sigma}_{\mathsf{in},1}$ Without loss of generality, we can assume that $\tilde{x}_0 \neq \tilde{x}_1$, $\tilde{\sigma}_0^{\mathsf{pk}} = \tilde{\sigma}_1^{\mathsf{pk}}$, and that $\mathsf{Verify}_{\mathsf{in}}(\mathsf{pk}_{\mathsf{in}}, \mathsf{vk}_{\mathsf{in}}, \tilde{x}_0, \tilde{\sigma}_{\mathsf{in},0}) = 1 = \mathsf{Verify}_{\mathsf{in}}(\mathsf{pk}_{\mathsf{in}}, \mathsf{vk}_{\mathsf{in}}, \tilde{x}_1, \tilde{\sigma}_{\mathsf{in},1})$. Otherwise, the output of the experiment is always 0, and the adversary's distinguishing advantage is correspondingly 0. Next, in $\mathsf{Expt}^{\mathsf{ch}}_{\mathcal{A},\mathcal{S},\Pi_{\mathsf{HS}}}(\lambda, 1)$, the challenger constructs a trapdoor td by invoking $\mathsf{td} \leftarrow \mathcal{S}_{\mathsf{in}}^{\mathsf{Ext}}(\mathsf{pk}_{\mathsf{in}}, \mathsf{vk}_{\mathsf{in}}, (\tilde{x}_0, \tilde{\sigma}_{\mathsf{in},0}), (\tilde{x}_1, \tilde{\sigma}_{\mathsf{in},1}))$.
- Let pk', $x \in \{0,1\}$, $\sigma = (\sigma^{\mathsf{pk}}, \sigma^{\mathsf{m}})$ be a query $\mathcal{A}$ makes to the challenger. If $\mathsf{Verify}(\mathsf{pk}', \mathsf{vk}, x, \sigma) = 1$, then the challenger proceeds as follows:

– In $\mathsf{Expt}^{\mathsf{ch}}_{\mathcal{A},\mathcal{S},\Pi_{\mathsf{HS}}}(\lambda, 0)$, the challenger parses $\sigma^{\mathsf{m}} = \sigma_{\mathsf{in}}$, and computes $\sigma^*_{\mathsf{in}} \leftarrow \mathsf{Hide}_{\mathsf{in}}(\mathsf{vk}_{\mathsf{in}}, x, \sigma_{\mathsf{in}})$. It replies to the adversary with $\sigma^* \leftarrow (\sigma^{\mathsf{pk}}, \sigma^*_{\mathsf{in}})$.

– In $\mathsf{Expt}^{\mathsf{ch}}_{\mathcal{A},\mathcal{S},\Pi_{\mathsf{HS}}}(\lambda, 1)$, the challenger parses $\sigma^{\mathsf{pk}} = (\boldsymbol{\sigma}_{\mathsf{out}}, \mathsf{pk}_{\mathsf{in}})$, and computes the signature $\sigma^*_{\mathsf{in}} \leftarrow \mathcal{S}^{\mathsf{Gen}}(\mathsf{pk}_{\mathsf{in}}, \mathsf{vk}_{\mathsf{in}}, \mathsf{td}, x, \varepsilon)$. It returns $\sigma^* = (\sigma^{\mathsf{pk}}, \sigma^*_{\mathsf{in}})$.

Since $\mathsf{Verify}(\mathsf{pk}_{\mathsf{in}}, \mathsf{vk}_{\mathsf{in}}, \tilde{x}_0, \tilde{\sigma}_{\mathsf{in},0}) = 1 = \mathsf{Verify}(\mathsf{pk}_{\mathsf{in}}, \mathsf{vk}_{\mathsf{in}}, \tilde{x}_1, \tilde{\sigma}_{\mathsf{in},1})$, and $\tilde{\sigma}^{\mathsf{pk}}_0 = \tilde{\sigma}^{\mathsf{pk}}_1$, we have that td is a valid trapdoor for $\mathcal{S}^{\mathsf{Ext}}_{\mathsf{in}}$. Since $\Pi_{\mathsf{HS,in}}$ is context-hiding, the message-dependent component σ^*_{in} of the final signature σ^* generated by $\mathcal{S}^{\mathsf{Gen}}$ in $\mathsf{Expt}^{\mathsf{ch}}_{\mathcal{A},\mathcal{S},\Pi_{\mathsf{HS}}}(\lambda, 1)$ is statistically indistinguishable from σ^*_{in} generated by the challenger in $\mathsf{Expt}^{\mathsf{ch}}_{\mathcal{A},\mathcal{S},\Pi_{\mathsf{HS}}}(\lambda, 0)$. The claim follows. ☐

Theorem 3.21 (Compactness). *Fix a security parameter λ. Suppose $\Pi_{\mathsf{HS,in}}$ and $\Pi_{\mathsf{HS,out}}$ satisfy compactness (Definition 3.10), and moreover, the size of a homomorphically-evaluated public key output by $\mathsf{PrmsEval}_{\mathsf{out}}(C, \cdot)$ is $\mathrm{poly}(\lambda, d)$, where d is a bound on the depth of the circuit C. Then, Construction 3.17 satisfies compactness.*

Proof. Follows immediately by construction. Specifically, the signature output by $\mathsf{SigEval}$ consists of compact signatures output by $\mathsf{SigEval}_{\mathsf{out}}$ and $\mathsf{SigEval}_{\mathsf{in}}$, and a homomorphically-evaluated public key output by $\mathsf{PrmsEval}_{\mathsf{out}}$. Therefore, the size of the signatures depend only on $|C(\mathbf{x})|$ and is independent of $|\mathbf{x}|$. ☐

Instantiating the construction. We note that both $\Pi_{\mathsf{HS,in}}$ and $\Pi_{\mathsf{HS,out}}$ can be instantiated by Construction 3.11 in Section 3.2. In particular, Construction 3.11 satisfies the additional compactness requirement on the size of the public keys needed in Theorem 3.21. This yields the following corollary:

Corollary 3.22 (Adaptively-Secure Homomorphic Signatures). *Fix a security parameter λ and a message length $\ell = \mathrm{poly}(\lambda)$. Let $\mathcal{C} = \{\mathcal{C}_\lambda\}_{\lambda \in \mathbb{N}}$ be a function class where $\mathcal{C}_\lambda$ consists of Boolean circuits of depth up to $d = d(\lambda)$ on ℓ-bit inputs. Then, under the SIS assumption, there exists a homomorphic signature scheme $\Pi_{\mathsf{HS}} = (\mathsf{PrmsGen}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{PrmsEval}, \mathsf{SigEval}, \mathsf{Hide}, \mathsf{Verify}, \mathsf{VerifyFresh}, \mathsf{VerifyHide})$ with message space $\{0, 1\}$, message length ℓ, and function class $\mathcal{C}$ that satisfies adaptive unforgeability (Definition 3.5), context-hiding (Definition 3.9), and compactness (Definition 3.10).*

3.3 Preprocessing NIZKs from Homomorphic Signatures

In this section, we begin by formally defining the notion of a non-interactive zero-knowledge argument in the preprocessing model (i.e., "preprocessing NIZKs"). This notion was first introduced by De Santis et al. [DMP88], who also gave the first candidate construction of a preprocessing NIZK from one-way functions. Multiple works have since proposed additional candidates of preprocessing NIZKs from one-way functions [LS90, Dam92, IKOS07] or oblivious transfer [KMO89]. However, all of these constructions are *single-theorem*: the proving or verification key cannot be reused for multiple theorems without compromising either soundness or zero-knowledge. We provide a more detailed discussion of existing preprocessing NIZK constructions in Remark 3.33.

Definition 3.23 (NIZK Arguments in the Preprocessing Model)**.** Let $\mathcal{R}$ be an NP relation, and let $\mathcal{L}$ be its corresponding language. A non-interactive zero-knowledge (NIZK) argument for $\mathcal{L}$ in the preprocessing model consists of a tuple of three algorithms $\Pi_{\mathsf{PPNIZK}} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ with the following properties:

- $\mathsf{Setup}(1^\lambda) \to (k_P, k_V)$: On input the security parameter λ, the setup algorithm (implemented in a "preprocessing" step) outputs a proving key k_P and a verification key k_V.
- $\mathsf{Prove}(k_P, \mathbf{x}, \mathbf{w}) \to \pi$: On input the proving key k_P, a statement $\mathbf{x}$, and a witness $\mathbf{w}$, the prover's algorithm outputs a proof π.
- $\mathsf{Verify}(k_V, \mathbf{x}, \pi) \to \{0,1\}$: On input the verification key k_V, a statement $\mathbf{x}$, and a proof π, the verifier either accepts (with output 1) or rejects (with output 0).

Moreover, Π_{PPNIZK} should satisfy the following properties:

- **Completeness:** For all $\mathbf{x}, \mathbf{w}$ where $\mathcal{R}(\mathbf{x}, \mathbf{w}) = 1$, if we take $(k_P, k_V) \leftarrow \mathsf{Setup}(1^\lambda)$;

$$\Pr[\pi \leftarrow \mathsf{Prove}(k_P, \mathbf{x}, \mathbf{w}) : \mathsf{Verify}(k_V, \mathbf{x}, \pi) = 1] = 1.$$

- **Soundness:** For all efficient adversaries $\mathcal{A}$, if we take $(k_P, k_V) \leftarrow \mathsf{Setup}(1^\lambda)$, then

$$\Pr[(\mathbf{x}, \pi) \leftarrow \mathcal{A}^{\mathsf{Verify}(k_V, \cdot, \cdot)}(k_P) : \mathbf{x} \notin \mathcal{L} \wedge \mathsf{Verify}(k_V, \mathbf{x}, \pi) = 1] = \operatorname{negl}(\lambda).$$

- **Zero-Knowledge:** For all efficient adversaries $\mathcal{A}$, there exists an efficient simulator $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2)$ such that if we take $(k_P, k_V) \leftarrow \mathsf{Setup}(1^\lambda)$ and $\tau_V \leftarrow \mathcal{S}_1(1^\lambda, k_V)$, we have that

$$\left| \Pr[\mathcal{A}^{\mathcal{O}_0(k_P, \cdot, \cdot)}(k_V) = 1] - \Pr[\mathcal{A}^{\mathcal{O}_1(k_V, \tau_V, \cdot, \cdot)}(k_V) = 1] \right| = \operatorname{negl}(\lambda),$$

where the oracle $\mathcal{O}_0(k_P, \mathbf{x}, \mathbf{w})$ outputs $\mathsf{Prove}(k_P, \mathbf{x}, \mathbf{w})$ if $\mathcal{R}(\mathbf{x}, \mathbf{w}) = 1$ and $\perp$ otherwise, and the oracle $\mathcal{O}_1(k_V, \tau_V, \mathbf{x}, \mathbf{w})$ outputs $\mathcal{S}_2(k_V, \tau_V, \mathbf{x})$ if $\mathcal{R}(\mathbf{x}, \mathbf{w}) = 1$ and $\perp$ otherwise.

Remark 3.24 (Comparison to NIZKs in the CRS Model)**.** Our zero-knowledge definition in Definition 3.23 does *not* allow the simulator to choose the verification state k_V. We can also consider a slightly weaker notion of zero-knowledge where the simulator also chooses the verification state:

- **Zero-Knowledge:** For all efficient adversaries $\mathcal{A}$, there exists an efficient simulator $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2)$ such that if we take $(k_P, k_V) \leftarrow \mathsf{Setup}(1^\lambda)$ and $(\tilde{k}_V, \tilde{\tau}_V) \leftarrow \mathcal{S}_1(1^\lambda)$, we have that

$$\left| \Pr[\mathcal{A}^{\mathsf{Prove}(k_P, \cdot, \cdot)}(k_V) = 1] - \Pr[\mathcal{A}^{\mathcal{O}(\tilde{k}_V, \tilde{\tau}_V, \cdot, \cdot)}(\tilde{k}_V) = 1] \right| = \operatorname{negl}(\lambda),$$

where the oracle $\mathcal{O}(\tilde{k}_V, \tilde{\tau}_V, \mathbf{x}, \mathbf{w})$ outputs $\mathcal{S}_2(\tilde{k}_V, \tilde{\tau}_V, \mathbf{x})$ if $\mathcal{R}(\mathbf{x}, \mathbf{w}) = 1$ and $\perp$ otherwise.

We note that this definition of zero-knowledge captures the standard notion of NIZK arguments in the common reference string (CRS) model. Specifically, in the CRS model, the Setup algorithm outputs a single CRS σ. The proving and verification keys are both defined to be σ.

Preprocessing NIZKs from homomorphic signatures. As described in Section 3.1, we can combine a homomorphic signature scheme (for general circuits) with any CPA-secure symmetric encryption scheme to obtain a preprocessing NIZK for general NP languages. We give our construction and security analysis below. Combining the lattice-based construction of homomorphic signatures (Construction 3.11) with Fact 2.6, we obtain the first multi-theorem preprocessing NIZK from standard lattice assumptions (Corollary 3.27). In Remark 3.28, we note that a variant of Construction 3.25 also gives a *publicly-verifiable* preprocessing NIZK.

Construction 3.25 (Preprocessing NIZKs from Homomorphic Signatures)**.** Fix a security parameter λ, and define the following quantities:

- Let $\mathcal{R}\colon \{0,1\}^n \times \{0,1\}^m \to \{0,1\}$ be an NP relation and $\mathcal{L}$ be its corresponding language.
- Let $\Pi_{\mathsf{enc}} = (\mathsf{SE.KeyGen}, \mathsf{SE.Encrypt}, \mathsf{SE.Decrypt})$ be a symmetric encryption scheme with message space $\{0,1\}^m$ and secret-key space $\{0,1\}^\rho$.
- For a message $\mathbf{x} \in \{0,1\}^n$ and ciphertext ct from the ciphertext space of Π_{enc}, define the function $f_{\mathbf{x},\mathsf{ct}}(k_{\mathsf{SE}}) := \mathcal{R}(\mathbf{x}, \mathsf{SE.Decrypt}(k_{\mathsf{SE}}, \mathsf{ct}))$.
- Let $\Pi_{\mathsf{HS}} = (\mathsf{PrmsGen}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{PrmsEval}, \mathsf{SigEval}, \mathsf{Hide}, \mathsf{Verify}, \mathsf{VerifyFresh}, \mathsf{VerifyHide})$ be a homomorphic signature scheme with message space $\{0,1\}$, message length ρ, and function class $\mathcal{C}$ that includes all functions of the form $f_{\mathbf{x},\mathsf{ct}}$.

We construct a preprocessing NIZK argument $\Pi_{\mathsf{NIZK}} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ as follows:

- $\mathsf{Setup}(1^\lambda) \to (k_P, k_V)$: First, generate a secret key $k_{\mathsf{SE}} \leftarrow \mathsf{SE.KeyGen}(1^\lambda)$. Next, generate $\mathsf{pp}_{\mathsf{HS}} \leftarrow \mathsf{PrmsGen}(1^\lambda, 1^\rho)$ and a signing-verification key-pair $(\mathsf{vk}_{\mathsf{HS}}, \mathsf{sk}_{\mathsf{HS}}) \leftarrow \mathsf{KeyGen}(1^\lambda)$. Next, sign the symmetric key $\boldsymbol{\sigma}_k \leftarrow \mathsf{Sign}(\mathsf{pp}_{\mathsf{HS}}, \mathsf{sk}_{\mathsf{HS}}, k_{\mathsf{SE}})$ and output
$$k_P = (k_{\mathsf{SE}}, \mathsf{pp}_{\mathsf{HS}}, \mathsf{vk}_{\mathsf{HS}}, \boldsymbol{\sigma}_k) \quad \text{and} \quad k_V = (\mathsf{pp}_{\mathsf{HS}}, \mathsf{vk}_{\mathsf{HS}}, \mathsf{sk}_{\mathsf{HS}}).$$
- $\mathsf{Prove}(k_P, \mathbf{x}, \mathbf{w}) \to \pi$: If $\mathcal{R}(\mathbf{x}, \mathbf{w}) = 0$, output $\perp$. Otherwise, parse $k_P = (k_{\mathsf{SE}}, \mathsf{pp}_{\mathsf{HS}}, \mathsf{vk}_{\mathsf{HS}}, \boldsymbol{\sigma}_k)$. Let $\mathsf{ct} \leftarrow \mathsf{SE.Encrypt}(k_{\mathsf{SE}}, \mathbf{w})$, and $C_{\mathbf{x},\mathsf{ct}}$ be the circuit that computes the function $f_{\mathbf{x},\mathsf{ct}}$ defined above. Compute the signature $\sigma'_{\mathbf{x},\mathsf{ct}} \leftarrow \mathsf{SigEval}(C_{\mathbf{x},\mathsf{ct}}, \mathsf{pp}_{\mathsf{HS}}, k_{\mathsf{SE}}, \boldsymbol{\sigma}_k)$ and then $\sigma^*_{\mathbf{x},\mathsf{ct}} \leftarrow \mathsf{Hide}(\mathsf{vk}_{\mathsf{HS}}, 1, \sigma'_{\mathbf{x},\mathsf{ct}})$. It outputs the proof $\pi = (\mathsf{ct}, \sigma^*_{\mathbf{x},\mathsf{ct}})$.

- $\mathsf{Verify}(k_V, \mathbf{x}, \pi) \to \{0,1\}$: Parse $k_V = (\mathsf{pp}_{\mathsf{HS}}, \mathsf{vk}_{\mathsf{HS}}, \mathsf{sk}_{\mathsf{HS}})$ and $\pi = (\mathsf{ct}, \sigma^*_{\mathbf{x},\mathsf{ct}})$. Let $C_{\mathbf{x},\mathsf{ct}}$ be the circuit that computes $f_{\mathbf{x},\mathsf{ct}}$ defined above. Then, compute $\mathsf{pk}_{\mathbf{x},\mathsf{ct}} \leftarrow \mathsf{PrmsEval}(C_{\mathbf{x},\mathsf{ct}}, \mathsf{pp}_{\mathsf{HS}})$, and output $\mathsf{VerifyHide}(\mathsf{pk}_{\mathbf{x},\mathsf{ct}}, \mathsf{vk}_{\mathsf{HS}}, 1, \sigma^*_{\mathbf{x},\mathsf{ct}})$.

Theorem 3.26 (Preprocessing NIZKs from Homomorphic Signatures)**.** *Let λ be a security parameter and $\mathcal{R}$ be an* NP *relation (and let $\mathcal{L}$ be its corresponding language). Let Π_{NIZK} be the NIZK argument in the preprocessing model from Construction 3.25 (instantiated with a symmetric encryption scheme Π_{enc} and a homomorphic signature scheme Π_{HS}). If Π_{enc} is CPA-secure and Π_{HS} satisfies evaluation correctness (Definition 3.3), hiding correctness (Definition 3.4), selective unforgeability (Definition 3.5, Remark 3.6), and context-hiding against honest signers (Definition 3.8), then Π_{NIZK} is a NIZK argument for $\mathcal{R}$ in the preprocessing model.*

We give the proof of Theorem 3.26 in Section 3.6.1 at the end of this chapter. Combining Construction 3.25 with the homomorphic signature construction Π_{HS} from Construction 3.11 and any LWE-based CPA-secure encryption scheme (Fact 2.6), we have the following corollary.

Corollary 3.27 (Preprocessing NIZKs from Lattices)**.** *Under the LWE assumption, there exists a multi-theorem preprocessing NIZK for* NP*.*

Remark 3.28 (Publicly-Verifiable Preprocessing NIZK)**.** Observe that the verification algorithm in Construction 3.25 does not depend on the signing key $\mathsf{sk}_{\mathsf{HS}}$ of the signature scheme. Thus, we can consider a variant of Construction 3.25 where the verification key does *not* contain $\mathsf{sk}_{\mathsf{HS}}$, and thus, the verification state can be made *public*. This does not compromise soundness because the prover's state already includes the other components of the verification key. However, this publicly-verifiable version of the scheme does not satisfy zero-knowledge according to the strong notion of zero-knowledge in Definition 3.23. This is because without the signing key, the simulator is no longer able to simulate the signatures in the simulated proofs. However, if we consider the weaker notion of zero-knowledge from Remark 3.24 where the simulator chooses the verification key for the preprocessing NIZK, then the publicly-verifiable version of the scheme is provably secure. Notably, when the simulator constructs the verification key, it also chooses (and stores) the signing key for the homomorphic signature scheme. This enables the simulator to simulate signatures when generating the proofs. The resulting construction is a publicly-verifiable preprocessing NIZK (i.e., a "designated-prover" NIZK)

Remark 3.29 (Argument Length Approaching the Witness Size)**.** The proofs in our preprocessing NIZK argument from Construction 3.25 consists of an encryption ct of the witness and a homomorphic signature σ with respect to a circuit C that implements the decryption function of the encryption scheme and the NP relation $\mathcal{R}$. Suppose the relation $\mathcal{R}$ can be implemented by a Boolean circuit of depth d. Using CPA-secure encryption with additive overhead (Fact 2.6), $|\mathsf{ct}| = |\mathbf{w}| + \mathrm{poly}(\lambda)$, where $|\mathbf{w}|$ is the length of a witness to $\mathcal{R}$. If the homomorphic signature is compact (Definition 3.10), then $|\sigma| = \mathrm{poly}(\lambda, d')$ where d' is a bound on the depth of the circuit C. Since the decryption function can be implemented by a circuit of depth $\mathrm{poly}(\lambda)$, we have that $d' = \mathrm{poly}(d, \lambda)$. This means that the overall size of the arguments in our candidate is $|\mathbf{w}| + \mathrm{poly}(\lambda, d)$. The overhead (on top of the NP witness) is *additive* in the security parameter and the *depth* of the NP relation. This is asymptotically shorter than the length of the proofs in NIZK constructions based on trapdoor permutations [FLS90, DDO+01] or pairings [GOS06, Gro10, GOS12], where the dependence is on the *size* of the circuit computing $\mathcal{R}$, and the overhead is *multiplicative* in the security parameter. Thus, our NIZK candidate gives a construction where the argument size approaches the *witness length*. Previously, Gentry et al. [GGI+15] gave a generic way to achieve these asymptotics by combining NIZKs with FHE. The advantage of our approach is that we only rely on lattice assumptions, while the Gentry et al. [GGI+15] compiler additionally assumes the existence of a NIZK scheme (which prior to this work, did not follow from standard lattice assumptions).

Remark 3.30 (Arguments with Common Witness)**.** The proofs in our preprocessing NIZK arguments from Construction 3.25 consists of an encryption of the witness together with a signature. This means that if the prover uses the same *witness* to prove multiple (distinct) statements, then the prover does not need to include a fresh encryption of its witness with every proof. It can send the encrypted witness once and then give multiple signatures with respect to the *same* encrypted witness. In particular, if a prover uses the same witness $\mathbf{w}$ to prove m statements, the total size of the proof is $|\mathbf{w}| + m \cdot \mathrm{poly}(\lambda, d)$, where d is a bound on the depth of the (possibly different) NP relation associated with the m statements. Effectively, the additional overhead of proving multiple statements using a common witness is *independent* of the witness size, and thus, the cost of transmitting the encrypted witness can be *amortized* across multiple proofs. We leverage this observation to implement a *succinct* version of the classic Goldreich-Micali-Wigderson compiler [GMW86, GMW87] in Section 3.5.1. We note that this amortization is also possible if we first apply the FHE-based transformation of Gentry et al. [GGI+15] to any NIZK construction. In our case, our NIZK candidate naturally satisfies this property.

Remark 3.31 (Preprocessing NIZKs from Homomorphic MACs)**.** We note that we can also instantiate Construction 3.25 with a *homomorphic MAC* [GW13, CF13, CFGN14, Cat14] to obtain a multi-theorem preprocessing NIZK. Although the resulting NIZK will not be publicly verifiable (Remark 3.28), a homomorphic MAC is a simpler cryptographic primitive that may be easier to construct, and thus, enable new constructions of multi-theorem preprocessing NIZKs from weaker assumptions. Many existing constructions of homomorphic MACs do not satisfy all of the necessary properties: the lattice-based construction in [GW13] is only secure against adversaries that can make a *bounded* number of verification queries, while the construction based on one-way functions in [CF13] do not provide context-hiding. The construction based on the ℓ-Diffie-Hellman inversion assumption [CF13, CFGN14] gives a context-hiding homomorphic MAC for bounded-degree polynomials (which suffices for verifying NC^1 computations)—here, $\ell = \mathrm{poly}(\lambda)$ is a parameter that scales with the *degree* of the computation being verified. Together with a group-based PRF with evaluation in

NC^1 (e.g., the Naor-Reingold PRF [NR97]), we can use Construction 3.25 to obtain a preprocessing NIZK for general NP languages from the ℓ-Diffie-Hellman inversion assumption.[6] We leave it as an interesting open problem to build context-hiding homomorphic MACs that suffice for preprocessing NIZKs from weaker (and static) cryptographic assumptions (e.g., the DDH assumption).

Remark 3.32 (Preprocessing NIZK Proofs from Extractable Homomorphic Commitments)**.** Construction 3.25 gives a NIZK *argument* in the preprocessing model. This is because in the proof of Theorem 3.26, soundness of the preprocessing NIZK reduces to *computational* unforgeability of the underlying homomorphic signature scheme. We can modify Construction 3.25 to obtain a NIZK *proof* by substituting a context-hiding *statistically-binding* homomorphic commitment in place of the homomorphic signature. This means that the homomorphic commitment scheme satisfies "statistical unforgeability:" a *computationally-unbounded* adversary cannot take a commitment to a message $\mathbf{x}$ and open it to a commitment on any value $y \neq f(\mathbf{x})$ with respect to the function f. Then, the resulting preprocessing NIZK achieves statistical soundness. We can instantiate the statistically-binding homomorphic commitment using the extractable homomorphic trapdoor function from Gorbunov et al. [GVW15b, Appendix B]. The specific construction is a variant of the Gorbunov et al. homomorphic signature scheme (Construction 3.11), where the public verification key $\mathsf{vk} = \mathbf{A}$ is chosen to be a public key of the GSW fully homomorphic encryption [GSW13] scheme.

While homomorphic commitments enable a preprocessing NIZK proof system, it is unclear how to efficiently implement the preprocessing step without relying on general-purpose MPC. In contrast, instantiating Construction 3.25 using homomorphic signatures yields a scheme where the preprocessing can be implemented directly using oblivious transfer (and does not require non-black-box use of the homomorphic signature scheme). For this reason, we focus on preprocessing NIZK arguments from homomorphic signatures in the remainder of this work.

Remark 3.33 (Preprocessing NIZKs from Weaker Assumptions)**.** By definition, any NIZK argument (or proof) system in the CRS model is also a preprocessing NIZK (according to the notion of zero-knowledge from Remark 3.24). In the CRS model (and without random oracles), there are several main families of assumptions known to imply NIZKs: number-theoretic conjectures such as quadratic residuosity [BFM88, DMP87, BDMP91],[7] trapdoor permutations [FLS90, DDO$^+$01, Gro10], pairings [GOS06, GOS12], or indistinguishability obfuscation [SW14]. In the designated-verifier setting, constructions are also known from additively homomorphic encryption [CD04, DFN06, CG15]. A number of works have also studied NIZKs in the preprocessing model, and several constructions have been proposed from one-way functions [DMP88, LS90, Dam92, IKOS07] and oblivious transfer [KMO89]. Since lattice-based assumptions imply one-way functions [Ajt96, Reg05],

[6]For this construction, we require that the NP relation can be implemented by an NC^1 circuit. While any NP relation can be represented as a depth-2 circuit (by including the intermediate wires of the NP circuit as part of the witness), the length of the preprocessing NIZK is now proportional to the *circuit size*, rather than the size of the witness.

[7]Some of these schemes [BFM88, DMP87] are "bounded" in the sense that the prover can only prove a small number of theorems whose total size is bounded by the length of the CRS.

oblivious transfer [PVW08], and additively homomorphic encryption [Reg05], one might think that we can already construct NIZKs in the preprocessing model from standard lattice assumptions. To our knowledge, this is not the case:

- The NIZK constructions of [DMP88, LS90, Dam92] are *single-theorem* NIZKs, and in particular, zero-knowledge does not hold if the prover uses the same proving key to prove multiple statements. In these constructions, the proving key contains secret values, and each proof reveals a subset of the prover's secret values. As a result, the verifier can combine multiple proofs together to learn additional information about each statement than it could have learned had it only seen a single proof. Thus, the constructions in [DMP88, LS90, Dam92] do not directly give a multi-theorem NIZK.

  A natural question to ask is whether we can use the transformation by Feige et al. [FLS90] who showed how to generically boost a NIZK (in the CRS model) with single-theorem zero-knowledge to obtain a NIZK with multi-theorem zero-knowledge. The answer turns out to be negative: the [FLS90] transformation critically relies on the fact that the prover algorithm is publicly computable, or equivalently, that the prover algorithm does not depend on any secrets.[8] This is the case in the CRS model, since the prover algorithm depends only on the CRS, but in the preprocessing model, the prover's algorithm can depend on a (secret) proving key k_P. In the case of [DMP88, LS90, Dam92], the proving key must be kept private for zero-knowledge. Consequently, the preprocessing NIZKs of [DMP88, LS90, Dam92] do not give a general multi-theorem NIZK in the preprocessing model.

- The (preprocessing) NIZK constructions based on oblivious transfer [KMO89], the "MPC-in-the-head" paradigm [IKOS07], and the ones based on homomorphic encryption [CD04, DFN06, CG15] are designated-verifier, and in particular, are vulnerable to the "verifier rejection" problem. Specifically, soundness is compromised if the prover can learn the verifier's response to multiple adaptively-chosen statements and proofs. For instance, in the case of [KMO89], an oblivious transfer protocol is used to hide the verifier's challenge bits; namely, the verifier's challenge message is fixed during the preprocessing, which means the verifier uses the *same* challenge to verify every proof. A prover that has access to a proof-verification oracle is able to reconstruct the verifier's challenge bit-by-bit and compromise soundness of the resulting NIZK construction. A similar approach is taken in the preprocessing NIZK construction of [IKOS07].

From the above discussion, the only candidates of general multi-theorem NIZKs in the preprocessing model are the same as those in the CRS model. Thus, this work provides the first candidate

[8] At a high-level, the proof in [FLS90] proceeds in two steps: first show that single-theorem zero knowledge implies single-theorem witness indistinguishability, and then that single-theorem witness indistinguishability implies multi-theorem witness indistinguishability. The second step relies on a hybrid argument, which requires that it be possible to *publicly* run the prover algorithm. This step does not go through if the prover algorithm takes in a secret state unknown to the verifier.

construction of a multi-theorem NIZK in the preprocessing model from standard lattice assumptions. It remains an open problem to construct multi-theorem NIZKs from standard lattice assumptions in the standard CRS model.

3.4 Blind Homomorphic Signatures

One limitation of preprocessing NIZKs is that we require a trusted setup to generate the proving and verification keys. One solution is to have the prover and the verifier run a (malicious-secure) two-party computation protocol (e.g., [LP07]) to generate the proving and verification keys. However, generic MPC protocols are often costly and require making *non-black-box* use of the underlying homomorphic signature scheme. In this section, we describe how this step can be efficiently implemented using a new primitive called *blind homomorphic signatures*. We formalize our notion in the model of universal composability [Can01], which we review in Section 3.4.1. This has the additional advantage of allowing us to realize the stronger notion of a preprocessing universally-composable NIZK (UC-NIZK) from standard lattice assumptions. We then define the ideal blind homomorphic functionality in the UC framework in Section 3.4.2. Finally, we give our UC-NIZK construction and then describe several applications to boosting the security of MPC in Section 3.5.

3.4.1 The Universal Composability Framework

In this section, we briefly review the universal composability (UC) framework. We refer to [Can01] for the full details. The description here is adapted from the presentation in [MW16, Appendix A] and [GS18, Appendix A]. Readers familiar with UC security can safely skip this section, and we include it only for completeness.

The UC framework. We work in the standard universal composability framework with static corruptions. The UC framework defines an environment $\mathcal{Z}$ (modeled as an efficient algorithm) that is invoked on the security parameter 1^λ and an auxiliary input $z \in \{0,1\}^*$. The environment oversees the protocol execution in one of two possible experiments:

- The *ideal world execution* involves dummy parties $\tilde{P}_1, \ldots, \tilde{P}_n$, an ideal adversary $\mathcal{S}$ (also called a "simulator") who may corrupt some of the dummy parties, and an ideal functionality $\mathcal{F}$.
- The *real world execution* involves parties $P_1, \ldots, P_n$ (modeled as efficient algorithms) and a real-world adversary $\mathcal{A}$ who may corrupt some of the parties.

In both cases, the environment $\mathcal{Z}$ chooses the inputs for the parties, receives the outputs from the uncorrupted parties, and can interact with the real/ideal world adversaries during the protocol execution. At the end of the protocol execution, the environment outputs a bit, which is defined to be the output of the experiment. More precisely, we define the following random variables:

- Let $\text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(1^\lambda, z)$ be the random variable for the output of the environment $\mathcal{Z}$ after interacting with the ideal world execution with adversary $\mathcal{S}$, the functionality $\mathcal{F}$ on security parameter λ and input z. We write $\text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$ to denote the ensemble $\left\{\text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(1^\lambda, z)\right\}_{\lambda\in\mathbb{N}, z\in\{0,1\}^*}$.
- Let $\text{REAL}_{\pi,\mathcal{A},\mathcal{Z}}(1^\lambda, z)$ denote the random variable for the output of the environment $\mathcal{Z}$ after interacting with the real world execution with adversary $\mathcal{A}$ and parties running a protocol π on security parameter λ and input z. We write $\text{REAL}_{\pi,\mathcal{A},\mathcal{Z}}$ to denote the ensemble $\left\{\text{REAL}_{\pi,\mathcal{A},\mathcal{Z}}(1^\lambda, z)\right\}_{\lambda\in\mathbb{N}, z\in\{0,1\}^*}$.

Definition 3.34. Fix $n \in \mathbb{N}$, let $\mathcal{F}$ be an n-ary functionality, and π be an n-party protocol. We say that the protocol π securely realizes $\mathcal{F}$ if for all efficient adversaries $\mathcal{A}$, there exists an ideal adversary $\mathcal{S}$ such that for all efficient environments $\mathcal{Z}$, we have that

$$\text{REAL}_{\pi,\mathcal{A},\mathcal{Z}} \overset{c}{\approx} \text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}.$$

Hybrid protocols. Hybrid protocols are protocols where, in addition to communicating as usual in the standard model of execution, the parties have access to (multiple copies of) an ideal functionality. More precisely, in a protocol execution in the $\mathcal{F}$-hybrid model (where $\mathcal{F}$ denotes an ideal functionality), the parties may give inputs and receive outputs from an unbounded number of copies of $\mathcal{F}$. The different copies of $\mathcal{F}$ are differentiated using a session ID (denoted sid). All of the inputs to each copy of $\mathcal{F}$ and the outputs from each copy of $\mathcal{F}$ have the same session ID. We can correspondingly extend Definition 3.34 to define the notion of a protocol π securely realizing a functionality $\mathcal{G}$ in the $\mathcal{F}$-hybrid model.

The universal composition operation. We now define the universal composition operation and state the universal composition theorem. Let ρ be an $\mathcal{F}$-hybrid protocol, and let Π be a protocol that securely realizes $\mathcal{F}$ (Definition 3.34). The composed protocol ρ^Π is the protocol where each invocation of the ideal functionality $\mathcal{F}$ in ρ is replaced by a fresh invocation of the protocol Π. Specifically, the first message sent to each copy of $\mathcal{F}$ (from any party) is replaced with the first message of Π (generated with the same input and sid associated with the particular copy of $\mathcal{F}$). Each output value generated by a copy of Π is treated as a message received from the corresponding copy of $\mathcal{F}$. Note that if Π is a $\mathcal{G}$-hybrid protocol (where $\mathcal{G}$ is an arbitrary ideal functionality), then ρ^Π is also a $\mathcal{G}$-hybrid protocol.

The universal composition theorem. Let $\mathcal{F}$ be an ideal functionality. In its general form, the universal composition theorem [Can01] states that if Π is a protocol that securely realizes $\mathcal{F}$, then for any $\mathcal{F}$-hybrid protocol ρ that securely realizes $\mathcal{G}$, the composed protocol ρ^Π securely realizes $\mathcal{G}$. We state the formal theorem below:

Theorem 3.35 (Universal Composition [Can01, Corollary 15])**.** *Let $\mathcal{F}, \mathcal{G}$ be ideal functionalities, and let Π be a protocol that securely realizes $\mathcal{F}$. If ρ securely realizes $\mathcal{G}$ in the $\mathcal{F}$-hybrid model, then the composed protocol ρ^{Π} securely realizes $\mathcal{G}$.*

UC functionalities. We now review several UC functionalities: the ideal common reference string (CRS), the oblivious transfer (OT), the zero-knowledge (ZK), and the general UC functionality that we use in this chapter.

The CRS functionality. The common reference string (CRS) functionality $\mathcal{F}_{\text{CRS}}^{\mathcal{D}}$ (parameterized by an efficiently-sampleable distribution $\mathcal{D}$) samples and outputs a string from $\mathcal{D}$. The formal specification from [CR03] is as follows:

Functionality $\mathcal{F}_{\text{CRS}}^{\mathcal{D}}$

The ideal CRS functionality $\mathcal{F}_{\text{CRS}}^{\mathcal{D}}$ is parameterized by an efficiently-sampleable distribution $\mathcal{D}$ and runs with parties $P_1, \ldots, P_n$ and an ideal adversary $\mathcal{S}$. Its behavior is as follows:

- Upon activation with session ID sid, the functionality samples $\sigma \leftarrow \mathcal{D}$ and sends (sid, σ) to the adversary $\mathcal{S}$.
- On receiving sid from a party P_i, send (sid, σ) to P_i.

Figure 3.1: The $\mathcal{F}_{\text{CRS}}^{\mathcal{D}}$ functionality [CR03].

The OT functionality. The oblivious transfer (OT) functionality $\mathcal{F}_{\text{OT}}^{s}$ (parameterized by the message length s) is a two-party functionality between a sender $\mathbf{S}$ and a receiver $\mathbf{R}$. The sender's input consists of two messages $\mathbf{x}_0, \mathbf{x}_1 \in \{0,1\}^s$ and the receiver's input consists of a bit $b \in \{0,1\}$. At the end of the protocol execution, the receiver learns $\mathbf{x}_b$ (and nothing else), and the sender learns nothing. These requirements are captured by the OT functionality $\mathcal{F}_{\text{OT}}^{s}$ from [CLOS02] defined as follows:

Functionality $\mathcal{F}_{\mathrm{OT}}^{s}$

The ideal OT functionality $\mathcal{F}_{\mathrm{OT}}^{s}$ is parameterized by a message length s and runs with a sender $\mathbf{S}$, a receiver $\mathbf{R}$, and an ideal adversary $\mathcal{S}$. Its behavior is as follows:

- Upon receiving a message $(\mathsf{sid}, \mathsf{sender}, \mathbf{x}_0, \mathbf{x}_1)$ from $\mathbf{S}$ where $\mathbf{x}_0, \mathbf{x}_1 \in \{0,1\}^s$, store the tuple $(\mathsf{sid}, \mathbf{x}_0, \mathbf{x}_1)$.
- Upon receiving a message $(\mathsf{sid}, \mathsf{receiver}, b)$ from $\mathbf{R}$, check if a tuple of the form $(\mathsf{sid}, \hat{\mathbf{x}}_0, \hat{\mathbf{x}}_1)$ has been stored for some pair of messages $\hat{\mathbf{x}}_0, \hat{\mathbf{x}}_1$. If so, send $(\mathsf{sid}, \hat{\mathbf{x}}_b)$ to $\mathbf{R}$, sid to the adversary, and halt. If not, send nothing to $\mathbf{R}$, but continue running.

Figure 3.2: The $\mathcal{F}_{\mathrm{OT}}^{s}$ functionality [CLOS02]

For simplicity of notation, we define a block-wise OT functionality $\mathcal{F}_{\mathrm{OT}}^{\ell,s}$ where the sender's input consists of ℓ pairs of messages $\{(\mathbf{x}_{i,0}, \mathbf{x}_{i,1})\}_{i\in[\ell]}$, where each $\mathbf{x}_{i,0}, \mathbf{x}_{i,1} \in \{0,1\}^s$ and the receiver's input consists of ℓ bits $b_1, \ldots, b_\ell \in \{0,1\}$. At the end of the protocol execution, the receiver learns the messages $\mathbf{x}_{1,b_1}, \ldots, \mathbf{x}_{\ell,b_\ell}$ (and nothing else), and the sender learns nothing. The block-wise OT functionality can be securely realized from the standard OT functionality $\mathcal{F}_{\mathrm{OT}}^{s}$ via the universal composition theorem [Can01].

Functionality $\mathcal{F}_{\mathrm{OT}}^{\ell,s}$

The ideal OT functionality $\mathcal{F}_{\mathrm{OT}}^{\ell,s}$ is parameterized by the number of messages ℓ and message length s, and runs with a sender $\mathbf{S}$, a receiver $\mathbf{R}$, and an ideal adversary $\mathcal{S}$. Its behavior is as follows:

- Upon receiving a message $(\mathsf{sid}, \mathsf{sender}, \{(\mathbf{x}_{i,0}, \mathbf{x}_{i,1})\}_{i\in[\ell]})$ from $\mathbf{S}$ where $\mathbf{x}_{i,\beta} \in \{0,1\}^s$ for $i \in [\ell]$, $\beta \in \{0,1\}$, store $\big(\mathsf{sid}, \{(\mathbf{x}_{i,0}, \mathbf{x}_{i,1})\}_{i\in[\ell]}\big)$.
- Upon receiving a message $(\mathsf{sid}, \mathsf{receiver}, (b_1, \ldots, b_\ell))$ from $\mathbf{R}$ for $b_1, \ldots, b_\ell \in \{0,1\}$, check if a tuple of the form $\big(\mathsf{sid}, \{(\hat{\mathbf{x}}_{i,0}, \hat{\mathbf{x}}_{i,1})\}_{i\in[\ell]}\big)$ has been stored for some choice of $\hat{\mathbf{x}}_{i,\beta} \in \{0,1\}^\ell$ where $i \in [\ell]$ and $\beta \in \{0,1\}$. If so, send $(\mathsf{sid}, \{\hat{\mathbf{x}}_{i,b_i}\}_{i\in[\ell]})$ to $\mathbf{R}$, sid to the adversary, and halt. If not, send nothing to $\mathbf{R}$, but continue running.

Figure 3.3: The $\mathcal{F}_{\mathrm{OT}}^{\ell,s}$ functionality.

The ZK functionality. The zero-knowledge (ZK) functionality is a two-party functionality between a prover $\mathcal{P}$ and a verifier $\mathcal{V}$. The prover is able to send the functionality a description of an NP relation $\mathcal{R}$, a statement $\mathbf{x}$ to be proven along with a witness $\mathbf{w}$. The functionality forwards the relation and the statement $\mathbf{x}$ to the verifier if and only if $\mathcal{R}(\mathbf{x}, \mathbf{w}) = 1$. Our definition is inherently multi-theorem; namely, the prover can prove arbitrarily many statements (possibly with respect to different NP relations). We distinguish between different proof sub-sessions by associating a unique sub-session ID ssid with each sub-session. Our definition is adapted from the one given in [CLOS02].

Functionality $\mathcal{F}_{\mathsf{ZK}}$

The ideal ZK functionality runs with a prover $\mathcal{P}$, a verifier $\mathcal{V}$ and an ideal adversary $\mathcal{S}$. Its behavior is as follows:

- Upon receiving a message $(\mathsf{sid}, \mathsf{ssid}, \mathsf{prove}, \mathcal{R}, \mathbf{x}, \mathbf{w})$ from $\mathcal{P}$ where $\mathcal{R}$ is an NP relation, if $\mathcal{R}(\mathbf{x}, \mathbf{w}) = 1$, then send $(\mathsf{sid}, \mathsf{ssid}, \mathsf{proof}, \mathcal{R}, \mathbf{x})$ to $\mathcal{V}$ and $\mathcal{S}$. Otherwise, ignore the message.

Figure 3.4: The $\mathcal{F}_{\mathsf{ZK}}$ functionality.

The general UC functionality. Let $f\colon (\{0,1\}^{\ell_{\text{in}}})^n \to (\{0,1\}^{\ell_{\text{out}}})^n$ be an arbitrary n-input function. The general UC-functionality $\mathcal{F}_f$ is parameterized with a function f and described in Figure 3.5. Our presentation is adapted from that in [GS18].

Functionality $\mathcal{F}_f$

The general UC functionality $\mathcal{F}_f$ is parameterized by a (possibly randomized) function $f : (\{0,1\}^{\ell_{\text{in}}})^n \to (\{0,1\}^{\ell_{\text{out}}})^n$ on n inputs, and runs with parties $\mathcal{P} = (P_1, \ldots, P_n)$, and an ideal adversary $\mathcal{S}$, as follows:

- Each party P_i sends $(\mathsf{sid}, \mathsf{input}, \mathcal{P}, P_i, \mathbf{x}_i)$ where $\mathbf{x}_i \in \{0,1\}^{\ell_{\text{in}}}$ to $\mathcal{F}_f$.
- After receiving inputs from each of the parties, the functionality computes $(\mathbf{y}_1, \ldots, \mathbf{y}_n) \leftarrow f(\mathbf{x}_1, \ldots, \mathbf{x}_n)$. For every party P_i that is corrupted, the functionality sends $\mathcal{S}$ the message $(\mathsf{sid}, \mathsf{output}, \mathcal{P}, P_i, \mathbf{y}_i)$.
- When the functionality receives a message $(\mathsf{sid}, \mathsf{finish}, \mathcal{P}, P_i)$ from $\mathcal{S}$, the ideal functionality sends $(\mathsf{sid}, \mathsf{output}, \mathcal{P}, P_i, \mathbf{y}_i)$ to P_i. The functionality $\mathcal{F}$ ignores the message if inputs from all parties in $\mathcal{P}$ have not been received.

Figure 3.5: The general UC functionality $\mathcal{F}_f$.

3.4.2 The Blind Homomorphic Signature Functionality

We now define the ideal blind homomorphic signature functionality $\mathcal{F}_{\text{BHS}}$. Our definition builds upon existing definitions of the ideal signature functionality $\mathcal{F}_{\text{SIG}}$ by Canetti [Can04] and the ideal blind signature functionality $\mathcal{F}_{\text{BLSIG}}$ by Fischlin [Fis06]. To simplify the presentation, we define the functionality in the two-party setting, where there is a special signing party (denoted $\mathbf{S}$) and a single receiver who obtains the signature (denoted $\mathbf{R}$). While this is a simpler model than the multi-party setting considered in [Can04, Fis06], it suffices for the applications we describe in this work.

Ideal signature functionalities. The $\mathcal{F}_{\text{SIG}}$ functionality from [Can04] essentially provides a "registry service" where a distinguished party (the signer) is able to register message-signature pairs.

Moreover, any party that possesses the verification key can check whether a particular message-signature pair is registered (and thus, constitutes a valid signature). The ideal functionality does not impose any restriction on the structure of the verification key or the legitimate signatures, and allows the adversary to choose those values. In a blind signature scheme, the signing process is replaced by an interactive protocol between the signer and the receiver, and the security requirement is that the signer does not learn the message being signed. To model this, the $\mathcal{F}_{\text{BLSIG}}$ functionality from [Fis06] asks the adversary to provide the description of a *stateless* algorithm $\mathsf{IdealSign}$ in addition to the verification key to the ideal functionality $\mathcal{F}_{\text{BLSIG}}$. For blind signing requests involving an *honest* receiver, the ideal functionality uses $\mathsf{IdealSign}$ to generate the signatures. The message that is signed (i.e., the input to $\mathsf{IdealSign}$) is not disclosed to either the signer or the adversary. This captures the intuitive requirement that the signer does not learn the message that is signed in a blind signature scheme. Conversely, if a corrupt user makes a blind signing request, then the ideal functionality asks the adversary to supply the signature that could result from such a request.

Capturing homomorphic operations. In a homomorphic signature scheme, a user possessing a signature $\boldsymbol{\sigma}$ on a message $\mathbf{x}$ should be able to compute a function g on $\boldsymbol{\sigma}$ to obtain a new signature σ^* on the message $g(\mathbf{x})$. In turn, the verification algorithm checks that σ^* is a valid signature on the value $g(\mathbf{x})$ *and* importantly, that it is a valid signature with respect to the function g. Namely, the signature is bound not only to the computed value $g(\mathbf{x})$ but also to the function g.[9] To extend the ideal signature functionality to support homomorphic operations on signatures, we begin by modifying the ideal functionality to maintain a mapping between *function-message pairs* and signatures (rather than a mapping between messages and signatures). In this case, a fresh signature $\boldsymbol{\sigma}$ (say, output by the blind signing protocol) on a message $\mathbf{x}$ would be viewed as a signature on the function-message pair $(f_{\mathsf{id}}, \mathbf{x})$, where f_{id} here denotes the identity function. Then, if a user subsequently computes a function g on $\boldsymbol{\sigma}$, the resulting signature σ^* should be viewed as a signature on the new pair $(g \circ f_{\mathsf{id}}, g(\mathbf{x})) = (g, g(\mathbf{x}))$. In other words, in a homomorphic signature scheme, signatures are bound to a function-message pair, rather than a single message.

Next, we introduce an additional *signature-evaluation* operation to the ideal functionality. There are several properties we desire from our ideal functionality:

- The ideal signature functionality allows the adversary to decide the structure of the signatures, so it is only natural that the adversary also decides the structure of the signatures output by the signature evaluation procedure.
- Signature evaluation should be compatible with the blind signing process. Specifically, the receiver should be able to compute on a signature it obtained from the blind signing functionality, and moreover, the computation (if requested by an honest receiver) should not reveal to the adversary on which signature or message the computation was performed.

[9] If there is no binding between σ^* and the function g, then we cannot define a meaningful notion of unforgeability.

- The computed signature should also hide the input message. In particular, if the receiver obtains a blind signature on a message $\mathbf{x}$ and later computes a signature σ^* on $g(\mathbf{x})$, the signature σ^* should not reveal the original (blind) message $\mathbf{x}$.

To satisfy these properties, the ideal functionality asks the adversary to additionally provide the description of a *stateless* signature evaluation algorithm $\mathsf{IdealEval}$ (in addition to $\mathsf{IdealSign}$). The ideal functionality uses $\mathsf{IdealEval}$ to generate the signatures when responding to evaluation queries. We capture the third property (that the computed signatures hide the input message to the computation) by setting the inputs to $\mathsf{IdealEval}$ to only include the function g that is computed and the output value of the computation $g(\mathbf{x})$. The input message $\mathbf{x}$ is not provided to $\mathsf{IdealEval}$.

Under our definition, the signature evaluation functionality takes as input a function-message pair $(f_{\mathsf{id}}, \mathbf{x})$, a signature $\boldsymbol{\sigma}$ on $(f_{\mathsf{id}}, \mathbf{x})$ (under the verification key vk of the signature scheme), and a description of a function g (to compute on $\mathbf{x}$). The output is a new signature σ^* on the pair $(g, g(\mathbf{x}))$. That is, σ^* is a signature on the value $g(\mathbf{x})$ with respect to the function g. When the evaluator is honest, the signature on $(g, g(\mathbf{x}))$ is determined by $\mathsf{IdealEval}(g, g(\mathbf{x}))$ (without going through the adversary). As discussed above, $\mathsf{IdealEval}$ only takes as input the function g and the value $g(\mathbf{x})$, and not the input; this means that the computed signature σ^* hides all information about $\mathbf{x}$ other than what is revealed by $g(\mathbf{x})$. When the evaluator is corrupt, the adversary chooses the signature on $(g, g(\mathbf{x}))$, subject to basic consistency requirements.[10] Once an evaluated signature is generated, the functionality registers the new signature σ^* on the pair $(g, g(\mathbf{x}))$. Our definition implicitly requires that homomorphic evaluation be non-interactive. Neither the adversary nor the signer is notified or participates in the protocol.

Preventing selective failures. In our definition, the functionalities $\mathsf{IdealSign}$ and $\mathsf{IdealEval}$ must either output $\perp$ on *all* inputs, or output $\perp$ on *none* of the inputs. This captures the property that a malicious signer cannot mount a *selective failure* attack against an honest receiver, where the function of whether the receiver obtains a signature or not in the blind signing protocol varies depending on its input message. In the case of the blind signing protocol, we do allow a malicious signer to cause the protocol to fail, but this failure event must be *independent* of the receiver's message. We capture this in the ideal functionality by allowing a corrupt signer to dictate whether a blind signing execution completes successfully or not. However, the corrupt signer must decide whether a given protocol invocation succeeds or fails *independently* of the receiver's message.

Simplifications and generalizations. In defining our ideal blind homomorphic signature functionality, we impose several restrictions to simplify the description and analysis. We describe these briefly here, and note how we could extend the functionality to provide additional generality. Note

[10]The adversary is not allowed to re-register a signature that was previously declared invalid (according to the verification functionality) as a valid signature.

that all of the applications we consider (Section 3.5) only require the basic version of the functionality (Figure 3.6), and not its generalized variants.

- **One-time signatures.** The ideal blind homomorphic signature functionality supports blind signing of a *single* message. Namely, the ideal blind signing functionality only responds to the first signing request from the receiver and ignores all subsequent requests. Moreover, the ideal functionality only supports signature evaluation requests after a signature has been successfully issued by the ideal signing functionality. We capture this via a ready flag that is only set at the conclusion of a successful signing operation. We can relax this single-signature restriction, but at the cost of complicating the analysis.

- **Single-hop evaluation.** Our second restriction on the ideal blind homomorphic signature functionality is we only consider "single-hop" homomorphic operations: that is, we only allow homomorphic operations on fresh signatures. In the ideal functionality, we capture this by having the signature evaluation functionality ignore all requests to compute on function-message pairs $(f, \mathbf{x})$ where $f \neq f_{\mathsf{id}}$ is not the identity function. A more general definition would also consider "multi-hop" evaluation where a party can perform arbitrarily many sequential operations on a signature. The reason we present our definition in the simpler single-hop setting is because existing constructions of homomorphic signatures [GVW15b] (which we leverage in our construction) do not support the multi-hop analog of our definition. This is because under our definition, the ideal evaluation functionality essentially combines the homomorphic evaluation with the context-hiding transformation in standard homomorphic signature schemes. The current homomorphic signature candidate [GVW15b] does not support homomorphic computation after performing context-hiding, and so, cannot be used to realize the more general "multi-hop" version of our functionality. For this reason, we give our definition in the single-hop setting.

We give the formal specification of the ideal blind homomorphic signature functionality $\mathcal{F}_{\text{BHS}}$ in Figure 3.6.

3.4.3 Constructing Blind Homomorphic Signatures

In Figure 3.7, we give the formal description of our blind homomorphic signature protocol Π_{BHS} in the $\mathcal{F}_{\text{OT}}^{\ell,s}$-hybrid model (Figure 3.3).[11] Here, we provide a brief overview of the construction. As discussed in Section 3.1, our construction combines homomorphic signatures with any UC-secure oblivious transfer protocol [CLOS02]. The key-generation, signature-verification, and signature-evaluation operations in Π_{BHS} just correspond to running the underlying Π_{HS} algorithms.

[11] For the protocol description and its security proof, we use the vector notation $\mathbf{x}$ to represent the messages (in order to be consistent with the homomorphic signature notation).

Functionality $\mathcal{F}_{\text{BHS}}$

The ideal blind homomorphic signature functionality $\mathcal{F}_{\text{BHS}}$ runs with a signer $\mathbf{S}$, a receiver $\mathbf{R}$, and an ideal adversary $\mathcal{S}$. The functionality is parameterized by a message length ℓ and a function class $\mathcal{H}$. We write f_{id} to denote the identity function.

Key Generation: Upon receiving a value $(\mathsf{sid}, \mathsf{keygen})$ from the signer $\mathbf{S}$, send $(\mathsf{sid}, \mathsf{keygen})$ to the adversary $\mathcal{S}$. After receiving $(\mathsf{sid}, \mathsf{vkey}, \mathsf{vk})$ from $\mathcal{S}$, give $(\mathsf{sid}, \mathsf{vkey}, \mathsf{vk})$ to $\mathbf{S}$ and record vk. Then, initialize an empty list $\mathcal{L}$, and a ready flag (initially unset).

Signature Generation: If a signature-generation request has already been processed, ignore the request. Otherwise, upon receiving a value $(\mathsf{sid}, \mathsf{sign}, \mathsf{vk}, \mathbf{x})$ from the receiver $\mathbf{R}$ (for some message $\mathbf{x} \in \{0,1\}^{\ell}$), send $(\mathsf{sid}, \mathsf{signature})$ to $\mathcal{S}$, and let $(\mathsf{sid}, \mathsf{IdealSign}, \mathsf{IdealEval})$ be the response from $\mathcal{S}$, where $\mathsf{IdealSign}$ and $\mathsf{IdealEval}$ are functions that either output $\perp$ on *all* inputs or on *no* inputs. Record the tuple $(\mathsf{IdealSign}, \mathsf{IdealEval})$. If $\mathbf{S}$ is honest, send $(\mathsf{sid}, \mathsf{signature})$ to $\mathbf{S}$ to notify it that a signature request has taken place. If $\mathbf{S}$ is corrupt, then send $(\mathsf{sid}, \mathsf{sig\text{-}success})$ to $\mathcal{S}$ and let (sid, b) be the response from $\mathcal{S}$. If $b \neq 1$, send $(\mathsf{sid}, \mathsf{signature}, (f_{\mathsf{id}}, \mathbf{x}), \perp)$ to $\mathbf{R}$. Otherwise, proceed as follows:

- If $\mathbf{R}$ is honest, generate $\sigma \leftarrow \mathsf{IdealSign}(\mathbf{x})$, and send $(\mathsf{sid}, \mathsf{signature}, (f_{\mathsf{id}}, \mathbf{x}), \sigma)$ to $\mathbf{R}$.
- If $\mathbf{R}$ is corrupt, send $(\mathsf{sid}, \mathsf{sign}, \mathbf{x})$ to $\mathcal{S}$ to obtain $(\mathsf{sid}, \mathsf{signature}, (f_{\mathsf{id}}, \mathbf{x}), \sigma)$.

If $(\mathsf{vk}, (f_{\mathsf{id}}, \mathbf{x}), \sigma, 0) \in \mathcal{L}$, abort. Otherwise, add $(\mathsf{vk}, (f_{\mathsf{id}}, \mathbf{x}), \sigma, 1)$ to $\mathcal{L}$, and if $\sigma \neq \perp$, set the flag ready.

Signature Verification: Upon receiving an input $(\mathsf{sid}, \mathsf{verify}, \mathsf{vk}', (f, \mathbf{x}), \sigma)$ from a party $\mathbf{P} \in \{\mathbf{S}, \mathbf{R}\}$, proceed as follows:

- *Correctness:* If $f \notin \mathcal{H}$, then set $t = 0$. If $\mathsf{vk} = \mathsf{vk}'$ and $(\mathsf{vk}, (f, \mathbf{x}), \sigma, 1) \in \mathcal{L}$, then set $t = 1$.
- *Unforgeability:* Otherwise, if $\mathsf{vk} = \mathsf{vk}'$, the signer $\mathbf{S}$ has not been corrupted, and there does not exist $(\mathsf{vk}, (f_{\mathsf{id}}, \mathbf{x}'), \sigma', 1) \in \mathcal{L}$ for some $\mathbf{x}'$, σ' where $\mathbf{x} = f(\mathbf{x}')$, then set $t = 0$, and add $(\mathsf{vk}, (f, \mathbf{x}), \sigma, 0)$ to $\mathcal{L}$.
- *Consistency:* Otherwise, if there is already an entry $(\mathsf{vk}', (f, \mathbf{x}), \sigma, t') \in \mathcal{L}$ for some t', set $t = t'$.
- Otherwise, send $(\mathsf{sid}, \mathsf{verify}, \mathsf{vk}', (f, \mathbf{x}), \sigma)$ to the adversary $\mathcal{S}$. After receiving $(\mathsf{sid}, \mathsf{verified}, (f, \mathbf{x}), \sigma, \tau)$ from $\mathcal{S}$, set $t = \tau$ and add $(\mathsf{vk}', (f, \mathbf{x}), \sigma, \tau)$ to $\mathcal{L}$.

Send $(\mathsf{sid}, \mathsf{verified}, (f, \mathbf{x}), \sigma, t)$ to $\mathbf{P}$. If $t = 1$, we say the signature successfully verified.

Signature Evaluation: If the ready flag has not been set, then ignore the request. Otherwise, upon receiving an input $(\mathsf{sid}, \mathsf{eval}, \mathsf{vk}, g, (f, \mathbf{x}), \sigma)$ from a party $\mathbf{P} \in \{\mathbf{S}, \mathbf{R}\}$, ignore the request if $f \neq f_{\mathsf{id}}$. If $f = f_{\mathsf{id}}$, then apply the signature verification procedure to $(\mathsf{sid}, \mathsf{verify}, \mathsf{vk}, (f, \mathbf{x}), \sigma)$, but do *not* forward the output to $\mathbf{P}$. If the signature does not verify, then ignore the request. Otherwise, proceed as follows:

- If $g \notin \mathcal{H}$, then set $\sigma^* = \perp$.
- Otherwise, if $\mathbf{P}$ is honest, compute $\sigma^* \leftarrow \mathsf{IdealEval}(g, g(\mathbf{x}))$.
- Otherwise, if $\mathbf{P}$ is corrupt, send $(\mathsf{sid}, \mathsf{eval}, g, (f, \mathbf{x}), \sigma)$ to $\mathcal{S}$ to obtain $(\mathsf{sid}, \mathsf{signature}, (g, g(\mathbf{x})), \sigma^*)$.

Finally, send $(\mathsf{sid}, \mathsf{signature}, (g, g(\mathbf{x})), \sigma^*)$ to $\mathbf{P}$. If $\sigma^* \neq \perp$ and $(\mathsf{vk}, (g, g(\mathbf{x})), \sigma^*, 0) \in \mathcal{L}$, abort. If $\sigma^* \neq \perp$ and $(\mathsf{vk}, (g, g(\mathbf{x})), \sigma^*, 0) \notin \mathcal{L}$, add $(\mathsf{vk}, (g, g(\mathbf{x})), \sigma^*, 1)$ to $\mathcal{L}$.

Figure 3.6: The $\mathcal{F}_{\text{BHS}}$ functionality.

Protocol Π_{BHS} in the $\mathcal{F}_{\text{OT}}^{\ell,s}$-Hybrid Model

Let λ be a security parameter and $\mathcal{H}$ be a class of functions from $\{0,1\}^\ell$ to $\{0,1\}$. For a parameter $t \in \mathbb{N}$, we define $f_{\mathsf{recon}} \colon \{0,1\}^{t\ell} \to \{0,1\}^\ell$ to be a share-reconstruction function $(\mathbf{w}_1, \ldots, \mathbf{w}_t) \mapsto \bigoplus_{i \in [t]} \mathbf{w}_i$. Let $\Pi_{\mathsf{HS}} = (\mathsf{PrmsGen}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{PrmsEval}, \mathsf{SigEval}, \mathsf{Hide}, \mathsf{Verify}, \mathsf{VerifyFresh}, \mathsf{VerifyHide})$ be a decomposable homomorphic signature scheme with message space $\{0,1\}$, message length ℓ, and function class $\mathcal{H}'$ where $\mathcal{H}'$ contains all functions of the form $f \circ f_{\mathsf{recon}}$ where $f \in \mathcal{H}$. We assume that the signer $\mathbf{S}$ and receiver $\mathbf{R}$ has access to the ideal functionality $\mathcal{F}_{\text{OT}}^{\ell,s}$ where s is the length of the signatures in Π_{HS}.

Key Generation: Upon receiving an input $(\mathsf{sid}, \mathsf{keygen})$, the signer $\mathbf{S}$ computes a set of public parameters $\mathsf{pp} = \left\{\mathsf{pk}_{i,j}\right\}_{i \in [t], j \in [\ell]} \leftarrow \mathsf{PrmsGen}(1^\lambda, 1^{t\ell})$, and a pair of keys $(\mathsf{vk}', \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$. It stores $(\mathsf{sid}, \mathsf{sk})$, sets $\mathsf{vk} = (\mathsf{pp}, \mathsf{vk}')$, and outputs $(\mathsf{sid}, \mathsf{vkey}, \mathsf{vk})$. Finally, the signer initializes the ready flag (initially unset).

Signature Generation: If the signer or receiver has already processed a signature-generation request, then they ignore the request. Otherwise, they proceed as follows:

- **Receiver:** On input $(\mathsf{sid}, \mathsf{sign}, \mathsf{vk}, \mathbf{x})$, where $\mathsf{vk} = (\mathsf{pp}, \mathsf{vk}')$ and $\mathbf{x} \in \{0,1\}^\ell$, the receiver chooses t shares $\mathbf{w}_1, \ldots, \mathbf{w}_t \xleftarrow{\text{R}} \{0,1\}^\ell$ where $\bigoplus_{i \in [t]} \mathbf{w}_i = \mathbf{x}$. Then, for each $i \in [t]$, it sends $\big((\mathsf{sid}, i), \mathsf{receiver}, \mathbf{w}_i\big)$ to $\mathcal{F}_{\text{OT}}^{\ell,s}$. It also initializes the ready flag (initially unset). Note that if vk is not of the form $(\mathsf{pp}, \mathsf{vk}')$ where $\mathsf{pk}' = \left\{\mathsf{pk}_{i,j}\right\}_{i \in [t], j \in [\ell]}$, the receiver outputs $(\mathsf{sid}, \mathsf{signature}, (f_{\mathsf{id}}, \mathbf{x}), \perp)$.
- **Signer:** On input $(\mathsf{sid}, \mathsf{signature})$, the signer generates signatures $\sigma_{i,j}^{\mathsf{pk}} \leftarrow \mathsf{SignPK}(\mathsf{pk}_{i,j}, \mathsf{sk})$ and $\sigma_{i,j,b}^{\mathsf{m}} \leftarrow \mathsf{SignM}(\mathsf{pk}_{i,j}, \mathsf{sk}, b, \sigma_{i,j}^{\mathsf{pk}})$, and sets $\sigma_{i,j,b} = (\sigma_{i,j}^{\mathsf{pk}}, \sigma_{i,j,b}^{\mathsf{m}})$ for all $i \in [t]$, $j \in [\ell]$ and $b \in \{0,1\}$. The signer then sends $\big((\mathsf{sid}, i), \mathsf{sender}, \{(\sigma_{i,j,0}, \sigma_{i,j,1})\}_{j \in [\ell]}\big)$ to $\mathcal{F}_{\text{OT}}^{\ell,s}$. In addition, $\mathbf{S}$ sends the message-independent components $\left\{\sigma_{i,j}^{\mathsf{pk}}\right\}_{i \in [t], j \in [\ell]}$ to $\mathbf{R}$, and sets the ready flag.

Let $\left\{\tilde{\sigma}_{i,j}^{\mathsf{pk}}\right\}_{i \in [t], j \in [\ell]}$ be the message-independent signatures that $\mathbf{R}$ receives from $\mathbf{S}$, and $\{\tilde{\sigma}_{i,j}\}_{i \in [t], j \in [\ell]}$ be the signatures $\mathbf{R}$ receives from the different $\mathcal{F}_{\text{OT}}^{\ell,s}$ invocations. For all $i \in [t]$ and $j \in [\ell]$, the receiver checks that $\mathsf{VerifyFresh}(\mathsf{pk}_{i,j}, \mathsf{vk}', w_{i,j}, \tilde{\sigma}_{i,j}) = 1$, and moreover, that the message-independent component of $\tilde{\sigma}_{i,j}$ matches $\tilde{\sigma}_{i,j}^{\mathsf{pk}}$ it received from the signer. If any check fails, then $\mathbf{R}$ outputs $(\mathsf{sid}, \mathsf{signature}, (f_{\mathsf{id}}, \mathbf{x}), \perp)$. Otherwise, it evaluates $\boldsymbol{\sigma} \leftarrow \mathsf{SigEval}\big(f_{\mathsf{recon}}, \mathsf{pp}, (\mathbf{w}_1, \ldots, \mathbf{w}_t), (\boldsymbol{\sigma}_1, \ldots, \boldsymbol{\sigma}_t)\big)$, where $\boldsymbol{\sigma}_i = (\tilde{\sigma}_{i,1}, \ldots, \tilde{\sigma}_{i,\ell})$ for all $i \in [t]$. The receiver also sets the ready flag and outputs $\big(\mathsf{sid}, \mathsf{signature}, (f_{\mathsf{id}}, \mathbf{x}), \boldsymbol{\sigma}\big)$.

Signature Verification: Upon receiving an input $(\mathsf{sid}, \mathsf{verify}, \mathsf{vk}, (f, \mathbf{x}), \boldsymbol{\sigma})$ where $\mathsf{vk} = (\mathsf{pp}, \mathsf{vk}')$, party $\mathbf{P} \in \{\mathbf{S}, \mathbf{R}\}$ first checks if $f \notin \mathcal{H}$ and sets $t = 0$ if this is the case. Otherwise, it computes $\mathsf{pk}_f \leftarrow \mathsf{PrmsEval}(f \circ f_{\mathsf{recon}}, \mathsf{pp})$. If $f = f_{\mathsf{id}}$, then it sets $t \leftarrow \mathsf{Verify}(\mathsf{pk}_f, \mathsf{vk}', \mathbf{x}, \boldsymbol{\sigma})$, and if $f \neq f_{\mathsf{id}}$, it sets $t \leftarrow \mathsf{VerifyHide}(\mathsf{pk}_f, \mathsf{vk}', \mathbf{x}, \boldsymbol{\sigma})$. It outputs $(\mathsf{sid}, \mathsf{verified}, \mathbf{x}, \boldsymbol{\sigma}, t)$.

Signature Evaluation: If the ready flag has not been set, then ignore the request. Otherwise, upon receiving an input $(\mathsf{sid}, \mathsf{eval}, \mathsf{vk}, g, (f, \mathbf{x}), \boldsymbol{\sigma})$, party $\mathbf{P} \in \{\mathbf{S}, \mathbf{R}\}$ ignores the request if $f \neq f_{\mathsf{id}}$. If $f = f_{\mathsf{id}}$, $\mathbf{P}$ runs the signature-verification procedure on input $(\mathsf{sid}, \mathsf{verify}, \mathsf{vk}, (f, \mathbf{x}), \boldsymbol{\sigma})$ (but does not produce an output). If the signature does not verify, then ignore the request. Otherwise, it parses $\mathsf{vk} = (\mathsf{pp}, \mathsf{vk}')$, computes $\mathsf{pk}_{\mathsf{recon}} \leftarrow \mathsf{PrmsEval}(f_{\mathsf{recon}}, \mathsf{pp})$ and computes $\sigma' \leftarrow \mathsf{SigEval}(g, \mathsf{pk}_{\mathsf{recon}}, \mathbf{x}, \boldsymbol{\sigma})$, and $\sigma^* \leftarrow \mathsf{Hide}(\mathsf{vk}', g(\mathbf{x}), \sigma')$. It outputs $(\mathsf{sid}, \mathsf{signature}, (g, g(\mathbf{x})), \sigma^*)$.

Figure 3.7: The Π_{BHS} protocol.

The blind signing protocol is interactive and relies on OT. Since we use a bitwise homomorphic signature scheme, a signature on an ℓ-bit message consists of ℓ signatures, one for each bit of the message. In the first step of the blind signing protocol, the signer constructs two signatures (one for the bit 0 and one for the bit 1) for each bit position of the message. The receiver then requests the signatures corresponding to the bits of its message using the OT protocol. Intuitively, the OT protocol ensures that the signer does not learn which set of signatures the receiver requested and the receiver only learns a single signature for each bit position. However, this basic scheme is vulnerable to a "selective-failure" attack where the signer strategically generates *invalid* signatures for certain bit positions of the message $\mathbf{x}$. As a result, whether the receiver obtains a valid signature on its entire message becomes *correlated* with its message itself. To prevent this selective-failure attack, we use the standard technique of having the receiver first split its message $\mathbf{x}$ into a number of random shares $\mathbf{w}_1, \ldots, \mathbf{w}_t$ where $\mathbf{x} = \bigoplus_{i \in [t]} \mathbf{w}_i$. Instead of asking for a signature on $\mathbf{x}$ directly, it instead asks for a signature on the shares $\mathbf{w}_1, \ldots, \mathbf{w}_t$. Since the signatures on the shares $\mathbf{w}_1, \ldots, \mathbf{w}_t$ are homomorphic, the receiver can still compute a signature on the original message $\mathbf{x}$ and hence, correctness of signing is preserved. Moreover, as we show in the proof of Theorem 3.36, unless the malicious signer correctly guesses *all* of the shares of $\mathbf{w}_1, \ldots, \mathbf{w}_t$ the receiver chose, the probability that the receiver aborts (due to receiving an invalid signature) is *independent* of $\mathbf{x}$ no matter how the malicious signer generates the signatures. We formally summarize the security properties of Π_{BHS} in the following theorem, but defer its proof to Section 3.6.2 at the end of this chapter.

Theorem 3.36 (Blind Homomorphic Signatures)**.** *Fix a security parameter λ. Define parameters ℓ, t, and s as in Π_{BHS} (Figure 3.7) where $t = \omega(\log \lambda)$. Let $\mathcal{H}$ be a function class over $\{0,1\}^\ell$ and let Π_{HS} be a homomorphic signature scheme for the message space $\{0,1\}$ and function class $\mathcal{H}'$ such that for any function $f \in \mathcal{H}$, we have $f \circ f_{\mathsf{recon}} \in \mathcal{H}'$, where f_{recon} is the share-reconstruction function from Figure 3.7. Suppose that Π_{HS} satisfies correctness (Definitions 3.2, 3.3, and 3.4), unforgeability (Definition 3.5), and context-hiding (Definition 3.9). Then, the protocol Π_{BHS} (when instantiated with Π_{HS}) securely realizes the ideal functionality $\mathcal{F}_{\text{BHS}}$ (Figure 3.6) with respect to function class $\mathcal{H}$ in the presence of (static) malicious adversaries in the $\mathcal{F}_{\text{OT}}^{\ell,s}$-hybrid model.*

Blind homomorphic signatures from LWE. Combining the fully-secure homomorphic signature scheme from Construction 3.17 (based on [GVW15b]) with the lattice-based UC-secure oblivious transfer protocol from [PVW08], we obtain a blind homomorphic signature scheme from standard lattice assumptions. We describe our instantiation below.

Fact 3.37 (Oblivious Transfer from LWE [PVW08])**.** Let λ be a security parameter and define parameters $\ell, s = \text{poly}(\lambda)$. Then, under the LWE assumption, there exists a protocol Π_{OT} that security realizes the ideal OT functionality $\mathcal{F}_{\text{OT}}^{\ell,s}$ (Figure 3.3) in the presence of malicious adversaries in the CRS model (and assuming static corruptions). Moreover, the protocol Π_{OT} is *round-optimal*: it consists of one message from the receiver to the signer and one from the receiver to the signer.

Corollary 3.38 (Blind Homomorphic Signatures from LWE)**.** *Let λ be a security parameter. Then, under the LWE assumption, for all $d = \mathrm{poly}(\lambda)$, there exists a protocol Π'_{BHS} that securely realizes $\mathcal{F}_{\text{BHS}}$ for the class of depth-d Boolean circuits in the presence of malicious adversaries in the CRS model (and assuming static corruptions). Moreover, the protocol Π'_{BHS} satisfies the following properties:*

- *The key-generation, signature-verification, and signature-evaluation protocols are non-interactive.*
- *The signature-generation protocol (i.e., blind signing) is a two-round interactive protocol between the signer and the receiver (one message each way).*
- *The length of a signature is $\mathrm{poly}(\lambda, d)$.*

Proof. Let Π_{BHS} be the protocol from Figure 3.7 instantiated with the homomorphic signature scheme from Construction 3.17. By Theorem 3.36 and Corollary 3.22,[12] protocol Π_{BHS} securely realizes $\mathcal{F}_{\text{BHS}}$ in the $\mathcal{F}_{\text{OT}}^{\ell,s}$-hybrid model, for some $\ell, s = \mathrm{poly}(\lambda)$. We let Π'_{BHS} be the protocol obtained by instantiating the functionality $\mathcal{F}_{\text{OT}}^{\ell,s}$ in Π_{BHS} with the protocol from Fact 3.37. Security of Π'_{BHS} then follows from the universal composition theorem (Theorem 3.35) [Can01]. Key generation, signature verification, and signature evaluation in Π'_{BHS} simply corresponds to invoking the associated functionalities of the underlying homomorphic signature scheme, and thus, are non-interactive. The signature length is also inherited from Π_{HS}. The blind signing protocol reduces to a single invocation of $\mathcal{F}_{\text{OT}}^{\ell,s}$, which by Fact 3.37, can be implemented by just two rounds of interaction. □

Remark 3.39 (Size of CRS in Corollary 3.38)**.** In the lattice-based OT construction of [PVW08], a single CRS can only be used for a bounded number of OTs. The blind signing protocol in Π'_{BHS} from Corollary 3.38 requires $\ell \cdot \mathrm{poly}(\lambda)$ invocations of OT, where ℓ is the message length. Thus, instantiating Π'_{BHS} requires a CRS of length $\mathrm{poly}(\ell, \lambda)$. In our preprocessing UC-NIZK (Section 3.5), $\ell = \mathrm{poly}(\lambda)$, and so a CRS of size $\mathrm{poly}(\lambda)$ suffices to obtain a preprocessing UC-NIZK for general NP languages. It is an open problem to build a lattice-based UC-secure OT protocol in the CRS model with a *reusable* CRS.

3.5 Universally-Composable Preprocessing NIZKs

In this section, we show how to combine blind homomorphic signatures with CPA-secure encryption to obtain UC-NIZKs in the preprocessing model from standard lattice assumptions. We give our protocol Π_{ZK} in the $\mathcal{F}_{\text{BHS}}$-hybrid model in Figure 3.8. Next, we state the formal security theorem and describe how to instantiate it from standard lattice assumptions. We give the proof of Theorem 3.40 in Section 3.6.3 at the end of this chapter.

[12]Note that we are using the fact that hardness of LWE also implies hardness of SIS (with corresponding parameters).

Protocol Π_{ZK} in the $\mathcal{F}_{\text{BHS}}$-Hybrid Model

Let λ be a security parameter and $\Pi_{\mathsf{enc}} = (\mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt})$ be a CPA-secure encryption scheme. We assume that the prover $\mathcal{P}$ and the verifier $\mathcal{V}$ have access to the ideal functionality $\mathcal{F}_{\text{BHS}}$, where $\mathcal{P}$ is the receiver $\mathbf{R}$ and $\mathcal{V}$ is the signer $\mathbf{S}$. For any NP relation $\mathcal{R}$, define the Boolean-valued function $\mathsf{CheckWitness}_{\mathcal{R},\mathsf{ct},\mathbf{x}}$, parameterized by $\mathcal{R}$, a statement $\mathbf{x}$, and a ciphertext ct as follows: on input a secret key sk, $\mathsf{CheckWitness}_{\mathcal{R},\mathsf{ct},\mathbf{x}}(\mathsf{sk})$ outputs 1 if and only if $\mathcal{R}(\mathbf{x}, \mathsf{Decrypt}(\mathsf{sk}, \mathsf{ct})) = 1$, and 0 otherwise. We implicitly assume that $\mathsf{CheckWitness}_{\mathcal{R},\mathsf{ct},\mathbf{x}} \in \mathcal{H}$, where $\mathcal{H}$ is the function class associated with $\mathcal{F}_{\text{BHS}}$.

Preprocessing phase: In the preprocessing phase, the prover and verifier do the following:

1. The verifier sends $(\mathsf{sid}, \mathsf{keygen})$ to $\mathcal{F}_{\text{BHS}}$ and receives in response a verification key vk. The verifier sends vk to the prover. Subsequently, when the verifier receives $(\mathsf{sid}, \mathsf{signature})$ from $\mathcal{F}_{\text{BHS}}$, it sets the ready flag.
2. The prover begins by sampling a secret key $\mathsf{sk} \leftarrow \mathsf{KeyGen}(1^\lambda)$. Then, it requests a signature on sk under vk by sending $(\mathsf{sid}, \mathsf{sign}, \mathsf{vk}, \mathsf{sk})$ to $\mathcal{F}_{\text{BHS}}$. The prover receives a signature σ_{sk} from $\mathcal{F}_{\text{BHS}}$. If $\sigma_{\mathsf{sk}} = \perp$, then the prover aborts.

Prover: On input a tuple $(\mathsf{sid}, \mathsf{ssid}, \mathsf{prove}, \mathcal{R}, \mathbf{x}, \mathbf{w})$ where $\mathcal{R}(\mathbf{x}, \mathbf{w}) = 1$, the prover proceeds as follows:

1. Encrypt the witness $\mathbf{w}$ to obtain a ciphertext $\mathsf{ct} \leftarrow \mathsf{Encrypt}(\mathsf{sk}, \mathbf{w})$.
2. Submit $(\mathsf{sid}, \mathsf{eval}, \mathsf{vk}, \mathsf{CheckWitness}_{\mathcal{R},\mathsf{ct},\mathbf{x}}, (f_{\mathsf{id}}, \mathsf{sk}), \sigma_{\mathsf{sk}})$ to $\mathcal{F}_{\text{BHS}}$ to obtain a signature σ^*.
3. Set $\pi = (\mathsf{ct}, \sigma^*)$ and send $(\mathsf{sid}, \mathsf{ssid}, \mathsf{proof}, \mathcal{R}, \mathbf{x}, \pi)$ to the verifier.

Verifier: When the verifier receives a tuple $(\mathsf{sid}, \mathsf{ssid}, \mathsf{proof}, \mathcal{R}, \mathbf{x}, \pi)$, it ignores the request if the ready flag has not been set. Otherwise, it parses $\pi = (\mathsf{ct}, \sigma)$, and ignores the message if π does not have this form. Otherwise, it submits $(\mathsf{sid}, \mathsf{verify}, \mathsf{vk}, (\mathsf{CheckWitness}_{\mathcal{R},\mathsf{ct},\mathbf{x}}, 1), \sigma)$ to $\mathcal{F}_{\text{BHS}}$. If the signature is valid (i.e., $\mathcal{F}_{\text{BHS}}$ replies with 1), then the verifier accepts and outputs $(\mathsf{sid}, \mathsf{ssid}, \mathsf{proof}, \mathcal{R}, \mathbf{x})$. Otherwise the verifier ignores the message.

Figure 3.8: Preprocessing ZK argument in the $\mathcal{F}_{\text{BHS}}$-hybrid model.

Theorem 3.40 (Preprocessing Zero-Knowledge Arguments)**.** *Let $\Pi_{\mathsf{enc}} = (\mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt})$ be a CPA-secure encryption scheme. Then, the protocol Π_{ZK} in Figure 3.8 (instantiated with Π_{enc}) securely realizes $\mathcal{F}_{\mathsf{ZK}}$ in the presence of (static) malicious adversaries in the $\mathcal{F}_{\text{BHS}}$-hybrid model.*

Corollary 3.41 (Preprocessing UC-NIZKs from LWE)**.** *Let λ be a security parameter. Then, under the LWE assumption, for all $d = \mathrm{poly}(\lambda)$, there exists a protocol Π_{NIZK} that securely realizes $\mathcal{F}_{\mathsf{ZK}}$ in the presence of (static) malicious adversaries in the CRS model for all NP relations $\mathcal{R}$ that can be computed by a circuit of depth at most d. The protocol Π_{NIZK} satisfies the following properties:*

- *The (one-time) preprocessing phase is a two-round protocol between the prover and the verifier.*
- *The prover's and verifier's algorithms are both non-interactive.*
- *If $\mathcal{R}$ is an NP relation, then the length of a proof of membership for the language associated with $\mathcal{R}$ is $m + \mathrm{poly}(\lambda, d)$, where m is the size of the witness associated with $\mathcal{R}$.*

Proof. Fix a depth bound $d = \mathrm{poly}(\lambda)$. First, we can instantiate the CPA-secure encryption scheme

$\Pi_{\mathsf{enc}} = (\mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt})$ in Figure 3.8 from lattices using Fact 2.6. Let d' be a bound on the depth of the circuit that computes the $\mathsf{CheckWitness}_{\mathcal{R},\mathsf{ct},\mathbf{x}}$ function in Figure 3.8. Note that $d' = \mathrm{poly}(\lambda, d)$, since the depth of the relation $\mathcal{R}$ is bounded by d and the depth of the $\mathsf{Decrypt}$ function is $\mathrm{poly}(\lambda)$. By Corollary 3.38, under the LWE assumption, there exists a protocol Π_{BHS} that securely realized $\mathcal{F}_{\text{BHS}}$ for the class of all depth-d' Boolean circuits in the presence of (static) malicious adversaries. The claim then follows by combining Theorem 3.40 with Corollary 3.38 and the universal composition theorem (Theorem 3.35). We now check the additional properties:

- The preprocessing phase corresponds to the blind signing protocol of Π_{BHS}, which is a two-round protocol between the signer and the verifier.
- The prover's algorithm corresponds to signature evaluation while the verifier's algorithm corresponds to signature verification. Both of these are non-interactive in Π_{BHS}.
- The length of a proof for an NP relation $\mathcal{R}$ consists of an encryption of the witness under Π_{enc} (of size $m + \mathrm{poly}(\lambda)$) and a signature under Π_{BHS} (of size $\mathrm{poly}(\lambda, d)$). The total size is bounded by $m + \mathrm{poly}(\lambda, d)$. □

3.5.1 Applications to MPC

In this section, we describe several applications of our preprocessing UC-NIZKs to boosting the security of MPC protocols. First, we show that combining our construction with the round-optimal semi-malicious MPC protocol of Mukherjee-Wichs [MW16] yields a round-optimal malicious-secure MPC protocol where the communication complexity only depends on the size of the inputs/outputs in a *reusable preprocessing* model (Remark 3.44) from lattices. Then, we show that by leveraging the observation in Remark 3.30, we obtain a *succinct* version of the GMW [GMW86, GMW87] compiler from lattice assumptions.

Malicious-secure MPC from lattices. Previously, Mukherjee and Wichs showed how to construct a two-round MPC protocol with UC-security against semi-malicious adversaries from standard lattice assumptions [MW16]. Their protocol has several notable properties, including optimal round complexity and near-optimal communication complexity: namely, the total communication between the parties depends only on the *length* of the parties' inputs and outputs, and *not* on the complexity (i.e., circuit size) of the underlying computation. Achieving this latter property is often referred to as breaking the "circuit-size barrier" for secure computation [BGI16].

The Mukherjee-Wichs construction (as well as its predecessor [AJL$^+$12]) achieve security against semi-malicious adversaries, and rely on general-purpose NIZKs to achieve full security against malicious adversaries without increasing the round complexity. However, since NIZKs are not known to follow from standard lattice assumptions in the CRS model, the security of the malicious-secure protocols cannot be reduced to a single set of hardness assumptions (for instance, we need to combine lattice assumptions with other number-theoretic assumptions).

Using our lattice-based preprocessing NIZKs, we can obtain malicious-secure MPC in a preprocessing model while basing security *exclusively* on standard lattice assumptions. Specifically, in the preprocessing step, the parties would execute the preprocessing protocol of our UC-NIZK construction (Figure 3.8, Corollary 3.41). In the online phase of the protocol, the parties essentially have access to an ideal zero-knowledge functionality, and so, we can apply the same semi-malicious to malicious boosting described in [AJL$^+$12, MW16] to obtain a protocol with full malicious security. The round complexity and communication complexity of the online phase of the protocol is unchanged from that of the Mukherjee-Wichs construction. Moreover, our preprocessing protocol has several appealing properties: it is not only independent of the party's inputs, but it is also (almost) independent of the computation being performed (it depends only *polylogarithmically* on the depth of the online computation). This means that the same preprocessing can in fact be *reused* across many protocol executions, provided that the computations have bounded depth. In fact we can make the preprocessing completely independent of the online computation if we make an additional circular security assumption (c.f., Corollary 3.43). We state our conclusions more precisely below:

Fact 3.42 (MPC from Multi-Key FHE [MW16])**.** Let λ be a security parameter, and $f\colon (\{0,1\}^{\ell_{\text{in}}})^n \to (\{0,1\}^{\ell_{\text{out}}})^n$ be an arbitrary n-input function. Let C_f be the circuit that computes f, and let d_f be its depth. Then, under the LWE assumption, there exists a protocol Π_f that securely realizes $\mathcal{F}_f$ in the presence of (static) *semi-malicious* adversaries in the CRS model and assuming the parties have access to an authenticated broadcast channel. Recall that $\mathcal{F}_f$ is the general UC functionality for computing the function f (Figure 3.5). Moreover, the protocol π_f satisfies the following properties:

- **Optimal round complexity:** The protocol Π_f is a two-round protocol.
- **Low communication complexity:** The total communication complexity of the protocol is $(\ell_{\text{in}} + \ell_{\text{out}}) \cdot \text{poly}(\lambda, n, d_f)$. In other words, the total communication depends only on the security parameter, the length of the inputs, the length of the outputs, and the *depth* of the computation (rather than the size $|C_f|$). Moreover, if we make an additional *circular security* assumption, then the total communication complexity becomes $(\ell_{\text{in}} + \ell_{\text{out}}) \cdot \text{poly}(\lambda, n)$, which is completely *independent* of the complexity of the computation f. This is essentially the best we can hope for from an MPC protocol for $\mathcal{F}_f$.

Corollary 3.43 (Malicious-Secure MPC in the Preprocessing Model from Lattices)**.** *Let λ be a security parameter, and let $f\colon (\{0,1\}^{\ell_{\text{in}}})^n \to (\{0,1\}^{\ell_{\text{out}}})^n$ be an arbitrary n-input function. Let C_f be the circuit that computes f, and let d_f be its depth. Then, under the LWE assumption, there exists a protocol Π_f that securely realizes $\mathcal{F}_f$ in the presence of (static)* malicious *adversaries in the CRS model (and assuming the parties have access to an authenticated broadcast channel). The protocol Π_f*

splits into two sub-protocols: a preprocessing protocol $\Pi_f^{(\mathrm{pre})}$ *and an online protocol* $\Pi_f^{(\mathrm{online})}$ *with the following properties:*

- ***Reusable preprocessing:*** *The total computational and communication complexity of the preprocessing protocol* $\Pi_f^{(\mathrm{pre})}$ *is* $\mathrm{poly}(n, \lambda, \log d_f)$*. Notably, the preprocessing is* independent *of the size of each party's inputs and the overall size* $|C_f|$ *of the computation. Because the preprocessing only depends* logarithmically *on the depth of* C_f *(and not its size), the same precomputation can be* reused *across many* parallel *evaluations of* C_f *(which would increase the size of the computation, but not its depth). Moreover, if we make the additional circular security assumption from Fact 3.42, then the total computational and communication complexity is* $\mathrm{poly}(n, \lambda)$*, and completely independent of the function* f*.*
- ***Optimal online round complexity:*** *The online protocol* $\Pi_f^{(\mathrm{online})}$ *consists of two rounds of communication.*
- ***Low online communication complexity:*** *The total communication complexity of the online protocol* $\Pi_f^{(\mathrm{online})}$ *is* $(\ell_{\mathrm{in}} + \ell_{\mathrm{out}}) \cdot \mathrm{poly}(\lambda, n, d_f)$*. If we make the additional circular security assumption from Fact 3.42, then the total communication complexity of* $\Pi_f^{(\mathrm{online})}$ *is again essentially optimal:* $(\ell_{\mathrm{in}} + \ell_{\mathrm{out}}) \cdot \mathrm{poly}(\lambda, n)$*.*

Proof. Follows by applying the generic semi-malicious-to-malicious compiler of [AJL+12, Appendix E] to the MPC protocol described in Fact 3.42 along with our UC-NIZKs in the preprocessing model from LWE. □

Remark 3.44 (Reusable Preprocessing)**.** A nice property satisfied by our MPC protocol in the preprocessing model is that the preprocessing is *reusable*. Namely, we only have to run the preprocessing protocol once, provided that all of the computations in the online phase can be implemented by circuits of some bounded depth. In fact, if we are willing to make an additional circular security assumption, the preprocessing is entirely independent of the computation. We note that many classic MPC protocols that leverage preprocessing for better online efficiency do not provide reusable preprocessing [Bea91, DPSZ12]. In these cases, the complexity of the preprocessing phase scales with the *size* of the circuit that is computed in the online phase as opposed to the depth (e.g., the classic technique of Beaver multiplication triples [Bea91] requires generating a single triple for every multiplication gate that will be evaluated during the online phase of the protocol). Having a reusable preprocessing protocol enables us to amortize the cost of the preprocessing across many different computations.

Remark 3.45 (Non-Reusable Preprocessing from Weaker Assumptions)**.** An alternative approach to boosting the Mukherjee-Wichs protocol to provide malicious security in the preprocessing model is to use a bounded-theorem preprocessing NIZK, which can in turn be instantiated from one-way functions [DMP88, LS90, Dam92] or oblivious transfer [KMO89]. One drawback of this approach is that the preprocessing is no longer reusable across multiple computations (since each NIZK system can only be used to prove an *a priori* bounded number of statements). As a result, the round complexity and the computational costs of the preprocessing protocol can no longer be amortized across multiple protocol executions. Moreover, it is unclear that the original bounded-theorem NIZK candidates satisfy the stronger property of universal composability. As such, they cannot be directly applied to achieve malicious security of the Mukherjee-Wichs construction in the UC model.

A succinct GMW compiler from lattices. As discussed in Remark 3.30, if a prover wants to prove m statements (each of which can be checked by a circuit of depth at most d) using the *same* witness $\mathbf{w}$, then the total length of all of the arguments will be $|\mathbf{w}| + m \cdot \text{poly}(\lambda, d)$. In particular, the length of the common witness can be *amortized* across many statements. We can leverage this property to obtain a "succinct" version of the classic GMW compiler [GMW86, GMW87] that transforms any MPC protocol Π for some function f in the semi-honest model to a protocol Π' for the same function f in the malicious model. We begin by briefly recalling the "GMW compiler:"

- **Input commitment:** First, the parties commit to their (private) inputs.
- **Coin tossing:** The parties engage in a secure coin-tossing protocol to determine the (secret) randomness each party uses in the protocol execution. At the end of this step, each party has a (private) random string as well as a commitment to every other party's randomness.
- **Protocol emulation:** During the protocol execution, the parties run the semi-honest protocol Π. Whenever the parties send a message, they include a NIZK argument that their message was computed according to the specification of Π on inputs and randomness that are consistent with their committed inputs and randomness.

The NIZK arguments bind each user to following the semi-honest protocol as described. In the UC-model, Canetti et al. [CLOS02] showed an analog of the GMW compiler based on UC-NIZKs.

Our preprocessing NIZKs from lattices gives a new instantiation of the GMW compiler from standard lattice assumptions. Our construction has the appealing property that the communication overhead of the compiler protocol Π' is essentially *independent* of the parties' computational complexity in the semi-honest protocol Π. We give a concrete comparison below:

- Using traditional NIZKs based on trapdoor permutations [FLS90, DDO$^+$01] or pairings [GOS06, GOS12], the total size of the NIZK proofs is proportional to the size of each party's computation. Thus, the communication overhead of Π' compared to the original protocol Π on each round r is $\text{poly}(\lambda, n, |C_r|)$, where λ is the security parameter, n is the number of parties, and C_r is the circuit that checks whether a party's message on round r is consistent with the protocol specification Π as well as the party's committed inputs and randomness.

- In the GMW protocol, each party uses the *same* witness to construct their proofs in each round of the protocol (the witness is their private input and randomness). Thus, using the trick described in Remark 3.30, the parties only have to communicate an encryption of their input and randomness *once* at the beginning of the protocol. Thereafter, on each round r of the protocol execution, the size of each proof is $\text{poly}(\lambda, n, d_r)$, where d_r is a bound on the *depth* of the consistency check circuit C_r defined above. Since d_r can be significantly smaller than C_r, the communication overhead of using our lattice-based preprocessing NIZK to instantiate the GMW compiler can lead to substantial asymptotic savings.

As was also noted in Remark 3.30, a similar savings in communication is also possible by first applying the FHE-based transformation from [GGI$^+$15] to any NIZK construction to obtain a NIZK with the same proof size as that of the construction in Corollary 3.41, and then using the resulting construction to implement the GMW compiler. Compared to this alternative approach, our construction has the advantage that it can be instantiated directly from lattice assumptions (and does not additionally assume the existence of a NIZK). Moreover, our construction is likely more efficient since we do not have to incur the cost of composing FHE decryption with NIZK verification in addition to performing FHE evaluation.

3.6 Proofs from this Chapter

In this section, we give the formal proofs of Theorem 3.26 (Section 3.6.1), Theorem 3.36 (Section 3.6.2), and Theorem 3.6.3 (Section 3.6.3).

3.6.1 Proof of Theorem 3.26

We show completeness, soundness, zero-knowledge separately.

Completeness. Take any statement $\mathbf{x}$ and witness $\mathbf{w}$ where $\mathcal{R}(\mathbf{x}, \mathbf{w}) = 1$. Let $(k_P, k_V) \leftarrow \mathsf{Setup}(1^\lambda)$, where $k_P = (k_{\mathsf{SE}}, \mathsf{pp}_{\mathsf{HS}}, \mathsf{vk}_{\mathsf{HS}}, \boldsymbol{\sigma}_k)$. Take $(\mathsf{ct}, \sigma^*_{\mathbf{x},\mathsf{ct}}) \leftarrow \mathsf{Prove}(k_P, \mathbf{x}, \mathbf{w})$. By correctness of Π_{enc},

$$C_{\mathbf{x},\mathsf{ct}}(k_{\mathsf{SE}}) = \mathcal{R}(\mathbf{x}, \mathsf{SE.Decrypt}(k_{\mathsf{SE}}, \mathsf{ct})) = \mathcal{R}(\mathbf{x}, \mathbf{w}) = 1.$$

Completeness of Π_{NIZK} then follows from evaluation correctness (Definition 3.3) and hiding correctness (Definition 3.4) of Π_{HS}.

Soundness. At a high-level, soundness follows from (selective) unforgeability of Π_{HS} (Definition 3.5, Remark 3.6). An adversary that succeeds in breaking soundness must produce a statement $\mathbf{x} \notin \mathcal{L}$, a ciphertext ct and a signature $\sigma^*_{\mathbf{x},\mathsf{ct}}$ on the message 1 with respect to the function $C_{\mathbf{x},\mathsf{ct}}$. Since $\mathbf{x} \notin \mathcal{L}$, there does not exist any witness $\mathbf{w} \in \{0,1\}^m$ where $\mathcal{R}(\mathbf{x}, \mathbf{w}) = 1$, which means that there are no inputs to $C_{\mathbf{x},\mathsf{ct}}$ where the output is 1. More formally, suppose there is an adversary $\mathcal{A}$ that breaks soundness of Π_{NIZK} with advantage ε. We use $\mathcal{A}$ to construct an adversary that breaks selective unforgeability of $\mathcal{B}$. Algorithm $\mathcal{B}$ works as follows:

1. At the beginning of the selective unforgeability game, algorithm $\mathcal{B}$ generates a secret key $k_{\mathsf{SE}} \leftarrow \mathsf{SE.KeyGen}(1^\lambda)$, and sends k_{SE} to the challenger. The challenger replies with the public parameters $\mathsf{pp}_{\mathsf{HS}}$, the verification key $\mathsf{vk}_{\mathsf{HS}}$ and a signature $\boldsymbol{\sigma}_k$.
2. Algorithm $\mathcal{B}$ sets $k_P = (k_{\mathsf{SE}}, \mathsf{pp}_{\mathsf{HS}}, \mathsf{vk}_{\mathsf{HS}}, \boldsymbol{\sigma}_k)$ and gives k_P to $\mathcal{A}$.
3. Whenever $\mathcal{A}$ makes an oracle query to the verification oracle, algorithm $\mathcal{B}$ answers according to the specification in Construction 3.25. Note that the verification algorithm only depends on $\mathsf{pp}_{\mathsf{HS}}$ and $\mathsf{vk}_{\mathsf{HS}}$, both of which are known to $\mathcal{B}$ (and in fact $\mathcal{A}$). Notably, the secret signing key $\mathsf{sk}_{\mathsf{HS}}$ is not needed to run Verify.
4. At the end of the game, when $\mathcal{A}$ outputs a statement $\mathbf{x}$ and a proof $\pi = (\mathsf{ct}, \sigma^*_{\mathbf{x},\mathsf{ct}})$, algorithm $\mathcal{B}$ gives the circuit $C_{\mathbf{x},\mathsf{ct}}$, the message 1, and the signature $\sigma^*_{\mathbf{x},\mathsf{ct}}$ to the challenger.

By construction, algorithm $\mathcal{B}$ *perfectly* simulates the prover key for $\mathcal{A}$. Thus, with probability ε, algorithm $\mathcal{A}$ outputs $\mathbf{x} \notin \mathcal{L}$ such that $\sigma^*_{\mathbf{x},\mathsf{ct}}$ is a valid signature on the message 1 with respect to the function $C_{\mathbf{x},\mathsf{ct}}$. By definition, $C_{\mathbf{x},\mathsf{ct}}(k_{\mathsf{SE}}) = 0$, so $\sigma^*_{\mathbf{x},\mathsf{ct}}$ is a valid forgery. Soundness follows.

Zero-Knowledge. At a high-level, zero-knowledge follows by CPA-security of the encryption scheme and weak context-hiding of the homomorphic signature scheme. Since Π_{HS} is weak context-hiding (Definition 3.8), there exists an efficient simulator $\mathcal{S}_{\mathsf{ch}}$ that can simulate the signatures output by the Hide algorithm. We use $\mathcal{S}_{\mathsf{ch}}$ to construct the zero-knowledge simulator $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2)$:

- On input the security parameter λ and the verification state $k_V = (\mathsf{pp}_{\mathsf{HS}}, \mathsf{vk}_{\mathsf{HS}}, \mathsf{sk}_{\mathsf{HS}})$ where $\mathsf{pp}_{\mathsf{HS}} = (\mathsf{pk}_1, \dots, \mathsf{pk}_\rho)$, algorithm $\mathcal{S}_1$ samples a secret key $k_{\mathsf{SE}} \leftarrow \mathsf{SE.KeyGen}(1^\lambda)$. Next, it computes $\boldsymbol{\sigma}^{\mathsf{pk}}_k \leftarrow \mathsf{SignPK}(\mathsf{pp}, \mathsf{sk}_{\mathsf{HS}})$, and outputs the state $\tau_V = (k_{\mathsf{SE}}, \boldsymbol{\sigma}^{\mathsf{pk}}_k)$.
- On input the verification state $k_V = (\mathsf{pp}_{\mathsf{HS}}, \mathsf{vk}_{\mathsf{HS}}, \mathsf{sk}_{\mathsf{HS}})$, the simulation state $\tau_V = (k_{\mathsf{SE}}, \boldsymbol{\sigma}^{\mathsf{pk}}_k)$, and a statement $\mathbf{x} \in \{0,1\}^n$, the simulator algorithm $\mathcal{S}_2$ begins by constructing a ciphertext $\mathsf{ct} \leftarrow \mathsf{SE.Encrypt}(k_{\mathsf{SE}}, 0^m)$. Then, it computes $\mathsf{pk}_{\mathbf{x},\mathsf{ct}} \leftarrow \mathsf{PrmsEval}(C_{\mathbf{x},\mathsf{ct}}, \mathsf{pp}_{\mathsf{HS}})$, $\sigma^{\mathsf{pk}}_{\mathbf{x},\mathsf{ct}} \leftarrow \mathsf{SigEvalPK}(C_{\mathbf{x},\mathsf{ct}}, \mathsf{pp}_{\mathsf{HS}}, \boldsymbol{\sigma}^{\mathsf{pk}}_k)$, and finally, it simulates the signature by computing $\sigma^{\mathsf{m}}_{\mathbf{x},\mathsf{ct}} \leftarrow \mathcal{S}_{\mathsf{ch}}(\mathsf{pk}_{\mathbf{x},\mathsf{ct}}, \mathsf{vk}_{\mathsf{HS}}, \mathsf{sk}_{\mathsf{HS}}, 1, \sigma^{\mathsf{pk}}_{\mathbf{x},\mathsf{ct}})$, and outputs the simulated proof $\pi = (\mathsf{ct}, \sigma^*_{\mathbf{x},\mathsf{ct}})$, where $\sigma^*_{\mathbf{x},\mathsf{ct}} = (\sigma^{\mathsf{pk}}_{\mathbf{x},\mathsf{ct}}, \sigma^{\mathsf{m}}_{\mathbf{x},\mathsf{ct}})$.

To complete the proof, we use a hybrid argument:

- Hyb_0: This is the experiment where the adversary has access to $\mathcal{O}_0$, where $\mathcal{O}_0(k_P, \mathbf{x}, \mathbf{w}) := \mathsf{Prove}(k_P, \mathbf{x}, \mathbf{w})$.

- Hyb_1: Same as Hyb_0, except the $\mathsf{Prove}(k_P, \mathbf{x}, \mathbf{w})$ queries are handled as follows:
  1. The challenger first computes $\mathsf{ct} \leftarrow \mathsf{SE.Encrypt}(k_{\mathsf{SE}}, \mathbf{w})$.
  2. Next, it computes the public key $\mathsf{pk}_{\mathbf{x},\mathsf{ct}} \leftarrow \mathsf{PrmsEval}(C_{\mathbf{x},\mathsf{ct}}, \mathsf{pp}_{\mathsf{HS}})$, a public signature component $\sigma^{\mathsf{pk}}_{\mathbf{x},\mathsf{ct}} \leftarrow \mathsf{SigEvalPK}(C_{\mathbf{x},\mathsf{ct}}, \mathsf{pp}_{\mathsf{HS}}, \boldsymbol{\sigma}^{\mathsf{pk}}_k)$, and finally, a simulated signature $\sigma^{\mathsf{m}}_{\mathbf{x},\mathsf{ct}} \leftarrow \mathcal{S}_{\mathsf{ch}}(\mathsf{pk}_{\mathbf{x},\mathsf{ct}}, \mathsf{vk}_{\mathsf{HS}}, \mathsf{sk}_{\mathsf{HS}}, 1, \sigma^{\mathsf{pk}}_{\mathbf{x},\mathsf{ct}})$. Here $\boldsymbol{\sigma}_k = (\boldsymbol{\sigma}^{\mathsf{pk}}_k, \boldsymbol{\sigma}^{\mathsf{m}}_{\mathsf{sk}})$ is the signature on k_{SE} the challenger generated from Setup (and is part of the proving key k_P).
  3. Finally, the challenger responds with $\pi = (\mathsf{ct}, \sigma^*_{\mathbf{x},\mathsf{ct}})$, where $\sigma^*_{\mathbf{x},\mathsf{ct}} = (\sigma^{\mathsf{pk}}_{\mathbf{x},\mathsf{ct}}, \sigma^{\mathsf{m}}_{\mathbf{x},\mathsf{ct}})$.
- Hyb_2: Same as Hyb_1, except the challenger replaces the encryption of $\mathbf{w}$ with an encryption of 0^m when answering the $\mathsf{Prove}(k_P, \mathbf{x}, \mathbf{w})$ queries.
- Hyb_3: This is the experiment where the adversary has access to $\mathcal{O}_1$, where $\mathcal{O}_1(k_V, \tau_V, \mathbf{x}, \mathbf{w}) := \mathcal{S}_2(k_V, \tau_V, \mathbf{x})$.

We now briefly argue that each pair of hybrids are computationally indistinguishable:

- Hybrids Hyb_0 and Hyb_1 are computationally indistinguishable by weak context-hiding security of Π_{HS}. Specifically, if $\mathcal{A}$ is able to distinguish Hyb_0 and Hyb_1, then we can construct an adversary $\mathcal{B}$ that breaks context-hiding as follows:
  1. At the beginning of the game, algorithm $\mathcal{B}$ receives a signing and a verification key $(\mathsf{vk}_{\mathsf{HS}}, \mathsf{sk}_{\mathsf{HS}})$ from the challenger. It then samples parameters $\mathsf{pp}_{\mathsf{HS}} \leftarrow \mathsf{PrmsGen}(1^\lambda, 1^\rho)$, a symmetric key $k_{\mathsf{SE}} \leftarrow \mathsf{SE.KeyGen}(1^\lambda)$ and a signature $\boldsymbol{\sigma}_k \leftarrow \mathsf{Sign}(\mathsf{pp}_{\mathsf{HS}}, \mathsf{sk}_{\mathsf{HS}}, k_{\mathsf{SE}})$. Algorithm $\mathcal{B}$ constructs the verification key $k_V = (\mathsf{pp}_{\mathsf{HS}}, \mathsf{vk}_{\mathsf{HS}}, \mathsf{sk}_{\mathsf{HS}})$ and sends it to $\mathcal{A}$.
  2. When $\mathcal{A}$ makes an oracle query on a pair $(\mathbf{x}, \mathbf{w})$ where $\mathcal{R}(\mathbf{x}, \mathbf{w}) = 1$, algorithm $\mathcal{B}$ simulates the response by first computing $\mathsf{ct} \leftarrow \mathsf{SE.Encrypt}(k_{\mathsf{SE}}, \mathbf{w})$. Next, it computes $\sigma_{\mathbf{x},\mathsf{ct}} \leftarrow \mathsf{SigEval}(C_{\mathbf{x},\mathsf{ct}}, \mathsf{pp}_{\mathsf{HS}}, k_{\mathsf{SE}}, \boldsymbol{\sigma}_k)$ and parses the result as $\sigma_{\mathbf{x},\mathsf{ct}} = (\sigma^{\mathsf{pk}}_{\mathbf{x},\mathsf{ct}}, \sigma'_{\mathbf{x},\mathsf{ct}})$. It also computes $\mathsf{pk}_{\mathbf{x},\mathsf{ct}} \leftarrow \mathsf{PrmsEval}(C_{\mathbf{x},\mathsf{ct}}, \mathsf{pp}_{\mathsf{HS}})$, and sends the public key $\mathsf{pk}_{\mathbf{x},\mathsf{ct}}$, the message 1, and the signature $(\sigma^{\mathsf{pk}}_{\mathbf{x},\mathsf{ct}}, \sigma'_{\mathbf{x},\mathsf{ct}})$ to the context-hiding challenger. The challenger replies with a refreshed signature $\sigma^*_{\mathbf{x},\mathsf{ct}}$. Algorithm $\mathcal{B}$ responds to the query with $(\mathsf{ct}, \sigma^*_{\mathbf{x},\mathsf{ct}})$.
  3. At the end of the experiment, $\mathcal{B}$ outputs whatever $\mathcal{A}$ outputs.

  By construction, if the signatures returned by the context-hiding challenger are generated using the Hide algorithm, then $\mathcal{B}$ perfectly simulates Hyb_0, while if the signatures are generated using the simulator, then $\mathcal{B}$ perfectly simulates Hyb_1. Indistinguishability of the two hybrids thus follows by context-hiding.
- Hybrids Hyb_1 and Hyb_2 are computationally indistinguishable by CPA-security of Π_{enc}. Specifically, the challenger's logic in Hyb_1 and Hyb_2 does not depend on k_{SE}, so we can simulate the two hybrid experiments given access to an encryption oracle. Note that the signature

component $\boldsymbol{\sigma}_k^{\mathsf{pk}}$ needed to respond to queries in Hyb_1 and Hyb_2 is only the *public* component of the signature (and can be generated without knowledge of the actual secret key k_{SE}).

- Hybrids Hyb_2 and Hyb_3 are identical experiments. Namely, the behavior of the challenger in Hyb_2 precisely coincides with the behavior in the experiment where the adversary is given access to the oracle $\mathcal{O}_1(k_V, \tau_V, \mathbf{x}, \mathbf{w}) := \mathcal{S}_2(k_V, \tau_V, \mathbf{x})$.

Since each pair of hybrid experiments are computationally indistinguishable, we conclude that Π_{NIZK} provides zero-knowledge. ☐

3.6.2 Proof of Theorem 3.36

Let $\mathcal{A}$ be a static adversary that interacts with the environment $\mathcal{Z}$, a signer $\mathbf{S}$, and receiver $\mathbf{R}$ running the real protocol Π_{BHS} (Figure 3.7). We construct an ideal world adversary (simulator) $\mathcal{S}$ that interacts with the environment $\mathcal{Z}$, the ideal functionality $\mathcal{F}_{\text{BHS}}$, and dummy parties $\tilde{\mathbf{S}}$, $\tilde{\mathbf{R}}$ such that no environment $\mathcal{Z}$ can distinguish an interaction with $\mathcal{A}$ in the real protocol from one with $\mathcal{S}$ in the ideal world.

We begin by describing the simulator $\mathcal{S}$. At the beginning of the protocol execution, the simulator $\mathcal{S}$ begins by simulating an execution of Π_{BHS} with adversary $\mathcal{A}$. In particular, $\mathcal{S}$ simulates the environment $\mathcal{Z}$, the behavior of the honest parties, as well as the ideal OT functionality $\mathcal{F}_{\text{OT}}^{\ell,s}$ in the simulated protocol execution with $\mathcal{A}$. Algorithm $\mathcal{A}$ begins by declaring which parties it wants to corrupt, and $\mathcal{S}$ corrupts the analogous set of dummy parties in the ideal execution (e.g., if $\mathcal{A}$ corrupts the signer $\mathbf{S}$, then $\mathcal{S}$ corrupts the dummy signer $\tilde{\mathbf{S}}$). The simulation then proceeds as follows.

Simulating the communication with the environment. Whenever the simulator $\mathcal{S}$ receives an input from the environment $\mathcal{Z}$, it forwards the input to $\mathcal{A}$ (as if it came from the environment in the simulated protocol execution). Whenever $\mathcal{A}$ writes a message on its output tape (in the simulated protocol execution), the simulator $\mathcal{S}$ writes the same output on its own output tape (to be read by the environment).

Simulating the key-generation phase. In the key-generation phase, the simulator $\mathcal{S}$ proceeds as follows, depending on whether the signer $\tilde{\mathbf{S}}$ is corrupt:

- *The signer is honest.* When $\mathcal{S}$ receives a value $(\mathsf{sid}, \mathsf{keygen})$ from $\mathcal{F}_{\text{BHS}}$, the simulator generates $\mathsf{pp} \leftarrow \mathsf{PrmsGen}(1^\lambda, 1^{t\ell})$, $(\mathsf{sk}, \mathsf{vk}') \leftarrow \mathsf{KeyGen}(1^\lambda)$, and stores $(\mathsf{sid}, \mathsf{sk})$. It sets $\mathsf{vk} = (\mathsf{pp}, \mathsf{vk}')$, and sends $(\mathsf{sid}, \mathsf{vkey}, \mathsf{vk})$ to $\mathcal{F}_{\text{BHS}}$.

- *The signer is corrupt.* When $\mathcal{Z}$ activates a corrupt signer $\tilde{\mathbf{S}}$ on input $(\mathsf{sid}, \mathsf{keygen})$, $\mathcal{S}$ activates the signer $\mathbf{S}$ with the same input $(\mathsf{sid}, \mathsf{keygen})$ in its simulated copy of Π_{BHS}. Let $(\mathsf{sid}, \mathsf{vkey}, \mathsf{vk})$ be the verification key output by $\mathbf{S}$ (as decided by $\mathcal{A}$). The simulator $\mathcal{S}$ then sends a request

$(\mathsf{sid}, \mathsf{keygen})$ to $\mathcal{F}_{\text{BHS}}$ (on behalf of $\tilde{\mathbf{S}}$), and responds to the key-generation request from $\mathcal{F}_{\text{BHS}}$ with the tuple $(\mathsf{sid}, \mathsf{vkey}, \mathsf{vk})$.

Simulating the signature-generation phase. The simulator $\mathcal{S}$ simulates the signing protocol as follows, depending on whether the signer $\tilde{\mathbf{S}}$ is corrupt:

- *The signer is honest.* We first describe how the simulator $\mathcal{S}$ constructs the ideal algorithms $(\mathsf{IdealSign}, \mathsf{IdealEval})$ when it receives a query $(\mathsf{sid}, \mathsf{signature})$ from $\mathcal{F}_{\text{BHS}}$. Let $\mathsf{vk} = (\mathsf{pp}, \mathsf{vk}')$ and sk be the parameters the simulator sampled in the key-generation phase (since $\tilde{\mathbf{S}}$ is honest, the simulator chose the secret signing key). The simulator then defines the $\mathsf{IdealSign}$ and $\mathsf{IdealEval}$ algorithms (with $\mathsf{pp}, \mathsf{vk}', \mathsf{sk}$ hard-wired) as follows:

  – $\mathsf{IdealSign}(\mathbf{x})$: On input $\mathbf{x} \in \{0,1\}^\ell$:

    1. Sample shares $\mathbf{w}_1, \ldots, \mathbf{w}_t \xleftarrow{\text{R}} \{0,1\}^\ell$ such that $\bigoplus_{i \in [t]} \mathbf{w}_i = \mathbf{x}$.
    2. Generate $(\boldsymbol{\sigma}_1, \ldots, \boldsymbol{\sigma}_t) \leftarrow \mathsf{Sign}\big(\mathsf{pp}, \mathsf{sk}, (\mathbf{w}_1, \ldots, \mathbf{w}_t)\big)$.
    3. Return $\mathsf{SigEval}\big(f_{\mathsf{recon}}, \mathsf{pp}, (\mathbf{w}_1, \ldots, \mathbf{w}_t), (\boldsymbol{\sigma}_1, \ldots, \boldsymbol{\sigma}_t)\big)$.

  – $\mathsf{IdealEval}(g, x)$: On input a function $g \in \mathcal{H}$ and a value $x \in \{0,1\}$:

    1. Compute $\mathsf{pk}_g \leftarrow \mathsf{PrmsEval}(g \circ f_{\mathsf{recon}}, \mathsf{pp})$.
    2. Sign $\sigma \leftarrow \mathsf{Sign}(\mathsf{pk}_g, \mathsf{sk}, x)$.
    3. Return $\mathsf{Hide}(\mathsf{vk}', x, \sigma)$.

  The simulator replies to $\mathcal{F}_{\text{BHS}}$ with $(\mathsf{IdealSign}, \mathsf{IdealEval})$. If the receiver is honest, then this completes the simulation for the signing request. Conversely, if the receiver is corrupt, then the simulator $\mathcal{S}$ proceeds as follows:

  – When $\mathcal{Z}$ activates the receiver $\tilde{\mathbf{R}}$ on input $(\mathsf{sid}, \mathsf{sign}, \mathsf{vk}, \mathbf{x})$, the simulator forwards $(\mathsf{sid}, \mathsf{sign}, \mathsf{vk}, \mathbf{x})$ to $\mathbf{R}$ (which is under the control of $\mathcal{A}$) in the simulated protocol execution (as if it came from $\mathcal{A}$'s environment).

  – After $\mathbf{R}$ sends inputs $\big((\mathsf{sid}, i), \mathsf{receiver}, \mathbf{w}_i\big)$ for all $i \in [t]$ to the ideal OT functionality $\mathcal{F}_{\text{OT}}^{\ell,s}$ in the simulated protocol execution, the simulator computes $\mathbf{x} \leftarrow \bigoplus_{i \in [t]} \mathbf{w}_i$. If this is *not* the first signing request from $\mathbf{R}$, then the simulator ignores the request. Otherwise, the simulator sends $(\mathsf{sid}, \mathsf{sign}, \mathsf{vk}, \mathbf{x})$ to $\mathcal{F}_{\text{BHS}}$.

  – When $\mathcal{F}_{\text{BHS}}$ sends $(\mathsf{sid}, \mathsf{sign}, \mathbf{x})$ to $\mathcal{S}$ to choose the signature on behalf of $\tilde{\mathbf{R}}$, the simulator constructs signatures $\boldsymbol{\sigma}_i \leftarrow \mathsf{Sign}(\mathsf{pp}_i, \mathsf{sk}, \mathbf{w}_i)$ and sends $\big((\mathsf{sid}, i), \boldsymbol{\sigma}_i\big)$ to $\mathbf{R}$ for $i \in [t]$. For the message-independent components of the signatures, $\mathcal{S}$ parses $\boldsymbol{\sigma}_i = (\boldsymbol{\sigma}_i^{\mathsf{pk}}, \boldsymbol{\sigma}_i^{\mathsf{m}})$ for $i \in [t]$, and sends $\left\{\boldsymbol{\sigma}_i^{\mathsf{pk}}\right\}_{i \in [t]}$ to $\mathbf{R}$. The simulator also computes $\boldsymbol{\sigma} \leftarrow \mathsf{SigEval}(f_{\mathsf{recon}}, \mathsf{pp}, (\mathbf{w}_1, \ldots, \mathbf{w}_t), (\boldsymbol{\sigma}_1, \ldots, \boldsymbol{\sigma}_t))$, and sends $(\mathsf{sid}, \mathsf{signature}, (f_{\mathsf{id}}, \mathbf{x}), \boldsymbol{\sigma})$, where $\mathbf{x} = \bigoplus_{i \in [t]} \mathbf{w}_i$, to $\mathcal{F}_{\text{BHS}}$.

- *The signer is corrupt.* If the receiver $\tilde{\mathbf{R}}$ is also corrupt, then $\mathcal{S}$ determines the behavior of $\tilde{\mathbf{S}}$ and $\tilde{\mathbf{R}}$ using $\mathcal{A}$ (who controls the behavior of $\mathbf{S}$ and $\mathbf{R}$ in the simulated protocol execution). Specifically, the simulator proceeds as follows:
  - When the environment activates $\tilde{\mathbf{R}}$ with an input $(\mathsf{sid}, \mathsf{sign}, \mathsf{vk}, \mathbf{x})$, the simulator activates the receiver $\mathbf{R}$ in its simulated protocol execution with the same input.
  - The simulator simulates the ideal OT functionality $\mathcal{F}_{\text{OT}}^{\ell,s}$ in its simulated protocol execution exactly according to the specification of $\mathcal{F}_{\text{OT}}^{\ell,s}$ in Figure 3.3.
  - The simulator echoes any output of $\mathcal{A}$ (to the environment).

  Note that in this case where the signer and receiver are both corrupt, the simulator $\mathcal{S}$ *never* interacts with the ideal functionality. Conversely, if the receiver $\tilde{\mathbf{R}}$ is honest, then the simulator proceeds as follows:
  - When the ideal functionality sends a query $(\mathsf{sid}, \mathsf{signature})$ to $\mathcal{S}$, the simulator needs to respond with a specification of the ideal signing and evaluation functionalities $\mathsf{IdealSign}$ and $\mathsf{IdealEval}$. The simulator starts by performing several basic checks:
    1. The simulator begins by activating the signer $\mathbf{S}$ with the input $(\mathsf{sid}, \mathsf{signature})$ in its simulated execution of the protocol. Let $\big((\mathsf{sid}, i), \mathsf{sender}, \{(\sigma_{i,j,0}, \sigma_{i,j,1})\}_{j \in [\ell]}\big)$ for $i \in [t]$ be the inputs $\mathbf{S}$ sends to $\mathcal{F}_{\text{OT}}^{\ell,s}$, and let $\left\{\sigma_{i,j}^{\mathsf{pk}}\right\}_{i \in [t], j \in [\ell]}$ be the message-independent components $\mathbf{S}$ sends to $\mathbf{R}$ in the simulated protocol execution. Note that in the real protocol execution, the receiver $\mathbf{R}$ only interacts with $\mathcal{F}_{\text{OT}}^{\ell,s}$ and does not send any messages to $\mathbf{S}$ (so $\mathcal{S}$ does not need to simulate any messages on behalf of $\mathbf{R}$).
    2. Let vk be the verification key $\mathcal{S}$ chose during key-generation. The simulator parses the verification key as $\mathsf{vk} = (\mathsf{pp}, \mathsf{vk}')$ where $\mathsf{pp} = \left\{\mathsf{pk}_{i,j}\right\}_{i \in [t], j \in [\ell]}$. If the verification key does not have this structure, then the simulator defines the ideal signing and evaluation functions $\mathsf{IdealSign}$ and $\mathsf{IdealEval}$ to always output $\perp$.
    3. Otherwise, the simulator parses $\sigma_{i,j,b} = (\sigma_{i,j,b}^{\mathsf{pk}}, \sigma_{i,j,b}^{\mathsf{m}})$ for $i \in [t]$, $j \in [\ell]$, $b \in \{0,1\}$. We say that a signature $\sigma_{i,j,b}$ is "valid" if
    $$\sigma_{i,j,b}^{\mathsf{pk}} = \sigma_{i,j}^{\mathsf{pk}} \quad \text{and} \quad \mathsf{VerifyFresh}(\mathsf{pk}_{i,j}, \mathsf{vk}', b, \sigma_{i,j,b}) = 1, \tag{3.1}$$
    and otherwise, we say that $\sigma_{i,j,b}$ is "invalid." Then, if there exists indices $i \in [t]$ and $j \in [\ell]$ where $\sigma_{i,j,0}$ and $\sigma_{i,j,1}$ are both invalid, the simulator defines the signing and evaluation functions $\mathsf{IdealSign}$ and $\mathsf{IdealEval}$ to always output $\perp$.
    4. Finally, the simulator checks if for *all* $j \in [\ell]$, there exists $i \in [t]$ where $\sigma_{i,j,0}$ and $\sigma_{i,j,1}$ are both valid. If this is not the case, then $\mathcal{S}$ defines the ideal signing and evaluation functions $\mathsf{IdealSign}$ and $\mathsf{IdealEval}$ to always output $\perp$.

If all of the checks pass, then there exists i^*, j^* where $\sigma_{i^*,j^*,0}$ and $\sigma_{i^*,j^*,1}$ are both valid. In this case, the simulator uses the context-hiding simulator $\mathcal{S}^{\mathsf{ch}} = (\mathcal{S}^{\mathsf{Ext}}, \mathcal{S}^{\mathsf{Gen}})$ from Definition 3.9 to extract a simulation trapdoor $\mathtt{td} \leftarrow \mathcal{S}^{\mathsf{Ext}}(\mathsf{pk}_{i^*,j^*}, \mathsf{vk}', (0, \sigma_{i^*,j^*,0}), (1, \sigma_{i^*,j^*,1}))$. Then, the simulator defines the functions $(\mathsf{IdealSign}, \mathsf{IdealEval})$ as follows. Note that the public keys pp, the simulation trapdoor td, and the message-independent signature components $\left\{\sigma_{i,j}^{\mathsf{pk}}\right\}_{i \in [t], j \in [\ell]}$ are hard-wired in the description of the algorithms.

- $\mathsf{IdealSign}(\mathbf{x})$: On input $\mathbf{x} \in \{0,1\}^\ell$:
  1. First, the ideal signing algorithm initializes $\mathbf{w}_1, \ldots, \mathbf{w}_t \leftarrow 0^\ell$.
  2. By assumption, for all $i \in [t]$ and $j \in [\ell]$, there is *at least* one $b \in \{0,1\}$ where $\sigma_{i,j,b}$ is valid. Now, for all $i \in [t]$ and $j \in [\ell]$, if there is *exactly* one bit $b \in \{0,1\}$ where $\sigma_{i,j,b}$ is valid, then the simulator sets $w_{i,j} = b$.
  3. For all remaining indices $i \in [t]$ and $j \in [\ell]$ where both $\sigma_{i,j,0}$ and $\sigma_{i,j,1}$ are valid, the simulator samples $w_{i,j} \xleftarrow{\text{R}} \{0,1\}$, subject to the restriction that $\bigoplus_{i \in [t]} \mathbf{w}_i = \mathbf{x}$. Note that this constraint is always satisfiable since for all $j \in [\ell]$, there is at least one $i \in [t]$ where both $\sigma_{i,j,0}$ and $\sigma_{i,j,1}$ are valid by assumption.
  4. Then, for all $i \in [t]$, the algorithm sets $\boldsymbol{\sigma}_i = (\sigma_{i,1,w_{i,1}}, \ldots, \sigma_{i,\ell,w_{i,\ell}})$, and outputs the signature $\mathsf{SigEval}\big(f_{\mathsf{recon}}, \mathsf{pp}, (\mathbf{w}_1, \ldots, \mathbf{w}_t), (\boldsymbol{\sigma}_1, \ldots, \boldsymbol{\sigma}_t)\big)$.
- $\mathsf{IdealEval}(g, x)$: On input a function $g \in \mathcal{H}$, and a value $x \in \{0,1\}$:
  1. Compute $\mathsf{pk}_g \leftarrow \mathsf{PrmsEval}(g \circ f_{\mathsf{recon}}, \mathsf{pp})$.
  2. Compute $\sigma_g^{\mathsf{pk}} \leftarrow \mathsf{SigEvalPK}\big(g \circ f_{\mathsf{recon}}, \mathsf{pp}, (\boldsymbol{\sigma}_1^{\mathsf{pk}}, \ldots, \boldsymbol{\sigma}_t^{\mathsf{pk}})\big)$, where $\boldsymbol{\sigma}_i^{\mathsf{pk}} = (\sigma_{i,1}^{\mathsf{pk}}, \ldots, \sigma_{i,\ell}^{\mathsf{pk}})$.
  3. Return $\mathcal{S}^{\mathsf{Gen}}(\mathsf{pk}_g, \mathsf{vk}', \mathsf{td}, x, \sigma_g^{\mathsf{pk}})$

– When the ideal functionality sends $(\mathsf{sid}, \mathsf{sig\text{-}success})$ to $\mathcal{S}$, the simulator responds as follows. First, let $\{(\sigma_{i,j,0}, \sigma_{i,j,1})\}_{i \in [t], j \in [\ell]}$ be the set of signatures the signer provided to the ideal OT functionality and $\left\{\sigma_{i,j}^{\mathsf{pk}}\right\}_{i \in [t], j \in [\ell]}$ be the set of message-independent public components sent by $\mathbf{S}$ in the simulated protocol execution. As before, we say that $\sigma_{i,j,b}$ is valid if and only if Eq. (3.1) holds. First, if the simulator previously defined $\mathsf{IdealSign}$ and $\mathsf{IdealEval}$ to $\perp$, then it replies with $(\mathsf{sid}, 0)$. Otherwise, let n be the number of indices $i \in [t]$, $j \in [\ell]$, and $b \in \{0,1\}$ where $\sigma_{i,j,b}$ is invalid. Then, with probability $1 - 2^{-n}$, the simulator responds with $(\mathsf{sid}, 0)$. With probability 2^{-n}, the simulator responds with $(\mathsf{sid}, 1)$.

Simulating the signature-verification phase. When the environment activates $\tilde{\mathbf{P}} \in \left\{\tilde{\mathbf{S}}, \tilde{\mathbf{R}}\right\}$ on input $(\mathsf{sid}, \mathsf{verify}, \mathsf{vk}', (f, \mathbf{x}), \boldsymbol{\sigma})$, the simulator $\mathcal{S}$ proceeds as follows:

- If $\tilde{\mathbf{P}}$ is honest and the simulator $\mathcal{S}$ receives a query $(\mathsf{sid}, \mathsf{verify}, \mathsf{vk}', (f, \mathbf{x}), \boldsymbol{\sigma})$ from $\mathcal{F}_{\text{BHS}}$, the simulator first parses $\mathsf{vk}' = (\mathsf{pp}', \mathsf{vk}'')$. It then computes $\mathsf{pk}'_f \leftarrow \mathsf{PrmsEval}(f \circ f_{\mathsf{recon}}, \mathsf{pp}')$ and sets $t \leftarrow \mathsf{VerifyHide}(\mathsf{pk}'_f, \mathsf{vk}'', \mathbf{x}, \boldsymbol{\sigma})$ if $f \neq f_{\mathsf{id}}$, and $t \leftarrow \mathsf{Verify}(\mathsf{pk}'_f, \mathsf{vk}'', \mathbf{x}, \boldsymbol{\sigma})$ if $f = f_{\mathsf{id}}$. It returns $(\mathsf{sid}, \mathsf{verified}, \mathbf{x}, \boldsymbol{\sigma}, t)$ to $\mathcal{F}_{\text{BHS}}$.

- If $\tilde{\mathbf{P}}$ is corrupted, then $\mathcal{S}$ activates the party $\mathbf{P}$ with the input $(\mathsf{sid}, \mathsf{verify}, \mathsf{vk}', (f, \mathbf{x}), \boldsymbol{\sigma})$ in its simulated copy of Π_{BHS}. Let $(\mathsf{sid}, \mathsf{verified}, \mathbf{x}, \boldsymbol{\sigma}, t)$ be the output by $\mathbf{P}$. The simulator forwards $(\mathsf{sid}, \mathsf{verified}, \mathbf{x}, \boldsymbol{\sigma}, t)$ to the environment. Note that the simulator does not interact with the ideal functionality $\mathcal{F}_{\text{BHS}}$ in this case.

Simulating the signature-evaluation phase. When the environment activates $\tilde{\mathbf{P}} \in \left\{\tilde{\mathbf{S}}, \tilde{\mathbf{R}}\right\}$ on an input $(\mathsf{sid}, \mathsf{eval}, \mathsf{vk}, g, (f, \mathbf{x}), \boldsymbol{\sigma})$, where $f = f_{\mathsf{id}}$, the simulator $\mathcal{S}$ proceeds as follows:

- If $\tilde{\mathbf{P}}$ is honest, then $\mathcal{S}$ only needs to simulate the verification request (if asked by the ideal functionality). The simulator responds to the verification request using the procedure described above (for simulating the verification queries).
- If $\tilde{\mathbf{P}}$ is corrupt, then $\mathcal{S}$ activates party $\mathbf{P}$ with the input $(\mathsf{sid}, \mathsf{eval}, \mathsf{vk}, g, (f, \mathbf{x}), \boldsymbol{\sigma})$ in its simulated copy of Π_{BHS}. Let $(\mathsf{sid}, \mathsf{signature}, (g, g(\mathbf{x})), \sigma')$ be the output by $\mathbf{P}$. The simulator forwards $(\mathsf{sid}, \mathsf{signature}, (g, g(\mathbf{x})), \sigma')$ to the environment. Note that the simulator does *not* interact with the ideal functionality $\mathcal{F}_{\text{BHS}}$ in this case.

To complete the proof, we show that no efficient environment $\mathcal{Z}$ can distinguish the output of the real execution with the adversary $\mathcal{A}$ from the output of the ideal execution with the simulator $\mathcal{S}$. Our argument considers several distinct cases, depending on whether the signer and receiver are honest or corrupt.

Lemma 3.46. *If both the signer and the receiver are honest, then for all efficient environments $\mathcal{Z}$, we have that* $\text{IDEAL}_{\mathcal{F}_{\text{BHS}}, \mathcal{S}, \mathcal{Z}} \overset{c}{\approx} \text{REAL}_{\Pi_{\text{BHS}}, \mathcal{A}, \mathcal{Z}}$.

Proof. We proceed via a hybrid argument:

- Hyb_0: This is the real distribution $\text{REAL}_{\Pi_{\text{BHS}}, \mathcal{A}, \mathcal{Z}}$.
- Hyb_1: Same as Hyb_0, except we modify the honest parties' behavior as follows:
  - At the beginning of the experiment, initialize $\mathbf{x}^* \leftarrow \bot$.
  - At the end of a signing request, let $(\mathsf{sid}, \mathsf{signature}, (f_{\mathsf{id}}, \mathbf{x}), \boldsymbol{\sigma})$ be the signature output by the receiver. Update $\mathbf{x}^* \leftarrow \mathbf{x}$. If any party issued a verification request of the form $(\mathsf{sid}, \mathsf{signature}, (f_{\mathsf{id}}, \mathbf{x}), \boldsymbol{\sigma})$ prior to the signing request, then the experiment aborts with output $\bot$.
  - Let vk be the verification key generated by the signer in the key-generation phase. When the environment activates a party on a verification request $(\mathsf{sid}, \mathsf{verify}, \mathsf{vk}', (f, \mathbf{x}), \boldsymbol{\sigma})$ where $\mathsf{vk}' = \mathsf{vk}$ and $\mathbf{x} \neq f(\mathbf{x}^*)$, then the party outputs $(\mathsf{sid}, \mathsf{verified}, (f, \mathbf{x}), \boldsymbol{\sigma}, 0)$. Otherwise, the output is determined as in Hyb_0.
- Hyb_2: This is the ideal distribution $\text{IDEAL}_{\mathcal{F}_{\text{BHS}}, \mathcal{S}, \mathcal{Z}}$.

We now show that the outputs of each pair of consecutive hybrid experiments are computationally indistinguishable.

Claim 3.47. *Suppose Π_{HS} satisfies unforgeability (Definition 3.5). Then, the outputs of* Hyb_0 *and* Hyb_1 *are computationally indistinguishable.*

Proof. Suppose there exists an environment $\mathcal{Z}$ (and an adversary $\mathcal{A}$) such that the outputs of Hyb_0 and Hyb_1 are distinguishable. We use $\mathcal{Z}$ and $\mathcal{A}$ to construct an adversary $\mathcal{B}$ that breaks unforgeability (Definition 3.5) of Π_{HS}. Algorithm $\mathcal{B}$ operates according to the specification of the unforgeability security experiment $\mathsf{Expt}^{\mathsf{uf}}_{\mathcal{A},\Pi_{\mathsf{HS}}}(\lambda)$, and simulates an execution of Hyb_0 or Hyb_1 for the environment $\mathcal{Z}$ (and adversary $\mathcal{A}$). Specifically, $\mathcal{B}$ simulates the behavior of the honest signer and receiver in the protocol execution experiment:

- At the beginning of the unforgeability security game, algorithm $\mathcal{B}$ receives public keys pp and a verification key vk' from the challenger. It also initializes $\mathbf{x}^* \leftarrow \perp$.

- When $\mathcal{Z}$ activates the signer $\mathbf{S}$ to run the key-generation protocol with a query $(\mathsf{sid}, \mathsf{keygen})$, algorithm $\mathcal{B}$ simulates the honest signer's behavior by outputting $(\mathsf{sid}, \mathsf{vkey}, (\mathsf{pp}, \mathsf{vk}'))$.

  By definition of the unforgeability experiment $\mathsf{Expt}^{\mathsf{uf}}_{\mathcal{A},\Pi_{\mathsf{HS}}}(\lambda)$, the unforgeability challenger samples $\mathsf{pp} \leftarrow \mathsf{PrmsGen}(1^\lambda, 1^{t\ell})$, and $(\mathsf{sk}, \mathsf{vk}') \leftarrow \mathsf{KeyGen}(1^\lambda)$. Thus, algorithm $\mathcal{B}$ perfectly simulates the signer's behavior in Hyb_0 and Hyb_1.

- For signing queries, after $\mathcal{Z}$ activates the receiver $\mathbf{R}$ with a tuple $(\mathsf{sid}, \mathsf{sign}, \mathsf{vk}, \mathbf{x})$ and the signer $\mathbf{S}$ with a tuple $(\mathsf{sid}, \mathsf{signature})$, algorithm $\mathcal{B}$ samples $\mathbf{w}_1, \dots, \mathbf{w}_t \xleftarrow{\text{R}} \{0,1\}^\ell$ such that $\bigoplus_{i \in [t]} \mathbf{w}_i = \mathbf{x}$ and submits $(\mathbf{w}_1, \dots, \mathbf{w}_t)$ to the unforgeability challenger to receive $(\boldsymbol{\sigma}_1, \dots, \boldsymbol{\sigma}_t)$. It computes $\boldsymbol{\sigma} \leftarrow \mathsf{SigEval}(f_{\mathsf{recon}}, \mathsf{pp}, (\mathbf{w}_1, \dots, \mathbf{w}_t), (\boldsymbol{\sigma}_1, \dots, \boldsymbol{\sigma}_t))$ and simulates the receiver's output as $(\mathsf{sid}, \mathsf{signature}, (f_{\mathsf{id}}, \mathbf{x}), \boldsymbol{\sigma})$. In addition, $\mathcal{B}$ sets $\mathbf{x}^* \leftarrow \mathbf{x}$.

  In $\mathsf{Expt}^{\mathsf{uf}}_{\mathcal{A},\Pi_{\mathsf{HS}}}(\lambda)$, the challenger computes $(\boldsymbol{\sigma}_1, \dots, \boldsymbol{\sigma}_t) \leftarrow \mathsf{Sign}\big(\mathsf{pp}, \mathsf{sk}, (\mathbf{w}_1, \dots, \mathbf{w}_t)\big)$, exactly as in Hyb_0 and Hyb_1. Thus, $\mathcal{B}$ perfectly simulates the signing queries in Hyb_0 and Hyb_1.

- For verification and evaluation queries, $\mathcal{B}$ implements the same procedure as in Hyb_0 and Hyb_1. None of these queries require knowledge of the secret signing key sk, and thus, can be perfectly simulated by $\mathcal{B}$.

- At any point during the simulation, if $\mathcal{Z}$ activates a party on a verification request of the form $(\mathsf{sid}, \mathsf{verify}, \mathsf{vk}, (f, \mathbf{x}), \boldsymbol{\sigma})$ where $f(\mathbf{x}^*) \neq \mathbf{x}$ and $\boldsymbol{\sigma}$ is a valid signature on $(f, \mathbf{x})$, then $\mathcal{B}$ does the following:

  – If $f = f_{\mathsf{id}}$, then $\mathcal{B}$ computes $\boldsymbol{\sigma}^* \leftarrow \mathsf{Hide}(\mathsf{vk}', \mathbf{x}, \boldsymbol{\sigma})$ and sends the tuple $(f_{\mathsf{recon}}, \mathbf{x}, \boldsymbol{\sigma}^*)$ to the unforgeability challenger as its forgery.

  – Otherwise, $\mathcal{B}$ sends the tuple $(f \circ f_{\mathsf{recon}}, \mathbf{x}, \boldsymbol{\sigma})$ to the unforgeability challenger as its forgery.

Since the only difference between Hyb_0 and Hyb_1 is the additional checks in the signing and verification protocols, if the outputs of Hyb_0 and Hyb_1 are distinguishable with non-negligible advantage ε, then one of the following conditions must hold with probability ε:

- The receiver's output in the signing request is a tuple $(\mathsf{sid}, \mathsf{signature}, (f_{\mathsf{id}}, \mathbf{x}), \boldsymbol{\sigma})$ and a party was activated to run a verification request on the tuple $(\mathsf{sid}, \mathsf{signature}, (f_{\mathsf{id}}, \mathbf{x}), \boldsymbol{\sigma})$ before the signing request. Since $\boldsymbol{\sigma}$ was output by an honest signing request, this means that $\boldsymbol{\sigma}$ is a valid signature on $\mathbf{x}$: namely, that $\mathsf{Verify}(\mathsf{pk}_{\mathsf{recon}}, \mathsf{vk}', \mathbf{x}, \boldsymbol{\sigma}) = 1$, where $\mathsf{pk}_{\mathsf{recon}} \leftarrow \mathsf{PrmsEval}(f_{\mathsf{recon}}, \mathsf{pp})$. Moreover, since the verification request occurred before the signing request, algorithm $\mathcal{B}$ would have submitted the tuple $(f_{\mathsf{recon}}, \mathbf{x}, \boldsymbol{\sigma}^*)$ to the unforgeability challenger where $\boldsymbol{\sigma}^* \leftarrow \mathsf{Hide}(\mathsf{vk}', \mathbf{x}, \boldsymbol{\sigma})$ *before* it made any signing queries to the unforgeability challenger. By hiding correctness, $\boldsymbol{\sigma}^*$ is a valid signature on $\mathbf{x}$ with respect to f_{recon}, and so $\mathcal{B}$ wins the unforgeability game.

- Otherwise, the environment must have activated a party on a verification query of the form $(\mathsf{sid}, \mathsf{verify}, \mathsf{vk}, (f, \mathbf{x}), \boldsymbol{\sigma})$ the successfully verifies in Hyb_0 but not in Hyb_1. First, since the signature $\boldsymbol{\sigma}$ verifies in Hyb_0, this means that $f \circ f_{\mathsf{recon}} \in \mathcal{H}'$ and in particular, that $\mathsf{VerifyHide}(\mathsf{pk}_{f \circ f_{\mathsf{recon}}}, \mathsf{vk}', \mathbf{x}, \boldsymbol{\sigma}) = 1$ where $\mathsf{pp}_{f \circ f_{\mathsf{recon}}} \leftarrow \mathsf{PrmsEval}(f \circ f_{\mathsf{recon}}, \mathsf{pp})$. Now, if the adversary $\mathcal{B}$ made a signing request to the unforgeability challenger on the message $(\mathbf{w}_1, \ldots, \mathbf{w}_t)$, then it would have also set $\mathbf{x}^* = \bigoplus_{i \in [t]} \mathbf{w}_i$. Since $\boldsymbol{\sigma}$ verifies in Hyb_0 but not in Hyb_1, the special condition in Hyb_1 must be satisfied which means

$$(f \circ f_{\mathsf{recon}})(\mathbf{w}_1, \ldots, \mathbf{w}_t) = f(\mathbf{x}^*) \neq \mathbf{x}.$$

  This means that $\boldsymbol{\sigma}$ is a valid signature on $\mathbf{x}$ with respect to the function $f \circ f_{\mathsf{recon}}$, and thus, is a valid forgery. Alternatively, if $\mathcal{B}$ never made a signing request to the unforgeability challenger, then $\boldsymbol{\sigma}$ is trivially a valid forgery.

In both cases, algorithm $\mathcal{B}$ breaks unforgeability of Π_{HS}, so we conclude that $\mathcal{B}$ has advantage ε in the unforgeability game. □

Claim 3.48. *The outputs of hybrids* Hyb_1 *and* Hyb_2 *are identically distributed.*

Proof. We consider the view of the environment $\mathcal{Z}$ in Hyb_1 and Hyb_2 during each phase of the protocol.

- *Key-generation*: For the key-generation phase, the simulator $\mathcal{S}$ in Hyb_2 exactly emulates the generation of pp and $\mathsf{vk} = (\mathsf{sk}, \mathsf{vk}')$ as defined in Hyb_1. Thus, the outputs of the honest parties in the key-generation phase of Hyb_1 and Hyb_2 are identically distributed.

- *Signature-generation*: In Hyb_2, since both $\mathbf{S}$ and $\mathbf{R}$ are honest, the signatures that the receiver obtains from $\mathcal{F}_{\text{BHS}}$ are determined by the ideal algorithm $\mathsf{IdealSign}$ that $\mathcal{S}$ provides to the functionality $\mathcal{F}_{\text{BHS}}$. Since $\mathcal{S}$ defines these algorithms exactly as in the protocol specification of Π_{BHS} using the identically-distributed signing key sk and verification key vk, the resulting signatures in Hyb_1 and Hyb_2 are identically distributed. Moreover, the same abort condition is present in both Hyb_1 and Hyb_2, so whenever an environment issues a query that causes the ideal functionality to abort in Hyb_2, the experiment also aborts in Hyb_1.

- *Signature-verification*: In Hyb_2, the ideal functionality $\mathcal{F}_{\text{BHS}}$ handles the signature verification queries $(\mathsf{sid}, \mathsf{verify}, \mathsf{vk}', (f, \mathbf{x}), \boldsymbol{\sigma})$. We consider the different possibilities:
  - If $f \notin \mathcal{H}$, then $\mathcal{F}_{\text{BHS}}$ always sets the verification bit $t = 0$. In this case, the honest parties in Hyb_1 also sets $t = 0$ according to the protocol specification.
  - Otherwise, if $\mathsf{vk} = \mathsf{vk}'$ and $(\mathsf{vk}, (f, \mathbf{x}), \boldsymbol{\sigma}, 1) \in \mathcal{L}$, then $\mathcal{F}_{\text{BHS}}$ sets $t = 1$. We consider several scenarios depending on how the entry $(\mathsf{vk}, (f, \mathbf{x}), \boldsymbol{\sigma}, 1) \in \mathcal{L}$ was added to $\mathcal{L}$. If $\boldsymbol{\sigma}$ was generated as the result of a signing or a evaluation request, then by correctness of Π_{HS}, the honest party in Hyb_1 also outputs 1. If the entry was added as a result of a previous verification request (which successfully verified), then because the honest party's verification algorithm in Π_{HS} is *deterministic* (and the signature verified previously), the party also outputs 1 in Hyb_1.
  - Otherwise, if $\mathsf{vk} = \mathsf{vk}'$, and there does not exist $(\mathsf{vk}, (f_{\mathsf{id}}, \mathbf{x}'), \boldsymbol{\sigma}', 1) \in \mathcal{L}$ for some $\mathbf{x}', \boldsymbol{\sigma}'$ where $\mathbf{x} = f(\mathbf{x}')$, then $\mathcal{F}_{\text{BHS}}$ sets $t = 0$. This corresponds to a setting where the receiver never makes a signing request on any $\mathbf{x}^* \in \{0,1\}^\ell$ where $\mathbf{x} = f(\mathbf{x}^*)$. This means the condition in Hyb_1 is satisfied, in which case the party's output is $(\mathsf{sid}, \mathsf{verified}, (f, \mathbf{x}'), \boldsymbol{\sigma}', 0)$. This matches the behavior in Hyb_2.
  - Otherwise, if there is already an entry $(\mathsf{vk}', (f, \mathbf{x}), \boldsymbol{\sigma}, t') \in \mathcal{L}$ for some t', the ideal functionality sets $\mathcal{F}_{\text{BHS}}$ sets $t = t'$. In the real protocol execution in Hyb_1, the honest verifier's decision algorithm is deterministic. Hence, if a signature previously verified (resp., failed to verify), it will continue to verify (resp., fail to verify).
  - Finally, if none of the above criterion apply, then the ideal functionality allows the simulator $\mathcal{S}$ to decide the verification response in Hyb_2. By construction, for an honest party, the simulator implements the same logic as that in the actual protocol Π_{BHS}.

  We conclude that the outputs of the honest parties in response to verification queries are identically distributed in Hyb_1 and Hyb_2.

- *Signature-evaluation*: In Hyb_2, since both parties $\mathbf{S}$ and $\mathbf{R}$ are honest, the resulting signatures that a party receives from $\mathcal{F}_{\text{BHS}}$ are fully determined by the ideal algorithm $\mathsf{IdealEval}$ that $\mathcal{S}$ provides to the functionality $\mathcal{F}_{\text{BHS}}$. Since $\mathcal{S}$ implements these algorithms exactly as in the protocol specification of Π_{BHS} using the identically-distributed signing key sk and verification key vk, the signatures output by the evaluation algorithm in Hyb_1 and Hyb_2 are identically distributed. Moreover, by correctness of Π_{HS} and construction of $\mathcal{S}$, the abort condition in $\mathcal{F}_{\text{BHS}}$ for evaluation queries is never triggered. $\square$

Lemma 3.46 now follows by combining Claims 3.47 and 3.48. $\square$

Lemma 3.49. *If the signer is honest and the receiver is corrupt, then for all efficient environments $\mathcal{Z}$, we have that* $\text{IDEAL}_{\mathcal{F}_{\text{BHS}},\mathcal{S},\mathcal{Z}} \overset{c}{\approx} \text{REAL}_{\Pi_{\text{BHS}},\mathcal{A},\mathcal{Z}}$.

Proof. We use a similar hybrid structure as that used in the proof of Lemma 3.46:

- Hyb_0: This is the real distribution $\text{REAL}_{\Pi_{\text{BHS}},\mathcal{A},\mathcal{Z}}$.
- Hyb_1: Same as Hyb_0, except we modify the honest signer's behavior as follows:
  - At the beginning of the experiment, initialize $\mathbf{x}^* \leftarrow \perp$.
  - During a signing request, let $\mathbf{w}_1, \ldots, \mathbf{w}_t$ be the messages $\mathbf{R}$ submits to $\mathcal{F}_{\text{OT}}^{\ell,s}$. Update $\mathbf{x}^* \leftarrow \bigoplus_{i \in [t]} \mathbf{w}_i$. If the environment activated the signer to make a verification request of the form $(\mathsf{sid}, \mathsf{signature}, (f_{\mathsf{id}}, \mathbf{x}^*), \boldsymbol{\sigma})$ where $\boldsymbol{\sigma}$ is a valid signature on $(f_{\mathsf{id}}, \mathbf{x}^*)$ prior to the signing request, then the experiment aborts with output $\perp$.
  - Let vk be the verification key generated by the signer in the key-generation phase. If the environment activates the honest signer on a verification request of the form $(\mathsf{sid}, \mathsf{verify}, \mathsf{vk}', (f, \mathbf{x}), \boldsymbol{\sigma})$ where $\mathsf{vk}' = \mathsf{vk}$ and $\mathbf{x} \neq f(\mathbf{x}^*)$, then the signer's output is set to $(\mathsf{sid}, \mathsf{verified}, (f, \mathbf{x}), \boldsymbol{\sigma}, 0)$. Otherwise, the output is determined as in Hyb_0.
- Hyb_2: This is the ideal distribution $\text{IDEAL}_{\mathcal{F}_{\text{BHS}},\mathcal{S},\mathcal{Z}}$.

Claim 3.50. *Suppose Π_{HS} satisfies unforgeability (Definition 3.5). Then, the outputs of Hyb_0 and Hyb_1 are computationally indistinguishable.*

Proof. Suppose there exists an environment $\mathcal{Z}$ and adversary $\mathcal{A}$ (that corrupts the receiver $\mathbf{R}$) such that the outputs of Hyb_0 and Hyb_1 are distinguishable. We use $\mathcal{Z}$ and $\mathcal{A}$ to construct an algorithm $\mathcal{B}$ that breaks unforgeability of Π_{HS}. In the reduction, algorithm $\mathcal{B}$ simulates the behavior of the honest signer for $\mathcal{Z}$ and $\mathcal{A}$ according to the protocol specification in Hyb_0 and Hyb_1. The overall argument follows a very similar structure as the proof of Claim 3.47, so we only give a sketch of how $\mathcal{B}$ simulates the execution of Hyb_0 and Hyb_1 below:

- As in the proof of Claim 3.47, algorithm $\mathcal{B}$ uses the public keys pp and the verification key vk' from the unforgeability challenger as the signer's verification key $\mathsf{vk} = (\mathsf{pp}, \mathsf{vk}')$.
- To simulate a signing protocol, after the receiver $\mathbf{R}$ (under the direction of $\mathcal{A}$) submits shares $\mathbf{w}_1, \ldots, \mathbf{w}_t \in \{0,1\}^\ell$ to $\mathcal{F}_{\text{OT}}^{\ell,s}$, algorithm $\mathcal{B}$ submits $(\mathbf{w}_1, \ldots, \mathbf{w}_t)$ to the unforgeability challenger to obtain the signatures $(\boldsymbol{\sigma}_1, \ldots, \boldsymbol{\sigma}_t)$, which it uses to simulate the response from $\mathcal{F}_{\text{OT}}^{\ell,s}$.

- Finally, algorithm $\mathcal{B}$ simulates the verification and evaluation queries to the honest signer as described in Hyb_0 and Hyb_1, since these operations only depend on the public parameters.

By an analogous argument to that in the proof of Claim 3.47, algorithm $\mathcal{B}$ correctly simulates the behavior of the honest signer in a protocol execution with $\mathcal{Z}$ and $\mathcal{A}$. Thus, with non-negligible probability, the environment will activate the honest signer on a signing or verification query whose behavior differs between Hyb_0 and Hyb_1. As in the proof of Claim 3.47, if either condition is satisfied, the environment's query enables $\mathcal{B}$ to break unforgeability of the signature scheme. □

Claim 3.51. *The outputs of* Hyb_1 *and* Hyb_2 *are identically distributed.*

Proof. We argue that the view of the environment $\mathcal{Z}$ is identically distributed in Hyb_1 and Hyb_2. The argument follows similarly to that in the proof of Claim 3.48. We sketch the key details below:

- *Key-generation*: The simulator $\mathcal{S}$ (in Hyb_2) implements the key-generation phase exactly according to the specification of the real protocol Π_{BHS} (in Hyb_1).

- *Signature-generation*: In Hyb_1, when the receiver $\mathbf{R}$ (under the direction of $\mathcal{A}$) submits shares $\mathbf{w}_1, \ldots, \mathbf{w}_t \in \{0,1\}^\ell$ to $\mathcal{F}_{\text{OT}}^{\ell,s}$, it receives in response from the $\mathcal{F}_{\text{OT}}^{\ell,s}$ functionality signatures $\boldsymbol{\sigma}_1, \ldots, \boldsymbol{\sigma}_t$ where $(\boldsymbol{\sigma}_1, \ldots, \boldsymbol{\sigma}_t) \leftarrow \mathsf{Sign}(\mathsf{pp}, \mathsf{sk}, (\mathbf{w}_1, \ldots, \mathbf{w}_t))$. This is precisely how $\mathcal{S}$ simulates the signing request for $\mathcal{A}$ in Hyb_2. Let $((f_{\mathsf{id}}, \mathbf{x}), \boldsymbol{\sigma})$ be the message-signature pair that the simulator $\mathcal{S}$ registers with the ideal functionality $\mathcal{F}_{\text{BHS}}$ at the end of the signing request in Hyb_2. If this pair is already registered with $\mathcal{F}_{\text{BHS}}$ as an invalid signature, then $\mathcal{F}_{\text{BHS}}$ aborts and the protocol execution halts in Hyb_2. By definition of Hyb_2 and $\mathcal{S}$, this is only possible if the environment activates the honest signer to make a verification request on the message-signature pair $((f_{\mathsf{id}}, \mathbf{x}), \boldsymbol{\sigma})$ prior to the signing request. This coincides with the abort condition in Hyb_1, and so we conclude that the output of the signature-generation phase in Hyb_1 and Hyb_2 is identically distributed.

- *Signature-verification*: Signature verification is a non-interactive procedure, so it suffices to argue that the outputs of the honest signer in response to the environment's queries are identically distributed in Hyb_1 and Hyb_2. By construction of $\mathcal{S}$, only verification and evaluation queries involving an *honest* party requires interacting with the ideal functionality. The argument then proceeds as in the proof of Claim 3.48.

- *Signature-evaluation*: Similar to the case of signature verification, signature evaluation is non-interactive, so it suffices to argue that the outputs of the honest signer in response to the environment's queries are identically distributed. This argument then proceeds as in the proof of Claim 3.48. □

Combining Claims 3.50 and 3.51, the lemma follows. □

Lemma 3.52. *If the signer is corrupt and the receiver is honest, then for all efficient environments $\mathcal{Z}$, we have that* $\text{IDEAL}_{\mathcal{F}_{\text{BHS}},\mathcal{S},\mathcal{Z}} \overset{c}{\approx} \text{REAL}_{\Pi_{\text{BHS}},\mathcal{A},\mathcal{Z}}$.

Proof. We proceed via a hybrid argument:

- Hyb_0: This is the real distribution $\text{REAL}_{\Pi_{\text{BHS}},\mathcal{A},\mathcal{Z}}$.
- Hyb_1: Same as Hyb_0 except we modify the honest receiver's behavior in the signature-generation protocol as follows. Let vk be the verification key chosen by the signer in the key-generation phase. Let $\big((\mathsf{sid}, i), \mathsf{sender}, \{(\sigma_{i,j,0}, \sigma_{i,j,1})\}_{i\in[t],j\in[\ell]}\big)$ be the set of signatures the signer submits to the ideal OT functionality $\mathcal{F}_{\text{OT}}^{\ell,s}$, and let $\left\{\sigma_{i,j}^{\mathsf{pk}}\right\}_{i\in[t],j\in[\ell]}$ be the message-independent signature components $\mathbf{S}$ sends to $\mathbf{R}$. The receiver *always* outputs $(\mathsf{sid}, \mathsf{signature}, (f_{\mathsf{id}}, \mathbf{x}), \perp)$ if any of the following conditions hold:
  - The signer's verification key vk cannot be written as $(\mathsf{pp}, \mathsf{vk}')$ where $\mathsf{pp} = \{\mathsf{pk}_{i,j}\}_{i\in[t],j\in[\ell]}$.
  - If there exists indices $i \in [t]$ and $j \in [\ell]$ where both $\sigma_{i,j,0}$ and $\sigma_{i,j,1}$ are invalid. We say that a signature $\sigma_{i,j,b}$ is valid if it satisfies Eq. (3.1).
  - If there exists $j \in [\ell]$ such that for all $i \in [t]$, at least one of $\sigma_{i,j,0}$ and $\sigma_{i,j,1}$ is invalid.

  Otherwise, the honest receiver implements the verification protocol as in the real scheme.
- Hyb_2: Same as Hyb_1 except we use the context-hiding simulator $\mathcal{S} = (\mathcal{S}^{\mathsf{Ext}}, \mathcal{S}^{\mathsf{Gen}})$ to generate the signatures the honest receiver $\mathbf{R}$ outputs on evaluation queries. Here, we assume that none of the conditions from Hyb_1 are satisfied (otherwise, the honest receiver outputs $\perp$ in the signing protocol and ignores all evaluation requests). In particular, we have the following:
  - Let $\big((\mathsf{sid}, i), \mathsf{sender}, \{(\sigma_{i,j,0}, \sigma_{i,j,1})\}_{i\in[t],j\in[\ell]}\big)$ be the set of signatures the signer submits to the ideal OT functionality $\mathcal{F}_{\text{OT}}^{\ell,s}$, and let $\left\{\sigma_{i,j}^{\mathsf{pk}}\right\}_{i\in[t],j\in[\ell]}$ be the message-independent signature components $\mathbf{S}$ sends to $\mathbf{R}$.
  - Since none of the conditions in Hyb_1 are satisfied, there exist indices i^*, j^* where $\sigma_{i^*,j^*,0}$ and $\sigma_{i^*,j^*,1}$ are both valid. Moreover, the verification key vk can be written as $\mathsf{vk} = (\mathsf{pp}, \mathsf{vk}')$ where $\mathsf{pp} = \{\mathsf{pk}_{i,j}\}_{i\in[t],j\in[\ell]}$. The experiment invokes the context-hiding simulator $\mathcal{S}^{\mathsf{Ext}}$ to extract a simulation trapdoor $\mathsf{td} \leftarrow \mathcal{S}^{\mathsf{Ext}}(\mathsf{vk}', (0, \sigma_{i^*,j^*,0}), (1, \sigma_{i^*,j^*,1}))$, and stores td. The receiver's signature is constructed using the same procedure from Hyb_1.
  - During signature evaluation, on input $(\mathsf{sid}, \mathsf{eval}, \mathsf{vk}, g, (f, \mathbf{x}), \boldsymbol{\sigma})$, $\mathbf{R}$ first applies the signature verification procedure on input $(\mathsf{sid}, \mathsf{verify}, \mathsf{vk}, (f, \mathbf{x}), \boldsymbol{\sigma})$. If the signature verifies, the receiver's signature is generated by computing $\mathsf{pk}_g \leftarrow \mathsf{PrmsEval}(g \circ f_{\mathsf{recon}}, \mathsf{pp})$, $\sigma_g^{\mathsf{pk}} \leftarrow \mathsf{SigEvalPK}\big(g \circ f_{\mathsf{recon}}, \mathsf{pp}, (\boldsymbol{\sigma}_1^{\mathsf{pk}}, \dots, \boldsymbol{\sigma}_t^{\mathsf{pk}})\big)$, where $\boldsymbol{\sigma}_i^{\mathsf{pk}} = (\sigma_{i,1}^{\mathsf{pk}}, \dots, \sigma_{i,\ell}^{\mathsf{pk}})$, and finally $\sigma^* \leftarrow \mathcal{S}^{\mathsf{Gen}}(\mathsf{pk}_g, \mathsf{vk}', \mathsf{td}, g(\mathbf{x}), \sigma_g^{\mathsf{pk}})$. The receiver's output is the tuple $(\mathsf{sid}, \mathsf{signature}, (g, g(\mathbf{x})), \sigma^*)$.
- Hyb_3: This is the ideal distribution $\text{IDEAL}_{\mathcal{F}_{\text{BHS}},\mathcal{S},\mathcal{Z}}$.

Claim 3.53. *Suppose $t = \omega(\log \lambda)$. Then, the outputs of hybrids Hyb_0 and Hyb_1 are statistically indistinguishable.*

Proof. The only difference between the two experiments is the additional checks in Hyb_1 which affects the honest receiver's output on signing queries. We consider each of the conditions separately, and argue that for each of them, the receiver's output in Hyb_1 is the same as that in Hyb_0, except with probability at most $2^{-(t-1)} = 2^{-\omega(\log \lambda)} = \operatorname{negl}(\lambda)$.

- Suppose that the signer's verification key is not well-formed: namely, that $\mathsf{vk} \neq (\mathsf{pp}, \mathsf{vk}')$ where $\mathsf{pp} = \left\{\mathsf{pk}_{i,j}\right\}_{i \in [t], j \in [\ell]}$. In this case, the receiver's signature is $\perp$ in both Hyb_0 and Hyb_1.
- Suppose there exists $i \in [t]$ and $j \in [\ell]$ where both $\sigma_{i,j,0}$ and $\sigma_{i,j,1}$ are invalid. In this case, the honest receiver in Hyb_0 outputs $\perp$ as its signature, which matches the behavior in Hyb_1.
- Suppose there exists $j \in [\ell]$ such that for all $i \in [t]$, at least one of $\sigma_{i,j,0}$ and $\sigma_{i,j,1}$ is invalid. We argue that in this case, the receiver outputs $\perp$ with probability at least $1 - 2^{-(t-1)}$ in Hyb_0. Without loss of generality, we can assume that exactly one of $\sigma_{i,j,0}$ and $\sigma_{i,j,1}$ for all $i \in [t]$ is invalid (the case where both are invalid is captured by the previous case). Let $b_1, \ldots, b_t \in \{0,1\}$ be such that σ_{i,j,b_i} is invalid, and let $\mathbf{x} = (x_1, \ldots, x_\ell) \in \{0,1\}^\ell$ be the receiver's message in the signing protocol. In the real protocol, the honest receiver samples $w_{i,j} \xleftarrow{\text{R}} \{0,1\}$ for all $i \in [t]$ such that $x_j = \bigoplus_{i \in [t]} w_{i,j}$. We consider two possibilities:
  - Suppose $x_j \neq \bigoplus_{i \in [t]} b_i$. This means that there exists $i \in [t]$ where $w_{i,j} \neq b_i$. In the real protocol, this means that the receiver obtains signature $\sigma_{i,j,w_{i,j}}$, which by assumption is invalid. In this case, the receiver in Hyb_0 outputs $\perp$ as its signature.
  - Suppose $x_j = \bigoplus_{i \in [t]} b_i$. Since $w_{i,j}$ are sampled uniformly at random subject to the constraint, with probability $2^{-(t-1)}$, it is the case that $w_{i,j} = b_i$ for all $i \in [t]$. In this case, the receiver in Hyb_0 does not output $\perp$ (since every signature it obtains is valid). With probability $1 - 2^{-(t-1)}$, there is an index $i \in [t]$ where $w_{i,j} \neq b_i$. In this case, the receiver obtains signature $\sigma_{i,j,w_{i,j}}$, which by assumption is invalid. Thus, we conclude that the receiver in Hyb_0 aborts with probability $1 - 2^{-(t-1)}$.

  In this case, the honest receiver in Hyb_0 outputs $\perp$ with probability at least $1 - 2^{-(t-1)}$, while in Hyb_1, the receiver outputs $\perp$ with probability 1. In both cases, the probability is taken over the receiver's random coins. Since $t = \omega(\log \lambda)$, we conclude that the statistical distance between the output distributions of Hyb_0 and Hyb_1 is negligible. $\square$

Claim 3.54. *Suppose Π_{HS} satisfies context-hiding (Definition 3.9). Then, the outputs of hybrids Hyb_1 and Hyb_2 are computationally indistinguishable.*

Proof. Suppose there exists an environment $\mathcal{Z}$ and adversary $\mathcal{A}$ (that corrupts the signer $\mathbf{S}$) such that the outputs of Hyb_1 and Hyb_2 are distinguishable. We use $\mathcal{Z}$ and $\mathcal{A}$ to construct an adversary

$\mathcal{B}$ that breaks context-hiding security (Definition 3.9) of Π_{HS}. Algorithm $\mathcal{B}$ begins by simulating the protocol execution in Hyb_1 and Hyb_2 for $\mathcal{Z}$ and $\mathcal{A}$. In the simulation, $\mathcal{B}$ is responsible for simulating the behavior of the honest receiver $\mathbf{R}$ and the ideal OT functionality $\mathcal{F}_{\mathrm{OT}}^{\ell,s}$.

- *Key-generation*: The key-generation protocol only involves $\mathcal{Z}$ and $\mathcal{A}$, so $\mathcal{B}$ does not need to simulate anything.
- *Signature-generation*: On a signature-generation query $(\mathsf{sid}, \mathsf{sign}, \mathsf{vk}, \mathbf{x})$, let $\{(\sigma_{i,j,0}, \sigma_{i,j,1})\}_{i \in [t], j \in [\ell]}$ be the signatures the signer $\mathbf{S}$ submits to the ideal OT functionality $\mathcal{F}_{\mathrm{OT}}^{\ell,s}$ in the simulated protocol execution (as directed by $\mathcal{Z}$ and $\mathcal{A}$). Additionally, let $\left\{\sigma_{i,j}^{\mathsf{pk}}\right\}_{i \in [t], j \in [\ell]}$ be the message-independent signature components the signer sends to the receiver. Algorithm $\mathcal{B}$ checks the three conditions in Hyb_1 and Hyb_2, and if any condition is satisfied, it defines the receiver's output to be $(\mathsf{sid}, \mathsf{signature}, (f_{\mathsf{id}}, \mathbf{x}), \perp)$.

  Otherwise, the verification key vk has the form $\mathsf{vk} = (\mathsf{pp}, \mathsf{vk}')$ where $\mathsf{pp} = \left\{\mathsf{pk}_{i,j}\right\}_{i \in [t], j \in [\ell]}$, and moreover, there exists indices i^*, j^* where $\sigma_{i^*,j^*,0}$ and $\sigma_{i^*,j^*,1}$ are both valid. Algorithm $\mathcal{B}$ submits the public key pk_{i^*,j^*}, the verification key vk', and the message-signature pairs $(0, \sigma_{i^*,j^*,0})$ and $(1, \sigma_{i^*,j^*,1})$ to the context-hiding challenger. Finally, algorithm $\mathcal{B}$ simulates the receiver's output according to the specification in Hyb_1 and Hyb_2.

  If the receiver's output is not $\perp$, algorithm $\mathcal{B}$ does the following. Let $\mathbf{w}_1, \ldots, \mathbf{w}_t$ where $\bigoplus_{i \in [t]} \mathbf{w}_i = \mathbf{x}$ be the bit-strings $\mathcal{B}$ chose when simulating the honest receiver. For $i \in [t]$, algorithm $\mathcal{B}$ defines $\boldsymbol{\sigma}_i = (\sigma_{i,1,w_{i,1}}, \ldots, \sigma_{i,\ell,w_{i,\ell}})$.
- *Signature-verification*: Algorithm $\mathcal{B}$ simulates the verification queries involving the receiver $\mathbf{R}$ as described in Hyb_1 and Hyb_2. Note that because signature verification is non-interactive, the environment $\mathcal{Z}$ and the adversary $\mathcal{A}$ completely dictate the behavior of verification queries to the corrupt signer.
- *Signature-evaluation*: Whenever $\mathcal{Z}$ activates the receiver on a signature-evaluation query $(\mathsf{sid}, \mathsf{eval}, \mathsf{vk}, g, (f, \mathbf{x}), \boldsymbol{\sigma})$, where $\mathsf{vk} = (\mathsf{pp}, \mathsf{vk}')$, algorithm $\mathcal{B}$ ignores the request if the receiver's signature in the signing protocol was $\perp$. Otherwise, it proceeds as follows:
  - As in Hyb_1 and Hyb_2, algorithm $\mathcal{B}$ checks that $f = f_{\mathsf{id}}$ and that σ is a valid signature on $(f, \mathbf{x})$. If the signature verifies, then $\mathcal{B}$ first computes $\mathsf{pk}_g \leftarrow \mathsf{PrmsEval}(g \circ f_{\mathsf{recon}}, \mathsf{pp})$. Then, it computes $\sigma' \leftarrow \mathsf{SigEval}(g \circ f_{\mathsf{recon}}, (\boldsymbol{\sigma}_1, \ldots, \boldsymbol{\sigma}_t))$ where $\boldsymbol{\sigma}_i$ is defined as in the signing protocol.
  - Algorithm $\mathcal{B}$ submits the public key pk_g, the message $g(\mathbf{x})$, and the signature σ' to the context-hiding challenger, and receives in response a signature σ^*. Algorithm $\mathcal{B}$ simulates the output of the honest receiver as $(\mathsf{sid}, \mathsf{signature}, (g, g(\mathbf{x})), \sigma^*)$.

Note that signature evaluation is non-interactive, the environment $\mathcal{Z}$ and the adversary $\mathcal{A}$ completely dictate the behavior of evaluation queries to the corrupt signer.

- At the end of the experiment, when $\mathcal{Z}$ outputs a bit, $\mathcal{B}$ outputs the same bit.

We now show that $\mathcal{B}$ breaks context-hiding security with the same advantage as $\mathcal{Z}$. By construction, the only difference between the two hybrid experiments Hyb_1 and Hyb_2 is the way the honest receiver's signatures are generated on evaluation queries. Note that if the honest receiver outputs $\perp$ in response to a signing query, then the honest receiver in Hyb_1 and Hyb_2 ignores all evaluation queries. In this case, the two experiments are identical. Thus, without loss of generality, we assume that the signing protocol succeeds. In this case, algorithm $\mathcal{B}$ submits a valid key, verification key, and message-signature pairs to the context-hiding challenger.

Now, assume that $\mathcal{B}$ does not abort during signature generation. Then, if the context-hiding challenger implements the hide algorithm using Hide, then the signatures output by $\mathcal{B}$ when simulating the honest evaluation queries are distributed according to Hyb_1, and $\mathcal{B}$ perfectly simulates an execution of Hyb_1 for $\mathcal{Z}$ and $\mathcal{A}$. Alternatively, if the context-hiding challenger implements the hide algorithm using $\mathcal{S}^{\mathsf{Gen}}$, then the signatures output by $\mathcal{B}$ when simulating the honest evaluation queries are distributed according to Hyb_2, and $\mathcal{B}$ perfectly simulates an execution of Hyb_2 for $\mathcal{Z}$ and $\mathcal{A}$. Thus, if $\mathcal{Z}$ is able to distinguish experiments Hyb_1 and Hyb_2, algorithm $\mathcal{B}$ breaks context-hiding of Π_{HS} with the same advantage. $\square$

Claim 3.55. *The outputs of hybrids* Hyb_2 *and* Hyb_3 *are identically distributed.*

Proof. We now show that the view of the environment $\mathcal{Z}$ when interacting with an adversary $\mathcal{A}$ in Hyb_2 is distributed identically with its view when interacting with the simulator $\mathcal{S}$ in Hyb_3.

- *Key-generation*: When the environment $\mathcal{Z}$ activates the signer on a key-generation query, algorithm $\mathcal{S}$ simply forwards the query to its simulated protocol execution with adversary $\mathcal{A}$ (as if it came from $\mathcal{A}$'s environment). Thus, the output of $\mathcal{S}$ in Hyb_3 is distributed identically to the output of $\mathcal{A}$ in Hyb_2.

- *Signature-generation*: By construction, $\mathcal{S}$ perfectly simulates the behavior of the ideal OT functionality $\mathcal{F}_{\mathrm{OT}}^{\ell,s}$, so it perfectly simulates the view of $\mathcal{A}$ in its simulated protocol execution (since $\mathcal{A}$ only interacts with $\mathcal{F}_{\mathrm{OT}}^{\ell,s}$). Thus, $\mathcal{S}$ perfectly simulates any interaction between the adversary and the environment that can occur during this phase.

  It suffices to argue that the output of the honest receiver in Hyb_2 and Hyb_3 is identically distributed on a query $(\mathsf{sid}, \mathsf{sign}, \mathsf{vk}, \mathbf{x})$, where vk can be parsed as $\mathsf{vk} = (\mathsf{pp}, \mathsf{vk}')$. Let $\{(\sigma_{i,j,0}, \sigma_{i,j,1})\}_{i \in [t], j \in [\ell]}$ be the signatures the signer submits to the ideal OT functionality $\mathcal{F}_{\mathrm{OT}}^{\ell,s}$, and let $\left\{\sigma_{i,j}^{\mathsf{pk}}\right\}_{i \in [t], j \in [\ell]}$ be the message-independent signature components the signer sends to the receiver. First, if any of the verification conditions in Hyb_2 (defined in the description of

Hyb_1) are satisfied, then the output of the honest receiver in Hyb_2 is $(\mathsf{sid}, \mathsf{signature}, (f_{\mathsf{id}}, \mathbf{x}), \perp)$. By construction, the simulator $\mathcal{S}$ implements an identical set of checks. If any of the conditions are satisfied, then the simulator defines the $\mathsf{IdealSign}$ function to output $\perp$ on *all* inputs. This means that in Hyb_3, the honest receiver also outputs $\perp$ as its signature in response to the signing request. We conclude that the behavior in Hyb_2 and Hyb_3 is identical whenever any of the conditions in Hyb_2 is triggered.

Now, consider the case where none of the conditions in Hyb_2 are satisfied. This means that for all $j \in [\ell]$, there is at least one $i_j \in [t]$ where *both* $\sigma_{i_j,j,0}$ and $\sigma_{i_j,j,1}$ are valid (according to the criterion in Eq. (3.1)). We consider the receiver's output in Hyb_2 and Hyb_3.

– In Hyb_2, the honest receiver chooses $\mathbf{w}_i \xleftarrow{\text{R}} \{0,1\}^\ell$ for all $i \in [t]$ such that $\mathbf{x} = \bigoplus_{i \in [t]} \mathbf{w}_i$. This is equivalent to first sampling $w_{i,j} \xleftarrow{\text{R}} \{0,1\}$ for all $j \in [\ell]$ and $i \neq i_j$ and setting $w_{i_j,j} \in \{0,1\}$ such that $\mathbf{x} = \bigoplus_{i \in [t]} \mathbf{w}_i$. Next, we say that an index $i \in [t]$ and $j \in [\ell]$ is "bad" if either $\sigma_{i,j,0}$ or $\sigma_{i,j,1}$ is invalid. For all bad indices i, j, define $b_{i,j} \in \{0,1\}$ so that $\sigma_{i,j,b_{i,j}}$ is valid. We consider two possibilities.

  * The receiver in Hyb_2 outputs $\perp$ as its signature if there is a bad index $i \in [t]$, $j \in [\ell]$ where $w_{i,j} \neq b_{i,j}$. Suppose there are n such bad indices. Since both $\sigma_{i_j,j,0}$ and $\sigma_{i_j,j,1}$ are valid for all $j \in [\ell]$, and all of the $w_{i,j}$'s are sampled uniformly at random for $i \neq i_j$, it follows that with probability $1 - 2^{-n}$ (over the randomness used to sample the $w_{i,j}$'s), there is at least one $i \neq i_j$ and $j \in [\ell]$ where $w_{i,j} \neq b_{i,j}$.
  * With probability 2^{-n}, for all bad indices $i \in [t]$, $j \in [\ell]$, we have that $w_{i,j} = b_{i,j}$. In this case, the honest receiver in Hyb_2 obtains valid signatures $\sigma_{i,j,w_{i,j}}$ from $\mathcal{F}_{\text{OT}}^{\ell,s}$ and constructs $\boldsymbol{\sigma}$ according to the specification of Hyb_2. Here, $w_{i,j} = b_{i,j}$ for all bad indices, and for all remaining indices $i \neq i_j$ and $j \in [\ell]$, the choice bit $w_{i,j}$ is uniformly random.

– In Hyb_3, the ideal signing algorithm $\mathsf{IdealSign}$ is used to generate the honest receiver's signature, and the simulator $\mathcal{S}$ decides whether the honest receiver outputs $\perp$ or the output of $\mathsf{IdealSign}$. By construction, if n is the number of bad indices, then $\mathcal{S}$ causes the honest receiver to output $\perp$ with probability $1 - 2^{-n}$, which is precisely the probability that the honest receiver outputs $\perp$ in Hyb_2. With probability 2^{-n}, the honest receiver outputs the signature computed by $\mathsf{IdealSign}$. We argue that in this case, the signature output by $\mathsf{IdealSign}$ is distributed identically to the signature that would have been constructed by the honest receiver in Hyb_2. By construction, $\mathsf{IdealSign}$ sets $\boldsymbol{\sigma}_i = (\sigma_{i,1,w_{i,1}}, \ldots, \sigma_{i,\ell,w_{i,\ell}})$ for all $i \in [t]$ where $w_{i,j} = b_{i,j}$ for all bad indices $i \in [t]$ and $j \in [\ell]$. For the remaining indices, $w_{i,j}$ is uniformly random subject to the restriction that $\bigoplus \mathbf{w}_i = \mathbf{x}$ where $\mathbf{w}_i = (w_{i,1}, \ldots, \mathbf{w}_{i,\ell})$. Observe that this is the *same* distribution from which the $w_{i,j}$ are sampled in Hyb_2. Finally, $\mathsf{IdealSign}$ constructs the final signature σ' by computing $\boldsymbol{\sigma} \leftarrow$

$\mathsf{SigEval}(f_{\mathsf{recon}}, \mathsf{pp}, (\mathbf{w}_1, \dots, \mathbf{w}_t), (\boldsymbol{\sigma}_1, \dots, \boldsymbol{\sigma}_t))$. This is precisely the behavior of the honest receiver in Hyb_2. In addition, by correctness of Π_{HS}, it will never be the case that $(\mathsf{vk}, (f_{\mathsf{id}}, \mathbf{x}), \boldsymbol{\sigma}, 0) \in \mathcal{L}$. Specifically, $\boldsymbol{\sigma}$ is a valid signature on $\mathbf{x}$ under vk, so the simulator $\mathcal{S}$ would never register it as an invalid signature in $\mathcal{F}_{\text{BHS}}$. (Because the signer is corrupt, the unforgeability criterion in signature verification is ignored).

We conclude that the output of the honest receiver in response to a signing query is identically distributed in Hyb_2 and Hyb_3.

- *Signature-verification*: If the environment $\mathcal{Z}$ activates the corrupt signer $\tilde{\mathbf{S}}$ on a verification query $(\mathsf{sid}, \mathsf{verify}, \mathsf{vk}', (f, \mathbf{x}), \boldsymbol{\sigma})$, the simulator $\mathcal{S}$ activates the real signer $\mathbf{S}$ (under the control of $\mathcal{A}$) in its simulated version of Π_{BHS}. Since the simulator $\mathcal{S}$ forwards $\mathbf{S}$'s output to $\mathcal{Z}$, the responses to these queries in Hyb_2 and Hyb_3 are identically distributed.

  Next, suppose the environment $\mathcal{Z}$ activates the honest receiver $\mathbf{R}$ on a signature verification query $(\mathsf{sid}, \mathsf{verify}, \mathsf{vk}', (f, \mathbf{x}), \boldsymbol{\sigma})$. In Hyb_3, the ideal functionality $\mathcal{F}_{\text{BHS}}$ handles the signature-verification queries. We consider the different possibilities below. Note that because the signer is assumed to be corrupt, the unforgeability condition is ignored.

  - If $f \notin \mathcal{H}$, then $\mathcal{F}_{\text{BHS}}$ sets the verification bit $t = 0$. This is the behavior in Hyb_2.
  - Otherwise, if $\mathsf{vk} = \mathsf{vk}'$ and $(\mathsf{vk}, (f, \mathbf{x}), \boldsymbol{\sigma}, 1) \in \mathcal{L}$, then $\mathcal{F}_{\text{BHS}}$ sets $t = 1$. We consider several scenarios depending on how the entry $(\mathsf{vk}, (f, \mathbf{x}), \boldsymbol{\sigma}, 1)$ was added to $\mathcal{L}$. If $\boldsymbol{\sigma}$ was generated as a result of a signing or a evaluation request involving the honest receiver, then by (evaluation and hiding) correctness of Π_{HS}, $\boldsymbol{\sigma}$ is a valid signature on $\mathbf{x}$, and the honest receiver in Hyb_2 would also accept the signature. If the entry was added as a result of a previous verification request (which successfully verified), then because the honest party's verification algorithm in Π_{HS} is *deterministic* and since the signature previously verified, then the honest receiver would also output 1 in Hyb_2.
  - Otherwise, if there is already an entry $(\mathsf{vk}', (f, \mathbf{x}), \boldsymbol{\sigma}, t') \in \mathcal{L}$ for some t', $\mathcal{F}_{\text{BHS}}$ sets $t = t'$. In the real protocol in Hyb_2, the honest verifier's algorithm is deterministic. Hence, if a signature previously verified (resp., failed to verify), it will continue to verify (resp., fail to verify).
  - Finally, if none of the above criterion apply, then the ideal functionality allows the simulator $\mathcal{S}$ to decide the verification response in Hyb_3. By construction, for the honest receiver, the simulator implements the same logic as in the real protocol in Hyb_2.

  We conclude that the output of the honest receiver in response to verification queries is identically distributed in Hyb_2 and Hyb_3.

- *Signature-evaluation*: By definition, for any signature evaluation query made by $\mathcal{Z}$ to $\tilde{\mathbf{S}}$ in Hyb_3, the simulator $\mathcal{S}$ invokes $\mathbf{S}$ (under the control of $\mathcal{A}$) in its simulated copy of Π_{BHS} and forwards $\mathbf{S}$'s output to $\mathcal{Z}$. Therefore, $\mathcal{Z}$'s views in Hyb_2 and Hyb_3 are identical.

  Next, suppose that the environment $\mathcal{Z}$ activates the honest receiver $\tilde{\mathbf{R}}$ on an evaluation query in Hyb_3. In this case, the ideal functionality first verifies the signature (as argued above, the outcome of the signature verification procedure is identically distributed in Hyb_2 and Hyb_3), and then invokes the $\mathsf{IdealEval}$ algorithm provided by $\mathcal{S}$ to construct the signature. By construction, $\mathsf{IdealEval}$ is precisely the algorithm used in Hyb_2 to generate the signatures (specifically, both $\mathsf{IdealEval}$ and the procedure in Hyb_2 simulate the signatures using the context-hiding simulator for Π_{HS}). We conclude that the output of the honest receiver in response to an evaluation query is identically distributed in Hyb_2 and Hyb_3.

We conclude that on all queries, the view of the environment $\mathcal{Z}$ in Hyb_2 and Hyb_3 is identically distributed. $\square$

Lemma 3.52 now follows by combining Claims 3.53, 3.54, and 3.55. $\square$

Lemma 3.56. *If both the signer and the receiver are corrupt, then* $\text{IDEAL}_{\mathcal{F}_{\text{BHS}},\mathcal{S},\mathcal{Z}} \equiv \text{REAL}_{\Pi_{\text{BHS}},\mathcal{A},\mathcal{Z}}$.

Proof. When both parties are corrupt, the simulator $\mathcal{S}$ only needs to simulate the behavior of the ideal OT functionality $\mathcal{F}_{\text{OT}}^{\ell,s}$ when simulating the protocol execution for adversary $\mathcal{A}$. Since $\mathcal{S}$ forwards all of the queries from $\mathcal{Z}$ to $\mathcal{A}$ (as if it came from $\mathcal{A}$'s environment in the simulated protocol execution), and moreover, $\mathcal{S}$ perfectly simulates the behavior of the $\mathcal{F}_{\text{OT}}^{\ell,s}$ functionality, the output of $\mathcal{S}$ in the ideal execution is distributed identically to the output of $\mathcal{A}$ in the real execution. $\square$

Theorem 3.36 now follows by combining Lemmas 3.46, 3.49, 3.52, and 3.56. $\square$

3.6.3 Proof of Theorem 3.40

Let $\mathcal{A}$ be a static adversary that interacts with the environment $\mathcal{Z}$, a prover $\mathcal{P}$, and a verifier $\mathcal{V}$ running the real protocol Π_{ZK} (Figure 3.8). We construct an ideal world adversary (simulator) $\mathcal{S}$ that interacts with the environment $\mathcal{Z}$, a dummy prover $\tilde{\mathcal{P}}$, a dummy verifier $\tilde{\mathcal{V}}$, and ideal functionality $\mathcal{F}_{\mathsf{ZK}}$ such that no environment $\mathcal{Z}$ can distinguish an interaction with $\mathcal{A}$ in the real execution from one with $\mathcal{S}$ in the ideal execution.

We begin by describing the simulator $\mathcal{S}$. At the beginning of the protocol execution, the simulator $\mathcal{S}$ begins by invoking the adversary $\mathcal{A}$. Algorithm $\mathcal{A}$ begins by declaring which parties it would like to corrupt, and $\mathcal{S}$ corrupts the corresponding set of dummy parties. The simulation then proceeds as follows.

Simulating the communication with the environment. Whenever the simulator $\mathcal{S}$ receives an input from the environment $\mathcal{Z}$, it forwards the input to $\mathcal{A}$ (as if it came from the environment in the simulated protocol execution). Whenever $\mathcal{A}$ writes a message on its output tape (in the simulated protocol execution), the simulator $\mathcal{S}$ writes the same output on its own output tape (to be read by the environment).

Simulating the ideal BHS functionality. At the beginning of the protocol execution, the simulator $\mathcal{S}$ initializes an empty list $\mathcal{L}$ to keep track of the signatures in the simulated instance of $\mathcal{F}_{\text{BHS}}$. The simulator $\mathcal{S}$ simulates the ideal BHS functionality exactly as described in Figure 3.6. Whenever the specification of $\mathcal{F}_{\text{BHS}}$ needs to interact with the ideal adversary, the simulator $\mathcal{S}$ forwards the request to $\mathcal{A}$ (as if it came from $\mathcal{F}_{\text{BHS}}$ in the simulated protocol execution), and uses the response from $\mathcal{A}$ to continue the simulation.

Simulating the preprocessing phase. In the preprocessing phase, the verifier and the prover never exchange any messages with each other. They only interact with the $\mathcal{F}_{\text{BHS}}$ functionality. As stated above, the simulator simulates the behavior of $\mathcal{F}_{\text{BHS}}$ exactly as described in Figure 3.6. If a party is corrupt, then the simulator uses $\mathcal{A}$ to determine the messages it sends to $\mathcal{F}_{\text{BHS}}$. If a party is honest, then $\mathcal{S}$ simulates the behavior of the honest party exactly as in the real protocol. Let $\widetilde{\mathsf{vk}}$ be the verification key the verifier sends to the prover in the simulated execution.

Simulating the proofs. After simulating the preprocessing phase, the simulator $\mathcal{S}$ proceeds as follows, depending on which parties are corrupt:

- *The prover is honest*: If the prover is honest, then the prover (in both the real and ideal executions) does nothing until it is activated by the environment. In the ideal execution, whenever the environment activates the prover on an input $(\mathsf{sid}, \mathsf{ssid}, \mathsf{prove}, \mathcal{R}, \mathbf{x}, \mathbf{w})$ where $\mathcal{R}(\mathbf{x}, \mathbf{w}) = 1$, then $\mathcal{S}$ receives a tuple $(\mathsf{sid}, \mathsf{ssid}, \mathsf{proof}, \mathcal{R}, \mathbf{x})$ from $\mathcal{F}_{\mathsf{ZK}}$. When this occurs, $\mathcal{S}$ simulates the request as follows. First, let $\widetilde{\mathsf{sk}}$ be the secret key the simulator $\mathcal{S}$ chose for the prover when simulating the preprocessing phase (since the prover is honest, $\mathcal{S}$ chooses the secret key). Then, $\mathcal{S}$ constructs a ciphertext $\widetilde{\mathsf{ct}} \leftarrow \mathsf{Encrypt}(\mathsf{sk}, 0^\tau)$, where τ denotes the length of the witness for relation $\mathcal{R}$. Next, $\mathcal{S}$ constructs a signature $\tilde{\sigma}^* \leftarrow \mathsf{IdealEval}(\mathsf{CheckWitness}_{\mathcal{R},\mathsf{ct},\mathbf{x}}, 1)$, where $\mathsf{IdealEval}$ is the ideal signature evaluation functionality that $\mathcal{A}$ chooses for $\mathcal{F}_{\text{BHS}}$. The simulator $\mathcal{S}$ constructs the simulated proof $\tilde{\pi} = (\widetilde{\mathsf{ct}}, \tilde{\sigma}^*)$, adds the signature $(\mathsf{vk}, (\mathsf{CheckWitness}_{\mathcal{R},\mathsf{ct},\mathbf{x}}, 1), \tilde{\sigma}^*, 1)$ to $\mathcal{L}$ (if an entry does not already exist), and sends $(\mathsf{sid}, \mathsf{ssid}, \mathsf{proof}, \mathbf{x}, \tilde{\pi})$ to the verifier in the simulated protocol execution.
- *The prover is corrupt*: First, if the verifier is also corrupt, then the simulator $\mathcal{S}$ only needs to simulate the BHS functionality $\mathcal{F}_{\text{BHS}}$. Specifically, whenever the environment activates the

prover on an input in the ideal execution, the simulator simply forwards the input to the (corrupt) prover in the simulated execution.

On the other hand, if the verifier is honest, then $\mathcal{S}$ proceeds as follows:

– At the beginning of the simulation, $\mathcal{S}$ initializes $\widetilde{\mathsf{sk}}$ to $\perp$. At any point in the simulated protocol execution, if the prover (as dictated by $\mathcal{A}$) makes a successful signing request to the $\mathcal{F}_{\text{BHS}}$ functionality, the simulator $\mathcal{S}$ updates $\widetilde{\mathsf{sk}}$ to be the message the prover submitted to the signing functionality. By definition of the $\mathcal{F}_{\text{BHS}}$ functionality, the prover can make at most one successful signing request to the $\mathcal{F}_{\text{BHS}}$.

– Whenever the environment activates the prover in the ideal execution on an input $(\mathsf{sid}, \mathsf{ssid}, \mathsf{prove}, \mathcal{R}, \mathbf{x}, \mathbf{w})$, the simulator $\mathcal{S}$ activates the prover in the simulated protocol execution on the same input.

– Whenever the prover in the simulated execution sends a message $(\mathsf{sid}, \mathsf{ssid}, \mathsf{proof}, \mathcal{R}, \mathbf{x}, \pi)$ to the verifier, the simulator parses $\pi = (\mathsf{ct}, \sigma^*)$. If π does not have this form or if $\widetilde{\mathsf{sk}} = \perp$, then $\mathcal{S}$ ignores the message. Otherwise, the simulator $\mathcal{S}$ submits the request $(\mathsf{sid}, \mathsf{verify}, \mathsf{vk}, (\mathsf{CheckWitness}_{\mathcal{R},\mathsf{ct},\mathbf{x}}, 1), \sigma^*)$ to its (simulated) ideal functionality $\mathcal{F}_{\text{BHS}}$. If the signature does not verify, then $\mathcal{S}$ ignores the request. Otherwise, it computes $\mathbf{w} \leftarrow \mathsf{Decrypt}(\widetilde{\mathsf{sk}}, \mathsf{ct})$ and outputs $(\mathsf{sid}, \mathsf{ssid}, \mathsf{proof}, \mathcal{R}, \mathbf{x})$ for the honest verifier in the simulated execution. In addition, $\mathcal{S}$ submits $(\mathsf{sid}, \mathsf{ssid}, \mathsf{prove}, \mathcal{R}, \mathbf{x}, \mathbf{w})$ to Π_{ZK} (on behalf of the prover $\tilde{\mathcal{P}}$).

To conclude the proof, we show that the environment cannot distinguish the output of the real execution with adversary $\mathcal{A}$ from an ideal execution with the simulator $\mathcal{S}$. We consider the two cases separately.

Lemma 3.57. *If the prover is honest, and Π_{enc} is a CPA-secure encryption scheme, then in the $\mathcal{F}_{\text{BHS}}$-hybrid model, $\text{REAL}_{\Pi_{\mathsf{ZK}},\mathcal{A},\mathcal{Z}} \overset{c}{\approx} \text{IDEAL}_{\mathcal{F}_{\mathsf{ZK}},\mathcal{S},\mathcal{Z}}$.*

Proof. Our proof proceeds via a hybrid argument:

- Hyb_0: This is the real distribution $\text{REAL}_{\Pi_{\mathsf{ZK}},\mathcal{A},\mathcal{Z}}$.

- Hyb_1: Same as Hyb_0, except when constructing proofs, the honest prover does not submit $(\mathsf{sid}, \mathsf{eval}, \mathsf{vk}, \mathsf{CheckWitness}_{\mathcal{R},\mathsf{ct},\mathbf{x}}, (f_{\mathsf{id}}, \mathsf{sk}), \sigma_{\mathsf{sk}})$ to $\mathcal{F}_{\text{BHS}}$ to obtain the signature σ^*. Instead, the signature is constructed as $\sigma^* \leftarrow \mathsf{IdealEval}(\mathsf{CheckWitness}_{\mathcal{R},\mathsf{ct},\mathbf{x}}, 1)$. Afterwards, the entry $(\mathsf{vk}, (\mathsf{CheckWitness}_{\mathcal{R},\mathsf{ct},\mathbf{x}}, 1), \sigma^*, 1)$ is added to $\mathcal{F}_{\text{BHS}}$ (if an entry does not already exist.)

- Hyb_2: Same as Hyb_1, except during the preprocessing phase, the honest prover sends $(\mathsf{sid}, \mathsf{sign}, \mathsf{vk}, \mathsf{sk}')$ to $\mathcal{F}_{\text{BHS}}$ where $\mathsf{sk}' \leftarrow \mathsf{KeyGen}(1^\lambda)$ is generated independently of sk. The ciphertexts in the encryption step are still generated using sk.

- Hyb_3: Same as Hyb_2, except the honest prover encrypts the all-zeroes string 0^τ (where τ is the bit-length of the witness) when constructing the proofs.
- Hyb_4: Same as Hyb_3, except the honest prover requests the signature on sk in the preprocessing step (instead of the dummy key sk').
- Hyb_5: This is the ideal distribution $\text{IDEAL}_{\mathcal{F}_{\mathsf{ZK}},\mathcal{S},\mathcal{Z}}$.

We now show that assuming Π_{enc} is CPA-secure, the outputs of each pair of consecutive hybrid experiments are computationally indistinguishable.

- Hybrids Hyb_0 and Hyb_1 are identical experiments according to the specification of $\mathcal{F}_{\text{BHS}}$. Specifically, since the prover is honest, the ideal functionality $\mathcal{F}_{\text{BHS}}$ answers the signature-evaluation queries using the ideal evaluation function $\mathsf{IdealEval}$, which is precisely the procedure described in Hyb_1.
- Hybrids Hyb_1 and Hyb_2 are computationally indistinguishable if the encryption scheme $(\mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt})$ is CPA-secure. First, the only difference in Hyb_1 and Hyb_2 is that in Hyb_2, the entry $(\mathsf{sid}, \mathsf{vk}, (f_{\mathsf{id}}, \mathsf{sk}), \sigma_{\mathsf{sk}})$ in $\mathcal{F}_{\text{BHS}}$ is replaced with the entry $(\mathsf{sid}, \mathsf{vk}, (f_{\mathsf{id}}, \mathsf{sk}'), \sigma_{\mathsf{sk}})$. We consider two cases, depending on whether the verifier is honest or corrupt.

  The verifier is honest: If the verifier is honest, then these two experiments are identically distributed. Specifically, the only queries the honest verifier makes to $\mathcal{F}_{\text{BHS}}$ are on (computed) signatures σ^* that are registered with $\mathcal{F}_{\text{BHS}}$.

  The verifier is corrupt: In this case, the adversary $\mathcal{A}$ can make arbitrary queries (on behalf of the verifier) to the $\mathcal{F}_{\text{BHS}}$ functionality. In addition, since the verifier is the signer (with respect to the $\mathcal{F}_{\text{BHS}}$ functionality), during signature verification, only the correctness and consistency conditions are checked (and in particular, *not* the unforgeability condition). This means that the view of adversary $\mathcal{A}$ is identically distributed in Hyb_1 and Hyb_2 unless $\mathcal{A}$ makes an evaluation or a verification query to $\mathcal{F}_{\text{BHS}}$ on a message of the form $(f_{\mathsf{id}}, \mathsf{sk})$ or $(f_{\mathsf{id}}, \mathsf{sk}')$. We first show that in Hyb_2, the probability that $\mathcal{A}$ makes a verification query to $\mathcal{F}_{\text{BHS}}$ on a message of the form $(f_{\mathsf{id}}, \mathsf{sk}')$ is negligible. By construction, in Hyb_2, the adversary's view is *completely* independent of sk', so we can effectively defer the sampling of sk' until *after* the adversary has made all of its verification queries. Since the adversary makes $\text{poly}(\lambda)$ verification queries, and sk' is drawn from a distribution with min-entropy at least $\omega(\log \lambda)$,[13] the probability (taken over the randomness of the key-generation algorithm) that sk' coincides with a message in the adversary's query is negligible.

[13]This is implied by CPA-security of the encryption scheme. Otherwise, the adversary has a noticeable probability of guessing the key for the encryption scheme, which trivially breaks CPA-security.

We now show that if there exists an adversary $\mathcal{A}$ and an environment $\mathcal{Z}$ such that the outputs of Hyb_1 and Hyb_2 are not computationally indistinguishable, then we can construct an adversary $\mathcal{B}$ that breaks CPA-security of Π_{enc}. Based on the above analysis, to achieve non-negligible distinguishing advantage, algorithm $\mathcal{A}$ has to issue a verification query on the message $(f_{\mathsf{id}}, \mathsf{sk})$ to the ideal functionality $\mathcal{F}_{\text{BHS}}$ with non-negligible probability. Algorithm $\mathcal{B}$ simulates an instance of the protocol execution environment according to Hyb_2 as follows:

- Then, $\mathcal{B}$ starts the protocol execution experiment by activating the environment $\mathcal{Z}$. Algorithm $\mathcal{B}$ now simulates the protocol execution experiment as described in Hyb_2.
- To simulate the honest prover during the preprocessing phase, $\mathcal{B}$ leaves the secret key sk unspecified (since it is not needed in the simulation). It samples $\mathsf{sk}' \leftarrow \mathsf{KeyGen}(1^\lambda)$ for the honest prover and simulates the rest of the preprocessing as described in Hyb_2.
- Whenever the environment activates the prover to construct a proof on a statement-witness pair $(\mathbf{x}, \mathbf{w})$ for a relation $\mathcal{R}$, algorithm $\mathcal{B}$ simulates the honest prover in Hyb_2 by first checking that $\mathcal{R}(\mathbf{x}, \mathbf{w}) = 1$. If so, then $\mathcal{B}$ submits an encryption query on the pair $(\mathbf{w}, \mathbf{w})$ to the encryption oracle to obtain a ciphertext ct. Algorithm $\mathcal{B}$ simulates the rest of the protocol exactly as described in Hyb_2.
- Algorithm $\mathcal{B}$ simulates the $\mathcal{F}_{\text{BHS}}$ functionality according to the specification of Hyb_1 and Hyb_2 (the behavior of $\mathcal{F}_{\text{BHS}}$ is identical in the two hybrids, and does not depend on sk).
- At the end of the protocol execution experiment, algorithm $\mathcal{B}$ chooses a random bit-string $\xi \xleftarrow{\text{R}} \{0,1\}^\lambda$. It makes ξ chosen-message queries to the encryption oracle on pairs $(\xi_1, 0), \ldots, (\xi_\lambda, 1)$ to obtain ciphertexts $\mathsf{ct}_1, \ldots, \mathsf{ct}_\lambda$. Then, for each verification query made by adversary $\mathcal{A}$ to $\mathcal{F}_{\text{BHS}}$ on a message of the form $(f_{\mathsf{id}}, \widehat{\mathsf{sk}})$ for some $\widehat{\mathsf{sk}}$, algorithm $\mathcal{B}$ checks to see if for all $i \in [\lambda]$, $\mathsf{Decrypt}(\widehat{\mathsf{sk}}, \mathsf{ct}_i) = \xi_i$. If this holds for all $i \in [\lambda]$, then $\mathcal{B}$ outputs 1. Otherwise, it outputs 0.

By definition, we see that $\mathcal{B}$ perfectly simulates the protocol execution according to the specification in Hyb_2. By assumption, with non-negligible probability ε, algorithm $\mathcal{A}$ will issue a query to $\mathcal{F}_{\text{BHS}}$ on the message $(f_{\mathsf{id}}, \mathsf{sk})$. This means that with probability ε, there is some $\widehat{\mathsf{sk}}$ where $\widehat{\mathsf{sk}} = \mathsf{sk}$.

- Suppose $\mathcal{B}$ is interacting with the encryption oracle $\mathcal{O}_0$ in the CPA-security game. In this case, if there is a message of the form $(f_{\mathsf{id}}, \widehat{\mathsf{sk}})$ where $\widehat{\mathsf{sk}} = \mathsf{sk}$, then $\mathcal{B}$ outputs 1 by correctness of the encryption scheme. In this case, $\mathcal{B}$ outputs 1 with probability at least ε.
- Suppose instead that $\mathcal{B}$ is interacting with the encryption oracle $\mathcal{O}_1$. In this case, the ciphertexts $\mathsf{ct}_1, \ldots, \mathsf{ct}_\lambda$ and keys $\widehat{\mathsf{sk}}$ are all independent of ξ_i. Thus, union bounding over all of the messages of the form $(f_{\mathsf{id}}, \widehat{\mathsf{sk}})$ appearing in the verification queries to $\mathcal{F}_{\text{BHS}}$, the probability that $\mathcal{B}$ outputs 1 (taken over the choice of ξ_i's) is at most $\text{poly}(\lambda)/2^\lambda = \text{negl}(\lambda)$.

We conclude that $\mathcal{B}$ is able to break CPA-security with non-negligible probability $\varepsilon - \mathrm{negl}(\lambda)$. Thus, if the encryption scheme is CPA-secure, the outputs of hybrids Hyb_1 and Hyb_2 are computationally indistinguishable.

- Hybrids Hyb_2 and Hyb_3 are computationally indistinguishable if Π_{enc} is CPA-secure. Observe that none of the logic in Hyb_2 and Hyb_3 depend on the secret key sk, and all of the messages can be simulated given access to an encryption oracle $\mathsf{Encrypt}(\mathsf{sk}, \cdot)$. By CPA-security of Π_{enc}, we conclude that the outputs of these two hybrid experiments are computationally indistinguishable.

- Hybrids Hyb_3 and Hyb_4 are computationally indistinguishable if Π_{enc} is CPA-secure. The argument follows by the same logic as that used to argue computational indistinguishability of Hyb_1 and Hyb_2.

- Hybrids Hyb_4 and Hyb_5 are identically distributed. By construction, the honest prover's behavior in Hyb_4 precisely coincides with the behavior of the simulated prover in Hyb_5. Thus, the outputs of $\mathcal{A}$ in Hyb_4 are distributed exactly as the outputs of $\mathcal{S}$ in Hyb_5. Moreover, if the verifier is honest, then the outputs of the honest verifier in Hyb_4 are distributed identically to the outputs in Hyb_5; this follows by appealing to the correctness property of the ideal $\mathcal{F}_{\mathrm{BHS}}$ functionality. We conclude that the output distribution of Hyb_4 is identical to that of the ideal execution.

Since each pair of hybrid arguments is computationally indistinguishable, the lemma follows. □

Lemma 3.58. *If the prover is corrupt, then in the $\mathcal{F}_{\mathrm{BHS}}$-hybrid model, we have that* $\mathrm{REAL}_{\Pi_{\mathsf{ZK}},\mathcal{A},\mathcal{Z}} \equiv \mathrm{IDEAL}_{\mathcal{F}_{\mathsf{ZK}},\mathcal{S},\mathcal{Z}}$.

Proof. In the case where the prover is corrupt, we show that the output of the real and ideal protocol executions are identically distributed. We consider two cases.

The verifier is corrupt: If the verifier is also corrupt, then the simulator $\mathcal{S}$ is only responsible for simulating the $\mathcal{F}_{\mathrm{BHS}}$ functionality. Since $\mathcal{S}$ simulates the ideal BHS functionality exactly as described in Figure 3.6, the output of $\mathcal{S}$ is identically distributed as the output of $\mathcal{A}$ in the real execution, and the claim follows.

The verifier is honest: If the verifier is honest, we show that $\mathcal{S}$ perfectly simulates the behavior of the honest verifier in the simulated protocol execution. By construction, $\mathcal{S}$ perfectly simulates the behavior of the honest verifier in the preprocessing phase. Next, in the real execution, the honest verifier only responds when it receives a tuple of the form $(\mathsf{sid}, \mathsf{ssid}, \mathsf{proof}, \mathcal{R}, \mathsf{x}, \pi)$ from the prover. We show that the simulation is correct:

- Suppose in the real scheme, the verifier has not set the ready flag. This corresponds to the setting where the prover has never made a signing request to $\mathcal{F}_{\mathrm{BHS}}$. In this case, the verifier ignores the request. In the simulated protocol execution, if the prover never makes a signing request to $\mathcal{F}_{\text{BHS}}$, then $\widetilde{\mathsf{sk}} = \bot$, and the verifier also ignores the request.

- Suppose in the real scheme, the proof π does not have the form (ct, σ). In this case, the verifier also ignores the request. This is precisely how $\mathcal{S}$ simulates the honest verifier's behavior in the simulated protocol execution.

- Otherwise, in the real scheme, the honest verifier parses the proof as $\pi = (\mathsf{ct}, \sigma)$, and submits $(\mathsf{sid}, \mathsf{verify}, \mathsf{vk}, (\mathsf{CheckWitness}_{\mathcal{R},\mathsf{ct},\mathbf{x}}, 1), \sigma)$ to $\mathcal{F}_{\text{BHS}}$. We consider several cases:

  Case 1: Suppose that $(\mathsf{vk}, (\mathsf{CheckWitness}_{\mathcal{R},\mathsf{ct},\mathbf{x}}, 1), \sigma, 1) \in \mathcal{L}$, where $\mathcal{L}$ is the list of signatures maintained by $\mathcal{F}_{\text{BHS}}$. In this case, $\mathcal{F}_{\text{BHS}}$ declares the signature valid, and the honest verifier in the real scheme accepts the proof by outputting $(\mathsf{sid}, \mathsf{ssid}, \mathsf{proof}, \mathcal{R}, \mathbf{x})$. According to the specification of $\mathcal{F}_{\text{BHS}}$, there are two possible ways for $(\mathsf{vk}, (\mathsf{CheckWitness}_{\mathcal{R},\mathsf{ct},\mathbf{x}}, 1), \sigma, 1)$ to be added to $\mathcal{L}$:

  - The prover made a successful evaluation query with function $\mathsf{CheckWitness}_{\mathcal{R},\mathsf{ct},\mathbf{x}}$ on some input sk where $\mathsf{CheckWitness}_{\mathcal{R},\mathsf{ct},\mathbf{x}}(\mathsf{sk}) = 1$, and moreover, there is an entry $(\mathsf{vk}, (f_{\mathsf{id}}, \mathsf{sk}), \sigma', 1) \in \mathcal{L}$ for some σ'.
  - The prover previously made a verification query on $(\mathsf{vk}, (\mathsf{CheckWitness}_{\mathcal{R},\mathsf{ct},\mathbf{x}}, 1), \sigma, 1)$ and the adversary decided the verification result. According to the $\mathcal{F}_{\text{BHS}}$ specification, the adversary chooses the verification output only if there exists $(\mathsf{vk}, (f_{\mathsf{id}}, \mathsf{sk}), \sigma', 1) \in \mathcal{L}$ for some σ' where $\mathsf{CheckWitness}_{\mathcal{R},\mathsf{ct},\mathbf{x}}(\mathsf{sk}) = 1$.

  We conclude that in this case, there exist sk and σ' where $(\mathsf{vk}, (f_{\mathsf{id}}, \mathsf{sk}), \sigma', 1) \in \mathcal{L}$ and $\mathsf{CheckWitness}_{\mathcal{R},\mathsf{ct},\mathbf{x}}(\mathsf{sk}) = 1$. Since the verifier is honest, by the specification of $\mathcal{F}_{\text{BHS}}$, this is possible only if the prover has previously made a successful signing request on sk. This means that in the simulated protocol execution, the prover must have submitted a signing request to $\mathcal{F}_{\text{BHS}}$ on message sk. By construction of the simulator, $\widetilde{\mathsf{sk}} = \mathsf{sk}$. Now, in the simulation, $\mathcal{S}$ computes $\mathsf{Decrypt}(\widetilde{\mathsf{sk}}, \mathsf{ct})$ to obtain $\mathbf{w}$. Since $\mathsf{CheckWitness}_{\mathcal{R},\mathsf{ct},\mathbf{x}}(\mathsf{sk}) = 1$, this means that $\mathcal{R}(\mathbf{x}, \mathbf{w}) = 1$. In the ideal execution, the simulator sends $(\mathsf{sid}, \mathsf{ssid}, \mathsf{proof}, \mathcal{R}, \mathbf{x}, \mathbf{w})$ to Π_{ZK}, which by definition forwards the output $(\mathsf{sid}, \mathsf{ssid}, \mathsf{proof}, \mathcal{R}, \mathbf{x})$ to the dummy verifier. Thus, in this case, the honest verifier's behavior in both the real and ideal executions is identical.

  Case 2: Suppose that $(\mathsf{vk}, (\mathsf{CheckWitness}_{\mathcal{R},\mathsf{ct},\mathbf{x}}, 1), \sigma, 1) \notin \mathcal{L}$. We consider two possibilities.

  - If there does not exist an entry $(\mathsf{vk}, (f_{\mathsf{id}}, \mathsf{sk}), \sigma', 1)$ where $\mathsf{CheckWitness}_{\mathcal{R},\mathsf{ct},\mathbf{x}}(\mathsf{sk}) = 1$ in the list $\mathcal{L}$ for some sk and σ', then by the unforgeability condition, the ideal functionality $\mathcal{F}_{\text{BHS}}$ declares the signature invalid, and the honest verifier in the real scheme ignores the message. Since $\mathcal{S}$ simulates the ideal functionality $\mathcal{F}_{\text{BHS}}$ perfectly, the simulator $\mathcal{S}$ also ignores the message in the simulated execution.

– On the other hand, if $\mathcal{L}$ does contain an entry $(\mathsf{vk}, (f_{\mathsf{id}}, \mathsf{sk}), \sigma', 1)$ for some sk and σ' where $\mathsf{CheckWitness}_{\mathcal{R},\mathsf{ct},\mathbf{x}}(\mathsf{sk}) = 1$, then $\mathcal{F}_{\text{BHS}}$ allows the adversary to decide whether the signature is valid or not. If the adversary declares the signature invalid, then in both the real and the simulated executions, the verifier ignores the message. If the adversary declares the signature valid, then in the real execution, the verifier accepts the proof and outputs $(\mathsf{sid}, \mathsf{ssid}, \mathsf{proof}, \mathcal{R}, \mathbf{x})$. In the simulated execution, because there does exist $(\mathsf{vk}, (f_{\mathsf{id}}, \mathsf{sk}), \sigma', 1) \in \mathcal{L}$ where $\mathsf{CheckWitness}_{\mathcal{R},\mathsf{ct},\mathbf{x}}(\mathsf{sk}) = 1$, we can apply the same analysis from Case 1 to argue that in the ideal execution, the simulated verifier also accepts the proof and outputs $(\mathsf{sid}, \mathsf{ssid}, \mathsf{proof}, \mathcal{R}, \mathbf{x})$. Moreover, in this case, $\mathcal{S}$ also forwards $(\mathsf{sid}, \mathsf{ssid}, \mathsf{proof}, \mathcal{R}, \mathbf{x}, \mathbf{w})$ where $\mathbf{w} \leftarrow \mathsf{Decrypt}(\widetilde{\mathsf{sk}}, \mathsf{ct})$ and $\mathcal{R}(\mathbf{x}, \mathbf{w}) = 1$ to Π_{ZK}. This means that the honest verifier in the ideal execution also outputs $(\mathsf{sid}, \mathsf{ssid}, \mathsf{proof}, \mathcal{R}, \mathbf{x})$.

From the above analysis, we see that in all cases, the behavior of the honest verifier in both the real execution and the ideal execution is identical. Moreover, algorithm $\mathcal{S}$ perfectly simulates the view of $\mathcal{A}$ in the simulated protocol execution. The lemma follows. $\square$

Combining Lemmas 3.57 and 3.58, we conclude that the Π_{ZK} protocol securely realizes $\mathcal{F}_{\mathsf{ZK}}$ in the presence of malicious adversaries in the $\mathcal{F}_{\text{BHS}}$-hybrid model. $\square$

3.7 Chapter Summary

In this chapter, we constructed the first multi-theorem preprocessing NIZK arguments from standard lattice assumptions by way of context-hiding homomorphic signatures. We then introduced a new cryptographic primitive (blind homomorphic signatures) which provided an efficient way to implement the preprocessing step of our new NIZK candidate. We conclude with two directions for future work.

Towards NIZKs in the CRS model. Our techniques do not directly generalize to the CRS model. While it is possible to obtain a *publicly-verifiable* preprocessing NIZK (i.e., make the verification key k_V public), our construction critically relies on the prover state being hidden. This is because the prover state contains the *secret key* the prover uses to encrypt its witness in the proofs, so publishing this compromises zero-knowledge. Nonetheless, we believe that having a better understanding of NIZKs in the preprocessing model provides a useful stepping stone towards the goal of building NIZKs from lattices in the CRS model, and we leave this as an exciting open problem.

Preprocessing NIZKs from other assumptions? Our work gives the first construction of a multi-theorem preprocessing NIZK from standard lattice assumptions. It is an interesting challenge to obtain multi-theorem preprocessing NIZKs from other assumptions that are currently not known to imply NIZKs in the CRS model. For instance, a natural target would be to construct multi-theorem NIZKs in the preprocessing model from the decisional Diffie-Hellman (DDH) assumption.

Chapter 4

Succinct Non-Interactive Arguments (SNARGs)

In this chapter, we turn our attention to constructing *succinct* non-interactive argument systems (SNARGs) from lattice-based assumptions. Recall from Chapter 1 that an argument system for NP is succinct if its communication complexity is *polylogarithmic* in the running time of the NP verifier. We begin by providing some background on the existing approaches for constructing SNARGs. Then in Section 4.1, we provide a technical overview of our construction.

Designated-verifier arguments. In a designated-verifier argument system, there is a setup algorithm that generates a public proving key and a secret verification state. Only the verifier who knows the verification state is able to verify proofs, and soundness holds provided that the prover does not know the secret verification state. In many applications like bootstrapping obfuscation[1] and verifiable computation [WB15], designated-verifier SNARGs suffice. A key question that arises in the design and analysis of designated verifier arguments is whether the same common reference string can be reused for multiple proofs—that is, whether the proof system provides multi-theorem soundness. As discussed in Chapter 3, in many designated-verifier argument (and proof) systems, if the prover can choose its queries in a way that induces noticeable correlations between the outputs of the verification oracle and the secret verification state, then the adversary can potentially compromise the soundness of the scheme. Thus, special care is needed to construct designated-verifier argument systems in the multi-theorem setting.

[1]The application of designated-verifier SNARGs to bootstrapping obfuscation is described in the full version of this chapter [BISW17]. Because this thesis is primarily focused on constructing non-interactive argument systems, we omit the details here.

SNARGs from linear-only encryption. Bitansky et al. [BCI$^+$13] introduced a generic compiler for building designated-verifier SNARGs in the "preprocessing model" based on a notion called "linear-only" encryption.[2] In the preprocessing model, the setup algorithm that constructs the CRS can run in time that depends polynomially on a time bound T of the computations that will be verified (this is in contrast to the notion of a "fully succinct" SNARG, which stipulates that the running time of the setup algorithm is $\text{polylog}(T)$). In the designated-verifier setting, the preprocessing algorithm outputs a prover key (i.e., the public CRS), and a secret verification key.[3] The resulting scheme can then be used to verify computations that run in time at most T. The compiler of [BCI$^+$13] can be decomposed into an information-theoretic transformation and a cryptographic transformation, which we outline here:

- First, they restrict the interactive proof model to only consider "affine-bounded" provers. An affine-bounded prover is only able to compute affine functions (over a ring) of the verifier's queries.[4] Bitansky et al. give several constructions of succinct two-message interactive proofs in this restricted model by applying a generic transformation to existing "linear PCP" constructions.

- Next, they introduce a new cryptographic primitive called linear-only encryption, which is a (public-key) encryption scheme that *only* supports linear homomorphisms on ciphertexts. Bitansky et al. show that combining a linear-only encryption scheme with the affine-restricted interactive proofs from the previous step suffices to construct a designated-verifier SNARG in the preprocessing model. The construction is quite natural: the CRS for the SNARG system is a linear-only encryption of what would be the verifier's first message. The prover then homomorphically computes its response to the verifier's encrypted queries. The linear-only property of the encryption scheme constrains the prover to only using affine strategies. This ensures soundness for the SNARG. To check a proof, the verifier decrypts the prover's responses and applies the decision algorithm for the underlying two-message proof system. Bitansky et al. give several candidate instantiations for their linear-only encryption scheme based on Paillier encryption [Pai99] as well as bilinear maps [Jou00, BF01].

Linear PCPs. Like [BCI$^+$13], our SNARG constructions rely on linear PCPs.[5] A linear PCP of length m over a finite field $\mathbb{F}$ is an oracle computing a linear function $\boldsymbol{\pi} : \mathbb{F}^m \to \mathbb{F}$. On any

[2]Gennaro et al. [GGPR13] also described a similar technique based on additively homomorphic encodings and quadratic span programs to construct publicly-verifiable and designated-verifier SNARGs.

[3]When describing SNARGs, the term "preprocessing SNARGs" refers to designated-verifier SNARGs with non-succinct precomputation. This use of the term "preprocessing" differs somewhat from the notion of preprocessing NIZKs from Chapter 3 in that a preprocessing NIZK also includes argument systems where the proving key is kept secret.

[4]Bitansky et al. [BCI$^+$13] refer to this as "linear-only," even though the prover is allowed to compute affine functions. To be consistent with their naming conventions, we will primarily write "linear-only" to refer to "affine-only."

[5]Linear PCPs were first used by Ishai et al. [IKO07] to construct efficient *interactive* argument systems from any homomorphic encryption scheme. Subsequently, Bitansky et al. [BCI$^+$13] gave a construction of succinct *non-interactive* arguments from linear PCPs together with a linear-only encryption scheme.

query $\mathbf{q} \in \mathbb{F}^m$, the linear PCP oracle responds with $\mathbf{q}^T\boldsymbol{\pi}$. More generally, if k queries are made to the linear PCP oracle, the k queries can be packed into the columns of a query matrix $\mathbf{Q} \in \mathbb{F}^{m\times k}$. The response of the linear PCP oracle can then be written as $\mathbf{Q}^T\boldsymbol{\pi}$. We provide more details in Section 4.3. In this work, we instantiate our linear PCPs using the quadratic span programs of Gennaro et al. [GGPR13].

Complexity measures for SNARGs. For a security parameter λ, we measure the asymptotic cost of achieving soundness error $2^{-\lambda}$ against provers (modeled as a Boolean circuit) of size 2^λ. In this work, we are primarily interested in minimizing the prover complexity and the proof size.

4.1 Summary of Results and Technical Overview

In this section, we summarize our main results on constructing preprocessing SNARGs based on a new notion called linear-only vector encryption. Our results builds upon and extends the framework introduced by Bitansky et al. [BCI+13].

New compiler for preprocessing SNARGs. The preprocessing SNARGs we construct in this work enjoy several advantages over those of [BCI+13]. We enumerate some of them below:

- **Direct construction of SNARGs from linear PCPs.** Our compiler gives a *direct* compilation from linear PCPs over a finite field $\mathbb{F}$ into a preprocessing SNARG. In contrast, the compiler in [BCI+13] first constructs a two-message linear interactive proof from a linear PCP by introducing an additional linear consistency check. The additional consistency check not only increases the communication complexity of their construction, but also introduces a soundness error $O(1/|\mathbb{F}|)$. As a result, their construction only provides soundness when working over a large field (that is, when $|\mathbb{F}|$ is superpolynomial in the security parameter). By using a direct compilation of linear PCPs into SNARGs, we avoid both of these problems. Our construction does not require any additional consistency checks and moreover, it preserves the soundness of the underlying linear PCP. Thus, as long as the underlying linear PCP is statistically sound, applying our compiler yields a computationally sound argument (even if $|\mathbb{F}|$ is small).

- **Constructing linear PCPs with strong soundness.** As noted before, constructing multi-theorem designated-verifier SNARGs can be quite challenging. In [BCI+13], this is handled at the information-theoretic level (by constructing interactive proof systems satisfying a notion of "strong" or "reusable" soundness) and at the cryptographic level (by introducing strengthened definitions of linear-only encryption). A key limitation in their approach is that the information-theoretic construction of two-round interactive proof systems again requires linear PCPs over superpolynomial-sized fields. As we discuss in Remark 4.26, using large fields incurs additional overhead when using lattice-based encryption schemes to compiler linear PCPs to preprocessing

SNARGs. In this work, we show how to apply soundness amplification to standard linear PCPs with constant soundness error against linearly-bounded provers (and which do not necessarily satisfy strong soundness) to obtain strong, statistically-sound linear PCPs against affine-bounded provers over polynomial-size fields. Coupled with our direct compilation of linear PCPs to preprocessing SNARGs, we obtain multi-theorem designated-verifier SNARGs.

We describe our construction of strong statistically sound linear PCPs against affine provers from linear PCPs with constant soundness error against linear provers in Section 4.3. Applying our transformation to linear PCPs based on the Walsh-Hadamard code [ALM$^+$92] as well as those based on quadratic-span programs (QSPs) [GGPR13], we obtain two linear PCPs with strong statistical soundness against affine provers over polynomial-size fields.

From linear PCPs to preprocessing SNARGs. The primary tool we use construction of preprocessing SNARGs from linear PCPs is a new cryptographic primitive we call linear-only *vector encryption.* A vector encryption scheme is an encryption scheme where the plaintexts are vectors of ring (or field) elements. Next, we extend the notion of linear-only encryption [BCI$^+$13] to the context of vector encryption. We say that a vector encryption scheme is linear-only if the only homomorphisms it supports is addition (and scalar multiplication) of vectors.

Our new notion of linear-only vector encryption gives an immediate method of compiling an k-query linear PCP (over a finite field $\mathbb{F}$) into a designated-verifier SNARG. The construction works as follows. In a k-query linear PCP over $\mathbb{F}$, the verifier's query can be written as a matrix $\mathbf{Q} \in \mathbb{F}^{m \times k}$ where m is the query length of the linear PCP. The linear PCP oracle's response is $\mathbf{Q}^T \boldsymbol{\pi}$ where $\boldsymbol{\pi} \in \mathbb{F}^m$ is the proof. To compile this linear PCP into a preprocessing SNARG, we use a linear-only vector encryption scheme with plaintext space $\mathbb{F}^k$. The setup algorithm takes the verifier's query matrix $\mathbf{Q}$ (which is *independent* of the statement being proved) and encrypts each row of $\mathbf{Q}$ using the vector encryption scheme. The key observation is that the product $\mathbf{Q}^T \boldsymbol{\pi}$ is a linear combination of the rows of $\mathbf{Q}$. Thus, the prover can homomorphically compute an encryption of $\mathbf{Q}^T \boldsymbol{\pi}$. To check the proof, the verifier decrypts to obtain the prover's responses and then invokes the decision algorithm for the underlying linear PCP. Soundness is ensured by the linear-only property of the underlying vector encryption scheme. The advantage of linear-only vector encryption (as opposed to standard linear-only encryption) is that the prover is constrained to evaluating a single linear function on *all* of the query vectors simultaneously. This insight enables us to remove the extra consistency check introduced in [BCI$^+$13], and thus, avoids the soundness penalty $O(1/|\mathbb{F}|)$ incurred by the consistency check.[6] Consequently, we can instantiate our transformation with statistically-sound linear PCPs over *any* finite field $\mathbb{F}$. We describe our construction in Section 4.4.

[6]This is the main difference between our approach and that taken in [BCI$^+$13]. By making the *stronger* assumption of linear-only *vector encryption*, we avoid the need for an extra consistency check, thus allowing for a *direct* compilation from linear PCPs to SNARGs. In contrast, [BCI$^+$13] relies on the weaker assumption of linear-only encryption, but requires an extra step of first constructing a two-message linear interactive proof (incorporating the consistency check) from the linear PCP.

New lattice-based SNARG candidates. We then conjecture that the Regev-based [Reg05] encryption scheme of Peikert, Vaikuntanathan, and Waters [PVW08] is a secret-key linear-only vector encryption scheme over $\mathbb{Z}_p^k$ where p is a prime whose bit-length is polynomial in the security parameter λ. Then, applying our generic compiler from linear PCPs to SNARGs (Construction 4.14) to our new linear PCP constructions over polynomial-size fields $\mathbb{Z}_p$, we obtain a lattice-based construction of a designated-verifier SNARG (for Boolean circuit satisfiability) in the preprocessing model.[7] Specifically, starting with a QSP-based linear PCP [GGPR13], we obtain the first lattice-based SNARG that is quasi-optimally succinct (i.e., proof size $\widetilde{O}(\lambda)$ to achieve soundness error $2^{-\lambda}$ against 2^{λ}-size provers). Direct instantiation of the Bitansky et al. construction with a Regev-based candidate for linear-only encryption yields a SNARG with proof size $\widetilde{O}(\lambda^2)$ in order to achieve soundness $2^{-\lambda}$ against provers of size 2^{λ} (Remark 4.26). Thus, for Boolean circuit satisfiability, using lattice-based linear-only *vector encryption* provides concrete advantages over vanilla linear-only encryption.

4.2 Succinct Non-Interactive Arguments

We now review the definition of succinct non-interactive argument (SNARG) systems. We specialize our definitions to the problem of Boolean circuit satisfiability.

Definition 4.1 (Succinct Non-Interactive Arguments)**.** Let $\mathcal{C} = \{C_n\}_{n\in\mathbb{N}}$ be a family of Boolean circuits. A succinct non-interactive argument (SNARG) for the relation $\mathcal{R}_{\mathcal{C}}$ (and associated language $\mathcal{L}_{\mathcal{C}}$) is a tuple of algorithms $\Pi_{\mathsf{SNARG}} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ defined as follows:

- $\mathsf{Setup}(1^\lambda, 1^n) \to (\sigma, \tau)$: On input the security parameter λ and the circuit family parameter n, the setup algorithm outputs a common reference string σ and a verification state τ.
- $\mathsf{Prove}(\sigma, \mathbf{x}, \mathbf{w}) \to \pi$: On input the reference string σ, a statement $\mathbf{x}$, and a witness $\mathbf{w}$, the prove algorithm outputs a proof π.
- $\mathsf{Verify}(\tau, \mathbf{x}, \pi) \to \{0, 1\}$: On input the verification state τ, a statement $\mathbf{x}$, and a proof π, the verification algorithm outputs 1 if it "accepts" the proof, and 0 otherwise.

Moreover, Π_{SNARG} satisfies the following properties:

- **Completeness:** For all $n \in \mathbb{N}$ and $(\mathbf{x}, \mathbf{w}) \in \mathcal{R}_{C_n}$,

$$\Pr[(\sigma, \tau) \leftarrow \mathsf{Setup}(1^\lambda, 1^n); \pi \leftarrow \mathsf{Prove}(\sigma, \mathbf{x}, \mathbf{w}) : \mathsf{Verify}(\tau, \mathbf{x}, \pi) = 1] = 1.$$

- **Soundness:** Depending on the notion of soundness:

[7]While it would be preferable to obtain a construction based on the hardness of standard lattice assumptions like learning with errors (LWE) [Reg05], the separation results of Gentry and Wichs [GW11] suggest that stronger, non-falsifiable assumptions may be necessary to construct SNARGs for general NP languages.

  - **Adaptive Soundness:** For all $n \in \mathbb{N}$ and every polynomial-size prover P^*,

$$\Pr[(\sigma, \tau) \leftarrow \mathsf{Setup}(1^\lambda, 1^n); (\mathbf{x}, \pi) \leftarrow P^*(\sigma) : \mathsf{Verify}(\tau, \mathbf{x}, \pi) = 1 \text{ and } \mathbf{x} \notin \mathcal{L}_{C_n}] = \operatorname{negl}(\lambda).$$

  - **Non-adaptive Soundness:** For all $n \in \mathbb{N}$ and every polynomial-size prover P^*, and all statements $\mathbf{x} \notin \mathcal{L}_{C_n}$,

$$\Pr[(\sigma, \tau) \leftarrow \mathsf{Setup}(1^\lambda, 1^n); \pi \leftarrow P^*(\sigma, \mathbf{x}) : \mathsf{Verify}(\tau, \mathbf{x}, \pi) = 1] = \operatorname{negl}(\lambda).$$

- **Succinctness:** Depending on the notion of succinctness:
  - **Fully Succinct:** There exists a universal polynomial p (independent of $\mathcal{C}$) such that Setup runs in time $p(\lambda + \log |C_n|)$, Verify runs in time $p(\lambda + |\mathbf{x}| + \log |C_n|)$, and the length of the proof output by Prove is bounded by $p(\lambda + \log |C_n|)$.
  - **Preprocessing:** There exists a universal polynomial p (independent of $\mathcal{C}$) such that Setup runs in time $p(\lambda + |C_n|)$, Verify runs in time $p(\lambda + |\mathbf{x}| + \log |C_n|)$ and the length of the proof output by Prove is bounded by $p(\lambda + \log |C_n|)$.

Before proceeding, we give some intuition on the different soundness and succinctness notions. A SNARG is *adaptive* if the prover can choose the statement after seeing the reference string σ; otherwise, it is *non-adaptive*. Next, a SNARG is *fully-succinct* if the setup algorithm is efficient (runs in time polylogarithmic in the size of the circuit); otherwise, the SNARG is a *preprocessing* SNARG. For both fully-succinct as well as preprocessing SNARGs, the verifier's runtime and the length of the proof grow *polylogarithmically* in the size of the underlying circuit.

Public vs. designated verifiability. A SNARG is *publicly verifiable* if the verification state τ is allowed to be public. Alternatively, a *designated-verifier* SNARG is one where security only holds if τ remains secret. In this work, we focus on constructing designated-verifier SNARGs.

Multi-theorem SNARGs. A useful property for SNARGs to have is the ability to *reuse* the same reference string σ for multiple proofs. This is particularly important in the case of preprocessing SNARGs where an expensive precomputation stage is needed to construct the reference string. This multi-theorem setting can be modeled by imposing a stronger requirement where soundness should hold even if the adversary has access to a proof verification oracle. While this stronger soundness requirement follows immediately if the SNARG system is publicly verifiable, the same is not true in the designated-verifier setting. In fact, by issuing carefully crafted queries to the proof verification oracle, a dishonest prover can potentially learn information about the secret verification state, and thus, compromise the soundness of the SNARG system.

In this work, we construct a compiler that combines an information-theoretic primitive (linear PCPs) with a cryptographic primitive (linear-only vector encryption) to obtain a designated-verifier SNARG. Correspondingly, we address this core issue of reusability at both the information-theoretic level (via a stronger soundness definition) as well as at the cryptographic level (via a stronger notion of linear-only encryption). We give more details in Section 4.4.3.

Other properties. In addition to the basic properties outlined above, there are numerous additional notions that pertain to SNARGs. In some applications, the soundness requirement is strengthened to an extractability property—that is, whenever a prover is able to convince the verifier that a statement $\mathbf{x}$ is in the language, there is also an (efficient) extraction algorithm that is able to extract a witness $\mathbf{w}$ for $\mathbf{x}$ such that $(\mathbf{x}, \mathbf{w}) \in \mathcal{R}$. This yields succinct arguments of knowledge (SNARKs).

Another commonly considered notion in the context of succinct non-interactive arguments is zero-knowledge. As shown by Bitansky et al. [BCCT12], preprocessing SNARKs can be combined with (possibly non-succinct) non-interactive arguments of knowledge (NIZK arguments) to obtain zero-knowledge SNARKs (i.e., "zkSNARKs") in the preprocessing model. We also refer to [Gro10, Lip12, PHGR13, BCG$^+$13, GGPR13, BCI$^+$13, Lip13, BCTV14, DFGK14, Lip16] for constructions and implementations of zero-knowledge SNARKs. In the context of constructing zero-knowledge SNARGs from linear PCPs, we note that we can leverage the ideas from [BCI$^+$13] to realize succinct zero-knowledge arguments. Since this is not the primary focus of this work, we will not consider zero-knowledge in the remainder of this work.

4.3 Linear PCPs

In this section, we review the definition of linear probabilistically checkable proofs (linear PCPs). In a k-query linear PCP system for a binary relation $\mathcal{R}$ over a finite field $\mathbb{F}$, the proof consists of a vector $\boldsymbol{\pi} \in \mathbb{F}^m$ and the PCP oracle is restricted to computing a linear function on the verifier's query vector. Specifically, on input a query matrix $\mathbf{Q} \in \mathbb{F}^{m \times k}$, the PCP oracle responds with $\mathbf{y} = \mathbf{Q}^T \boldsymbol{\pi} \in \mathbb{F}^k$. We now give a formal definition adapted from [BCI$^+$13].

Definition 4.2 (Linear PCPs [BCI$^+$13])**.** Let $\mathcal{R}$ be a binary relation, $\mathbb{F}$ be a finite field, $\mathcal{P}$ be a deterministic prover algorithm, and $\mathcal{V}$ be a probabilistic oracle verification algorithm. Then, $(\mathcal{P}, \mathcal{V})$ is a k-query linear PCP for $\mathcal{R}$ over $\mathbb{F}$ with soundness error ε and query length m if it satisfies the following requirements:

- **Syntax:** For a vector $\boldsymbol{\pi} \in \mathbb{F}^m$, the verification algorithm $\mathcal{V}^{\boldsymbol{\pi}} = (\mathcal{Q}, \mathcal{D})$ consists of an input-oblivious probabilistic query algorithm $\mathcal{Q}$ and a deterministic decision algorithm $\mathcal{D}$. The query algorithm $\mathcal{Q}$ generates a query matrix $\mathbf{Q} \in \mathbb{F}^{m \times k}$ (independently of the statement $\mathbf{x}$) and some state information st. The decision algorithm $\mathcal{D}$ takes the statement $\mathbf{x}$, the state st, and the response vector $\mathbf{y} = \mathbf{Q}^T \boldsymbol{\pi} \in \mathbb{F}^k$ and either "accepts" or "rejects."

- **Completeness:** For every $(\mathbf{x}, \mathbf{w}) \in \mathcal{R}$, the output of $\mathcal{P}(\mathbf{x}, \mathbf{w})$ is a vector $\boldsymbol{\pi} \in \mathbb{F}^m$ such that $\mathcal{V}^{\boldsymbol{\pi}}(\mathbf{x})$ accepts with probability 1.

- **Soundness:** For all $\mathbf{x}$ where $(\mathbf{x}, \mathbf{w}) \notin \mathcal{R}$ for all $\mathbf{w}$ and for all vectors $\boldsymbol{\pi}^* \in \mathbb{F}^m$, the probability that $\mathcal{V}^{\boldsymbol{\pi}^*}(\mathbf{x})$ accepts is at most ε.

We say that $(\mathcal{P}, \mathcal{V})$ is an input-oblivious k-query linear PCP for $\mathcal{R}$ over $\mathbb{F}$ with *knowledge error* ε and query length d if $(\mathcal{P}, \mathcal{V})$ satisfies the properties above, but the soundness property is replaced by the following (stronger) knowledge property:

- **Knowledge:** There exists a knowledge extractor $\mathcal{E}$ such that for every vector $\boldsymbol{\pi}^* \in \mathbb{F}^d$, if $\mathcal{V}^{\boldsymbol{\pi}^*}(\mathbf{x})$ accepts with probability at least ε, then $\mathcal{E}^{\boldsymbol{\pi}^*}(\mathbf{x})$ outputs $\mathbf{w}$ such that $(\mathbf{x}, \mathbf{w}) \in \mathcal{R}$. As with soundness, we can correspondingly define a notion of *knowledge against affine strategies.*

We say that $(\mathcal{P}, \mathcal{V})$ is statistically sound if $\varepsilon(\kappa) = \text{negl}(\kappa)$, where κ is a statistical security parameter.

Soundness against affine provers. In Definition 4.2, we only required soundness (correspondingly, knowledge) to hold against provers that employ a *linear* strategy, and not an *affine* strategy. Our construction of SNARGs (Section 4.4), will require the stronger property that soundness holds against provers using an affine strategy—that is, a strategy which can be described by a tuple $\mathbf{\Pi} = (\boldsymbol{\pi}, \mathbf{b})$ where $\boldsymbol{\pi} \in \mathbb{F}^m$ represents a linear function and $\mathbf{b} \in \mathbb{F}^k$ represents an affine shift. Then, on input a query matrix $\mathbf{Q} \in \mathbb{F}^{m \times k}$, the response vector is constructed by evaluating the affine relation $\mathbf{y} = \mathbf{Q}^T \boldsymbol{\pi} + \mathbf{b}$. We now define this stronger notion of soundness against an affine prover.

Definition 4.3 (Soundness Against Affine Provers)**.** Let $\mathcal{R}$ be a relation and $\mathbb{F}$ be a finite field. A linear PCP $(\mathcal{P}, \mathcal{V})$ is a k-query linear PCP for $\mathcal{R}$ over $\mathbb{F}$ with soundness error ε against affine provers if it satisfies the requirements in Definition 4.2 with the following modifications:

- **Syntax:** For any affine function $\mathbf{\Pi} = (\boldsymbol{\pi}, \mathbf{b})$, the verification algorithm $\mathcal{V}^{\mathbf{\Pi}}$ is still specified by a tuple $(\mathcal{Q}, \mathcal{D})$. Algorithms $\mathcal{Q}, \mathcal{D}$ are the same as in Definition 4.2, except that the response vector $\mathbf{y}$ computed by the PCP oracle is an affine function $\mathbf{y} = \mathbf{Q}^T \boldsymbol{\pi} + \mathbf{b} \in \mathbb{F}^k$ of the query matrix $\mathbf{Q}$ rather than a linear function.

- **Soundness against affine provers:** For all $\mathbf{x}$ where $(\mathbf{x}, \mathbf{w}) \notin \mathcal{R}$ for all $\mathbf{w}$, and for all affine functions $\mathbf{\Pi}^* = (\boldsymbol{\pi}^*, \mathbf{b}^*)$ where $\boldsymbol{\pi}^* \in \mathbb{F}^m$ and $\mathbf{b}^* \in \mathbb{F}^k$, the probability that $\mathcal{V}^{\mathbf{\Pi}^*}(\mathbf{x})$ accepts is at most ε.

Algebraic complexity. There are many ways one can measure the complexity of a linear PCP system such as the number of queries or the number of field elements in the verifier's queries. Another important metric also considered in [BCI+13] is the *algebraic complexity* of the verifier. In particular, the verifier's query algorithm $\mathcal{Q}$ and decision algorithm $\mathcal{D}$ can both be viewed as multivariate polynomials (equivalently, arithmetic circuits) over the finite field $\mathbb{F}$. We say that the query algorithm $\mathcal{Q}$ has degree d_Q if the output of $\mathcal{Q}$ can be computed by a collection of multivariate polynomials of maximum degree d_Q in the verifier's choice of randomness. Similarly, we say that the decision algorithm $\mathcal{D}$ has degree d_D if the output of $\mathcal{D}$ can be computed by a multivariate polynomial of maximum degree d_D in the prover's response and the verification state.

Strong soundness. In this work, we focus on constructing designated-verifier SNARGs. An important consideration that arises in the design of designated-verifier SNARGs is whether the same reference string σ can be reused across many proofs. This notion is formally captured by stipulating that the SNARG system remains sound even if the prover has access to a proof-verification oracle. While this property naturally follows from soundness if the SNARG system is publicly-verifiable, the same is not true in the designated-verifier setting. Specifically, in the designated-verifier setting, soundness is potentially compromised if the responses of the proof-verification oracle is correlated with the verifier's secrets. Thus, to construct a *multi-theorem* designated-verifier SNARG, we require linear PCPs with a stronger soundness property, which we state below.

Definition 4.4 (Strong Soundness [BCI$^+$13])**.** A k-query linear PCP $(\mathcal{P}, \mathcal{V})$ with soundness error ε satisfies strong soundness if for every input $\mathbf{x}$ and every proof $\boldsymbol{\pi}^* \in \mathbb{F}^m$, either $\mathcal{V}^{\boldsymbol{\pi}^*}(\mathbf{x})$ accepts with probability 1 or with probability at most ε.

Roughly speaking, in a linear PCP that satisfies strong soundness, *every* linear PCP prover either causes the linear PCP verifier to accept with probability 1 or with bounded probability. This prevents correlation attacks where a malicious prover is able to submit (potentially malformed) proofs to the verifier and seeing responses that are correlated with the verifier's secrets. We can define an analogous notion of strong soundness against affine provers.

4.3.1 Constructing Linear PCPs with Strong Soundness

A natural first question is whether linear PCPs with strong soundness against affine provers exist. Previously, Bitansky et al. [BCI$^+$13] gave two constructions of algebraic linear PCPs for Boolean circuit satisfaction problems: one from the Hadamard-based PCP of Arora et al. [ALM$^+$92], and another from the quadratic span programs (QSPs) of Gennaro et al. [GGPR13]. In both cases, the linear PCP is defined over a finite field $\mathbb{F}$ and the soundness error scales inversely with $|\mathbb{F}|$. Thus, the linear PCP is statistically sound only if $|\mathbb{F}|$ is *superpolynomial* in the (statistical) security parameter. If we use the Bitansky et al. [BCI$^+$13] compiler in conjunction with traditional lattice-based encryption schemes (c.f., [Reg05]) to compiler the linear PCP into a preprocessing SNARG, using large fields incurs a cost in both the prover complexity as well as the proof size. For example, using fields of size $2^{\Omega(\kappa)}$ (needed, for example, to achieve statistical soundness $2^{-\kappa}$) incurs a *multiplicative* overhead $\Omega(\kappa)$ in both the prover complexity as well as the proof size.

In this section, we show that starting from any linear PCP with *constant* soundness error against linear provers, we can generically obtain a linear PCP that is statistically sound against affine provers. Our generic transformation consists of two steps. The first is a standard soundness amplification step where the verifier makes κ sets of independently generated queries (of the underlying linear PCP scheme) to the PCP oracle, where κ is a statistical security parameter. The verifier accepts only if the prover's responses to all κ sets of queries are valid. Since the queries are independently generated, each of the κ sets of responses (for a false statement) is accepted with probability at most ε (where ε is proportional to $1/|\mathbb{F}|$). Thus, an honest verifier only accepts with probability at most $\varepsilon^\kappa = \text{negl}(\kappa)$.

However, this basic construction does not achieve *strong* soundness against *affine* provers. For instance, a malicious linear PCP prover using an affine strategy could selectively corrupt the responses to exactly one set of queries (by applying an affine shift to its response for a single set of queries). When this selective corruption is applied to a well-formed proof and the verifier's decision algorithm has low algebraic complexity, then the verifier will accept with some noticeable probability less than 1, which is sufficient to break strong soundness. To address this problem, the verifier first applies a (secret) random linear shift to its queries before submitting them to the PCP oracle. This ensures that any prover using an affine strategy with a non-zero offset will corrupt its responses to *every* set of queries, and the proof will be rejected with overwhelming probability. We now describe our generic construction in more detail.

Construction 4.5 (Statistically-Sound Linear PCPs over Small Fields)**.** Fix a statistical security parameter κ. Let $\mathcal{R}$ be a binary relation, $\mathbb{F}$ be a finite field, and $\left(\mathcal{P}^{(\text{weak})}, \mathcal{V}^{(\text{weak})}\right)$ be an k-query linear PCP for $\mathcal{R}$, where $\mathcal{V}^{(\text{weak})} = \left(\mathcal{Q}^{(\text{weak})}, \mathcal{D}^{(\text{weak})}\right)$. Define the (κk)-query linear PCP $(\mathcal{P}, \mathcal{V})$ where $\mathcal{V} = (\mathcal{Q}, \mathcal{D})$ as follows:

- **Prover's Algorithm $\mathcal{P}$:** On input $(\mathbf{x}, \mathbf{w})$, output $\mathcal{P}^{(\text{weak})}(\mathbf{x}, \mathbf{w})$.
- **Verifier's Query Algorithm $\mathcal{Q}$:** The query algorithm invokes $\mathcal{Q}^{(\text{weak})}$ a total of κ times to obtain (independent) query matrices $\mathbf{Q}_1, \ldots, \mathbf{Q}_\kappa \in \mathbb{F}^{m \times k}$ and state information $\mathsf{st}_1, \ldots, \mathsf{st}_\kappa$. It constructs the concatenated matrix $\mathbf{Q} = [\mathbf{Q}_1 | \mathbf{Q}_2 | \cdots | \mathbf{Q}_\kappa] \in \mathbb{F}^{m \times \kappa k}$. Finally, it chooses a random matrix $\mathbf{Y} \overset{\text{R}}{\leftarrow} \mathbb{F}^{\kappa k \times \kappa k}$ and outputs the queries $\mathbf{Q}' = \mathbf{Q}\mathbf{Y}$ and state $\mathsf{st} = (\mathsf{st}_1, \ldots, \mathsf{st}_\kappa, \mathbf{Y}')$ where $\mathbf{Y}' = (\mathbf{Y}^T)^{-1}$.
- **Verifier's Decision Algorithm $\mathcal{D}$:** On input the statement $\mathbf{x}$, the prover's response vector $\mathbf{a}' \in \mathbb{F}^{\kappa k}$ and the state $\mathsf{st} = (\mathsf{st}_1, \ldots, \mathsf{st}_\kappa, \mathbf{Y}')$, the verifier's decision algorithm computes $\mathbf{a} = \mathbf{Y}'\mathbf{a}' \in \mathbb{F}^{\kappa k}$. Next, it writes $\mathbf{a}^T = [\mathbf{a}_1^T | \mathbf{a}_2^T | \cdots | \mathbf{a}_\kappa^T]$ where each $\mathbf{a}_i \in \mathbb{F}^k$ for $i \in [\kappa]$. Then, for each $i \in [\kappa]$, the verifier runs $\mathcal{D}^{(\text{weak})}(\mathbf{x}, \mathbf{a}_i, \mathsf{st}_i)$ and accepts if $\mathcal{D}^{(\text{weak})}$ accepts for all κ instances. It rejects otherwise.

Theorem 4.6 (Statistically-Sound Linear PCPs over Small Fields)**.** *Fix a statistical security parameter κ. Let $\mathcal{R}$ be a binary relation, $\mathbb{F}$ be a finite field, and $(\mathcal{P}^{(\text{weak})}, \mathcal{V}^{(\text{weak})})$ be a strongly-sound*

k-query linear PCP for $\mathcal{R}$ with constant soundness error $\varepsilon \in [0,1)$ against linear provers. If $|\mathbb{F}| > d_D$, where d_D is the degree of the verifier's decision algorithm $\mathcal{D}^{(\text{weak})}$, then the linear PCP $(\mathcal{P}, \mathcal{V})$ from Construction 4.5 is a (κk)-query linear PCP for $\mathcal{R}$ with strong statistical soundness against affine provers.

Proof. Completeness follows immediately from completeness of the underlying linear PCP system, so it suffices to check that the linear PCP is statistically sound against affine provers. Take any statement $\mathbf{x}$, and consider an affine prover strategy $\mathbf{\Pi}^* = (\boldsymbol{\pi}^*, \mathbf{b}^*)$, where $\boldsymbol{\pi}^* \in \mathbb{F}^m$ and $\mathbf{b}^* \in \mathbb{F}^{\kappa k}$. We consider two cases:

- Suppose $\mathbf{b}^* \neq 0^{\kappa k}$. Then, the decision algorithm $\mathcal{D}$ starts by computing

$$\mathbf{a} = \mathbf{Y}'\mathbf{a}' = \mathbf{Y}'(\mathbf{Y}^T \mathbf{Q}^T \boldsymbol{\pi}^* + \mathbf{b}^*) = \mathbf{Q}^T \boldsymbol{\pi}^* + \mathbf{Y}'\mathbf{b}^* \in \mathbb{F}^{\kappa k}.$$

  Next, the verifier invokes the decision algorithm $\mathcal{D}^{(\text{weak})}$ for the underlying linear PCP on the components of $\mathbf{a}$. By assumption, $\mathcal{D}^{(\text{weak})}$ is a polynomial of maximum degree d_D in the components of the prover's response $\mathbf{a}$, and by extension, in the components of the matrix $\mathbf{Y}'$. Since $\mathbf{b}^*$ is non-zero, this is a non-zero polynomial in the $\mathbf{Y}'$. Since $\mathbf{Y}'$ is sampled uniformly at random (and independently of $\mathbf{Q}, \boldsymbol{\pi}^*, \mathbf{b}^*$), by the Schwartz-Zippel lemma (Lemma 2.2), $\mathcal{D}^{(\text{weak})}(\mathbf{x}, \mathbf{a}_i, \mathsf{st}_i)$ accepts with probability at most $d_D / |\mathbb{F}|$ for each $i \in [\kappa]$. Thus, the verifier rejects with probability at least $1 - (d_D / |\mathbb{F}|)^\kappa = 1 - \text{negl}(\kappa)$ since $|\mathbb{F}| > d_D$.

- Suppose $\mathbf{b}^* = 0^{\kappa k}$. Then, the prover's strategy is a linear function $\boldsymbol{\pi}^*$. Since the underlying PCP satisfies strong soundness against linear provers, it follows that $\mathcal{D}^{(\text{weak})}(\mathbf{a}_i, \mathsf{st}_i)$ either accepts with probability 1 or with probability at most ε. In the former case, $\mathcal{D}$ also accepts with probability 1. In the latter case, because the verifier constructs the κ queries to the underlying linear PCP independently, $\mathcal{D}$ accepts with probability at most $\varepsilon^\kappa = \text{negl}(\kappa)$. We conclude that the proof system $(\mathcal{P}, \mathcal{V})$ satisfies strong soundness against affine provers. □

Remark 4.7 (Efficiency of Transformation)**.** Construction 4.5 incurs a κ overhead in the number of queries made to the PCP oracle and a quadratic overhead in the algebraic complexity of the verifier's decision algorithm. Specifically, the degree of the verifier's decision algorithm in Construction 4.5 is d_D^2, where d_D is the degree of the verifier's decision algorithm in the underlying linear PCP. The quadratic factor arises from undoing the linear shift in the prover's responses before applying the decision algorithm of the underlying linear PCP. In many existing linear PCP systems, the verifier's decision algorithm has low algebraic complexity (e.g., $d_D = 2$ for both the Hadamard-based linear PCP [ALM+92] as well as the QSP-based linear PCP [GGPR13]), so the verifier's algebraic complexity only increases modestly. However, the increase in degree means that we can no longer leverage pairing-based linear-only one-way encodings [BCI+13] to construct publicly-verifiable SNARGs (since these techniques only apply when the algebraic complexity of the verifier's decision algorithm is exactly 2). No such limitations apply in the designated-verifier setting.

Remark 4.8 (Comparison with [BCI+13, Lemma C.3])**.** Bitansky et al. [BCI+13, Lemma C.3] previously showed that any algebraic linear PCP over a finite field $\mathbb{F}$ with soundness error ε is also strongly sound with soundness error $\varepsilon' = \max\left\{\varepsilon, \frac{d_Q d_D}{|\mathbb{F}|}\right\}$. For sufficiently large fields $\mathbb{F}$ (e.g., when $|\mathbb{F}|$ is superpolynomial), statistical soundness implies strong statistical soundness. However, when $|\mathbb{F}|$ is polynomial, then their lemma is insufficient to argue strong statistical soundness of the underlying linear PCP. In contrast, using our construction (Construction 4.5), any linear PCP with just constant soundness against linear provers can be used to construct an algebraic linear PCP with strong statistical soundness against affine provers (at the cost of increasing the query complexity and the verifier's algebraic complexity).

Concrete instantiations. Applying Construction 4.5 to the algebraic linear PCPs for Boolean circuit satisfaction of Bitansky et al. [BCI+13], we obtain statistically-sound linear PCPs for Boolean circuit satisfaction over small finite fields. In the following, fix a (statistical) security parameter κ and let C be a Boolean circuit of size s.

- Starting from the Hadamard-based PCP of Arora et al. [ALM+92] over a finite field $\mathbb{F}$, there exists a 3-query linear PCP with strong soundness error $2/|\mathbb{F}|$. The algebraic complexity of the decision algorithm for this PCP is $d_D = 2$. Applying Construction 4.5 and working over any finite field where $|\mathbb{F}| > 2$, we obtain a (3κ)-query linear PCP with strong statistical soundness against affine provers and where queries have length $O(s^2)$.
- Starting from the quadratic span programs of Gennaro et al. [GGPR13], there exists a 3-query linear PCP over any (sufficiently large) finite field $\mathbb{F}$ with strong soundness error $O(s/|\mathbb{F}|)$. The algebraic complexity of the decision algorithm for this PCP is $d_D = 2$. Applying Construction 4.5 and working over a sufficiently large finite field of size $|\mathbb{F}| = \widetilde{O}(s)$, we obtain a (3κ)-query linear PCP with strong statistical soundness against affine provers where queries have length $O(s)$.

4.4 SNARGs from Linear-Only Vector Encryption

In this section, we introduce the notion of a linear-only vector encryption scheme. We then show how linear-only vector encryption can be directly combined with the linear PCPs from Section 4.3 to obtain multi-theorem designated-verifier preprocessing SNARGs in the standard model. Then, we describe a candidate instantiation of our linear-only vector encryption scheme using the LWE-based encryption scheme of Peikert, Vaikuntanathan, and Waters [PVW08]. Our notion of linear-only vector encryption is a direct generalization of the notion of linear-only encryption first introduced by Bitansky et al. [BCI+13].

4.4.1 Linear-Only Vector Encryption

A vector encryption scheme is an encryption scheme where the message space is a vector of ring elements. In this section, we take $\mathbb{Z}_p$ as the underlying ring and $\mathbb{Z}_p^k$ as the message space (for some dimension k). We introduce the basic schema below:

Definition 4.9 (Vector Encryption Scheme over $\mathbb{Z}_p^k$)**.** A secret-key vector encryption scheme over $\mathbb{Z}_p^k$ consists of a tuple of algorithms $\Pi_{\mathsf{venc}} = (\mathsf{Setup}, \mathsf{Encrypt}, \mathsf{Decrypt})$ with the following properties:

- $\mathsf{Setup}(1^\lambda, 1^k) \to \mathsf{sk}$: The setup algorithm takes as input the security parameter λ and the dimension k of the message space and outputs the secret key sk.
- $\mathsf{Encrypt}(\mathsf{sk}, \mathbf{v}) \to \mathsf{ct}$: The encryption algorithm takes as input the secret key sk and a message vector $\mathbf{v} \in \mathbb{Z}_p^k$ and outputs a ciphertext ct.
- $\mathsf{Decrypt}(\mathsf{sk}, \mathsf{ct}) \to \mathbb{Z}_p^k \cup \{\perp\}$: The decryption algorithm takes as input the secret key sk and a ciphertext ct and either outputs a message vector $\mathbf{v} \in \mathbb{Z}_p^k$ or a special symbol $\perp$ (to denote an invalid ciphertext).

We can define the usual notions of correctness and CPA-security (Definition 2.1) for a vector encryption scheme. Next, we say that a vector encryption scheme over $\mathbb{Z}_p^k$ is additively homomorphic if given encryptions $\mathsf{ct}_1, \mathsf{ct}_2$ of two vectors $\mathbf{v}_1, \mathbf{v}_2 \in \mathbb{Z}_p^k$, respectively, there is a public operation[8] that allows one to compute an encryption ct_{12} of the (component-wise) sum $\mathbf{v}_1 + \mathbf{v}_2 \in \mathbb{Z}_p^k$. Note that additively homomorphic vector encryption can be constructed directly from any additively homomorphic encryption scheme by simply encrypting each component of the vector separately. However, when leveraging vector encryption to build efficient SNARGs, we require that our encryption scheme satisfies a more restrictive homomorphism property. We define this now.

Linear-only vector encryption. Intuitively, we say that a vector encryption scheme is *linear-only* if the only homomorphic operations the adversary can perform on ciphertexts is evaluate affine functions on the underlying plaintext vectors. At a high level, we model this property by requiring that whenever an adversary $\mathcal{A}$ constructs a ciphertext ct' from a collection of ciphertexts $\mathsf{ct}_1, \ldots, \mathsf{ct}_m$, there is an efficient extractor $\mathcal{E}$ that can produce an affine function that "explains" the ciphertext. We give the formal definition below (adapted from the corresponding definitions in [BCI$^+$13]):

Definition 4.10 (Linear-Only Vector Encryption [BCI$^+$13, adapted])**.** Fix a security parameter λ. A secret-key vector encryption scheme $\Pi_{\mathsf{venc}} = (\mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt})$ for a message space $\mathbb{Z}_p^k$ is *linear-only* if for all efficient adversaries $\mathcal{A}$, there exists an efficient extractor $\mathcal{E}$ such that for all auxiliary inputs $z \in \{0,1\}^\lambda$, and any plaintext generation algorithms $\mathcal{M}$ (on input 1^k, algorithm $\mathcal{M}$ outputs a

[8] In principle, homomorphic evaluation might require additional *public* parameters to be published by the setup algorithm. For simplicity of presentation, we will assume that no additional parameters are required, but all of our notions extend to the setting where the setup algorithm outputs a public evaluation key.

vector in $\mathbb{Z}_p^k$), we have that for $\mathsf{sk} \leftarrow \mathsf{KeyGen}(1^\lambda, 1^k)$, $(\mathbf{v}_1, \ldots, \mathbf{v}_m) \leftarrow \mathcal{M}(1^k)$, $\mathsf{ct}_i \leftarrow \mathsf{Encrypt}(\mathsf{sk}, \mathbf{v}_i)$ for all $i \in [m]$, $\mathsf{ct}' \leftarrow \mathcal{A}(\{\mathsf{ct}_i\}_{i\in[m]}\,; z)$, $(\boldsymbol{\pi}, \mathbf{b}) \leftarrow \mathcal{E}(\{\mathsf{ct}_i\}_{i\in[m]}\,; z)$, $\mathbf{v}' \leftarrow [\mathbf{v}_1|\mathbf{v}_2|\cdots|\mathbf{v}_m] \cdot \boldsymbol{\pi} + \mathbf{b}$,

$$\Pr[\mathsf{Decrypt}(\mathsf{sk}, \mathsf{ct}') \neq \mathbf{v}'] = \operatorname{negl}(\lambda).$$

Linear targeted malleability. As noted in [BCI+13], in many scenarios, a weaker variant of linear-only encryption called *linear targeted malleability* suffices for constructing preprocessing SNARGs. We give this definition below (adapted from the corresponding definition from [BSW12]).

Definition 4.11 (Linear Targeted Malleability [BSW12, adapted])**.** Fix a security parameter λ. A (secret-key) vector encryption scheme $\Pi_{\mathsf{venc}} = (\mathsf{Setup}, \mathsf{Encrypt}, \mathsf{Decrypt})$ for a message space $\mathbb{Z}_p^k$ satisfies *linear targeted malleability* if for all efficient adversaries $\mathcal{A}$ and plaintext generation algorithms $\mathcal{M}$ (on input 1^k, algorithm $\mathcal{M}$ outputs vectors in $\mathbb{Z}_p^k$), there exists a (possibly computationally unbounded) simulator $\mathcal{S}$ such that for any auxiliary input $z \in \{0,1\}^{\operatorname{poly}(\lambda)}$, the following two distributions are computationally indistinguishable:

Real Distribution:

1. $\mathsf{sk} \leftarrow \mathsf{Setup}(1^\lambda, 1^k)$
2. $(s, \mathbf{v}_1, \ldots, \mathbf{v}_m) \leftarrow \mathcal{M}(1^k)$
3. $\mathsf{ct}_i \leftarrow \mathsf{Encrypt}(\mathsf{sk}, \mathbf{v}_i)$ for all $i \in [m]$
4. $\mathsf{ct}' \leftarrow \mathcal{A}(\{\mathsf{ct}_i\}_{i\in[m]}\,; z)$ where $\mathsf{Decrypt}(\mathsf{sk}, \mathsf{ct}') \neq \perp$
5. Output $\left(\{\mathbf{v}_i\}_{i\in[m]}\,, s, \mathsf{Decrypt}(\mathsf{sk}, \mathsf{ct}')\right)$

Ideal Distribution:

1. $(s, \mathbf{v}_1, \ldots, \mathbf{v}_m) \leftarrow \mathcal{M}(1^k)$
2. $(\boldsymbol{\pi}, \mathbf{b}) \leftarrow \mathcal{S}(z)$ where $\boldsymbol{\pi} \in \mathbb{Z}_p^m$, $\mathbf{b} \in \mathbb{Z}_p^k$
3. $\mathbf{v}' \leftarrow [\mathbf{v}_1|\mathbf{v}_2|\cdots|\mathbf{v}_m] \cdot \boldsymbol{\pi} + \mathbf{b}$
4. Output $\left(\{\mathbf{v}_i\}_{i\in[m]}\,, s, \mathbf{v}_i'\right)$

Remark 4.12 (Multiple Ciphertexts)**.** Similar to [BSW12, BCI+13], we can also define a variant of linear-only vector encryption (Definition 4.10) and linear targeted malleability (Definition 4.11) where the adversary is allowed to output multiple ciphertexts $\mathsf{ct}'_1, \ldots, \mathsf{ct}'_m$. In this case, the extractor $\mathcal{E}$ or the simulator $\mathcal{S}$ should output an affine function $(\boldsymbol{\Pi}, \mathbf{B})$ where $\boldsymbol{\Pi} \in \mathbb{Z}_p^{m\times m}$ and $\mathbf{B} \in \mathbb{Z}_p^{k\times m}$ that "explains" the ciphertexts $\mathsf{ct}'_1, \ldots, \mathsf{ct}'_m$. However, the simple variant we have defined above where the adversary just outputs a single ciphertext is sufficient for our construction.

Remark 4.13 (Auxiliary Input Distributions)**.** In Definitions 4.11 and 4.10, the simulator $\mathcal{S}$ and the extractor $\mathcal{E}$, respectively, are required to succeed for all auxiliary inputs $z \in \{0,1\}^{\operatorname{poly}(\lambda)}$. This seems like a very strong requirement since z can be used to encode difficult problems that the simulator or extractor needs to solve in order to correctly simulate the output distribution [BCPR14]. However, the definitions can be relaxed to only consider "benign" auxiliary-input distributions for which the property holds. For instance, in many scenarios, it suffices that the auxiliary input z is a uniformly random string.

4.4.2 From Linear-Only Vector Encryption to Preprocessing SNARGs

In this section, we give our construction of preprocessing SNARGs from linear-only vector encryption. The analysis of our preprocessing SNARG (Theorems 4.15 and 4.17) follow analogously to the corresponding analysis in [BCI$^+$13, §6] and in Section 4.4.2. Thus, we only state our theorems here and defer to the analysis in [BCI$^+$13] for the full details.

Construction 4.14 (SNARG from Linear-Only Vector Encryption)**.** Fix a prime p (so the ring $\mathbb{Z}_p$ is a field), and let $\mathcal{C} = \{C_n\}_{n \in \mathbb{N}}$ be a family of arithmetic circuits over $\mathbb{Z}_p$.[9] Let $\mathcal{R}_\mathcal{C}$ be the relation associated with $\mathcal{C}$. Let $(\mathcal{P}, \mathcal{V})$ be an k-query input-oblivious linear PCP for $\mathcal{C}$, where the verifier $\mathcal{V} = (\mathcal{Q}, \mathcal{D})$ can be decomposed into a query-generation algorithm $\mathcal{Q}$ and a decision algorithm $\mathcal{D}$. Let $\Pi_{\mathsf{venc}} = (\mathsf{Setup}, \mathsf{Encrypt}, \mathsf{Decrypt})$ be a secret-key vector encryption scheme for $\mathbb{Z}_p^k$. Our single-theorem, designated-verifier SNARG $\Pi_{\mathsf{SNARG}} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ in the preprocessing model for $\mathcal{R}_\mathcal{C}$ is defined as follows:

- $\mathsf{Setup}(1^\lambda, 1^n) \to (\sigma, \tau)$: On input the security parameter λ and the circuit family parameter n, the setup algorithm first invokes the query algorithm $\mathcal{Q}$ for the linear PCP to obtain a query matrix $\mathbf{Q} \in \mathbb{Z}_p^{m \times k}$ and some state information st. Next, it generates a secret key for the vector encryption scheme $\mathsf{sk} \leftarrow \mathsf{Setup}(1^\lambda, 1^k)$. Then, it encrypts each row (an element of $\mathbb{Z}_p^k$) of the query matrix $\mathbf{Q}$. More specifically, for $i \in [m]$, let $\mathbf{q}_i \in \mathbb{Z}_p^k$ be the i^{th} row of $\mathbf{Q}$. Then, the setup algorithm computes ciphertexts $\mathsf{ct}_i \leftarrow \mathsf{Encrypt}(\mathsf{sk}, \mathbf{q}_i)$. Finally, the setup algorithm outputs the common reference string $\sigma = (\mathsf{ct}_1, \ldots, \mathsf{ct}_m)$ and the verification state $\tau = (\mathsf{sk}, \mathsf{st})$.

- $\mathsf{Prove}(\sigma, \mathbf{x}, \mathbf{w})$: On input a common reference string $\sigma = (\mathsf{ct}_1, \ldots, \mathsf{ct}_m)$, a statement $\mathbf{x}$, and a witness $\mathbf{w}$, the prover invokes the prover algorithm $\mathcal{P}$ for the linear PCP to obtain a vector $\boldsymbol{\pi} \in \mathbb{Z}_p^m$. Viewing $\mathsf{ct}_1, \ldots, \mathsf{ct}_m$ as vector encryptions of the rows of a query matrix $\mathbf{Q} \in \mathbb{Z}_p^{m \times k}$, the prover uses the linear homomorphic properties of Π_{venc} to homomorphically compute an encryption of the matrix vector product $\mathbf{Q}^T \boldsymbol{\pi}$. In particular, the prover homomorphically computes the sum $\mathsf{ct}' = \sum_{i \in [m]} \pi_i \cdot \mathsf{ct}_i$. The prover outputs the ciphertext ct' as its proof.

- $\mathsf{Verify}(\tau, \mathbf{x}, \pi)$: On input the (secret) verification state $\tau = (\mathsf{sk}, \mathsf{st})$, the statement $\mathbf{x}$, and the proof $\pi = \mathsf{ct}'$, the verifier decrypts the proof ct' using the secret key sk to obtain the prover's responses $\mathbf{a} \leftarrow \mathsf{Decrypt}(\mathsf{sk}, \mathsf{ct}')$. If $\mathbf{a} = \perp$, the verifier stops and outputs 0. Otherwise, it invokes the verification decision algorithm $\mathcal{D}$ on the statement $\mathbf{x}$, the responses $\mathbf{a}$, and the linear PCP verification state st to decide whether the proof is valid or not. The verification algorithm echoes the output of the decision algorithm.

Theorem 4.15 ([BCI$^+$13, Lemma 6.3, adapted])**.** *Fix a security parameter λ and a prime p. Let $\mathcal{C} = \{C_n\}_{n \in \mathbb{N}}$ be a family of arithmetic circuits over $\mathbb{F}_p$, $\mathcal{R}_\mathcal{C}$ be the relation associated with $\mathcal{C}$,*

[9]While we describe a SNARG for arithmetic circuit satisfiability (over $\mathbb{Z}_p$), the problem of Boolean circuit satisfiability easily reduces to arithmetic circuit satisfiability with only constant overhead [BCI$^+$13, Claim A.2].

and $(\mathcal{P}, \mathcal{V})$ be a linear PCP with soundness error $\varepsilon(\lambda)$ against affine provers for the relation $\mathcal{R}_\mathcal{C}$. Let $\Pi_{\mathsf{venc}} = (\mathsf{Setup}, \mathsf{Encrypt}, \mathsf{Decrypt})$ be a vector encryption scheme over $\mathbb{Z}_p$ with linear targeted malleability (Definition 4.11). Then applying Construction 4.14 to $(\mathcal{P}, \mathcal{V})$ and Π_{venc} yields a (non-adaptive) designated-verifier preprocessing SNARG for $\mathcal{R}_\mathcal{C}$ with soundness error $2 \cdot \varepsilon(\lambda) + \operatorname{negl}(\lambda)$.

Remark 4.16 (Adaptivity)**.** In Theorem 4.15, we showed that instantiating Construction 4.14 with a vector encryption scheme with linear targeted malleability and a linear PCP yields a non-adaptive SNARG in the preprocessing model. The same construction can be shown to satisfy *adaptive* soundness for proving efficiently-decidable statements. As noted in [BCI+13, Remark 6.5], we can relax Definition 4.11 and allow the adversary to additionally output an arbitrary string in the real distribution which the simulator must produce in the ideal distribution. Invoking Construction 4.14 with an encryption scheme that satisfies this strengthened linear targeted malleability definition yields a SNARG with adaptive soundness for the case of verifying deterministic polynomial-time computations. For verifying general NP computations, we can obtain adaptive soundness by conjecturing that the vector encryption scheme satisfies the stronger notion of linear-only property from Definition 4.10. We state this in Theorem 4.17.

Theorem 4.17 ([BCI+13, Lemma 6.2, adapted])**.** *Fix a security parameter λ and a prime p. Let $\mathcal{C} = \{C_n\}_{n \in \mathbb{N}}$ be a family of arithmetic circuits over $\mathbb{F}_p$, $\mathcal{R}_\mathcal{C}$ be the relation associated with $\mathcal{C}$, and $(\mathcal{P}, \mathcal{V})$ be a linear PCP with soundness error $\varepsilon(\lambda)$ against affine provers for the relation $\mathcal{R}_\mathcal{C}$. Let $\Pi_{\mathsf{venc}} = (\mathsf{Setup}, \mathsf{Encrypt}, \mathsf{Decrypt})$ be a linear-only vector encryption scheme over $\mathbb{Z}_p$ (Definition 4.10). Then, applying Construction 4.14 to $(\mathcal{P}, \mathcal{V})$ and Π_{venc} yields a adaptive designated-verifier preprocessing SNARG for $\mathcal{R}_\mathcal{C}$ with soundness error $\varepsilon(\lambda) + \operatorname{negl}(\lambda)$.*

Remark 4.18 (Multi-Theorem SNARGs)**.** Our basic notion of linear targeted malleability for vector encryption only suffices to construct a single-theorem SNARG. While the same construction can be shown secure for an adversary that is allowed to make any constant number of queries to a proof verification oracle, we are not able to prove that the construction is secure against a prover who makes polynomially many queries to the proof verification oracle. In Section 4.4.3, we present an analog of the strengthened version of linear-only encryption from [BCI+13, Appendix C] that suffices for constructing a multi-theorem SNARG. Combined with a linear PCP that is *strongly sound* against affine provers, Construction 4.14 can then be applied to obtain a multi-theorem, designated-verifier SNARG. This raises the question of whether the same construction using the weaker notion of linear targeted malleability also suffices when the underlying linear PCP satisfies strong soundness. While we do not know how to prove security from this weaker definition, we also do not know of any attacks. This is especially interesting because at the information-theoretic level, the underlying linear PCP satisfies *strong* soundness, which intuitively would suggest that the responses the malicious prover obtains from querying the proof verification oracle are *uncorrelated* with the verifier's state (strong soundness states that for any proof, either the verifier accepts with probability 1 or with negligible probability).

Remark 4.19 (Arguments of Knowledge)**.** Theorem 4.15 shows that instantiating Construction 4.14 with a linear PCP with soundness against affine provers and a vector encryption scheme with linear targeted malleability suffices for a SNARG. In fact, the same construction yields a SNARK (that is, a succinct non-interactive argument *of knowledge*) if the soundness requirement on the underlying linear PCP is replaced with a knowledge requirement (Definition 4.2), and the vector encryption scheme satisfies a variant of linear targeted malleability (Definition 4.11) where the simulator is required to be *efficient* (i.e., polynomially-sized). For more details, we refer to [BCI$^+$13, Lemma 6.3, Remark 6.4].

4.4.3 Multi-Theorem Designated-Verifier SNARGs

In Theorem 4.15 of Section 4.4.1, we showed how a vector encryption scheme satisfying linear targeted malleability can be combined with a linear PCP with soundness against affine provers to obtain a single-theorem, designated-verifier SNARG. In this section, we introduce a stronger notion of linear-only encryption that can be combined with linear PCPs satisfying strong soundness against affine provers to obtain multi-theorem SNARGs. However, as noted in Section 4.4.1, it is interesting to see whether the simpler definition of linear targeted malleability together with strongly sound linear PCPs already suffices in the multi-theorem setting. We begin by formally introducing the notion of a multi-theorem SNARG.

Definition 4.20 (Adaptive Multi-Theorem SNARG)**.** Let $\mathcal{C} = \{C_n\}_{n\in\mathbb{N}}$ be a family of Boolean circuits. Let $\Pi_{\mathsf{SNARG}} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ be a SNARG for the relation $\mathcal{R}_{\mathcal{C}}$ (with corresponding language $\mathcal{L}_{\mathcal{C}}$). Then, Π_{SNARG} is an adaptive multi-theorem SNARG if for all $n \in \mathbb{N}$ and all polynomial-size provers P^*,

$$\Pr[(\sigma,\tau) \leftarrow \mathsf{Setup}(1^\lambda, 1^n); (\mathbf{x},\pi) \leftarrow (P^*)^{\mathsf{Verify}(\tau,\cdot,\cdot)}(\sigma) : \mathsf{Verify}(\tau,\mathbf{x},\pi) = 1 \text{ and } \mathbf{x} \notin \mathcal{L}_{C_n}] = \operatorname{negl}(\lambda).$$

In other words, soundness should hold even if the prover has access to a proof verification oracle $\mathsf{Verify}(\tau,\cdot,\cdot)$.

Next, we introduce a stronger notion of linear-only encryption that can be used to obtain a multi-theorem SNARG (via the same construction as Construction 4.14). In order to show a SNARG system is multi-theorem, we need a way to simulate the prover's queries to the verification oracle. In past works [BP04b, BP04c, BCI$^+$13] this has been handled by defining an interactive extractability assumption. Here, we extend the definition in [BCI$^+$13, Definition C.6] to the vector case.

Definition 4.21 (Linear-Only with Interactive Extraction [BCI$^+$13])**.** Fix a security parameter λ. Let $\Pi_{\mathsf{venc}} = (\mathsf{Setup}, \mathsf{Encrypt}, \mathsf{Decrypt})$ be a secret-key vector encryption scheme where the message space consists of vectors of $\mathbb{Z}_p$-elements. Let $\mathcal{M}$ be a message generation algorithm that on input 1^k, outputs a sequence of vectors in $\mathbb{Z}_p^k$. Let $z \in \{0,1\}^{\operatorname{poly}(\lambda)}$ be some auxiliary input. We define the interactive linear-only extraction game between $\mathcal{A}$ and $\mathcal{E}$ as follows:

1. **Setup phase:**
   - $\mathsf{sk} \leftarrow \mathsf{Setup}(1^\lambda)$
   - $(\mathbf{v}_1, \ldots, \mathbf{v}_m) \leftarrow \mathcal{M}(1^k)$
   - $\mathsf{ct}_i \leftarrow \mathsf{Encrypt}(\mathsf{sk}, \mathbf{v}_i)$ for all $i \in [m]$
2. **Query phase:** for all $i \in [q]$ where $q = \text{poly}(\lambda)$:
   - $\mathsf{ct}'_i \leftarrow \mathcal{A}(\mathsf{ct}_1, \ldots, \mathsf{ct}_m; e_1, \ldots, e_{i-1}; z)$
   - $\mathbf{\Pi}_i \leftarrow \mathcal{E}(\mathsf{ct}_1, \ldots, \mathsf{ct}_m; i; z)$ where $\mathbf{\Pi}_i$ is either an affine function $(\boldsymbol{\pi}_i, \mathbf{b}_i)$ or $\perp$ where $\boldsymbol{\pi}_i \in \mathbb{Z}_p^m$ and $\mathbf{b} \in \mathbb{Z}_p^k$

We say that $\mathcal{A}$ wins the game if one of the following conditions hold:

- For some $i \in [q]$, $\mathsf{Decrypt}(\mathsf{sk}, \mathsf{ct}'_i) = \perp$ but $\mathbf{\Pi}_i \neq \perp$.
- For some $i \in [q]$, $\mathsf{Decrypt}(\mathsf{sk}, \mathsf{ct}'_i) \neq \perp$ and either $\mathbf{\Pi}_i = \perp$ or $\mathsf{Decrypt}(\mathsf{sk}, \mathsf{ct}'_i) \neq \mathbf{a}_i$ where $\mathbf{\Pi}_i = (\boldsymbol{\pi}_i, \mathbf{b}_i)$ and $\mathbf{a}_i = [\mathbf{v}_1 | \mathbf{v}_2 | \cdots \mathbf{v}_m] \cdot \boldsymbol{\pi} + \mathbf{b}$

Finally, we say that Π_{venc} satisfies *linear-only with interactive extraction* if for all polynomial-size interactive adversaries $\mathcal{A}$, there exists a polynomial-size interactive extractor $\mathcal{E}$ such that for any auxiliary input $z \in \{0,1\}^{\text{poly}(\lambda)}$, any plaintext generation algorithm $\mathcal{M}$, the probability that $\mathcal{A}$ wins the above interactive linear-only extraction game is negligible.

We note state the corresponding theorem that applying Construction 4.14 to a linear PCP with strong soundness against affine provers and a vector encryption scheme satisfying the stronger notion of linear-only encryption with interactive extraction suffices to construct a multi-theorem SNARG. The proof follows analogously to the proof in [BCI+13, Lemma C.8].

Theorem 4.22 (Multi-Theorem SNARG [BCI+13, adapted])**.** *Let $(\mathcal{P}, \mathcal{V})$ be a linear PCP that is strongly sound against affine provers. Let $\Pi_{\mathsf{venc}} = (\mathsf{Setup}, \mathsf{Encrypt}, \mathsf{Decrypt})$ be a linear-only encryption scheme that satisfies linear-only with interactive extraction (Definition 4.21). Then, applying Construction 4.14 to $(\mathcal{P}, \mathcal{V})$ and Π_{venc} yields an adaptive, multi-theorem SNARG in the preprocessing model. Moreover, if $(\mathcal{P}, \mathcal{V})$ satisfies* strong knowledge *against affine provers, then applying Construction 4.14 to $(\mathcal{P}, \mathcal{V})$ and Π_{venc} yields an adaptive, multi-theorem SNARK in the preprocessing model.*

4.5 Constructing Lattice-Based SNARGs

In this section, we describe one candidate instantiation of a lattice-based linear-only vector encryption scheme: namely, the variant of Regev encryption [Reg05] by Peikert, Vaikuntanathan, and Waters [PVW08]. Combined with Construction 4.14, this gives a lattice-based SNARG candidate.

4.5.1 The Peikert-Vaikuntanathan-Waters Encryption Scheme

The core building block in our new SNARG construction is a *vector encryption* scheme for $\mathbb{Z}_p^k$ that plausible satisfies our notion of linear targeted malleability (Definition 4.11). In particular, we conjecture that the Regev-based encryption scheme [Reg05] due to Peikert, Vaikuntanathan, and Waters [PVW08, §7.2] satisfies our required properties.

The PVW encryption scheme. We now review the encryption scheme due to Peikert, Vaikuntanathan, and Waters [PVW08, §7.2]. In our setting, it suffices to just consider the secret-key setting.

Construction 4.23 ([PVW08, §7.2, adapted])**.** Fix a security parameter λ, lattice parameters $n, m, q = \mathrm{poly}(\lambda)$, and an error distribution χ. Let k be the plaintext dimension and let $\mathbb{Z}_p^k$ be the plaintext space. The vector encryption scheme $\Pi_{\mathsf{venc}} = (\mathsf{Setup}, \mathsf{Encrypt}, \mathsf{Decrypt})$ with plaintext space $\mathbb{Z}_p^k$ is defined as follows:

- $\mathsf{Setup}(1^\lambda, 1^k) \to \mathsf{sk}$: The setup algorithm samples $\bar{\mathbf{A}} \xleftarrow{\text{R}} \mathbb{Z}_q^{n \times m}$, $\bar{\mathbf{S}} \xleftarrow{\text{R}} \mathbb{Z}_q^{n \times k}$, and $\bar{\mathbf{E}} \leftarrow \chi^{k \times m}$. Define the matrices $\mathbf{A} \in \mathbb{Z}_q^{(n+k) \times m}$ and $\mathbf{S} \in \mathbb{Z}_q^{(n+k) \times k}$ as follows:

$$\mathbf{A} = \begin{bmatrix} \bar{\mathbf{A}} \\ \bar{\mathbf{S}}^T \bar{\mathbf{A}} + \mathbf{E} \end{bmatrix} \qquad \mathbf{S} = \begin{bmatrix} -\bar{\mathbf{S}} \\ \mathbf{I}_k \end{bmatrix},$$

  where $\mathbf{I}_k \in \mathbb{Z}_q^{k \times k}$ is the k-by-k identity matrix. Finally, it outputs the secret key $\mathsf{sk} = (\mathbf{A}, \mathbf{S})$.

- $\mathsf{Encrypt}(\mathsf{sk}, \mathbf{v}) \to \mathbf{c}$: To encrypt a vector $\mathbf{v} \in \mathbb{Z}_p^k$, choose $\mathbf{r} \xleftarrow{\text{R}} \{0,1\}^m$ and output the ciphertext $\mathbf{c} \in \mathbb{Z}_q^{n+k}$ where

$$\mathbf{c} = \mathbf{A}\mathbf{r} + \begin{bmatrix} 0^n \\ \lfloor q/p \rceil \cdot \mathbf{v} \end{bmatrix}.$$

- $\mathsf{Decrypt}(\mathsf{sk}, \mathbf{c}) \to \mathbf{v}$: Compute and output $[[\mathbf{S}^T \mathbf{c}]_q]_p$.

4.5.2 Our Lattice-Based SNARG Candidate

We now state our concrete conjecture on the vector encryption scheme Π_{venc} from Section 4.5.1 that yields the first lattice-based candidate of a designated-verifier, preprocessing SNARG with quasi-optimal succinctness.

Conjecture 4.24 (Linear Targeted Malleability of Construction 4.23)**.** The vector encryption scheme Π_{venc} from Construction 4.23 satisfies exponentially-strong[10] linear targeted malleability

[10] Achieving soundness error that is inverse exponential (i.e., $2^{-\lambda}$) against provers of size 2^λ necessitates some kind of exponential hardness assumptions. We can relax this conjecture as necessary if we target weaker soundness requirements (i.e., $\mathrm{negl}(\lambda)$ soundness against all $\mathrm{poly}(\lambda)$-size provers). Having a concrete target facilitates comparisons between different SNARG candidates.

(Definition 4.11): namely, the distinguishing advantage of any adversary of size 2^λ in Definition 4.11 is bounded by $2^{-\Omega(\lambda)}$.

Corollary 4.25 (Lattice-Based SNARG)**.** *Let λ be a security parameter and let $\mathcal{C} = \{C_n\}_{n \in \mathbb{N}}$ be a family of Boolean circuits. Under Conjecture 4.24, there exists a succinct non-interactive argument system $\Pi_{\mathsf{SNARG}} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ for the relation $\mathcal{R}_{\mathcal{C}}$ with the following properties:*

- *The soundness error of Π_{SNARG} is $2^{-\lambda}$ against all provers of size 2^λ.*
- *The prover algorithm* Prove *can be implemented by a circuit of size $\widetilde{O}(\lambda \cdot |C_n|)$.*
- *The length of a proof is $\widetilde{O}(\lambda)$.*

Proof. Under Conjecture 4.24, we can apply Construction 4.14 in conjunction with algebraic linear PCPs to obtain designated-verifier SNARGs in the preprocessing model (Theorem 4.15). In particular, we take our underlying linear PCP to be that obtained by combining Construction 4.5 with linear PCPs based on the quadratic span programs of Gennaro et al. [GGPR13]. We consider the concrete asymptotics below:

- **Prover complexity.** In Construction 4.14, the prover performs m homomorphic operations on the encrypted vectors, where m is the length of the underlying linear PCP. When instantiating the vector encryption scheme Π_{venc} over the plaintext space $\mathbb{Z}_p^k$ where $p = \text{poly}(\lambda)$, the ciphertexts consist of vectors of dimension $O(\lambda + k)$ over a ring of size $q = \text{poly}(\lambda)$.[11] Homomorphic operations on ciphertexts corresponds to scalar multiplication (by values from $\mathbb{Z}_p$) and vector additions. Since all operations are performed over a *polynomial-sized* domain, all of the basic arithmetic operations can be performed in polylog(λ) time. Thus, as long as the underlying linear PCP operates over a polynomial-sized field, the prover's overhead is $\widetilde{O}(m(\lambda + k))$.

  If the underlying linear PCP is instead instantiated with one based on the quadratic span programs of Gennaro et al. [GGPR13], then $m = \widetilde{O}(|C_n|)$ and $k = O(\lambda)$. The overall prover complexity in this case is $\widetilde{O}(\lambda \cdot |C_n|)$.

- **Proof length.** The proof in Construction 4.14 consist of a single ciphertext of the vector encryption scheme, which has length $\widetilde{O}(\lambda + k)$. Thus, in our candidate instantiation, the length of the proof is $\widetilde{O}(\lambda)$. □

Remark 4.26 (Comparison with [BCI$^+$13])**.** An alternative route to obtaining a lattice-based SNARG is to *directly* instantiate [BCI$^+$13] with Regev-based encryption. However, to achieve soundness error $2^{-\lambda}$, Bitansky et al. [BCI$^+$13] require a linear PCP (and a linear-only encryption

[11] More precisely, the ciphertexts are actually vectors of dimension $n + k$, where n is the dimension of the lattice in the LWE problem. Currently, the most effective algorithms for solving LWE rely either on BKW-style [BKW00, KF15] or BKZ-based attacks [SE94, CN11]. Based on our current understanding [LP11, CN11, KF15, BCD$^+$16], the best-known algorithms for LWE all require time $2^{\Omega(n/\log^c n)}$ for some constant c. Thus, in terms of a concrete security parameter λ, we set the lattice dimension to be $n = \widetilde{O}(\lambda)$.

scheme) over a field of size 2^λ. Instantiating the construction in [BCI+13] with Regev-based encryption over a plaintext space of size 2^λ, the resulting SNARGs have length $\widetilde{O}(\lambda^2)$ and the prover complexity is $\widetilde{O}(s\lambda^2)$. Another possibility is to instantiate [BCI+13] with Regev-based encryption over a polynomial-size field (thus incurring $1/\mathrm{poly}(\lambda)$-soundness error) and perform parallel repetition at the SNARG level to amplify the soundness. But this method suffers from the same drawback as above. While each individual SNARG instance (over a polynomial-size field) is quasi-optimally succinct, the size of the overall proof is still $\widetilde{O}(\lambda^2)$ and the prover's complexity remains at $\widetilde{O}(s\lambda^2)$. This is a factor λ worse than using linear-only vector encryption over a polynomial-size field.

Remark 4.27 (Comparison with Hash-Based SNARGs)**.** An alternative approach for constructing SNARGs is to start with Kilian's succinct *interactive* argument [Kil92] and apply the Fiat-Shamir heuristic [FS86] to obtain a SNARG in the random-oracle model (i.e., a CS proof [Mic00]). In fact, with a suitable heuristic instantiation of the random oracle, this method yields a quasi-optimal SNARG in terms of the prover's complexity (namely, the prover overhead is *additive* in the security parameter rather than multiplicative—we refer to Chapter 5 for a more thorough discussion of quasi-optimal SNARGs). However, CS proofs do not provide quasi-optimal succinctness. Recall that in Kilian's protocol, the prover first uses a Merkle hash tree to commit to a (standard) PCP for the statement being proved. The verifier then challenges the prover to open bits in the committed PCP, and checks that the revealed bits satisfy the PCP verification algorithm. Using the quasilinear PCPs of [Din06, BS08], Kilian's protocol achieves constant soundness error $\varepsilon < 1$ against provers of size 2^λ with $\widetilde{O}(\lambda)$ communication. We can use parallel repetition (using $\Omega(\lambda)$ challenges) to amplify the soundness to $\varepsilon^{-\Omega(\lambda)} = 2^{-\Omega(\lambda)}$. But then the proof is $\widetilde{O}(\lambda^2)$, and thus, no longer quasi-optimally succinct. A similar issue arises with other Kilian-bsaed protocols such as [BCCT12, BCC+17], which leverage extractable collision-resistant hash functions and single-server private information retrieval [CMS99].

Remark 4.28 (Arithmetic Circuit Satisfiability over Large Fields)**.** Construction 4.14 also applies to arithmetic circuit satisfiability over large finite fields (say, $\mathbb{Z}_p$ where $p = 2^\lambda$). However, if the size of the plaintext space for the vector encryption scheme Π_{venc} from Section 4.5.1 is 2^λ, then the bit-length of the ciphertexts becomes $\widetilde{O}(\lambda^2)$ bits. Consequently, the proof system is no longer quasi-optimally succinct. In contrast, the QSP-based constructions [GGPR13, BCI+13] remain quasi-optimally succinct for arithmetic circuit satisfiability over large fields.

Remark 4.29 (Multi-Theorem and Adaptive SNARGs.)**.** As noted in [BCI+13, Remark 5.6], encryption schemes that allow for "oblivious sampling" of ciphertexts cannot satisfy Definitions 4.10 and 4.21 (but they can still plausibly satisfy the weaker notion of linear targeted malleability from Definition 4.11 and considered in Conjecture 4.24). To satisfy the stronger notions of linear-only, it should be the case that the set of strings c where $\mathsf{Decrypt}(\mathsf{sk}, c) \neq \perp$ should be sparse (i.e., an adversary cannot simply sample a random string c that successfully decrypts to a valid vector).

Certainly, the vector encryption scheme from Section 4.5.1 does not satisfy this property since $\mathsf{Decrypt}(\mathsf{sk}, c) \neq \perp$ for all strings c in the ciphertext space.

A heuristic method to prevent oblivious sampling is to "sparsify" the ciphertext space using "double-encryption" [GGPR13, BCI$^+$13]. Specifically, an encryption of a vector $\mathbf{v} \in \mathbb{Z}_p^k$ consists of two encryptions $(\mathsf{ct}_1, \mathsf{ct}_2)$ of $\mathbf{v}$ under two independent secret keys sk_1 and sk_2, respectively. The secret key for the vector encryption includes the secret keys of both underlying schemes. During decryption, ct_1 and ct_2 are decrypted using sk_1 and sk_2, respectively. The decryption algorithm outputs $\perp$ if ct_1 and ct_2 do not decrypt to the same value; otherwise, the output is $\mathsf{Decrypt}(\mathsf{sk}_1, \mathsf{ct}_1)$. Intuitively, if the adversary was to sample random vectors from the ciphertext space, with overwhelming probability, the two ciphertexts will not decrypt to the same vector. We conjecture that this modified version of Regev-based vector encryption satisfies the stronger notion of linear-only encryption as defined in Definitions 4.10 and 4.21. Thus, with a factor of 2 overhead, we can obtain multi-theorem SNARGs (resp., SNARKs) based on any linear PCP satisfying strong soundness (resp., knowledge) against affine provers.

4.6 Chapter Summary

In this chapter, we constructed the first (designated-verifier) SNARG with quasi-optimal succinctness from standard lattice assumptions. Since our new SNARG candidates are lattice-based, they resist all currently-known quantum attacks. Another appealing property of our SNARGs is that proof verification is very simple (i.e., consists of taking an inner product over a finite field or a polynomial ring, followed by checking a quadratic relation on the inner products). In the next chapter, we further refine these methods to obtain a *quasi-optimal* SNARG—namely, a SNARG that *simultaneously* minimizes prover complexity as well as the proof size. We provide a concrete comparison our new lattice-based SNARG candidates with previous constructions in Section 5.5 (Table 5.1).

One limitation of our new lattice-based SNARGs is that they are all designated-verifier constructions. In contrast, many existing SNARG candidates in the random oracle model [Mic00] or from pairings [BCI$^+$13, GGPR13] are publicly verifiable. It is an important open problem to build publicly-verifiable SNARGs from lattices.

Chapter 5

Quasi-Optimal SNARGs

In this chapter, we introduce the notion of a *quasi-optimal* SNARG which is a SNARG that simultaneously minimizes both the prover complexity as well as the proof size. We begin by defining quasi-optimal SNARGs and then provide a technical overview of our main construction in Section 5.1.

Defining quasi-optimality. As in Chapter 4, in all of our SNARG constructions, we measure the asymptotic cost of achieving soundness error $2^{-\lambda}$ against provers of size 2^{λ}. We say that a SNARG (for Boolean circuit satisfiability) is quasi-optimally succinct if the proof size is $\widetilde{O}(\lambda)$, and moreover, that it is quasi-optimal if the prover complexity is $\widetilde{O}(|C|) + \text{poly}(\lambda, \log |C|)$, where C is the Boolean circuit.[1] In Lemma 5.54, we show that this notion of quasi-optimality is tight (up to polylogarithmic factors) in the following sense: assuming NP does not have succinct *proofs*, the length of any succinct argument system that provides this soundness guarantee is necessarily $\Omega(\lambda)$.

Previous SNARG constructions. Prior to this work, the only SNARG candidate that satisfies quasi-optimal prover complexity is Micali's CS proofs [Mic00]. However, to achieve $2^{-\lambda}$ soundness, the length of a CS proof is $\Omega(\lambda^2)$, which is not quasi-optimally succinct. Conversely, if we just consider SNARGs that are quasi-optimally succinct, we have many candidates based on bilinear maps [Gro10, Lip12, GGPR13, BCI$^+$13, Lip13, DFGK14, Gro16]. In all of these constructions, the SNARG proof consists of a *constant* number of bilinear group elements. To construct the proof, however, the prover has to perform a group operation for every gate in the underlying circuit, and since each group element is $\Omega(\lambda)$ bits, the prover overhead is at least multiplicative in λ. Consequently, none of the existing constructions are quasi-optimal in terms of prover complexity. Alternatively, our lattice-based candidate from Section 4.5 (Corollary 4.25) also satisfies quasi-optimal succinctness. The construction, though, suffers from a similar limitation: the prover needs to operate on an LWE

[1] We write $\widetilde{O}(\cdot)$ to suppress factors that are *polylogarithmic* in the circuit size $|C|$ and the security parameter λ.

ciphertext per gate in the circuit, which also introduces a multiplicative overhead $\Omega(\lambda)$ in the prover's computational cost.

Quasi-optimal linear MIPs. In this chapter, we give the first construction of a quasi-optimal SNARG for Boolean circuit satisfiability from a concrete cryptographic assumption. Our construction follows a similar structure as that used in Chapter 4. Specifically, we decompose the construction into two components: an information-theoretic component (linear MIPs), and a cryptographic component (linear-only vector encryption). We give a brief description of the information-theoretic primitive we construct in this work: a *quasi-optimal* linear MIP. At the end of this section, we discuss why the general PCPs and linear PCPs that have featured in previous SNARG constructions (including the one from Section 4.5) do not seem sufficient for building quasi-optimal SNARGs.

To briefly recall, a linear PCP over a finite field $\mathbb{F}$ is an oracle computing a linear function $\boldsymbol{\pi} \colon \mathbb{F}^m \to \mathbb{F}$. On any query $\mathbf{q} \in \mathbb{F}^m$, the linear PCP oracle responds with the inner product $\mathbf{q}^T\boldsymbol{\pi} = \langle \mathbf{q}, \boldsymbol{\pi} \rangle \in \mathbb{F}$. Linear MIPs directly generalize linear PCPs to the setting where there are ℓ independent proof oracles $(\boldsymbol{\pi}_1, \ldots, \boldsymbol{\pi}_\ell)$, each implementing a linear function $\boldsymbol{\pi}_i \colon \mathbb{F}^m \to \mathbb{F}$. In the linear MIP model, the verifier's queries consist of a ℓ-tuple $(\mathbf{q}_1, \ldots, \mathbf{q}_\ell)$ where each $\mathbf{q}_i \in \mathbb{F}^m$. For each query $\mathbf{q}_i \in \mathbb{F}^m$ to the proof oracle $\boldsymbol{\pi}_i$, the verifier receives the response $\langle \mathbf{q}_i, \boldsymbol{\pi}_i \rangle$. We give a formal definition in Section 5.2.1.

In this work, we say that a linear MIP for Boolean circuit satisfiability is quasi-optimal if the MIP prover (for proving satisfiability of a circuit C) can be implemented by a circuit of size $\widetilde{O}(|C|) + \mathrm{poly}(\lambda, \log |C|)$, and the linear MIP provides soundness error $2^{-\lambda}$. As we note in Remark 5.3, existing linear PCP constructions (e.g., [BCI+13] or Construction 4.5) are not quasi-optimal: they either require embedding the Boolean circuit into an arithmetic circuit over a large field [BCI+13], or, in the case of Construction 4.5, rely on making $O(\lambda)$ queries, each of length $m = O(|C|)$.

Constructing quasi-optimal linear MIPs. Our work gives the first construction of a quasi-optimal linear MIP for Boolean circuit satisfiability. We refer to Section 5.1 for an overview of our construction and to Section 5.3 for the full description. At a high-level, our quasi-optimal linear MIP construction relies on two key ingredients: a robust circuit decomposition and a method for enforcing consistency.

Robust circuit decomposition. Our robust decomposition primitive takes a circuit C and produces from it a collection of constraints $f_1, \ldots, f_t$, each of which can be computed by a circuit of size roughly $|C|/t$. Each constraint reads a subset of the bits of a global witness (computed based on the statement-witness pair for C). The guarantee provided by the robust decomposition is that for any false statement $\mathbf{x}$ (that is, a statement $\mathbf{x}$ where for all witnesses $\mathbf{w}$, $C(\mathbf{x}, \mathbf{w}) = 0$), no single witness to $f_1, \ldots, f_t$ can simultaneously satisfy more than a *constant fraction* of the constraints. Now, to prove satisfiability of a circuit C, the prover instead proves that there is a consistent witness that simultaneously satisfies all of the constraints $f_1, \ldots, f_t$. Each of these proofs can be implemented by a standard linear PCP. The advantage of this approach is that for a false statement, only a constant fraction of the constraints can be satisfied (for any choice of witness), so even if each underlying linear PCP instance only provided *constant* soundness, the probability that the prover is able to satisfy *all* of the instances is amplified to $2^{-\Omega(t)} = 2^{-\Omega(\lambda)}$ if we let $t = \Theta(\lambda)$. Finally, even though the prover now has to construct t proofs for the t constraints, each of the constraints can themselves be computed by a circuit of size $\widetilde{O}(|C|/t)$. The robustness property of our decomposition is reminiscent of the relation between traditional PCPs and constraint-satisfaction problems, and one might expect that we could instantiate such a decomposition using PCPs. However, in our settings, we require that the decomposition be *input-independent*, which to the best of our knowledge, is not satisfied by existing (quasilinear) PCP constructions. We discuss this in more detail in Remark 5.22.

The robust decomposition can amplify soundness without introducing much additional overhead. The alternative approach of directly applying a constant-query linear PCP to check satisfiability of C has the drawback of only providing $1/\text{poly}(\lambda)$ soundness when working over a small field (i.e., as would be the case with Boolean circuit satisfiability). We give the formal definition of a robust decomposition in Section 5.3.1, and then show how to realize it by combining MPC protocols with polylogarithmic overhead [DIK10] with the "MPC-in-the-head" paradigm [IKOS07]. Since the notion of a robust decomposition is a very natural one, we believe that our construction is of independent interest and will have applications beyond quasi-optimal linear MIP constructions.

Enforcing consistency. The second ingredient we require is a way for the verifier to check that the individual proofs the prover constructs (for showing satisfiability of each constraint $f_1, \ldots, f_t$) are self-consistent. Our construction here relies on constructing randomized permutation decompositions, and we refer to Section 5.1 for the technical overview, and Section 5.3 for the full description.

Preprocessing SNARGs from linear MIPs. To complete our construction of quasi-optimal SNARGs, we show a generic compiler from linear MIPs to preprocessing SNARGs using a variant of the linear-only vector encryption scheme introduced in Section 4.4.1. Specifically, we require a linear-only vector encryption scheme where the underlying encryption scheme is a *polynomial ring* (as opposed to a finite field). We give our construction in Section 5.4. Combined with our information-theoretic construction of quasi-optimal linear MIPs, this yields the first quasi-optimal designated-verifier SNARG for Boolean circuit satisfiability in the preprocessing model (Corollaries 5.62 and Remark 5.63).

Why linear MIPs? A natural question to ask is whether our new linear MIP to preprocessing SNARG compiler provides any advantage over the existing compilers by Bitansky et al. [BCI$^+$13] or that from Chapter 4. The Bitansky et al. compiler relies on linear interactive proofs while our compiler from Section 4.4 relies on linear PCPs as the core information-theoretic building block. After all, any k-query, ℓ-prover linear MIP with query length m can be transformed into a $(k\ell)$-query linear PCP with query length $m\ell$ by concatenating the proofs of the different provers together, and likewise, padding the queries accordingly. While this still yields a quasi-optimal linear PCP (with sparse queries), applying the existing cryptographic compilers to this linear PCP incurs an additional prover overhead that is proportional to ℓ. In our settings, $\ell = \Theta(\lambda)$, so the resulting SNARG is no longer quasi-optimal. By directly compiling linear MIPs to preprocessing SNARGs, our compiler *preserves* the prover complexity of the underlying linear MIP, and so, combined with our quasi-optimal linear MIP construction, yields a quasi-optimal SNARG for Boolean circuit satisfiability.

Alternatively, one might ask whether a similar construction of quasi-optimal SNARGs is possible starting from standard PCPs or linear PCPs with quasi-optimal prover complexity. Existing techniques for compiling general PCPs [Mic00, BCCT12, BCC$^+$17] to succinct argument systems all rely on some form of cryptographic hashing to commit to the proof and then open up a small number of bits chosen by the verifier. Some of the hash-based constructions [Mic00, BCCT12, BCC$^+$17] based on traditional PCPs can achieve quasi-optimal prover complexity, but none achieves quasi-optimal succinctness (Remark 4.27). If instead we start with linear PCPs and apply the compiler in either [BCI$^+$13] or the one from Section 4.4 (Construction 4.14), the challenge is in constructing a quasi-optimal linear PCP that provides soundness error $2^{-\lambda}$ over a small field $\mathbb{F}$. As noted above (and in Remark 5.3), existing linear PCP constructions are *not* quasi-optimal for Boolean circuit satisfiability.

5.1 Quasi-Optimal Linear MIP Construction Overview

In this section, we give a technical overview of our quasi-optimal linear MIP construction for arithmetic circuit satisfiability over a finite field $\mathbb{F}$. Combined with our cryptographic compiler based on linear-only vector encryption over rings, this gives the first construction of a quasi-optimal SNARG from a concrete cryptographic assumption.

Robust circuit decomposition. The first ingredient we require in our quasi-optimal linear MIP construction is a *robust* way to decompose an arithmetic circuit $C \colon \mathbb{F}^{n'} \times \mathbb{F}^{m'} \to \mathbb{F}^{h'}$ into a collection of t constraint functions $f_1, \ldots, f_t$, where each constraint $f_i \colon \mathbb{F}^n \times \mathbb{F}^m \to \{0,1\}$ takes as input a common statement $\mathbf{x} \in \mathbb{F}^n$ and witness $\mathbf{w} \in \mathbb{F}^m$. More importantly, each constraint f_i can be computed by a small arithmetic circuit C_i of size roughly $|C|/t$. This means that each arithmetic circuit C_i may only need to read some subset of the components in $\mathbf{x}$ and $\mathbf{w}$. There is a mapping $\mathsf{inp} \colon \mathbb{F}^{n'} \to \mathbb{F}^n$ that takes as input a statement $\mathbf{x}'$ for C and outputs a statement $\mathbf{x}$ for $f_1, \ldots, f_t$, and another mapping $\mathsf{wit} \colon \mathbb{F}^{n'} \times \mathbb{F}^{m'} \to \mathbb{F}^m$ that takes as input a statement-witness pair $(\mathbf{x}', \mathbf{w}')$ for C, and outputs a witness $\mathbf{w}$ for $f_1, \ldots, f_t$. The decomposition must satisfy two properties: completeness and robustness. Completeness says that whenever a statement-witness pair $(\mathbf{x}', \mathbf{w}')$ is accepted by C, then $f_i(\mathbf{x}, \mathbf{w}) = 1$ for all i if we set $\mathbf{x} = \mathsf{inp}(\mathbf{x}')$ and $\mathbf{w} = \mathsf{wit}(\mathbf{x}', \mathbf{w}')$. Robustness says that for a false statement $\mathbf{x}' \in \mathbb{F}^{n'}$, there are no valid witnesses $\mathbf{w} \in \mathbb{F}^m$ that can simultaneously satisfy more than a constant fraction of the constraints $f_1(\mathbf{x}, \cdot), \ldots, f_t(\mathbf{x}, \cdot)$, where $\mathbf{x} = \mathsf{inp}(\mathbf{x}')$.

Roughly speaking, a robust decomposition allows us to reduce checking satisfiability of a large circuit C to checking satisfiability of many smaller circuits $C_1, \ldots, C_t$. The gain in performance will be due to our ability to check satisfiability of all of the $C_1, \ldots, C_t$ in parallel. The importance of robustness will be critical for soundness amplification. We give the formal definition of a robust decomposition in Section 5.3.1.

Instantiating the robust decomposition. In Section 5.3.1, we describe one way of instantiating the robust decomposition by applying the "MPC-in-the-head" paradigm of [IKOS07] to MPC protocols with polylogarithmic overhead [DIK10]. We give a brief overview here. For an arithmetic circuit $C \colon \mathbb{F}^{n'} \times \mathbb{F}^{m'} \to \mathbb{F}^{h'}$, the encoding of a statement-witness pair $(\mathbf{x}, \mathbf{w})$ will be the *views* of each party in a (simulated) t-party MPC protocol computing C on $(\mathbf{x}, \mathbf{w})$, where the bits of the input and witness are evenly distributed across the parties. Each of the constraint functions f_i checks that party i outputs 1 in the protocol execution (indicating an accepting input), and that the view of party i is *consistent* with the views of the other parties. This means that the only bits of the encoded witness that each constraint f_i needs to read are those that correspond to messages that were sent or received by party i. Then, using an MPC protocol where the computation and communication overhead is polylogarithmic in the circuit size (c.f., [DIK10]), and where the computational burden is evenly distributed across the computing parties, each $f_1, \ldots, f_t$ can be implemented by a circuit of size $\widetilde{O}(|C| / t)$. Robustness of the decomposition follows from security of the underlying MPC protocol. We give the complete description and analysis in Section 5.3.1.

Blueprint for linear MIP construction. The high-level idea behind our quasi-optimal linear MIP construction is as follows. We first apply a robust circuit decomposition to the input circuit to obtain a collection of constraints $f_1, \ldots, f_t$, which can be computed by smaller arithmetic circuits $C_1, \ldots, C_t$, respectively. Each arithmetic circuit takes as input a subset of the components of the statement $\mathbf{x} \in \mathbb{F}^n$ and the witness $\mathbf{w} \in \mathbb{F}^m$. In the following, we write $\mathbf{x}_i$ and $\mathbf{w}_i$ to denote the subset of the components of $\mathbf{x}$ and $\mathbf{w}$, respectively, that circuit C_i reads. We can now construct a linear MIP with t provers as follows. A proof of a true statement $\mathbf{x}'$ with witness $\mathbf{w}'$ consists of t proof vectors $(\boldsymbol{\pi}_1, \ldots, \boldsymbol{\pi}_t)$, where each proof $\boldsymbol{\pi}_i$ is a linear PCP proof that $C_i(\mathbf{x}_i, \cdot)$ is satisfiable. Then, in the linear MIP model, the verifier has oracle access to the linear functions $\boldsymbol{\pi}_1, \ldots, \boldsymbol{\pi}_t$, which it can use to check satisfiability of $C_i(\mathbf{x}_i, \cdot)$. Completeness of this construction is immediate from completeness of the robust decomposition.

Soundness is more challenging to argue. For any false statement $\mathbf{x}'$, robustness of the decomposition of C only ensures that for any witness $\mathbf{w} \in \mathbb{F}^m$, at least a constant fraction of the constraints $f_i(\mathbf{x}, \mathbf{w})$ will not be satisfied, where $\mathbf{x} = \mathsf{inp}(\mathbf{x}')$. However, this does *not* imply that a constant fraction of the individual circuits $C_i(\mathbf{x}_i, \cdot)$ is unsatisfiable. For instance, for all i, there could exist some witness $\mathbf{w}_i$

such that $C_i(\mathbf{x}_i, \mathbf{w}_i) = 1$. This does *not* contradict the robustness of the decomposition so long as the set of all satisfying witnesses $\{\mathbf{w}_i\}$ contain many "inconsistent" assignments. More specifically, we can view each $\mathbf{w}_i$ as assigning values to some subset of the components of the overall witness $\mathbf{w}$, and we say that a collection of witnesses $\{\mathbf{w}_i\}$ is consistent if whenever two witnesses $\mathbf{w}_i$ and $\mathbf{w}_j$ assign a value to the same component of $\mathbf{w}$, they assign the *same* value. Thus, robustness only ensures that the prover cannot find a *consistent* set of witnesses $\{\mathbf{w}_i\}$ that can simultaneously satisfy more than a fraction of the circuits C_i. Or equivalently, if $\mathbf{x}$ is the encoding of a false statement $\mathbf{x}'$, then a constant fraction of any set of witnesses $\{\mathbf{w}_i\}$ where $C_i(\mathbf{x}_i, \mathbf{w}_i) = 1$ must be mutually inconsistent.

The above analysis shows that it is insufficient for the prover to independently argue satisfiability of each circuit $C_i(\mathbf{x}_i, \cdot)$. Instead, we need the stronger requirement that the prover uses a *consistent* set of witnesses $\{\mathbf{w}_i\}$ when constructing its proofs $\boldsymbol{\pi}_1, \ldots, \boldsymbol{\pi}_t$. Thus, we need a way to bind each proof $\boldsymbol{\pi}_i$ to a specific witness $\mathbf{w}_i$, as well as a way for the verifier to check that the complete set of witnesses $\{\mathbf{w}_i\}$ are mutually consistent. For the first requirement, we introduce the notion of a *systematic linear PCP*, which is a linear PCP where the linear PCP proof vector $\boldsymbol{\pi}_i$ contains a copy of a witness $\mathbf{w}_i$ where $C_i(\mathbf{x}_i, \mathbf{w}_i) = 1$ (Definition 5.23). Now, given a collection of systematic linear PCP proofs $\boldsymbol{\pi}_1, \ldots, \boldsymbol{\pi}_t$, the verifier's goal is to decide whether the witnesses $\mathbf{w}_1, \ldots, \mathbf{w}_t$ embedded within $\boldsymbol{\pi}_1, \ldots, \boldsymbol{\pi}_t$ are mutually consistent. Since the witnesses $\mathbf{w}_i$ are part of the proof vectors $\boldsymbol{\pi}_i$, in the remainder of this section, we will simply assume that the verifier has oracle access to the linear function $\langle \mathbf{w}_i, \cdot \rangle$ for all i since such queries can be simulated using the proof oracle $\langle \boldsymbol{\pi}_i, \cdot \rangle$.

5.1.1 Consistency Checking

The robust decomposition ensures that for a false statement $\mathbf{x}'$, any collection of witnesses $\{\mathbf{w}_i\}$ where $C_i(\mathbf{x}_i, \mathbf{w}_i) = 1$ for all i is guaranteed to have many inconsistencies. In fact, there must always exists $\Omega(t)$ (mutually disjoint) pairs of witnesses that contain some inconsistency in their assignments. Ensuring soundness thus reduces to developing an efficient method for testing whether $\mathbf{w}_1, \ldots, \mathbf{w}_t$ constitute a consistent assignment to the components of $\mathbf{w}$ or not. This is the main technical challenge in constructing quasi-optimal linear MIPs, and our construction proceeds in several steps, which we describe below.

Notation. We begin by introducing some notation. First, we pack the different witnesses $\mathbf{w}_1, \ldots, \mathbf{w}_t \in \mathbb{F}^q$ into the rows of an *assignment matrix* $\mathbf{W} \in \mathbb{F}^{t \times q}$. Specifically, the i^{th} row of $\mathbf{W}$ is the witness $\mathbf{w}_i$. Next, we define the *replication structure* for the circuits $C_1, \ldots, C_t$ to be a matrix $\mathbf{A} \in [m]^{t \times q}$. Here, the $(i, j)^{\text{th}}$ entry $\mathbf{A}_{i,j}$ encodes the index in $\mathbf{w} \in \mathbb{F}^m$ to which the j^{th} entry in $\mathbf{w}_i$ corresponds. With this notation, we say that the collection of witnesses $\mathbf{w}_1, \ldots, \mathbf{w}_t$ are consistent if for all indices (i_1, j_1) and (i_2, j_2) where $\mathbf{A}_{i_1,j_1} = \mathbf{A}_{i_2,j_2}$, the assignment matrix satisfies $\mathbf{W}_{i_1,j_1} = \mathbf{W}_{i_2,j_2}$.

Checking global consistency. To check whether an assignment matrix $\mathbf{W} \in \mathbb{F}^{t \times q}$ is consistent with respect to the replication structure $\mathbf{A} \in [m]^{t \times q}$, we can leverage an idea from Groth [Gro09], and subsequently used in [IPS09] for performing similar consistency checks. The high-level idea is as follows. Take any index $z \in [m]$ and consider the positions $(i_1, j_1), \ldots, (i_d, j_d)$ where z appears in $\mathbf{A}$. In this way, we associate a disjoint set of Hamiltonian cycles over the entries of $\mathbf{A}$, one for each of the m components of $\mathbf{w}$. Let Π be a permutation over the entries in the matrix $\mathbf{A}$ such that Π splits into a product of the Hamiltonian cycles induced by the entries of $\mathbf{A}$. In particular, this means $\mathbf{A} = \Pi(\mathbf{A})$, and moreover, $\mathbf{W}$ is consistent with respect to $\mathbf{A}$ if and only if $\mathbf{W} = \Pi(\mathbf{W})$. The insight in [Gro09] is that the relation $\mathbf{W} = \Pi(\mathbf{W})$ can be checked using two sets of linear queries. First, the verifier draws vectors $\mathbf{r}_1, \ldots, \mathbf{r}_t \xleftarrow{\text{R}} \mathbb{F}^q$ and defines the matrix $\mathbf{R} \in \mathbb{F}^{t \times q}$ to be the matrix whose rows are $\mathbf{r}_1, \ldots, \mathbf{r}_t$. Next, the verifier computes the permuted matrix $\mathbf{R}' \leftarrow \Pi(\mathbf{R})$. Let $\mathbf{r}'_1, \ldots, \mathbf{r}'_t$ be the rows of $\mathbf{R}'$. Similarly, let $\mathbf{w}_1, \ldots, \mathbf{w}_t$ be the rows of $\mathbf{W}$. Finally, the verifier queries the linear MIP oracles $\langle \mathbf{w}_i, \cdot \rangle$ on $\mathbf{r}_i$ and $\mathbf{r}'_i$ for all i and checks the relation

$$\sum_{i \in [t]} \langle \mathbf{w}_i, \mathbf{r}_i \rangle \stackrel{?}{=} \sum_{i \in [t]} \langle \mathbf{w}_i, \mathbf{r}'_i \rangle \in \mathbb{F}. \tag{5.1}$$

By construction of Π, if $\mathbf{W} = \Pi(\mathbf{W})$, this check always succeeds. However, if $\mathbf{W} \neq \Pi(\mathbf{W})$, then by the Schwartz-Zippel lemma (Lemma 2.2), this check rejects with probability $1/|\mathbb{F}|$. When working over a polynomial-size field, this consistency check achieves $1/\text{poly}(\lambda)$ soundness (where λ is a security parameter). We can use repeated queries to amplify the soundness to $\text{negl}(\lambda)$ without sacrificing quasi-optimality. However, this approach cannot give a linear MIP with $2^{-\lambda}$ soundness and still retain prover overhead that is only polylogarithmic in λ (since we would require $\Omega(\lambda)$ repetitions). To overcome this problem, we require a more robust consistency checking procedure.

Checking pairwise consistency. The consistency check described above and used in [Gro09, IPS09] is designed for checking *global* consistency of all of the assignments in $\mathbf{W} \in \mathbb{F}^{t \times q}$. The main disadvantage of performing the global consistency check in Eq. (5.1) is that it only provides soundness $1/|\mathbb{F}|$, which is insufficient when $\mathbb{F}$ is small (e.g., in the case of Boolean circuit satisfiability). One way to amplify soundness is to replace the single global consistency check with $t/2$ *pairwise* consistency checks, where each pairwise consistency check affirms that the assignments in a (mutually disjoint) pair of rows of $\mathbf{W}$ are self-consistent. Specifically, each of the $t/2$ checks consists of two queries $(\mathbf{r}_i, \mathbf{r}_j)$ and $(\mathbf{r}'_i, \mathbf{r}'_j)$ to $\langle \mathbf{w}_i, \cdot \rangle$ and $\langle \mathbf{w}_j, \cdot \rangle$, constructed in exactly the same manner as in the global consistency check, except specialized to only checking for consistency in the assignments to the variables in rows i and j. Since all of the pairwise consistency checks are independent, if there are $\Omega(t)$ pairs of inconsistent rows, the probability that all $t/2$ checks pass is bounded by $2^{-\Omega(t)}$. This means that for the same cost as performing a *single* global consistency check, the verifier can perform $\Omega(t)$ pairwise consistency checks. As long as many of the pairs of rows the verifier checks contain inconsistencies, we achieve soundness amplification.

Recall from earlier that our robust decomposition guarantees that whenever $\mathbf{x}_1, \ldots, \mathbf{x}_t$ correspond to a false statement, any collection of witnesses $\{\mathbf{w}_i\}$ where $C_i(\mathbf{x}_i, \mathbf{w}_i)$ is satisfied for all i necessarily contains many pairs $\mathbf{w}_i$ and $\mathbf{w}_j$ that are inconsistent. Equivalently, many pairs of rows in the assignment matrix $\mathbf{W}$ contain inconsistencies. Now, if the verifier knew which pairs of rows of $\mathbf{W}$ are inconsistent, then the verifier can apply a pairwise consistency check to detect an inconsistent $\mathbf{W}$ with high probability. The problem, however, is that the verifier does not know *a priori* which pairs of rows in $\mathbf{W}$ are inconsistent, and so, it is unclear how to choose the rows to check in the pairwise consistency test. However, if we make the stronger assumption that not only are there many pairs of rows in $\mathbf{W}$ that contain inconsistent assignments, but also, that most of these inconsistencies appear in *adjacent* rows, then we can use a pairwise consistency test (where each test checks for consistency between an adjacent pair of rows) to decide if $\mathbf{W}$ is consistent or not. When the assignment matrix $\mathbf{W}$ has many inconsistencies in pairs of adjacent rows, we say that the inconsistency pattern of $\mathbf{W}$ is "regular," and can be checked using a pairwise consistency test.

Regularity-inducing permutations. To leverage the pairwise consistency check, we require that the assignment matrix $\mathbf{W}$ has a regular inconsistency structure that is amenable to a pairwise consistency check. To ensure this, we introduce the notion of a *regularity-inducing permutation.* Our construction relies on the observation that the assignment matrix $\mathbf{W}$ is consistent with a replication structure $\mathbf{A}$ if and only if $\Pi(\mathbf{W})$ is consistent with $\Pi(\mathbf{A})$, where Π is an arbitrary permutation over the entries of a t-by-q matrix. Thus, if we want to check consistency of $\mathbf{W}$ with respect to $\mathbf{A}$, it suffices to check consistency of $\Pi(\mathbf{W})$ with respect to $\Pi(\mathbf{A})$. Then, we say that a specific permutation Π is regularity-inducing with respect to a replication structure $\mathbf{A}$ if whenever $\mathbf{W}$ has many pairs of inconsistent rows with respect to $\mathbf{A}$ (e.g., $\mathbf{W}$ is a set of accepting witnesses to a false statement), then $\Pi(\mathbf{W})$ has many inconsistencies in pairs of *adjacent* rows with respect to $\Pi(\mathbf{A})$. In other words, a regularity-inducing permutation shuffles the entries of the assignment matrix such that any inconsistency pattern in $\mathbf{W}$ maps to a regular inconsistency pattern according to the replication structure $\Pi(\mathbf{A})$. In the construction, instead of performing the pairwise consistency test on $\mathbf{W}$, which can have an arbitrary inconsistency pattern, we perform it on $\Pi(\mathbf{W})$, which has a regular inconsistency pattern. We define the notion more formally in Section 5.3.2 and show how to construct regularity-inducing permutations in Section 5.3.4.

Decomposing the permutation. Suppose Π is a regularity-inducing permutation for the replication structure $\mathbf{A}$ associated with the circuits $C_1, \ldots, C_t$ from the robust decomposition of C. Robustness ensures that for any false statement $\mathbf{x}'$, for all collections of witnesses $\{\mathbf{w}_i\}$ where $C_i(\mathbf{x}_i, \mathbf{w}_i) = 1$ for all i, and $\mathbf{x} = \mathsf{inp}(\mathbf{x}')$, the permuted assignment matrix $\Pi(\mathbf{W})$ has inconsistencies in $\Omega(t)$ pairs of adjacent rows with respect to $\Pi(\mathbf{A})$. This can be detected with probability $1 - 2^{-\Omega(t)}$ by performing a pairwise consistency test on the matrix $\mathbf{W}' = \Pi(\mathbf{W})$. The problem, however, is that the verifier only has oracle access to $\langle \mathbf{w}_i, \cdot \rangle$, and it is unclear how to *efficiently* perform the pairwise consistency test on the permuted matrix $\mathbf{W}'$ given just oracle access to the rows $\mathbf{w}_i$ of the unpermuted matrix. Our solution here is to introduce another set of t linear MIP provers for each row $\mathbf{w}'_i$ of $\mathbf{W}' = \Pi(\mathbf{W})$. Thus, the verifier has oracle access to both the rows of the original assignment matrix $\mathbf{W}$, which it uses to check satisfiability of $C_i(\mathbf{x}_i, \cdot)$, as well as the rows of the permuted assignment matrix $\mathbf{W}'$, which it uses to check consistency of the assignments in $\mathbf{W}$. The verifier accepts only if both sets of checks pass. The problem with this basic approach is that there is no reason the prover chooses the matrix $\mathbf{W}'$ so as to satisfy the relation $\mathbf{W}' = \Pi(\mathbf{W})$. Thus, to ensure soundness from this approach, the verifier needs a mechanism to also check that $\mathbf{W}' = \Pi(\mathbf{W})$, given oracle access to the rows of $\mathbf{W}$ and $\mathbf{W}'$.

To facilitate this check, we decompose the permutation Π into a sequence of α permutations $(\Pi_1, \dots, \Pi_\alpha)$ where $\Pi = \Pi_\alpha \circ \cdots \circ \Pi_1$. Moreover, each of the intermediate permutations Π_i has the property that they themselves can be decomposed into $t/2$ independent permutations, each of which only permutes entries that appear in 2 distinct rows of the matrix. This "2-locality" property on permutations is amenable to the linear MIP model, and we show in Construction 5.29 a way for the verifier to efficiently check that two matrices $\mathbf{W}$ and $\mathbf{W}'$ (approximately) satisfy the relation $\mathbf{W} = \Pi_i(\mathbf{W}')$, where Π_i is 2-locally decomposable. To complete the construction, we have the prover provide not just the matrix $\mathbf{W}$ and its permutation $\mathbf{W}'$, but all of the intermediate matrices $\mathbf{W}_i = (\Pi_i \circ \Pi_{i-1} \circ \cdots \circ \Pi_1)(\mathbf{W})$ for all $i = 1, \dots, \alpha$. Since each of the intermediate permutations applied are 2-locally decomposable, there is an efficient procedure for the prover to check each relation $\mathbf{W}_i = \Pi_i(\mathbf{W}_{i-1})$, where we write $\mathbf{W}_0 = \mathbf{W}$ to denote the original assignment matrix. If each of the intermediate permutations are correctly implemented, then the verifier is assured that $\mathbf{W}' = \Pi(\mathbf{W})$, and it can apply the pairwise consistency check on $\mathbf{W}'$ to complete the verification process. We use a Beneš network to implement the decomposition. This ensures that the number of intermediate permutations required is only logarithmic in t, so introducing these additional steps only incurs logarithmic overhead, and does not compromise quasi-optimality of the resulting construction.

Randomized permutation decompositions. There is one additional complication in that the intermediate consistency checks $\mathbf{W}' \stackrel{?}{=} \Pi_i(\mathbf{W})$ are imperfect. They only ensure that *most* of the rows in $\mathbf{W}'$ agree with the corresponding rows in $\Pi_i(\mathbf{W})$. What this means is that when the prover crafts its sequence of permuted assignment matrices $\mathbf{W} = \mathbf{W}_0, \mathbf{W}_1, \dots, \mathbf{W}_\alpha$, it is able to "correct" a small number of inconsistencies that appear in $\mathbf{W}$ in each step. Thus, we must ensure that for the particular inconsistency pattern that appears in $\mathbf{W}$, the prover is not able to find a sequence of matrices $\mathbf{W}_1, \dots, \mathbf{W}_\alpha$, where each of them approximately implements the correct permutation at each step, but at the end, is able to correct all of the inconsistencies in $\mathbf{W}$. To achieve this, we rely on a *randomized permutation decomposition*, where the verifier samples a random sequence of intermediate permutations $\Pi_1, \dots, \Pi_\alpha$ that collectively implement the target regularity-inducing permutation Π. There are a number of technicalities that arise in the construction and its analysis, and we refer to Section 5.3.4 for the full description.

Putting the pieces together. To summarize, our quasi-optimal linear MIP for circuit satisfiability consists of two key components. First, we apply a robust decomposition to the circuit to obtain many constraints with the property that for a false statement, a malicious prover either cannot satisfy most of the constraints, or if it does satisfy all of the constraints, it must have used an assignment with many inconsistencies. The second key ingredient we introduce is an efficient way to check if there are many inconsistencies in the prover's assignments in the linear MIP model. Our construction here relies on first constructing a regularity-inducing permutation to enable a simple method for consistency checking, and then using a randomized permutation decomposition to enforce the consistency check. We give the formal description and analysis in Section 5.3.

5.2 Main Ingredients

In this section, we provide a brief overview of the two main ingredients we require in our quasi-optimal SNARG construction: linear multi-prover interactive proofs (Section 5.2.1) and routing networks (Section 5.2.2).

5.2.1 Linear MIPs

Our construction of quasi-optimal SNARGs relies on much of the same underlying infrastructure of linear proof systems described in Section 4.3. In this section, we further extend these notions and recall the definition of of a linear multi-prover interactive proof (linear MIP) introduced by Ishai et al. [IKO07]. Afterwards, we introduce the notion of a quasi-optimal linear MIP.

Definition 5.1 (Linear MIPs [IKO07, adapted]). Let $\mathcal{R}$ be a binary relation, $\mathbb{F}$ be a finite field, $\mathcal{P} = (P_1, \ldots, P_\ell)$ be a tuple of ℓ prover algorithms, and $\mathcal{V}$ be an oracle verifier algorithm. Then, the pair $(\mathcal{P}, \mathcal{V})$ is an (input-oblivious) k-query linear multi-prover interactive proof (MIP) with ℓ provers for $\mathcal{R}$ over $\mathbb{F}$ with soundness error ε and query length d if it satisfies the following requirements:

- **Syntax:** Each prover algorithm P_i (for $i \in [\ell]$) takes as input a statement $\mathbf{x}$ and a witness $\mathbf{w}$ and outputs a vector $\boldsymbol{\pi}_i \in \mathbb{F}^d$. We write $\mathcal{P}(\mathbf{x}, \mathbf{w})$ to denote the tuple $(P_1(\mathbf{x}, \mathbf{w}), \ldots, P_\ell(\mathbf{x}, \mathbf{w}))$. The verification algorithm $\mathcal{V}^{\boldsymbol{\pi}_1, \ldots, \boldsymbol{\pi}_\ell} = (\mathcal{Q}, \mathcal{D})$ consists of an input-oblivious probabilistic query-generation algorithm $\mathcal{Q}$ and a deterministic decision algorithm $\mathcal{D}$. The query algorithm $\mathcal{Q}$ generates a tuple of query matrices $\mathbf{Q}_1, \ldots, \mathbf{Q}_\ell \in \mathbb{F}^{d \times k}$ and some additional state information st. The decision algorithm $\mathcal{D}$ takes as input the statement $\mathbf{x}$, the verification state st, and the prover responses $\mathbf{y}_1, \ldots, \mathbf{y}_\ell$ where each $\mathbf{y}_i = \mathbf{Q}_i^T \boldsymbol{\pi}_i \in \mathbb{F}^k$, and either "accepts" (with output 1) or "rejects" (with output 0).

- **Completeness:** For every $(\mathbf{x}, \mathbf{w}) \in \mathcal{R}$, and setting $\boldsymbol{\pi}_i \leftarrow P_i(\mathbf{x}, \mathbf{w})$ for all $i \in [\ell]$, we have that $\mathcal{V}^{\boldsymbol{\pi}_1,\ldots,\boldsymbol{\pi}_\ell}(\mathbf{x})$ accepts with probability 1.
- **Soundness:** For every $\mathbf{x} \notin \mathcal{L}$, and all proof vectors $(\boldsymbol{\pi}_1^*, \ldots, \boldsymbol{\pi}_\ell^*)$ where each $\boldsymbol{\pi}_i^* \in \mathbb{F}^d$, the probability that $\mathcal{V}^{\boldsymbol{\pi}_1^*,\ldots,\boldsymbol{\pi}_\ell^*}(\mathbf{x})$ accepts is at most ε. As in Definition 4.2, we can define a corresponding notion of *soundness against affine provers* where soundness holds against provers who each implement a different affine strategy $(\boldsymbol{\pi}_i^*, \mathbf{b}_i^*) \in \mathbb{F}^d \times \mathbb{F}^k$.

Similar to Definition 4.2, we can replace the soundness property with a stronger knowledge property.

Definition 5.2 (Quasi-Optimal Linear MIPs)**.** Let λ be a security parameter, and C be an arithmetic circuit of size s over a finite field $\mathbb{F}$. A k-query linear MIP $(\mathcal{P}, \mathcal{V})$ with ℓ provers for $\mathcal{R}_C$ with soundness error $2^{-\lambda}$ is *quasi-optimal* if the prover $\mathcal{P} = (P_1, \ldots, P_\ell)$ can be implemented by an arithmetic circuit of size $\widetilde{O}(s) + \text{poly}(\lambda, \log s)$, where the $\widetilde{O}(\cdot)$ notation is suppressing terms that are polylogarithmic in s and λ.

Remark 5.3 (Existing Linear PCP Construction)**.** We can view the linear PCP constructions by Bitansky et al. [BCI+13] as well as Construction 4.5 from Section 4.3.1 as a linear MIP with a single prover. However, when viewed as linear MIPs, these constructions are *not* quasi-optimal for arithmetic circuit satisfiability over a polynomial-size field (i.e., for Boolean circuit satisfiability). To provide soundness error $2^{-\lambda}$, the aforementioned linear PCP constructions either embed the circuit satisfiability instance inside a field of size $2^{\Omega(\lambda)}$, or have query complexity $O(\lambda)$. In both cases, the prover complexity becomes $\Omega(\lambda s) + \text{poly}(\lambda, \log s)$. Thus, the existing SNARG constructions are not quasi-optimal for Boolean circuit satisfiability.

5.2.2 Routing Networks

Our quasi-optimal linear MIP construction in Section 5.3 relies on an efficient method for checking whether two matrices $\mathbf{W}, \mathbf{W}' \in \mathbb{F}^{t \times q}$ satisfy $\mathbf{W} = \Pi(\mathbf{W}')$ where Π is an arbitrary permutation over the entries of a t-by-q matrix. We begin by stating a lemma from [GHS12] that states that an arbitrary permutation Π over the entries of a t-by-q matrix can be decomposed into the composition of a small number of permutations, where each permutation implements a row-wise permutation or a column-wise permutation of the matrix entries.

Definition 5.4 (Matrix Permutations)**.** Fix integers t, q and let Π be a permutation over the entries of a t-by-q matrix. We say that Π is *row-wise restricted* if Π only permutes elements within the rows of the matrix (that is, the permutation only changes the column, and not the row, of each element). Similarly, we say that Π is *column-wise restricted* if Π only permutes elements within the columns of the matrix.

Lemma 5.5 ([GHS12, Lemma 1])**.** *Fix positive integers $t, q \in \mathbb{N}$, and let Π be a permutation over the entries of a t-by-q matrix. Then, there exist permutations Π_1, Π_2, Π_3 such that $\Pi = \Pi_3 \circ \Pi_2 \circ \Pi_1$,*

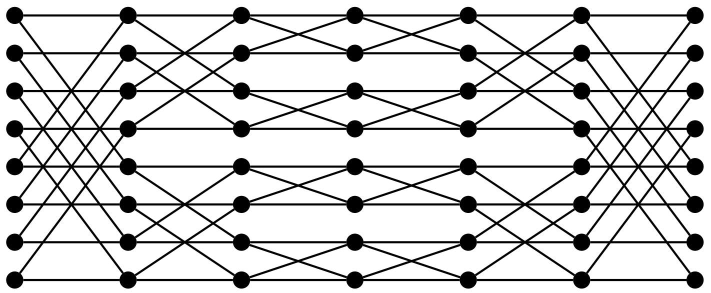
Figure 5.1: A Beneš network over $8 = 2^3$ nodes (BENEŠ$_3$).

where Π_1 *and* Π_3 *are row-wise restricted, and* Π_2 *is column-wise restricted. Moreover, there is an efficient algorithm to compute* Π_1, Π_2, Π_3 *given* Π.

Beneš networks. A Beneš permutation network [Ben64] is a special graph that can model all permutations Π on a collection of $m = 2^d$ elements. We give the precise definition below, and then state an elementary property on the structure of Beneš networks. We show an example of a Beneš network in Figure 5.1.

Definition 5.6 (Beneš Network [Ben64])**.** For a non-negative integer d, a d-dimensional Beneš network, denoted BENEŠ$_d$, is a directed graph with $2d + 1$ layers (labeled $0, \ldots, 2d$). Each layer contains $m = 2^d$ nodes (labeled $0, \ldots, 2^d - 1$), and edges only go from nodes in layer $i - 1$ to nodes in layer i for $i \in [2d]$. The first layer (layer 0) is the input layer and the final layer (layer $2d$) is the output layer. The graph structure is defined recursively as follows:

- A 0-dimensional Beneš network BENEŠ$_0$ consists of a single node.
- A d-dimensional Beneš network BENEŠ$_d$ consists of an input layer (with 2^d nodes) that feeds into two stacked BENEŠ$_{d-1}$ networks, which feed into an output layer. The edge configuration of the input and output layers are then defined as follows, for $i = 0, \ldots, 2^{d-1} - 1$:
  - **Input edges:** There is an edge from the i^{th} input of BENEŠ$_d$ to the i^{th} input of each of the BENEŠ$_{d-1}$ networks. There is also an edge from the $(i + 2^{d-1})^{\text{th}}$ input of BENEŠ$_d$ to the i^{th} input of each of the BENEŠ$_{d-1}$ networks.
  - **Output edges:** There is an edge from the i^{th} output of each of the BENEŠ$_{d-1}$ networks to the i^{th} output of BENEŠ$_d$. There is also an edge from the i^{th} output of each BENEŠ$_{d-1}$ network to the $(i + 2^{d-1})^{\text{th}}$ output of the BENEŠ$_d$ network.

Fact 5.7 (Structure of Beneš Networks)**.** Fix a positive integer $d \in \mathbb{N}$, and let $S = \{0, \ldots, 2^d - 1\}$. Then, a BENEŠ$_d$ network has the following structural properties:

- For $j \in \{0, \ldots, d\}$, we can partition the nodes in layer j into $\ell_j = 2^{d-j}$ disjoint sets $S_0^{(j)}, \ldots, S_{\ell_j - 1}^{(j)} \subseteq S$, each containing 2^j nodes, with the following properties:
  - For $k \in \{0, \ldots, 2^{d-j} - 1\}$, $S_k^{(j)}$ contains all indices $i \in S$ where the least-significant $d - j$ bits of i is equal to k.
  - For all nodes $i_s, i_t \in S_k^{(j)}$ for some k, there is a unique path of length j from the i_s^{th} input node to the i_t^{th} node in layer j of $\textsc{beneš}_d$.
  - For any two nodes $i_s \in S_k^{(j)}$, $i_t \in S_{k'}^{(j)}$ where $k \neq k'$, there are no paths from the i_s^{th} input node to the i_t^{th} node in layer j of $\textsc{beneš}_d$.
- For $j \in \{d, \ldots, 2d\}$, we can partition the nodes in layer j into $\ell_j = 2^{j-d}$ disjoint sets $S_0^{(j)}, \ldots, S_{\ell-1}^{(j)} \subseteq S$, each containing 2^{2d-j} nodes, with the following properties:
  - For $k \in \{0, \ldots, 2^{j-d} - 1\}$, $S_k^{(j)}$ contains all indices $i \in S$ where the least-significant $j - d$ bits of i is equal to k.
  - For all nodes $i_s, i_t \in S_k^{(j)}$ for some k, there is a unique path of length j from the i_s^{th} node in layer j to the i_t^{th} output node in $\textsc{beneš}_d$.
  - For all nodes $i_s \in S_k^{(j)}$, $i_t \in S_{k'}^{(j)}$ where $k \neq k'$, there are no paths from the i_s^{th} node in layer j to the i_t^{th} output node of $\textsc{beneš}_d$.

A key property of Beneš networks is that they are rearrangeable: any permutation Π on $m = 2^d$ values can be mapped to a set of 2^d node-disjoint paths in a d-dimensional Beneš network $\textsc{beneš}_d$ where the i^{th} path maps from input i to output $\Pi(i)$. We state the following fact from [Wak68, OTW71].

Fact 5.8 (Rearrangeability of Beneš Networks [Wak68, OTW71])**.** Let $d \in \mathbb{N}$ be a positive integer and let $S = \{0, \ldots, 2^d - 1\}$. For all permutations $\Pi \colon S \to S$ there exists a set of 2^d *node-disjoint* paths from each input node $i \in S$ in $\textsc{beneš}_d$ to its corresponding output node $\Pi(i)$. Moreover, these paths can be computed in time $O(d \cdot 2^d)$. We say that the paths $\mathcal{P}$ implement the permutation Π.

Randomized routing in Beneš networks. Fact 5.8 states that we can route any permutation Π on $[2^d]$ elements using a d-dimensional Beneš network. In our construction, we will use the following procedure to randomize the routing configuration.

Construction 5.9 (Randomized Routing)**.** Let $d \in \mathbb{N}$ be a positive integer and $S = \{0, \ldots, 2^d - 1\}$. For a permutation $\Pi \colon S \to S$ on S, let $\mathcal{P}$ be a collection of node-disjoint paths in $\textsc{beneš}_d$ that routes each input node $i \in S$ to its corresponding output node $\Pi(i)$. Namely, for each $i \in S$ and $j \in \{0, \ldots, 2d\}$, let $\mathcal{P}[i, j]$ be the value of node i in layer j of the $\textsc{beneš}_d$ network. We construct a new collection of paths $\mathcal{P}'$ as follows:

1. Initialize $\mathcal{P}' \leftarrow \mathcal{P}$. Then for $j \in [d]$ and $k = 0, \ldots, 2^{j-1} - 1$, do the following:

(a) Choose a random bit $b_{j,k} \xleftarrow{\text{R}} \{0,1\}$.

(b) For $i \in S$, let $i_1 i_2 \cdots i_d$ be the binary representation of i. Then, if $b_{j,k} = 1$, for all nodes $i \in S$ where $i_1 i_2 \cdots i_{j-1} = k$ and $i_j = 0$, swap the values $\mathcal{P}'[i, j']$ and $\mathcal{P}'[i + 2^{d-j}, j']$, for all $j' = j, \ldots, d - j$.

2. Output the randomized collection of paths $\mathcal{P}'$.

At a high level, Construction 5.9 takes a set of paths $\mathcal{P}$ implementing a specific permutation Π in a BENEŠ$_d$ network and produces a new set of paths $\mathcal{P}'$ in BENEŠ$_d$ that implement the same permutation. The procedure relies on the recursive structure of the Beneš network. For example, in the first layer of a BENEŠ$_d$ network, the input nodes are partitioned into two disjoint sets, one of which is routed using the top BENEŠ$_{d-1}$ network, and the other is routed using the bottom BENEŠ$_{d-1}$ network. The randomization procedure in Construction 5.9 maintains the same partitioning of input nodes, but each partition is either routed using the top BENEŠ$_{d-1}$ network or the bottom BENEŠ$_{d-1}$ network with equal probability. This process is then iteratively applied to permute the routing configuration for each of the BENEŠ$_{d-1}$ networks in the first layer, and so on. We state the formal correctness guarantee in the following lemma:

Lemma 5.10. *Let $d \in \mathbb{N}$ be a positive integer, $S = \{0, \ldots, 2^d - 1\}$, and $\Pi \colon S \to S$ be a permutation on S. Let $\mathcal{P}$ be a collection of node-disjoint paths in* BENEŠ$_d$ *that implements the permutation Π. Then, the new collection of paths $\mathcal{P}'$ obtained by applying the randomized routing procedure in Construction 5.9 to $\mathcal{P}$ is also a collection of node-disjoint paths in* BENEŠ$_d$ *that implements the same permutation Π.*

For any sequence of paths $\mathcal{P}$ implementing a permutation Π in a BENEŠ$_d$ network, the set $\mathcal{P}'$ of randomized paths output by Construction 5.9 has the property that if we consider the path of any *single* input node $i \in \{0, \ldots, 2^d - 1\}$ to $\Pi(i)$ in $\mathcal{P}'$, its path is distributed uniformly over all of the 2^d possible paths from i to $\Pi(i)$ in BENEŠ$_d$. We state the precise guarantee in the following lemma:

Lemma 5.11. *Let $d \in \mathbb{N}$ be a positive integer, $S = \{0, \ldots, 2^d - 1\}$, and $\Pi \colon S \to S$ be a permutation on S. Let $\mathcal{P}$ be a collection of node-disjoint paths in* BENEŠ$_d$ *that implements Π, and let $\mathcal{P}'$ be the set of randomized paths output by Construction 5.9 applied to $\mathcal{P}$. For a node $i \in S$, let $i = i_0, i_1, \ldots, i_{2d} = \Pi(i)$ denote the path of i in $\mathcal{P}'$. Then the following holds:*

- *For $j \in \{0, \ldots, d\}$, let $S_0^{(j)}, \ldots, S_{\ell_j - 1}^{(j)}$ be the partition of the nodes in layer j from Fact 5.7. Let $k \in \{0, \ldots, \ell_j - 1\}$ such that $i \in S_k^{(j)}$. Then, for all $i' \in S_k^{(j)}$, $\Pr[i_j = i'] = 1/\left|S_k^{(j)}\right| = 1/2^j$.*
- *For $j \in \{d, \ldots, 2d\}$, let $S_0^{(j)}, \ldots, S_{\ell_j - 1}^{(j)}$ be the partition of the nodes in layer j from Fact 5.7. Let $k \in \{0, \ldots, \ell_j - 1\}$ such that $\Pi(i) \in S_k^{(j)}$. Then, for all $i' \in S_k^{(j)}$, $\Pr[i_j = i'] = 1/\left|S_k^{(j)}\right| = 1/2^{2d-j}$.*

In both cases, the probability is taken over the random coins in the randomization algorithm (Construction 5.9).

Permutations from Beneš networks. We can view a collection of paths in a Beneš network as providing a systematic decomposition of an arbitrary permutation Π on $t = 2^d$ elements into a sequence of $\ell = O(\log t)$ permutations $\Pi_1, \ldots, \Pi_\ell$, where each $\Pi_1, \ldots, \Pi_\ell$ can be expressed as a product of disjoint 2-cycles, and $\Pi = \Pi_\ell \circ \cdots \circ \Pi_1$. More precisely, we can associate values $x_0, \ldots, x_{t-1}$ with the input nodes of the $\textsc{beneš}_d$ network (e.g., value x_i with the i^{th} input node). Given a path from an input node to an output node, we associate the value of the input node with every node along the path. Then, any collection of t node-disjoint paths $\mathcal{P}$ from input nodes to output nodes induces an assignment to every node in the network. Now, for any permutation Π on t values $x_0, \ldots, x_{t-1}$, we define $\Pi_1, \ldots, \Pi_\ell$ so that $(\Pi_i \circ \cdots \circ \Pi_1)(x_0, \ldots, x_{t-1})$ gives the values of the nodes in layer $i+1$ of $\textsc{beneš}_d$ on input $x_0, \ldots, x_{t-1}$ and paths determined by $\mathcal{P}$. The structure of the Beneš network ensures that each of the $\Pi_1, \ldots, \Pi_\ell$ is a product of *disjoint* 2-cycles. We say that permutations with this property (generalized to permutations over the entries of a matrix) are 2-locally decomposable (Definition 5.12). We give a more precise description of this decomposition in Construction 5.13. We formalize two properties satisfied by the decomposition in Lemma 5.14.

Definition 5.12 (2-Local Decomposability)**.** Fix an even integer $t \in \mathbb{N}$, an integer $q \in \mathbb{N}$, and set $t' = t/2$. Let Π be a permutation on the entries of a t-by-q matrix. We say that Π is 2-*locally decomposable* if there exists a partition $\{j_1, j_2\}, \ldots, \{j_{t'-1}, j_{t'}\}$ of $[t]$ and permutations $\Pi_1, \ldots, \Pi_{t'}$ over 2-by-q matrices such that for all matrices $\mathbf{W} \in \mathbb{F}^{q \times t}$, we have that $\hat{\mathbf{W}} = \Pi(\mathbf{W})$ if and only if for all $i \in [t']$,

$$\hat{\mathbf{W}}_{[j_{2i}, j_{2i+1}]} = \Pi_i\big(\mathbf{W}_{[j_{2i}, j_{2i+1}]}\big).$$

In other words, a permutation Π is 2-locally decomposable if Π can be written as a composition of $t' = t/2$ disjoint permutations that each operate on exactly two rows of the matrix.

Construction 5.13 (Randomized 2-Local Decomposition)**.** Let $t, q \in \mathbb{N}$ be integers where $t = 2^d$ for some $d \in \mathbb{N}$. Let Π be an arbitrary column-wise restricted permutation on the entries of a t-by-q matrix. The randomized 2-local decomposition of Π is a sequence of $2d$ matrices $\Pi_1, \ldots, \Pi_{2d}$ constructed as follows:

- For each column $i \in [q]$, let $\mathcal{P}_i$ be a collection of paths in a $\textsc{beneš}_d$ network that implements the permutation Π on the entries in column i. Let $\mathcal{P}_i'$ be the output of the randomized routing procedure in Construction 5.9 applied to $\mathcal{P}_i$.
- For $j \in [2d]$, we take Π_j to be a column-wise restricted permutation on t-by-q matrices. We write $\Pi_j^{(i)}$ to denote the permutation Π_j implements on column $i \in [q]$. We define $\Pi_1, \ldots, \Pi_{2d}$ so that for all $i \in [q]$ and $j \in [2d]$, $(\Pi_j^{(i)} \circ \cdots \circ \Pi_1^{(i)})(x_0, \ldots, x_{t-1})$ gives the values of the nodes in layer j of a $\textsc{beneš}_d$ network on input $(x_0, \ldots, x_{t-1})$ and using paths defined by $\mathcal{P}_i'$.

Lemma 5.14. *Let $t, q \in \mathbb{N}$ be integers where $t = 2^d$ for some $d \in \mathbb{N}$. Let Π be an arbitrary column-wise restricted permutation on the entries of a t-by-q matrix, and let $(\Pi_1, \ldots, \Pi_{2d})$ be the randomized*

2-local decomposition of Π *from Construction 5.13. The local decomposition satisfies the following properties:*

- *Each* $\Pi_1, \ldots, \Pi_{2d}$ *is a column-wise restricted and 2-locally decomposable.*
- $\Pi = \Pi_{2d} \circ \cdots \circ \Pi_1$.

Proof. The first property follows immediately from the structure of Beneš networks, and the second follows by construction. □

5.3 Quasi-Optimal Linear MIPs

In this section, we present our core information-theoretic construction of a linear MIP with quasi-optimal prover complexity. We refer to Section 5.1 for a high-level overview of the construction. In Sections 5.3.1 and 5.3.2, we introduce the key building blocks underlying our construction. We give the full construction of our quasi-optimal linear MIP in Section 5.3.3. We show how to instantiate our core building blocks in Section 5.3.4.

5.3.1 Robust Decomposition for Circuit Satisfiability

In this section, we formally define our notion of a robust decomposition of an arithmetic circuit. We then show how to instantiate the robust decomposition by combining the MPC-in-the-head paradigm [IKOS07] with robust MPC protocols with polylogarithmic overhead [DIK10]. We refer to the technical overview in Section 5.1 for a high-level description of how we implement our decomposition.

Definition 5.15 (Quasi-Optimal Robust Decomposition)**.** Let $C \colon \mathbb{F}^{n'} \times \mathbb{F}^{m'} \to \mathbb{F}^{h'}$ be an arithmetic circuit of size s over a finite field $\mathbb{F}$, $\mathcal{R}_C$ be its associated relation, and $\mathcal{L}_C \subseteq \mathbb{F}^{n'}$ be its associated language. A (t, δ)-robust decomposition of C consists of the following components:

- A collection of functions $f_1, \ldots, f_t$ where each function $f_i \colon \mathbb{F}^n \times \mathbb{F}^m \to \{0,1\}$ can be computed by an arithmetic circuit C_i of size $\widetilde{O}(s/t) + \operatorname{poly}(t, \log s)$. Note that a function f_i may only depend on a (fixed) subset of its input variables; in this case, its associated arithmetic circuit C_i only needs to take the (fixed) subset of dependent variables as input.
- An efficiently-computable mapping $\mathsf{inp} \colon \mathbb{F}^{n'} \to \mathbb{F}^n$ that maps between a statement $\mathbf{x}' \in \mathbb{F}^{n'}$ for C to a statement $\mathbf{x} \in \mathbb{F}^n$ for $f_1, \ldots, f_t$.
- An efficiently-computable mapping $\mathsf{wit} \colon \mathbb{F}^{n'} \times \mathbb{F}^{m'} \to \mathbb{F}^m$ that maps between a statement-witness pair $(\mathbf{x}', \mathbf{w}') \in \mathbb{F}^{n'} \times \mathbb{F}^{m'}$ to C to a witness $\mathbf{w} \in \mathbb{F}^m$ for $f_1, \ldots, f_t$.

Moreover, the decomposition must satisfy the following properties:

- **Completeness:** For all $(\mathbf{x}', \mathbf{w}') \in \mathcal{R}_C$, if we set $\mathbf{x} = \mathsf{inp}(\mathbf{x}')$ and $\mathbf{w} = \mathsf{wit}(\mathbf{x}', \mathbf{w}')$, then $f_i(\mathbf{x}, \mathbf{w}) = 1$ for all $i \in [t]$.
- **δ-Robustness:** For all statements $\mathbf{x}' \notin \mathcal{L}_C$, if we set $\mathbf{x} = \mathsf{inp}(\mathbf{x}')$, then it holds that for all $\mathbf{w} \in \mathbb{F}^m$, the set of indices $S_{\mathbf{w}} = \{i \in [t] : f_i(\mathbf{x}, \mathbf{w}) = 1\}$ satisfies $|S_{\mathbf{w}}| < \delta t$. In other words, any single witness $\mathbf{w}$ can only simultaneously satisfy at most a δ-fraction of the constraints.
- **Efficiency:** The mappings inp and wit can be computed by an arithmetic circuit of size $\widetilde{O}(s) + \mathrm{poly}(t, \log s)$.

Instantiating the robust decomposition. We now show how to instantiate our robust decomposition using secure multiparty computation (MPC) protocols with polylogarithmic overhead [DIK10]. Our decomposition follows the "MPC-in-the-head" paradigm of Ishai et al. [IKOS07]. In Remark 5.22, we discuss some of the similarities between our robust circuit decomposition and the duality between traditional PCPs and constraint satisfaction problems.

MPC preliminaries. We begin by reviewing some standard MPC definitions [Gol04, Can00]. Let t be the number of players, denoted $P_1, \ldots, P_t$. We assume that all players communicate synchronously over secure point-to-point channels. We model the functionality f computed by the t parties as an arithmetic circuit C over a finite field $\mathbb{F}$. In this section, it suffices to consider functionalities whose outputs consists of a single field element $\mathbb{F}$. We assume each party P_i has a common input $\mathbf{x} \in \mathbb{F}^n$ and a local input $\mathbf{w}_i \in \mathbb{F}^m$.

We specify a t-party MPC protocol Π by its next-message function. In particular, on input a party index $i \in [t]$, the public input $\mathbf{x}$, the party's local input $\mathbf{w}_i$ and randomness $\mathbf{r}_i$, and the messages P_i received $(m_1, \ldots, m_j)$ in the first j rounds of the protocol execution, $\Pi(i, \mathbf{s}, \mathbf{w}_i, \mathbf{r}_i, (m_1, \ldots, m_j))$ outputs a set of $t-1$ messages that P_i sends to each of the other parties in round $j+1$. The view of a party P_i, denoted view_i, in the protocol execution consists of its local input $\mathbf{w}_i$, randomness $\mathbf{r}_i$, and all of the messages that P_i both sends and receives[2] during the execution of Π. At the end of the protocol execution (or if Π signals an early termination), each party P_i also computes some output (as a function of its local state). We say that a pair of views view_i and view_j for two distinct parties P_i and P_j is consistent (with respect to Π and the public input $\mathbf{x}$) if the set of messages sent by P_i (in view_i) are identical to the messages P_j receives (in view_j). We now define the correctness and robustness requirement we require in our construction.

Definition 5.16 (Correctness)**.** An MPC protocol Π realizes a deterministic t-party functionality $f(\mathbf{x}, \mathbf{w}_1, \ldots, \mathbf{w}_n)$ with perfect correctness if on all inputs $\mathbf{x}, \mathbf{w}_1, \ldots, \mathbf{w}_n$, the probability (taken over each party's randomness) that the output of some player P_i is different from the output of f is 0.

[2]Typically, one defines the view of a party to only consist of the messages it receives during the computation. In our setting, it will be useful to also include the messages the party sends as part of the view, even though those messages can be computed implicitly from the other components in the view.

Definition 5.17 (δ-Robustness)**.** An MPC protocol Π realizes a deterministic t-party functionality with perfect δ-robustness if it is perfectly correct in the presence of a semi-honest adversary (as in Definition 5.16), and furthermore, for any adversary that corrupts up to δt parties, and for any input $(\mathbf{x}, \mathbf{w}_1, \ldots, \mathbf{w}_t)$, the following robustness property holds: if there are no inputs $(\mathbf{w}'_1, \ldots, \mathbf{w}'_t)$ where $f(\mathbf{x}, \mathbf{w}'_1, \ldots, \mathbf{w}'_t) = 1$, then the probability (taken over each party's randomness) that some uncorrupted party outputs 1 in an execution of Π where the inputs of the honest parties are consistent with $(\mathbf{x}, \mathbf{w}_1, \ldots, \mathbf{w}_t)$ is 0.

Note that in our settings, we do not require an additional privacy property from the MPC protocol. With this in mind, we now present our robust decomposition for an arithmetic circuit C.

Construction 5.18 (Robust Decomposition via MPC)**.** Let $\delta > 0$ be a constant and $t \in \mathbb{N}$ be an integer. Let $C \colon \mathbb{F}^{n'} \times \mathbb{F}^{m'} \to \mathbb{F}^{h'}$ be an arithmetic circuit, and Π_f be a t-party MPC protocol for the $t+1$-input function

$$f(\mathbf{x}, \mathbf{w}_1, \ldots, \mathbf{w}_n) = \begin{cases} 1 & \text{if } C(\mathbf{x}, \mathbf{w}_1 \| \mathbf{w}_2 \| \cdots \| \mathbf{w}_n) = 0^{h'} \\ 0 & \text{otherwise} \end{cases} \tag{5.2}$$

where each $\mathbf{w}_i$ is a vector of dimension $O(m'/t)$. Our (t, δ)-robust decomposition $(f_1, \ldots, f_t, \mathsf{inp}, \mathsf{wit})$ of C is then defined as follows:

- The input encoding function $\mathsf{inp} \colon \mathbb{F}^{n'} \to \mathbb{F}^n$, where $n' = n$, is the identity function.
- The witness encoding function wit takes as input a statement $\mathbf{x}' \in \mathbb{F}^{n'}$ and a witness $\mathbf{w}' \in \mathbb{F}^{m'}$ and simulates an execution of Π_f on inputs $\mathbf{x}'$ and $\mathbf{w}'_1, \ldots, \mathbf{w}'_t$ where $\mathbf{w}' = \mathbf{w}'_1 \| \mathbf{w}'_2 \| \cdots \| \mathbf{w}'_t$. Let $\mathsf{view}_1, \ldots, \mathsf{view}_t$ be the views of each of the t parties in the simulated MPC protocol. The output of the witness encoding functions is a new witness $\mathbf{w} = (\mathsf{view}_1, \ldots, \mathsf{view}_t)$ consisting of the views of the t parties in the execution of Π_f.
- Each of the constraint functions f_i for $i \in [t]$ takes as input the statement $\mathbf{x} \in \mathbb{F}^n$ and the witness $\mathbf{w} = (\mathsf{view}_1, \ldots, \mathsf{view}_t)$ and outputs 1 if the following conditions hold:
  - The output of party P_i (as determined by view_i) is 1 (indicating an accepting output).
  - The view view_i of party P_i is consistent with an honest evaluation of Π_f (with respect to the global input $\mathbf{x}'$). Recall that view_i includes not only the local state of party P_i and the set of messages P_i receives from the other parties during the protocol execution, but also the messages P_i sends to each of the other parties during the protocol execution. This step verifies that the messages P_i sends and its output are consistent with those that would be computed assuming an honest evaluation of Π_f.
  - The messages sent by party P_i (as specified in view_i) are consistent with the messages each of the other parties P_j receives (as specified in view_j).

In particular, each f_i only needs to read the components of the statement $\mathbf{x}$ that party P_i needs to read during the protocol execution. In addition, f_i only needs to read the components of $\mathbf{w}$ that pertain to party P_i: namely, view_i and the components of view_j (for all $j \neq i$) containing the messages P_j received from P_i.

Theorem 5.19. *Let $\delta > 0$ be a constant and $t \in \mathbb{N}$ be an integer. Let $C \colon \mathbb{F}^{n'} \times \mathbb{F}^{m'} \to \mathbb{F}^{h'}$ be an arithmetic circuit, and let Π_f be a perfectly δ-robust t-party MPC protocol for the function in Eq.* (5.2)*. Then, the decomposition $(f_1, \ldots, f_t, \mathsf{inp}, \mathsf{wit})$ in Construction 5.18 is $(t, 1-\delta)$-robust.*

Proof. We show completeness and robustness separately.

Completeness. If $(\mathbf{x}', \mathbf{w}') \in \mathcal{R}_C$, then by perfect correctness of Π_f, each of the honest parties will output 1. Moreover, in an honest protocol execution, all of the views view_i for $i \in [t]$ are internally and pairwise consistent.

Robustness. To show that the decomposition is $(1-\delta)$-robust, we need to show that for a statement $\mathbf{x}' \notin \mathcal{L}_C$, there is no setting of view $\mathbf{w} = (\mathsf{view}_1, \ldots, \mathsf{view}_t)$ that can satisfy more than a $(1-\delta)$-fraction of the constraints f_i. Our argument here is similar to the soundness analysis of the zero-knowledge proof systems from MPC in [IKOS07, Theorem 4.1]. At a high-level, suppose that all inconsistencies (if any) among the views $\mathsf{view}_1, \ldots, \mathsf{view}_t$ can be resolved by eliminating at most δt views. In this case, the protocol execution defined by $(\mathsf{view}_1, \ldots, \mathsf{view}_t)$ can be realized by an adversary that corrupts at most δt parties. But since Π_f is perfectly δ-robust, on input $\mathbf{x}'$, all of the uncorrupted parties will output 0. In this case, there are at most δt corrupted parties, so there are at least $(1-\delta)t$ honest parties P_i where the output is 0, and correspondingly, $f_i(\mathbf{x}, \mathbf{w}) = 0$ for those parties. Conversely, if there are more than δt views that are inconsistent, then for any view_i that has an inconsistency, $f_i(\mathbf{x}, \mathbf{w}) = 0$.

More formally, for a collection of views $\mathbf{w} = (\mathsf{view}_1, \ldots, \mathsf{view}_t)$, we say that the view of party P_i for $i \in [t]$ is "bad" if view_i is either inconsistent with an honest evaluation of Π_f on input $\mathbf{x}$, or if there is a discrepancy between an outgoing message from P_i in view_i and an incoming message for P_j (from P_i) in view_j. Let $B \subseteq [t]$ be the subset of "bad" parties. We consider two cases:

- If $|B| < \delta t$, then consider a protocol execution of Π_f where the adversary corrupts the set of parties B and behaves in a way so that the view of any party P_j for $j \notin B$ is view_j. By construction of the set B, this means that the views between any two parties $i, j \in [t] \setminus B$ are mutually consistent with an honest execution of Π_f on input $\mathbf{x}'$. Thus, since Π_f is perfectly δ-robust, the output of all parties P_j, where $j \notin B$ will be 0. Equivalently, for all $j \in [t] \setminus B$, $f_j(\mathbf{x}, \mathbf{w}) = 0$, so there are at most $(1-\delta)t$ indices i where $f_i(\mathbf{x}, \mathbf{w}) = 1$.
- If $|B| \geq \delta t$, then for all $i \in B$, $f_i(\mathbf{x}, \mathbf{w}) = 0$. This means that there are at most $(1-\delta)t$ indices $i \in [t] \setminus B$ where $f_i(\mathbf{x}, \mathbf{w}) = 1$, and the claim follows. □

Fact 5.20 ([DIK10])**.** Let $t \in \mathbb{N}$ be an integer. For any constant $0 < \delta < 1/3$ and arithmetic circuit $C(\mathbf{x}, \mathbf{w}_1, \ldots, \mathbf{w}_t)$ on $t + 1$ inputs with width $\Omega(t)$, there exists a t-party MPC protocol Π which computes C with perfect δ-robustness and where the total computational complexity is

$$|C| \cdot \operatorname{polylog}(t, |C|) + \operatorname{poly}(t, \log |C|) \cdot \operatorname{depth}(C)^2$$

Corollary 5.21. *Fix an integer $t \in \mathbb{N}$. Then, for any constant $0 < \delta < 1/3$, there exists a quasi-optimal $(t, 1 - \delta)$-robust decomposition for any arithmetic circuit $C \colon \mathbb{F}^n \times \mathbb{F}^m \to \mathbb{F}^h$ of size $\Omega(t)$.*

Proof. We instantiate Construction 5.18 using the information-theoretic MPC protocol from Fact 5.20. To obtain the required asymptotics, we make the following observations:

- First, we need to construct an arithmetic circuit that implements the functionality in Eq. (5.2). We construct the circuit $C' \colon \mathbb{F}^n \times \mathbb{F}^m \to \{0, 1\}$ from C as follows. Circuit C' first evaluates C on $(\mathbf{x}, \mathbf{w})$ and then projects the h outputs of C onto $\{0, 1\}$ (mapping 0 to 0 and all non-zero elements in $\mathbb{F}$ to 1). Projecting each element can be done with a circuit of size and depth $O(\log |\mathbb{F}|)$. Finally C' computes the Boolean AND on the negation of each of the projected output bits. When $\mathbb{F}$ is polynomial-sized (e.g., $|\mathbb{F}| = O(|C|)$), this transformation only adds logarithmic overhead to C. In particular, $\operatorname{depth}(C') = \operatorname{depth}(C) + O(\log |\mathbb{F}|)$ and $|C'| = |C| + O(h \cdot \log |\mathbb{F}|)$.

- For proof verification, we can also assume without loss of generality that the circuit C has constant depth. In particular, for verifying that $(\mathbf{x}, \mathbf{w})$ is a satisfying input to C, we can always construct a new circuit of size $|C|$ which takes as input the statement $\mathbf{x}$ and the value of each wire in $C(\mathbf{x}, \mathbf{w})$. The new circuit then simply checks that every wire is correctly computed, and that the output value of $C(\mathbf{x}, \mathbf{w})$ is $0^{h'}$. Coupled with the transformation from the previous step, we conclude that checking satisfiability of an arithmetic circuit C can always be reduced to checking satisfiability of a related circuit C' of size $O(|C| \log |C|)$ and depth $O(\log |C|)$. Invoking Fact 5.20 on the circuit C', we conclude that the inp and wit encoding functions satisfy the efficiency requirements of Definition 5.15.

- Moreover, when simulating an execution of the MPC protocol from Fact 5.20, we uniformly distribute the inputs (i.e., the bits of the witness) across the t parties. This ensures that the computational costs are distributed evenly across all t parties, and so the local computational complexity of each party becomes

$$|C| / t \cdot \operatorname{polylog}(t, |C|) + \operatorname{poly}(t, \log |C|).$$

Since each f_i is verifying integrity of view_i, we conclude that each f_i can be computed by a circuit of size $\widetilde{O}(|C| / t) + \operatorname{poly}(t, \log |C|)$. $\square$

Remark 5.22 (Robust Decomposition and Quasilinear PCPs)**.** Our robust decomposition essentially provides a way to convert a circuit satisfiability instance into checking satisfiability of a collection of smaller constraint functions defined over a common set of variables. This is reminiscent of viewing a traditional PCP (for a circuit satisfiability instance) as a constraint satisfaction problem (CSP), where each constraint in the CSP reads a small number of bits of the PCP. Thus, another potential way of obtaining a quasi-optimal robust decomposition is to use quasilinear PCPs [Din06, BS08]. Specifically, we view the PCP as a CSP instance; an encoding of a statement-witness pair corresponds to an assignment to the variables in the CSP, and the constraint functions in the robust decomposition simply implement the constraints of the CSP. However, with traditional PCPs, the variables on which each constraint depends varies with the statement being proved. One of the requirements of our robust decomposition is that each constraint only depends on a *fixed* subset of the bits of the encoded statement and witness, *irrespective* of the statement being proved. Thus, it is not clear how to leverage traditional PCPs to implement our robust decomposition.

In contrast, our MPC-based robust decomposition satisfies this *input-independence* property. Specifically, the components of the encoded statement-witness pair read by the i^{th} constraint just correspond to the view of the i^{th} party in the simulated MPC protocol, which is always a fixed subset of the encoded statement-witness pair, and independent of the statement being proved. It is an interesting problem to construct an input-independent quasilinear PCP, which may in turn yield another approach for realizing our robust decomposition primitive.

5.3.2 Consistency Checking

As described in Section 5.1, in our linear MIP construction, we first apply a robust decomposition to the input circuit C to obtain smaller arithmetic circuits $C_1, \dots, C_t$, each of which depends on some subset of the components of a witness $\mathbf{w} \in \mathbb{F}^m$. The proof then consists of a collection of systematic linear PCP proofs $\boldsymbol{\pi}_1, \dots, \boldsymbol{\pi}_t$ that $C_1, \dots, C_t$ are individually satisfiable. The second ingredient we require is a way for the verifier to check that the prover uses a consistent witness to construct the proofs $\boldsymbol{\pi}_1, \dots, \boldsymbol{\pi}_t$. In this section, we formally introduce the building blocks we use for the consistency check. We refer to Section 5.1.1 for an overview of our methods. First, we introduce a notion of a "systematic" linear PCP that enables these types of consistency checks (using linear queries).

Systematic linear PCPs. Recall from Section 5.1 that our linear MIP for checking satisfiability of a circuit C begins by applying a robust decomposition to the circuit C. The MIP proof is comprised of linear PCP proofs $\boldsymbol{\pi}_1, \dots, \boldsymbol{\pi}_t$ to show that each of the circuits $C_1(\mathbf{x}_1, \cdot), \dots, C_t(\mathbf{x}_t, \cdot)$ in the robust decomposition of C is satisfiable. Here, $\mathbf{x}_i$ denotes the bits of the statement $\mathbf{x}$ that circuit C_i reads. To provide soundness, the verifier needs to perform a sequence of consistency checks to ensure that the proofs $\boldsymbol{\pi}_1, \dots, \boldsymbol{\pi}_t$ are *consistent* with some witness $\mathbf{w}$. To facilitate this, we require that the underlying linear PCPs are *systematic*: namely, each proof $\boldsymbol{\pi}_i$ contains a copy of some witness $\mathbf{w}_i$ where $(\mathbf{x}_i, \mathbf{w}_i) \in \mathcal{R}_{C_i}$. The consistency check then affirms that the witnesses $\mathbf{w}_1, \ldots, \mathbf{w}_t$ associated with $\boldsymbol{\pi}_1, \ldots, \boldsymbol{\pi}_t$ are mutually consistent. We give the formal definition of a systematic linear PCP below, and then describe one such instantiation by Ben-Sasson et al. [BCG+13, Appendix E].

Definition 5.23 (Systematic Linear PCPs)**.** Let $(\mathcal{P}, \mathcal{V})$ be an input-oblivious k-query linear PCP for a relation $\mathcal{R}_C$ where $C \colon \mathbb{F}^n \times \mathbb{F}^m \to \mathbb{F}^h$. We say that $(\mathcal{P}, \mathcal{V})$ is *systematic* if the following conditions hold:

- On input a statement-witness pair $(\mathbf{x}, \mathbf{w}) \in \mathbb{F}^n \times \mathbb{F}^m$ the prover's output of $\mathcal{P}(\mathbf{x}, \mathbf{w})$ has the form $\boldsymbol{\pi} = [\mathbf{w}, \mathbf{p}] \in \mathbb{F}^d$, for some $\mathbf{p} \in \mathbb{F}^{d-m}$. In other words, the witness is included as part of the linear PCP proof vector.
- On input a statement $\mathbf{x}$ and given oracle access to a proof $\boldsymbol{\pi}^* = [\mathbf{w}^*, \mathbf{p}^*]$, the knowledge extractor $\mathcal{E}^{\boldsymbol{\pi}^*}(\mathbf{x})$ outputs $\mathbf{w}^*$.

Fact 5.24 ([BCG+13, Claim E.3])**.** Let $C \colon \mathbb{F}^n \times \mathbb{F}^m \to \mathbb{F}^h$ be an arithmetic circuit of size s over a finite field $\mathbb{F}$ where $|\mathbb{F}| > s$. There exists a systematic input-oblivious 5-query linear PCP $(\mathcal{P}, \mathcal{V})$ for $\mathcal{R}_{\mathcal{C}}$ over $\mathbb{F}$ with knowledge error $O(s/|\mathbb{F}|)$ and query length $O(s)$. Moreover, letting $\mathcal{V} = (\mathcal{Q}, \mathcal{D})$, the prover and verifier algorithms satisfy the following properties:

- the prover algorithm $\mathcal{P}$ is an arithmetic circuit of size $\widetilde{O}(s)$;
- the query-generation algorithm $\mathcal{Q}$ is an arithmetic circuit of size $O(s)$;
- the decision algorithm $\mathcal{D}$ is an arithmetic circuit of size $O(n)$.

Replication structures. Next, we introduce the notion of a replication structure induced by the decomposition $C_1, \ldots, C_t$, and define what it means for a collection of assignments to the circuit $C_1, \ldots, C_t$ to be consistent.

Definition 5.25 (Replication Structures and Inconsistency Matrices)**.** Fix integers $m, t, q \in \mathbb{N}$. A *replication structure* is a matrix $\mathbf{A} \in [m]^{t \times q}$. We say that a matrix $\mathbf{W} \in \mathbb{F}^{t \times q}$ is consistent with respect to a replication structure $\mathbf{A}$ if for all $i_1, i_2 \in [t]$ and $j_1, j_2 \in [q]$, whenever $\mathbf{A}_{i_1,j_1} = \mathbf{A}_{i_2,j_2}$, $\mathbf{W}_{i_1,j_1} = \mathbf{W}_{i_2,j_2}$. If there is a pair of indices (i_1, j_1) and (i_2, j_2) where this relation does not hold, then we say that there is an inconsistency in $\mathbf{W}$ (with respect to $\mathbf{A}$) at locations (i_1, j_1) and (i_2, j_2). For a replication structure $\mathbf{A} \in [m]^{t \times q}$ and a matrix of values $\mathbf{W} \in \mathbb{F}^{t \times q}$, we define the inconsistency matrix $\mathbf{B} \in \{0,1\}^{t \times q}$ where $\mathbf{B}_{i,j} = 1$ if and only if there is an inconsistency in $\mathbf{W}$ at location (i, j) with respect to the replication structure $\mathbf{A}$. In the subsequent analysis, we will sometimes refer to an arbitrary inconsistency matrix $\mathbf{B} \in \{0,1\}^{t \times q}$ (independent of any particular set of values $\mathbf{W}$ or replication structure $\mathbf{A}$).

Definition 5.26 (Consistent Inputs to Circuits)**.** Let $C_1, \ldots, C_t$ be a collection of circuits where each $C_i \colon \mathbb{F}^m \to \mathbb{F}^h$ only depends on at most $q \leq m$ components of an input vector $\mathbf{w} \in \mathbb{F}^m$. For each

$i \in [t]$, let $a_1^{(i)}, \ldots, a_q^{(i)} \in [m]$ be the indices of the q components of the input $\mathbf{w}$ on which C_i depends. The *replication structure* of $C_1, \ldots, C_t$ is the matrix $\mathbf{A} \in [m]^{t \times q}$, where the ith row of $\mathbf{A}$ is the vector $a_1^{(i)}, \ldots, a_q^{(i)}$ (namely, the subset of indices on which C_i depends). We say that a collection of inputs $\mathbf{w}_1, \ldots, \mathbf{w}_t \in \mathbb{F}^q$ to $C_1, \ldots, C_t$ is consistent if the assignment matrix $\mathbf{W}$, where the ith row of $\mathbf{W}$ is $\mathbf{w}_i$ for $i \in [t]$, is consistent with respect to the replication structure $\mathbf{A}$.

To simplify the analysis, we introduce the notion of an inconsistency graph for an assignment matrix $\mathbf{W} \in \mathbb{F}^{t \times q}$ with respect to a replication structure $\mathbf{A} \in [m]^{t \times q}$. At a high level, the inconsistency graph of $\mathbf{W}$ with respect to $\mathbf{A}$ is a graph with t nodes, one for each row of $\mathbf{W}$, and there is an edge between two nodes $i, j \in [t]$ if assignments $\mathbf{w}_i$ and $\mathbf{w}_j$ (in rows i and j of $\mathbf{W}$, respectively) contain an inconsistent assignment with respect to $\mathbf{A}$.

Definition 5.27 (Inconsistency Graph)**.** Fix positive integers $m, t, q \in \mathbb{N}$ and take a replication structure $\mathbf{A} \in [m]^{t \times q}$. For any assignment matrix $\mathbf{W} \in \mathbb{F}^{t \times q}$, we define the inconsistency graph $\mathcal{G}_{\mathbf{W},\mathbf{A}}$ of $\mathbf{W}$ with respect to $\mathbf{A}$ as follows:

- Graph $\mathcal{G}_{\mathbf{W},\mathbf{A}}$ is an undirected graph with t nodes, with labels in $[t]$. We associate node $i \in [t]$ with the ith row of $\mathbf{A}$.
- Graph $\mathcal{G}_{\mathbf{W},\mathbf{A}}$ has an edge between nodes i_1 and i_2 if there exists $j_1, j_2 \in [q]$ such that $\mathbf{A}_{i_1,j_1} = \mathbf{A}_{i_2,j_2}$ but $\mathbf{W}_{i_1,j_1} \neq \mathbf{W}_{i_2,j_2}$. In other words, there is an edge in $\mathcal{G}_{\mathbf{W},\mathbf{A}}$ whenever there is an inconsistency in the assignments to rows i_1 and i_2 in $\mathbf{W}$ (with respect to the replication structure $\mathbf{A}$).

Definition 5.28 (Regular Matchings)**.** Fix integers $m, t, q \in \mathbb{N}$ where t is even, and take any replication structure $\mathbf{A} \in [m]^{t \times q}$ and assignment matrix $\mathbf{W} \in \mathbb{F}^{t \times q}$. We say that the inconsistency graph $\mathcal{G}_{\mathbf{W},\mathbf{A}}$ contains a *regular* matching of size s if $\mathcal{G}_{\mathbf{W},\mathbf{A}}$ contains a matching[3] M of size s, where each edge $(v_1, v_2) \in M$ satisfies $(v_1, v_2) = (2i-1, 2i)$ for some $i \in [t/2]$. In other words, all matched edges are between nodes corresponding to *adjacent* rows in $\mathbf{W}$.

Having defined these notions, we can reformulate the guarantees provided by the (t, δ)-robust decomposition (Definition 5.15). For a constant $\delta > 0$, let $(f_1, \ldots, f_t, \mathsf{inp}, \mathsf{wit})$ be a (t, δ)-robust decomposition of a circuit C. Let $\mathbf{A}$ be the replication structure of the circuits $C_1, \ldots, C_t$ computing $f_1, \ldots, f_t$. Take any statement $\mathbf{x}' \notin \mathcal{L}_C$, and consider any collection of witnesses $\mathbf{w}_1, \ldots, \mathbf{w}_t$ where $C_i(\mathbf{x}_i, \mathbf{w}_i) = 1$ for all $i \in [t]$. As usual, $\mathbf{x}_i$ denotes the bits of $\mathbf{x} = \mathsf{inp}(\mathbf{x}')$ that C_i reads. Robustness of the decomposition ensures that no single $\mathbf{w}$ can be used to simultaneously satisfy more than a δ-fraction of the constraints. In particular, this means that there must exist $\Omega(t)$ pairs of witnesses $\mathbf{w}_i$ and $\mathbf{w}_j$ which are inconsistent. Equivalently, we say that the inconsistency graph $\mathcal{G}_{\mathbf{W},\mathbf{A}}$ contains a matching of size $\Omega(t)$. We prove this statement formally in Lemma 5.51.

[3]For a graph $\mathcal{G}$ with n nodes, labeled with the integers $1, \ldots, n$, a matching M is a set of edges $(i, k) \in [n] \times [n]$ with no common vertices.

Approximate consistency check. By relying on the robust decomposition, it suffices to construct a protocol where the verifier can detect whether the inconsistency graph $\mathcal{G}_{\mathbf{W},\mathbf{A}}$ of the prover's assignments $\mathbf{W}$ with respect to a replication structure $\mathbf{A}$ contains a large matching. To facilitate this, we first describe an algorithm to check whether two assignment matrices $\mathbf{W}, \mathbf{W}' \in \mathbb{F}^{t \times q}$ (approximately) satisfy the relation $\mathbf{W}' = \Pi(\mathbf{W})$ in the linear MIP model, where Π is a 2-locally decomposable permutation. This primitive can then be used directly to detect whether an inconsistency graph $\mathcal{G}_{\mathbf{W},\mathbf{A}}$ contains a *regular* matching (Corollary 5.32). Subsequently, we show how to permute the entries in $\mathbf{W}$ according to a permutation Π' so as to convert an arbitrary matching in $\mathcal{G}_{\mathbf{W},\mathbf{A}}$ into a regular matching in $\mathcal{G}_{\Pi'(\mathbf{W}),\Pi'(\mathbf{A})}$. Our construction of the approximate consistency check is a direct generalization of the pairwise consistency check procedure described in Section 5.1.1.

Construction 5.29 (Approximate Consistency Check)**.** Fix an even integer $t \in \mathbb{N}$, and let $P_1, \ldots, P_t$, $P_1', \ldots, P_t'$ be a collection of $2 \cdot t$ provers in a linear MIP system. For $i \in [t]$, let $\boldsymbol{\pi}_i \in \mathbb{F}^d$ be the proof vector associated with prover P_i and $\boldsymbol{\pi}_i' \in \mathbb{F}^d$ be the proof vector associated with prover P_i'. We can associate a matrix $\mathbf{W} \in \mathbb{F}^{t \times d}$ with provers $(P_1, \ldots, P_t)$, where the i^{th} row of $\mathbf{W}$ is $\boldsymbol{\pi}_i$. Similarly, we associate a matrix $\mathbf{W}'$ with provers $(P_1', \ldots, P_t')$. Let Π be a 2-locally decomposable permutation on the entries of a t-by-d matrix. Then, we describe the following linear MIP verification procedure for checking that $\mathbf{W}' \approx \Pi(\mathbf{W})$.

- **Verifier's query algorithm:** The verifier chooses a random matrix $\mathbf{R} \xleftarrow{\text{R}} \mathbb{F}^{t \times d}$, and sets $\mathbf{R}' \leftarrow \Pi(\mathbf{R})$. Let $\mathbf{r}_i$ and $\mathbf{r}_i'$ denote the i^{th} row of $\mathbf{R}$ and $\mathbf{R}'$, respectively. The query algorithm outputs the query $\mathbf{r}_i$ for prover P_i and the query $\mathbf{r}_i'$ to prover P_i'.

- **Verifier's decision algorithm:** Since Π is 2-locally decomposable, we can decompose Π into $t' = t/2$ independent permutations, $\Pi_1, \ldots, \Pi_{t'}$, where each Π_i only operates on a pair of rows (j_{2i-1}, j_{2i}), for all $i \in [t']$. Given responses $\mathbf{y}_i = \langle \boldsymbol{\pi}_i, \mathbf{r}_i \rangle \in \mathbb{F}$ and $\mathbf{y}_i' = \langle \boldsymbol{\pi}_i', \mathbf{r}_i' \rangle \in \mathbb{F}$ for $i \in [t]$, the verifier checks that the relation

$$\mathbf{y}_{j_{2i-1}} + \mathbf{y}_{j_{2i}} \stackrel{?}{=} \mathbf{y}'_{j_{2i-1}} + \mathbf{y}'_{j_{2i}},$$

  for all $i \in [t']$. The verifier accepts if the relations hold for all $i \in [t']$. Otherwise, it rejects.

By construction, we see that if $\mathbf{W}' = \Pi(\mathbf{W})$, then the verifier always accepts.

Lemma 5.30 (Consistency Check Soundness)**.** *Define t, Π, $\mathbf{W}$, and $\mathbf{W}'$ as in Construction 5.29. Then, if the matrix $\mathbf{W}'$ disagrees with $\Pi(\mathbf{W})$ on κ rows, the verifier in Construction 5.29 will reject with probability at least $1 - 2^{-\Omega(\kappa)}$.*

Proof. Consider the event where $\mathbf{W}'$ disagrees with $\hat{\mathbf{W}} = \Pi(\mathbf{W})$ on κ rows. We show that the probability of the verifier accepting in this case is bounded by $2^{-\Omega(\kappa)}$. In the linear MIP model, the verifier's decision algorithm corresponds to checking the following relation:

$$\left\langle \boldsymbol{\pi}_{j_{2i}}, \mathbf{r}_{j_{2i}} \right\rangle + \left\langle \boldsymbol{\pi}_{j_{2i+1}}, \mathbf{r}_{j_{2i+1}} \right\rangle \stackrel{?}{=} \left\langle \boldsymbol{\pi}'_{j_{2i}}, \mathbf{r}'_{j_{2i}} \right\rangle + \left\langle \boldsymbol{\pi}'_{j_{2i+1}}, \mathbf{r}'_{j_{2i+1}} \right\rangle. \tag{5.3}$$

By assumption, there are at least $\kappa/2$ indices $i \in [t]$ where $\mathbf{W}'_{[j_{2i-1}, j_{2i}]} \neq \hat{\mathbf{W}}_{[j_{2i-1}, j_{2i}]}$. By the Schwartz-Zippel lemma (Lemma 2.2), for the indices $i \in [t]$ where $\mathbf{W}'_{[j_{2i}, j_{2i+1}]} \neq \hat{\mathbf{W}}_{[j_{2i}, j_{2i+1}]}$, the relation in Eq. (5.3) holds with probability at most $1/|\mathbb{F}|$ (over the randomness used to sample $\mathbf{r}_{j_{2i-1}}$ and $\mathbf{r}_{j_{2i}}$) Since there are at least $\kappa/2$ such indices, the probability that Eq. (5.3) holds for all $i \in [t']$ is at most $(1/|\mathbb{F}|)^{\kappa/2} = 2^{-\Omega(\kappa)}$. Hence, the verifier rejects with probability $1 - 2^{-\Omega(\kappa)}$. □

The approximate consistency check from Construction 5.29 immediately gives a way to check whether an inconsistency graph $\mathcal{G}_{\mathbf{W},\mathbf{A}}$ contains a regular matching of size $\Omega(t)$. To show this, it suffices to exhibit a 2-locally decomposable permutation Π where the assignment matrix $\mathbf{W}$ is consistent on adjacent pairs of rows if and only if $\mathbf{W} = \Pi(\mathbf{W})$. The construction can be viewed as composing many copies of the global consistency check permutation used in [Gro09] (and described in Section 5.1.1), each applied to a pair of adjacent rows. We give the construction below.

Construction 5.31 (Pairwise Consistency in Adjacent Rows)**.** Fix integers $m, t, q \in \mathbb{N}$ with t even, and let $\mathbf{A} \in [m]^{t \times q}$ be a replication structure. Let $t' = t/2$. For each $i \in [t']$, let Π_i be a permutation over 2-by-q matrices such that Π_i splits into a disjoint set of Hamiltonian cycles based on the entries of $\mathbf{A}_{[2i-1,2i]}$. Define a permutation Π on t-by-q matrices where the action of Π on rows $2i-1$ and $2i$ is given by Π_i for all $i \in [t']$. By construction, the permutation Π is 2-locally decomposable, and moreover, $\mathbf{W} \in \mathbb{F}^{t \times q}$ is pairwise consistent on adjacent rows with respect to $\mathbf{A}$ if and only if $\mathbf{W} = \Pi(\mathbf{W})$.

Corollary 5.32. *Fix integers $m, t, q \in \mathbb{N}$ with t even. Let $\mathbf{A} \in [m]^{t \times q}$ be a replication structure, and Π be the pairwise consistency test permutation for $\mathbf{A}$ from Construction 5.31. Then, for any assignment matrix $\mathbf{W} \in \mathbb{F}^{t \times q}$ where the inconsistency graph $\mathcal{G}_{\mathbf{W},\mathbf{A}}$ contains a regular matching of size $\Omega(t)$, the verifier Construction 5.29 will reject the relation $\mathbf{W} \stackrel{?}{=} \Pi(\mathbf{W})$ with probability $1 - 2^{-\Omega(t)}$.*

Proof. Since $\mathcal{G}_{\mathbf{W},\mathbf{A}}$ contains a regular matching of size $\Omega(t)$, there are inconsistencies in $\Omega(t)$ pairs of adjacent rows of $\mathbf{W}$. By construction of Π, this means that $\mathbf{W}$ and $\Pi(\mathbf{W})$ differ on $\Omega(t)$ rows. The claim then follows by Lemma 5.30. □

Regularity-inducing permutations. Recall that our objective in the consistency check is to give an algorithm that detects whether an inconsistency graph $\mathcal{G}_{\mathbf{W},\mathbf{A}}$ contains a matching of size $\Omega(t)$. Corollary 5.32 gives a way to detect if the inconsistency graph $\mathcal{G}_{\mathbf{W},\mathbf{A}}$ contains a *regular* matching of size $\Omega(t)$ with soundness error $2^{-\Omega(t)}$. Thus, to perform the consistency check, we first construct a permutation Π on $\mathbf{W}$ such that whenever $\mathcal{G}_{\mathbf{W},\mathbf{A}}$ contain a matching of size $\Omega(t)$, the inconsistency graph $\mathcal{G}_{\Pi(\mathbf{W}),\Pi(\mathbf{A})}$ contains a regular matching of similar size $\Omega(t)$. We say that such permutations are *regularity-inducing.* While we are not able to construct a single permutation Π that is regularity-inducing for all assignment matrices $\mathbf{W}$, we are able to construct a *family* of permutations $(\Pi_1, \dots, \Pi_z)$ for a fixed replication structure $\mathbf{A}$ such that for all assignment matrices $\mathbf{W} \in \mathbb{F}^{t \times q}$, there is at least one $\beta \in [z]$ where $\mathcal{G}_{\Pi_\beta(\mathbf{W}), \Pi_\beta(\mathbf{A})}$ contains a regular matching of size $\Omega(t)$.

Definition 5.33 (Regularity-Inducing Permutations)**.** Fix integers $m, t, q \in \mathbb{N}$, and let $\mathbf{A} \in [m]^{t \times q}$ be a replication structure. Let Π be a permutation on t-by-q matrices and $\mathbf{W} \in \mathbb{F}^{t \times q}$ be a matrix such that the inconsistency graph $\mathcal{G}_{\mathbf{W},\mathbf{A}}$ contains a matching M of size s. We say that Π is ρ-*regularity-inducing* for $\mathbf{W}$ with respect to $\mathbf{A}$ if the inconsistency graph $\mathcal{G}_{\Pi(\mathbf{W}),\Pi(\mathbf{A})}$ contains a *regular* matching M' of size at least s/ρ. Moreover, there is a one-to-one correspondence between the edges in M' and a subset of the edges in M (as determined by Π). We say that $(\Pi_1, \dots, \Pi_z)$ is a collection of ρ-regularity-inducing permutations with respect to a replication structure $\mathbf{A}$ if for all $\mathbf{W} \in \mathbb{F}^{t \times q}$, there exists $\beta \in [z]$ such that Π_β is ρ-regularity-inducing for $\mathbf{W}$.

In this work, we will construct regularity-inducing permutations where $\rho = O(1)$. To simplify the following description, we will implicitly assume that $\rho = O(1)$. Given an assignment matrix $\mathbf{W}$ and a collection of ρ-regularity-inducing permutations $(\Pi_1, \dots, \Pi_z)$ for a replication structure $\mathbf{A}$, we can affirm that the inconsistency graph $\mathcal{G}_{\mathbf{W},\mathbf{A}}$ does not contain a matching of size $\Omega(t)$ by checking that each of the graphs $\mathcal{G}_{\Pi_\beta(\mathbf{W}),\Pi_\beta(\mathbf{A})}$ does not contain a regular matching of size $\Omega(t/\rho) = \Omega(t)$ for all $\beta \in [z]$ and assuming $\rho = O(1)$. By Corollary 5.32, each of these checks can be implemented in the linear MIP model using Construction 5.29. However, to apply the protocol in Construction 5.29 to $\Pi_\beta(\mathbf{W})$, the verifier requires oracle access to the individual rows of $\Pi_\beta(\mathbf{W})$. Thus, in the linear MIP construction, in addition to providing oracle access to the rows of the assignment matrix $\mathbf{W}$, we also provide the verifier oracle access to the rows of $\Pi_\beta(\mathbf{W})$ for all $\beta \in [z]$. Of course, a malicious MIP prover may provide the rows of a different matrix $\mathbf{W}' \in \mathbb{F}^{t \times q}$ (so as to pass the consistency check). Thus, the final ingredient we require is a way for the verifier to check that two matrices $\mathbf{W}, \mathbf{W}' \in \mathbb{F}^{t \times q}$ satisfy the relation $\mathbf{W}' = \Pi_\beta(\mathbf{W})$. Note that Construction 5.29 does not directly apply because the permutation Π_β is not necessarily 2-locally decomposable.

Decomposing the permutation. To complete the description, we now describe a way for the verifier to check that two matrices $\mathbf{W}, \mathbf{W}' \in \mathbb{F}^{t \times q}$ satisfy the relation $\mathbf{W}' = \Pi(\mathbf{W})$, for an *arbitrary* permutation Π. We assume that the verifier is given oracle access to the rows of $\mathbf{W}$ and $\mathbf{W}'$ in the linear MIP model. Construction 5.29 provides a way to check the relation whenever Π is 2-locally decomposable, so a natural starting point is to decompose the permutation Π into a sequence of 2-locally-decomposable permutations $\Pi_1, \dots, \Pi_\alpha$, where $\Pi = \Pi_\alpha \circ \cdots \circ \Pi_1$. This is possible, for instance, by first applying Lemma 5.5 and Construction 5.9 to Π. Then, the linear MIP proof consists of the initial and final matrices $\mathbf{W}$ and $\mathbf{W}'$, as well as the intermediate matrices $\mathbf{W}_i = (\Pi_i \circ \cdots \circ \Pi_1)(\mathbf{W})$. The linear MIP proof would consist of the rows of all of the matrices $\mathbf{W} = \mathbf{W}_0, \mathbf{W}_1, \dots, \mathbf{W}_\alpha = \mathbf{W}'$, and the verifier would apply Construction 5.29 to check that for all $\ell \in [\alpha]$, $\mathbf{W}_i = \Pi_i(\mathbf{W}_{i-1})$.

While this general approach seems sound, there is a subtle problem. The soundness guarantee for the consistency check in Construction 5.29 only states that on input $\mathbf{W}, \mathbf{W}'$ and a permutation Π, the verifier will only reject with probability $1 - 2^{\Omega(t)}$ when $\mathbf{W}'$ and $\Pi(\mathbf{W})$ differ on $\Omega(t)$ rows. This means that a malicious prover can provide a sequence of matrices $\mathbf{W}, \mathbf{W}_1, \ldots, \mathbf{W}_\alpha$ where each $\mathbf{W}_\ell$ differs from $\Pi_\ell(\mathbf{W}_{\ell-1})$ on a small number of rows (e.g., $o(t)$ rows), and in doing so, correct all of the inconsistent assignments that appear in the final matrix $\mathbf{W}_\alpha$.

Randomizing the decomposition. Abstractly, we can view the problem as follows. Let $\mathbf{B} \in \{0,1\}^{t \times q}$ be the inconsistency matrix for $\mathbf{W}$ with respect to $\mathbf{A}$ (Definition 5.25). In other words, $\mathbf{B}_{i,j} = 1$ whenever $\mathbf{W}_{i,j}$ encodes a value that is inconsistent with another assignment elsewhere in $\mathbf{W}$. Since $\mathcal{G}_{\mathbf{W},\mathbf{A}}$ contains a matching of size $\Omega(t)$, we know that there are at least $\Omega(t)$ rows in $\mathbf{B}$ that contain a 1. The permutation Π is chosen so that $\Pi(\mathbf{W})$ has a regular matching of size $\Omega(t)$ with respect to $\Pi(\mathbf{A})$. In particular, this means that the permuted inconsistency matrix $\Pi(\mathbf{B})$ contains a 1 in $\Omega(t)$ adjacent pairs of rows.

Consider the sequence of matrices $\mathbf{W}_1, \ldots, \mathbf{W}_\alpha$ chosen by the prover. Using the approximate pairwise consistency check, we can ensure that $\mathbf{W}_i$ agrees with $\Pi_i(\mathbf{W}_{i-1})$ on all but some κ_1 rows. Now suppose that there exists some $\ell \in [\alpha]$ where $\mathbf{B}_\ell = (\Pi_\ell \circ \cdots \circ \Pi_1)(\mathbf{B})$ has the property that all of the locations with a 1 in $\mathbf{B}$ appear in just κ_1 rows of $\mathbf{B}_\ell$. If this happens, then the malicious prover can construct $\mathbf{W}_1, \ldots, \mathbf{W}_{\ell-1}$ honestly, and then choose $\mathbf{W}_\ell$ such that $\mathbf{W}_\ell = \Pi_\ell(\mathbf{W}_{\ell-1})$ on all rows where $\mathbf{B}_\ell$ does not contain a 1, and set the values in the rows where $\mathbf{B}_\ell$ does contain a 1 to be consistent with the other rows of $\mathbf{W}$. Notably, all the entries in $\mathbf{W}_\ell$ are now consistent, and moreover, $\mathbf{W}_\ell$ differs from $\Pi_\ell(\mathbf{W}_{\ell-1})$ on at most κ_1 rows (and so, will not be detected with high probability by the pairwise consistency check). This means that from the verifier's perspective, the final matrix $\Pi(\mathbf{W})$ has no inconsistencies, and thus, the verifier's final pairwise consistency check passes with probability 1 (even though the original inconsistency graph $\mathcal{G}_{\mathbf{W},\mathbf{A}}$ contains a matching of size $\Omega(t)$). Thus, we require a stronger property on the permutation decomposition. It is not sufficient that there is a matching of size $\Omega(t)$ in the starting and ending configurations $\mathbf{W}$ and $\mathbf{W}'$. Rather, we need that the size of the matching in *every* step of the decomposition cannot shrink by too much, or equivalently, the intermediate permutations $\Pi_1, \ldots, \Pi_\alpha$ cannot "concentrate" all of the inconsistencies in $\mathbf{W}$ into a small number of rows (which the malicious prover can fix without being detected). We say permutation decompositions with this property are *non-concentrating.* We now formally define the notion of a non-concentrating permutation decomposition and what it means for a collection of permutation sequences to be non-concentrating.

Definition 5.34 (Non-Concentrating Permutations)**.** Fix positive integers $t, q \in \mathbb{N}$, and let $\Gamma = (\Pi_1, \ldots, \Pi_\alpha)$ be a sequence of permutations over t-by-q matrices. Let $\mathbf{B} \in \{0,1\}^{t \times q}$ be an inconsistency matrix. For $\ell \in [\alpha]$, define $\mathbf{B}_\ell = (\Pi_\ell \circ \cdots \circ \Pi_1)(\mathbf{B})$. We say that Γ is a sequence of (κ_1, κ_2)-*non-concentrating permutations* with respect to $\mathbf{B}$ if for all $\ell \in [\alpha]$, the inconsistency matrix $\mathbf{B}_\ell$ has the property that no subset of κ_1 rows contains more than κ_2 inconsistencies (indices where the value is 1). Next, we say a collection of permutation sequences $\Gamma^{(1)}, \ldots, \Gamma^{(\gamma)}$ where each $\Gamma^{(j)} = \big(\Pi_1^{(j)}, \ldots, \Pi_\alpha^{(j)}\big)$ is (κ_1, κ_2)-non-concentrating for a set $\mathcal{B} \subseteq \{0,1\}^{t \times q}$ of inconsistency matrices if for all $\mathbf{B} \in \mathcal{B}$, there is some $j \in [\gamma]$ such that $\Gamma^{(j)}$ is (κ_1, κ_2)-non-concentrating with respect to $\mathbf{B}$.

Putting the pieces together. To summarize, the goal of the consistency check is to decide whether the inconsistency graph $\mathcal{G}_{\mathbf{W},\mathbf{A}}$ of some assignment matrix $\mathbf{W}$ with respect to a replication structure $\mathbf{A}$ contains a matching of size $\Omega(t)$. Our strategy relies on the following:

- Let $(\Pi_1, \ldots, \Pi_z)$ be a collection of regularity-inducing permutations with respect to $\mathbf{A}$.
- For each $\beta \in [z]$, let $\Gamma_\beta^{(1)}, \ldots, \Gamma_\beta^{(\gamma)}$ be a collection of non-concentrating permutations that implement Π_β, where $\Gamma_\beta^{(j)} = (\Pi_{\beta,1}^{(j)}, \ldots, \Pi_{\beta,\alpha}^{(j)})$ for all $j \in [\gamma]$, and each of the intermediate permutations $\Pi_{\beta,\ell}^{(j)}$ are 2-locally decomposable for all $j \in [\gamma]$, $\beta \in [z]$, and $\ell \in [\alpha]$.

The proof then consists of the initial assignment matrix $\mathbf{W}$ in addition to all of the intermediate matrices $\mathbf{W}_{\beta,\ell}^{(j)} = \Pi_{\beta,\ell}^{(j)}(\mathbf{W}_{\beta,\ell-1}^{(j)})$, where we define $\mathbf{W}_{\beta,0}^{(j)} = \mathbf{W}$ for all $j \in [\gamma]$, $\beta \in [z]$. The verifier checks consistency of all of the intermediate matrices using Construction 5.29, and applies a pairwise consistency test (Construction 5.31) to each of $\mathbf{W}_{\beta,\alpha}^{(j)}$ for all $j \in [\gamma]$ and $\beta \in [z]$. The soundness argument then proceeds roughly as follows:

- Since $(\Pi_1, \ldots, \Pi_z)$ is regularity-inducing, there is some $\beta \in [z]$ where $\mathcal{G}_{\Pi_\beta(\mathbf{W}),\Pi_\beta(\mathbf{A})}$ contains a regular matching.
- Since $\Gamma_\beta^{(1)}, \ldots, \Gamma_\beta^{(\gamma)}$ is a collection of non-concentrating permutations that implement Π_β, and all of the intermediate consistency checks pass, then there must be some $j \in [\gamma]$ such that $\mathcal{G}_{\mathbf{W}_{\beta,\alpha}^{(j)},\Pi_\beta(\mathbf{A})}$ contains a regular matching of size $\Omega(t)$. The verifier then rejects with exponentially-small probability (in t) by soundness of the pairwise consistency test.

Finally, in our concrete instantiation (described in Section 5.3.4), we show how to construct our collection of regularity-inducing permutations and non-concentrating permutations sequences where $z = O(1)$, $\gamma = O(\log^3 t)$, $\alpha = \Theta(\log t)$. For this setting of parameters, the overall consistency check only incurs *polylogarithmic* overhead to the prover complexity and the proof size. In Section 5.3.3, we give the formal description and analysis of our linear MIP construction.

5.3.3 Quasi-Optimal Linear MIP Construction

In this section, we describe our quasi-optimal linear MIP for circuit satisfiability. We give our construction (Construction 5.35) but defer the security theorem (Theorem 5.50) and analysis to Section 5.3.5. By instantiating Construction 5.35 with the appropriate primitives (described in Sections 5.3.1 and 5.3.4), we obtain the first quasi-optimal linear MIP (Theorem 5.36).

Construction 5.35 (Linear MIP)**.** Fix parameters $t, \delta, k, \varepsilon, d, \rho, \kappa_1, \kappa_2$, and let C be an arithmetic circuit of size s over a finite field $\mathbb{F}$. The construction relies on the following ingredients:

- Let $(f_1, \ldots, f_t, \mathsf{inp}, \mathsf{wit})$ be a quasi-optimal (t, δ)-robust decomposition of C. Let C_i be the arithmetic circuit that computes each constraint $f_i \colon \mathbb{F}^n \times \mathbb{F}^m \to \{0, 1\}$.
- Let $(\mathcal{P}_1, \mathcal{V}_1), \ldots, (\mathcal{P}_t, \mathcal{V}_t)$ be k-query systematic linear PCP systems for circuits $C_1, \ldots, C_t$, respectively, with knowledge error ε and query length d.
- Let $\mathbf{A} \in [m]^{t \times q}$ be the replication structure of $C_1, \ldots, C_t$ (where q is a bound on the number of indices in a witness $\mathbf{w} \in \mathbb{F}^m$ on which each circuit depends). Let $\Pi_1, \ldots, \Pi_z$ be a collection of ρ-regularity-inducing permutations on t-by-q matrices with respect to the replication structure $\mathbf{A}$ (Definition 5.33).
- For $\beta \in [z]$, let $\mathcal{B}_\beta \subseteq \{0,1\}^{t \times q}$ be the set of inconsistency patterns where $\mathbf{B}$ and $\Pi_\beta(\mathbf{B})$ have at most one inconsistency in each row. Let $\Gamma_\beta^{(1)}, \ldots, \Gamma_\beta^{(\gamma)}$ be a collection of permutation sequences implementing Π_β that is (κ_1, κ_2)-non-concentrating for $\mathcal{B}_\beta$ (Definition 5.34). In particular, each $\Gamma_\beta^{(j)}$ is a sequence of α permutations $\big(\Pi_{\beta,1}^{(j)}, \ldots, \Pi_{\beta,\alpha}^{(j)}\big)$, where each intermediate permutation $\Pi_{\beta,\ell}^{(j)}$ is 2-locally decomposable.

The linear MIP with $t \cdot (1 + \alpha\gamma z)$ provers and query length d is defined as follows:

- **Syntax:** The linear MIP consists of $t \cdot (1 + \alpha\gamma z)$ provers. We label the provers as P_i and $P_{\beta,\ell,i}^{(j)}$ for $i \in [t]$, $j \in [\gamma]$, $\beta \in [z]$, and $\ell \in [\alpha]$. To simplify the description, we will often pack the proof vectors from different provers into the rows of a matrix (as in Construction 5.29). To recall, when we say we associate a matrix $\hat{\mathbf{W}} \in \mathbb{F}^{t \times d}$ with provers $(P_1, \ldots, P_t)$, we mean that the i^{th} row of $\hat{\mathbf{W}}$ is the proof vector assigned to prover P_i for all $i \in [t]$. Similarly, when we say the verifier distributes a query matrix $\mathbf{Q} \in \mathbb{F}^{t \times d}$ to provers $(P_1, \ldots, P_t)$, we mean that it submits the i^{th} row of $\mathbf{Q}$ as a query to P_i for all $i \in [t]$.
- **Prover's algorithm:** On input the statement $\mathbf{x}' \in \mathbb{F}^{n'}$ and witness $\mathbf{w}' \in \mathbb{F}^{m'}$, the prover prepares the proof vectors as follows:
  - **Linear PCP proofs.** First, the prover computes $\mathbf{x} \leftarrow \mathsf{inp}(\mathbf{x}')$ and $\mathbf{w} \leftarrow \mathsf{wit}(\mathbf{x}', \mathbf{w}')$. For each $i \in [t]$, it computes a proof $\boldsymbol{\pi}_i \leftarrow \mathcal{P}_i(\mathbf{x}_i, \mathbf{w}_i)$, where $\mathbf{x}_i$ and $\mathbf{w}_i$ denote the bits of the statement $\mathbf{x}$ and witness $\mathbf{w}$ on which circuit C_i depends, respectively. Since $(\mathcal{P}_i, \mathcal{V}_i)$ is a systematic linear PCP, we can write $\boldsymbol{\pi}_i = [\mathbf{w}_i, \mathbf{p}_i]$ where $\mathbf{w}_i \in \mathbb{F}^q$ and $\mathbf{p}_i \in \mathbb{F}^{d-q}$. For $i \in [t]$, the prover associates the vector $\boldsymbol{\pi}_i$ with P_i.
  - **Consistency proofs.** Let $\mathbf{W} \in \mathbb{F}^{t \times q}$ be the matrix where the i^{th} row is the vector $\mathbf{w}_i$. Now, for all $j \in [\gamma]$, $\beta \in [z]$, and $\ell \in [\alpha]$, let $\mathbf{W}_{\beta,\ell}^{(j)} = \big(\Pi_{\beta,\ell}^{(j)} \circ \Pi_{\beta,\ell-1}^{(j)} \circ \cdots \circ \Pi_{\beta,1}^{(j)}\big)(\mathbf{W})$. Let $\hat{\mathbf{W}}_{\beta,\ell}^{(j)} = \big[\mathbf{W}_{\beta,\ell}^{(j)}, 0^{t \times (d-q)}\big]$. The prover associates $\hat{\mathbf{W}}_{\beta,\ell}^{(j)}$ with provers $(P_{\beta,\ell,1}^{(j)}, \ldots, P_{\beta,\ell,t}^{(j)})$.

- **Verifier's query algorithm:** To simplify the description, we will sometimes state the query vectors the verifier submits to each prover P_i and $P^{(j)}_{\beta,\ell,i}$ rather than the explicit query matrices. The verifier's queries are constructed as follows:

  - **Linear PCP queries.** For $i \in [t]$, the verifier invokes the query generation algorithm $\mathcal{Q}_i$ for each of the underlying linear PCP instances $(\mathcal{P}_i, \mathcal{V}_i)$ to obtain a query matrix $\mathbf{Q}_i \in \mathbb{F}^{d \times k}$ and some state information st_i. The verifier gives $\mathbf{Q}_i$ to prover P_i, and saves the state $\mathsf{st} = (\mathsf{st}_1, \dots, \mathsf{st}_t)$.
  - **Routing consistency queries.** For all $j \in [\gamma]$, $\beta \in [z]$, and $\ell \in [\alpha]$, the verifier invokes the query generation algorithm of Construction 5.29 on permutation $\Pi^{(j)}_{\beta,\ell}$ to obtain two query matrices $\mathbf{R}^{(j)}_{\beta,\ell}$ and $\mathbf{S}^{(j)}_{\beta,\ell} \in \mathbb{F}^{t \times q}$. The verifier pads the matrices to obtain $\hat{\mathbf{R}}^{(j)}_{\beta,\ell} = \left[\mathbf{R}^{(j)}_{\beta,\ell}, 0^{t \times (d-q)}\right]$ and $\hat{\mathbf{S}}^{(j)}_{\beta,\ell} = \left[\mathbf{S}^{(j)}_{\beta,\ell}, 0^{t \times (d-q)}\right]$. There are two cases:
    * If $\ell = 1$, the verifier distributes the queries $\hat{\mathbf{R}}^{(j)}_{\beta,\ell}$ to provers $(P_1, \dots, P_t)$.
    * If $\ell > 1$, the verifier distributes the queries $\hat{\mathbf{R}}^{(j)}_{\beta,\ell}$ to provers $\left(P^{(j)}_{\beta,\ell-1,1}, \dots, P^{(j)}_{\beta,\ell-1,t}\right)$.

    In addition, the verifier distributes the queries $\hat{\mathbf{S}}^{(j)}_{\beta,\ell}$ to provers $\left(P^{(j)}_{\beta,\ell,1}, \dots, P^{(j)}_{\beta,\ell,t}\right)$. Intuitively, the verifier is applying the approximate consistency check from Construction 5.29 to every permutation $\Pi^{(j)}_{\beta,\ell}$.
  - **Pairwise consistency queries.** For each $\beta \in [z]$, let $\mathbf{A}_\beta = \Pi_\beta(\mathbf{A})$, and let Π'_β be the pairwise consistency test matrix for $\mathbf{A}_\beta$ (Construction 5.31). The verifier invokes the query generation algorithm of Construction 5.29 on permutation Π'_β to obtain two query matrices $\mathbf{R}_\beta$ and $\mathbf{S}_\beta \in \mathbb{F}^{t \times q}$. It pads the matrices to obtain $\hat{\mathbf{R}}_\beta = [\mathbf{R}_\beta, 0^{t \times (d-q)}]$ and $\hat{\mathbf{S}}_\beta = [\mathbf{S}_\beta, 0^{t \times (d-q)}]$. Next, it distributes $\hat{\mathbf{R}}_\beta$ and $\hat{\mathbf{S}}_\beta$ to $(P^{(j)}_{\beta,\alpha,1}, \dots, P^{(j)}_{\beta,\alpha,t})$ for all $j \in [\gamma]$. In this step, the verifier is checking pairwise consistency of the permuted assignment matrices $\mathbf{W}^{(j)}_{\beta,\alpha}$ for all $j \in [\gamma]$ and $\beta \in [z]$.

  In total, the verifier makes a total of $k + \alpha\gamma z$ queries to each prover P_i for $i \in [t]$. It makes $O(1)$ queries to the other provers.

- **Verifier's decision algorithm:** First, the verifier computes the statement $\mathbf{x} \leftarrow \mathsf{inp}(\mathbf{x}')$. For $i \in [t]$, let $\mathbf{x}_i$ denote the bits of $\mathbf{x}$ on which circuit C_i depends. The verifier processes the responses from each set of queries as follows:

  - **Linear PCP queries.** For $i \in [t]$, let $\mathbf{y}_i \in \mathbb{F}^k$ be the response of prover P_i to the linear PCP queries. For $i \in [t]$, the verifier invokes the decision algorithm $\mathcal{D}_i$ for each of the underlying linear PCP instances $(\mathcal{P}_i, \mathcal{V}_i)$ on the state st_i, the statement $\mathbf{x}_i$, and the response $\mathbf{y}_i$. It rejects the proof if $\mathcal{D}_i(\mathsf{st}_i, \mathbf{x}_i, \mathbf{y}_i) = 0$ for any $i \in [t]$.
  - **Consistency queries.** For each set of routing consistency query responses (for checking consistency of the intermediate permutations $\Pi^{(j)}_{\beta,\ell}$), and for each set of pairwise consistency

query responses (for checking consistency of the final configurations Π'_β), the verifier applies the decision algorithm from Construction 5.29, and rejects if any check fails.

If all of the checks pass, then the verifier accepts the proof.

Instantiating the construction. We defer the security analysis of Construction 5.35 to Section 5.3.5. In Section 5.3.4, we show how to instantiate the regularity-inducing permutations and the non-concentrating permutation sequences needed to apply Construction 5.35. Combining Construction 5.35 with these concrete instantiations (as well as our robust decomposition primitive from Section 5.3.1), we obtain a quasi-optimal linear MIP. We state the formal theorem below, and give the proof in Section 5.3.5.

Theorem 5.36 (Quasi-Optimal Linear MIP). *Fix a security parameter λ. Let $C \colon \mathbb{F}^n \times \mathbb{F}^m \to \mathbb{F}^h$ be an arithmetic circuit of size s over a $\mathrm{poly}(\lambda)$-size finite field $\mathbb{F}$ where $|\mathbb{F}| > s$. Then, there exists an input-oblivious k-query linear MIP $(\mathcal{P}, \mathcal{V})$ with $\ell = \widetilde{O}(\lambda)$ provers for $\mathcal{R}_C$ with soundness error $2^{-\lambda}$, query length $\widetilde{O}(s/\lambda) + \mathrm{poly}(\lambda, \log s)$, and $k = \mathrm{polylog}(\lambda)$. Moreover, letting $\mathcal{V} = (\mathcal{Q}, \mathcal{D})$, the prover and verifier algorithms satisfy the following properties:*

- *the prover algorithm $\mathcal{P}$ is an arithmetic circuit of size $\widetilde{O}(s) + \mathrm{poly}(\lambda, \log s)$;*
- *the query-generation algorithm $\mathcal{Q}$ is an arithmetic circuit of size $\widetilde{O}(s) + \mathrm{poly}(\lambda, \log s)$;*
- *the decision algorithm $\mathcal{D}$ is an arithmetic circuit of size $\widetilde{O}(\lambda n)$.*

Remark 5.37 (Soundness Against Affine Provers). To leverage our linear MIP to construct a SNARG, we require that the linear MIP provide soundness against affine provers. We note that Construction 5.35 inherits this property as long as the underlying linear PCPs and approximate consistency check primitives provide soundness against affine strategies. It is straightforward to see that Construction 5.29 remains sound even against affine adversarial strategies. Moreover, we can apply the transformation from Bitansky et al. [BCI+13, Construction 3.1] to existing linear PCPs to obtain linear PCPs with soundness against affine provers. Specifically, Bitansky et al. show how to take a k-query linear PCP over a finite field $\mathbb{F}$ with soundness error ε against linear provers to obtain a $(k+1)$-query linear PCP over $\mathbb{F}$ with soundness error $(\varepsilon + 1/|\mathbb{F}|)$ against affine provers by introducing an additional consistency check. In fact, the construction in [BCI+13] provides even stronger soundness guarantees, but those will not be needed in this work. Note that neither of these modifications increase the asymptotic complexity of Construction 5.35.

5.3.4 Constructing Randomized Permutation Decompositions

In this section, we show how to instantiate the underlying building blocks we require for performing our consistency checks. First, we construct a regularity-inducing permutation assuming that every value in the replication structure $\mathbf{A}$ appears at most twice. This assumption is satisfied, for instance, by the replication structure of our robust decomposition based on MPC from Section 5.3.1.

Construction 5.38 (Regularity-Inducing Permutations)**.** Fix integers $m, t, q \in \mathbb{N}$, with t even, and let $\mathbf{A} \in [m]^{t \times q}$ be a replication structure where every value in $\mathbf{A}$ appears at most twice. We construct permutations Π_1, Π_2 over the entries of $\mathbf{A}$ as follows:

- First, we partition $\mathbf{A}$ into $t' = t/2$ blocks, each containing a pair of rows: for $i \in [t']$, let $\mathbf{A}_i = \mathbf{A}_{[2(i-1)+1,2i]}$.
- We construct matrices $\mathbf{A}^{(1)}, \mathbf{A}^{(2)} \in [m]^{t \times q}$ as follows. For each block $i \in [t']$, we associate with it two sets of values $S_i^{(1)}, S_i^{(2)} \subseteq [m]$. For $j \in \{1, 2\}$, $S_i^{(j)}$ is the set of values that appear in the j^{th} row of $\mathbf{A}_i$, but not in any previous set $S_{i'}^{(j)}$ where $i' < i$. Matrix $\mathbf{A}^{(j)}$ is then constructed as follows:
    1. Initialize all values in $\mathbf{A}^{(j)}$ with $\perp$ to denote an "unassigned" position.
    2. Let $\mathbf{A}_i^{(j)} = \mathbf{A}_{[2(i-1)+1,2i]}^{(j)}$ denote the i^{th} block of $\mathbf{A}^{(j)}$. Let $v_1, \ldots, v_d \in [m]$ be the elements in $S_i^{(j)}$ where $d \le q$. Then, for $k \in [d]$, let c_{v_k} be the total number of times v_k appears in $\mathbf{A}$. Set the first c_{v_k} entries of column k of $\mathbf{A}_i^{(j)}$ to the value v_k.
    3. For all remaining values that appear in $\mathbf{A}$ but not $\mathbf{A}^{(j)}$, assign them arbitrarily to any unassigned position in $\mathbf{A}^{(j)}$.
- By construction, each $\mathbf{A}^{(j)}$ is a permutation of the entries in $\mathbf{A}$. Output the permutations Π_1, Π_2 where $\mathbf{A}^{(1)} = \Pi_1(\mathbf{A})$ and $\mathbf{A}^{(2)} = \Pi_2(\mathbf{A})$.

Lemma 5.39. *Fix integers $m, t, q \in \mathbb{N}$, with t even, and let $\mathbf{A} \in [m]^{t \times q}$ be a replication structure where every entry appears at most twice. Let Π_1, Π_2 be the permutations of $\mathbf{A}$ from Construction 5.38. Then, (Π_1, Π_2) is a collection of 2-regularity-inducing permutations (Definition 5.33) with respect to $\mathbf{A}$.*

Proof. Set $t' = t/2$, and for $i \in [t']$, let $S_i = \{2(i-1)+1, 2i\}$. Let M be the matching in $\mathcal{G}_{\mathbf{W},\mathbf{A}}$ of size s. Take any edge $(i_1, i_2) \in M$, and define $i_1', i_2' \in [t']$ so that $i_1 \in S_{i_1'}$ and $i_2 \in S_{i_2'}$. Without loss of generality, suppose $i_1' \le i_2'$. This means that there exists $j_1, j_2 \in [q]$ where $\mathbf{A}_{i_1,j_1} = \mathbf{A}_{i_2,j_2}$, but $\mathbf{W}_{i_1,j_1} \ne \mathbf{W}_{i_2,j_2}$. Let $v = \mathbf{A}_{i_1,j_1} \in [m]$. Since each entry in $\mathbf{A}$ appears at most twice, the first block in $\mathbf{A}$ that contains v is i_1'. We consider two cases:

- Suppose $i_1 = 1 \bmod 2$. This means that $v \in S_{i_1'}^{(1)}$ in Construction 5.38. Thus, Π_1 maps entries in positions (i_1, j_1) and (i_2, j_2) to some column in rows i_1 and $i_1 + 1$. Thus, $\mathcal{G}_{\mathbf{W}_1,\mathbf{A}_1}$ contains an inconsistency in rows $(i_1, i_1+1) \in S_{i_1'}$.
- Suppose $i_1 = 0 \bmod 2$. By an analogous argument as that used in the previous case, we conclude that $\mathcal{G}_{\mathbf{W}_2,\mathbf{A}_2}$ contains an inconsistency in rows $(i_1 - 1, i_1) \in S_{i_1'}$.

The above analysis shows that each edge $(i_1, i_2) \in M$ either contributes an edge $(i_1, i_1 + 1) \in S_{i_1'}$ to $\mathcal{G}_{\mathbf{W}_1,\mathbf{A}_1}$ or contribute an edge $(i_1 - 1, i_1) \in S_{i_1'}$ to $\mathcal{G}_{\mathbf{W}_2,\mathbf{A}_2}$. In other words, each edge (i_1, i_2) contributes a single edge to a *regular* matching in $\mathcal{G}_{\mathbf{W}_1,\mathbf{A}_1}$ or $\mathcal{G}_{\mathbf{W}_2,\mathbf{A}_2}$ Since $|M| \geq s$, we conclude that at least one of the graphs $\mathcal{G}_{\mathbf{W}_1,\mathbf{A}_1}, \mathcal{G}_{\mathbf{W}_2,\mathbf{A}_2}$ must contain a regular matching of size at least $s/2$. Moreover, the correspondence between the edges in the regular matching for graphs $\mathcal{G}_{\mathbf{W}_1,\mathbf{A}_1}$ and $\mathcal{G}_{\mathbf{W}_2,\mathbf{A}_2}$ and the edges in M is immediate from the above analysis. $\square$

Randomized permutation decomposition. Next, we show how to construct a sequence of non-concentrating 2-locally decomposable permutations. Recall that a non-concentrating sequence of permutations Γ implementing a permutation Π is an ordered set of permutations $(\Pi_1, \ldots, \Pi_\alpha)$ such that if we start with *any* inconsistency matrix $\mathbf{B} \in \{0,1\}^{t \times q}$ where both $\mathbf{B}$ and $\Pi(\mathbf{B})$ have at most one inconsistency in each row, then the positions of the inconsistencies in the intermediate matrices $\mathbf{B}_\ell = \Pi_\ell(\mathbf{B}_{\ell-1})$ and $\mathbf{B}_0 = \mathbf{B}$ do not concentrate in a small number of rows. We begin by giving a high-level outline of how we sample such sequences for a target permutation Π.

- First, we apply Lemma 5.5 to Π to obtain three permutations Π_1, Π_2, Π_3 where Π_1 and Π_3 are row-wise restricted and Π_2 is column-wise restricted. Moreover, the decomposition satisfies $\Pi = \Pi_3 \circ \Pi_2 \circ \Pi_1$.

- Next, we randomize the first row-rise restricted permutation Π_1. Our randomization procedure exploits the observation that if two entries j_1, j_2 in the same row have the same target column (under Π), then we can swap the entries j_1 and j_2 under Π_1 and undo the swap in Π_3. We describe this procedure in Construction 5.40. Then, we show in Theorem 5.42 that as long as there are many entries in each row of Π that map to the same column, this randomization procedure distributes the inconsistencies in $\mathbf{B}$ across many columns.

- After randomizing Π_1, we apply the randomized 2-local decomposition from Construction 5.9 based on randomized routing in a Beneš network to Π_2 to obtain a (randomized) sequence of 2-local permutations implementing Π_2. We show in Theorem 5.43 that over the randomness used to sample the randomized 2-local decomposition, the inconsistencies in $\mathbf{B}$ do not concentrate in a small number of rows. Intuitively, this follows from the fact that the inconsistencies in $\mathbf{B}$ are distributed across many columns (from the row-wise shuffling procedure in the previous step), and the fact that each column is independently randomized in Construction 5.9. We can then show that the probability that the inconsistencies across many columns concentrate in a small number of rows is exponentially small.

- Theorems 5.42 and 5.43 show that for a *fixed* inconsistency pattern $\mathbf{B}$, the probability that the inconsistencies in $\mathbf{B}$ concentrate in a small number of rows is exponentially small. Here, the probability is taken over the randomness used to sample the row-wise and column-wise decompositions. To construct a collection of permutation sequences $\Gamma^{(1)}, \ldots, \Gamma^{(\gamma)}$ such that for *all* inconsistency patterns $\mathbf{B}$, there is a non-concentrating sequence, we simply sample many independent sequences $\Gamma^{(j)}$. In Theorem 5.48, we show that if we sample $\gamma = O(\log^3 t)$ such sequences, then with probability $1 - 2^{-\Omega(t)}$, for all inconsistency patterns $\mathbf{B}$ that contain at most one inconsistency in each row, at least one of the sequences $\Gamma^{(j)}$ will be non-concentrating. We give the overall construction in Construction 5.47.

Construction 5.40 (Row-Wise Random Permutation Decomposition)**.** Fix positive integers $t, q \in \mathbb{N}$. Let Π be a permutation over the entries of a t-by-q matrix. The row-wise random permutation decomposition (Π_1, Π_2, Π_3) is then defined as follows:

- First let $(\hat{\Pi}_1, \hat{\Pi}_2, \hat{\Pi}_3)$ be the decomposition of Π from Lemma 5.5. In particular, $\hat{\Pi}_1$ and $\hat{\Pi}_3$ are row-wise restricted permutations and $\hat{\Pi}_2$ is a column-wise restricted permutation.
- Since $\hat{\Pi}_1$ is a row-wise restricted permutation, we can decompose it into a product of t independent permutations $\hat{\Pi}_1^{(1)}, \ldots, \hat{\Pi}_1^{(t)}$, where the ith permutation $\hat{\Pi}_1^{(i)} \colon [q] \to [q]$ is applied to the ith row of the matrix. Similarly, we can express $\hat{\Pi}_2$ as a product of q independent permutations $\hat{\Pi}_2^{(1)}, \ldots, \hat{\Pi}_2^{(q)}$, where each $\hat{\Pi}_2^{(j)} \colon [t] \to [t]$ is applied to the jth column of the matrix.
- For each $i \in [t]$, define the vector $\hat{\mathbf{c}}^{(i)} \in [t]^q$, where for all $j \in [q]$, $\hat{\mathbf{c}}_j^{(i)}$ is the permuted row-index of the $(i, j)^{\text{th}}$ entry of the matrix under $\hat{\Pi}_2 \circ \hat{\Pi}_1$. Specifically, $\hat{\mathbf{c}}_j^{(i)} = \hat{\Pi}_2^{(\hat{\Pi}_1^{(i)}(j))}(i)$. Then, define sets $\hat{T}_1^{(i)}, \ldots, \hat{T}_t^{(i)} \subseteq [q]$ where $\hat{T}_\beta^{(i)}$ is the set of indices $j \in [q]$ where $\hat{\mathbf{c}}_j^{(i)} = \beta$. Clearly, $\hat{T}_1^{(i)}, \ldots, \hat{T}_t^{(i)}$ form a partition for $[q]$. For each $\beta \in [t]$, sample a random permutation $\pi_\beta^{(i)} \colon [q] \to [q]$ that *randomly* permutes the indices in $\hat{T}_\beta^{(i)}$ and leaves the other indices unchanged. Define $\Pi_1^{(i)} = \pi_q^{(i)} \circ \cdots \circ \pi_1^{(i)} \circ \hat{\Pi}_1^{(i)}$ and Π_1 to be the row-wise restricted permutation where its action on row i is given by $\Pi_1^{(i)}$.
- Let $\Pi_2 = \hat{\Pi}_2$ and choose Π_3 such that $\Pi = \Pi_3 \circ \Pi_2 \circ \Pi_1$.

Lemma 5.41. *Fix positive integers $t, q \in \mathbb{N}$ and let Π be a permutation over the entries of a t-by-q matrix. Let (Π_1, Π_2, Π_3) be the row-wise random permutation decomposition of Π from Construction 5.40. Then, $\Pi = \Pi_3 \circ \Pi_2 \circ \Pi_1$, both Π_1 and Π_3 are row-wise restricted, and Π_2 is column-wise restricted.*

Proof. By construction, $\Pi = \Pi_3 \circ \Pi_2 \circ \Pi_1$. Let $\hat{\Pi}_1, \hat{\Pi}_2, \hat{\Pi}_3$ be the permutations used to construct Π_1, Π_2, Π_3 in Construction 5.40. Since $\hat{\Pi}_1$ is row-wise restricted and permutation $\hat{\Pi}_2$ is column-wise restricted, Π_1 and Π_2 are by construction row-wise restricted and column-wise restricted, respectively. It suffices to show that Π_3 is row-wise restricted.

Take any entry $(i, j) \in [t] \times [q]$, and let (i_t, j_t) be its image under permutation Π. Let $(i_1, j_1) = \Pi_1(i, j)$ and $(i_2, j_2) = \Pi_2(i_1, j_1)$. We show that $i_2 = i_t$, or equivalently, after applying $\Pi_2 \circ \Pi_1$, every entry (i, j) ends up in the same row as $\Pi(i, j)$. In this case, the permutation Π_3 only needs to changes the column of each entry, and the claim follows. We show this in a sequence of steps.

- Let $(i_1', j_1') = \hat{\Pi}_1(i, j)$ and $(i_2', j_2') = \hat{\Pi}_2(i_1', j_1') = \Pi_2(i_1', j_1')$, since $\Pi_2 = \hat{\Pi}_2$.
- For $j \in [q]$, let $\Pi_2^{(j)} \colon [t] \to [t]$ be the permutation Π_2 implements on the j^{th} column. In addition, let $\Pi_1^{(i)}, \hat{\Pi}_1^{(i)} \colon [q] \times [q]$ be the permutations Π_1 and $\hat{\Pi}_1$ implement on i^{th} row, respectively.
- Since Π_1, $\hat{\Pi}_1$, and $\hat{\Pi}_3$ are row-wise restricted, $i_1' = i = i_1$ and $i_2' = i_t$. Moreover, by definition, $i_2' = \Pi_2^{(j_1')}(i)$, $i_2 = \Pi_2^{(j_1)}(i)$, $j_1' = \hat{\Pi}_1^{(i)}(j)$, and $j_1 = \Pi_1^{(i)}(j)$.
- By construction,

$$j_1 = \Pi_1^{(i)}(j) = \big(\pi_q^{(i)} \circ \cdots \circ \pi_1^{(i)}\big)\big(\hat{\Pi}_1^{(i)}(j)\big) = \big(\pi_q^{(i)} \circ \cdots \circ \pi_1^{(i)}\big)(j_1').$$

  By design, each $\pi_\beta^{(i)}$ for $\beta \in [q]$ only permutes indices $j \in [q]$ where $\hat{\Pi}_2^{(\hat{\Pi}_1^{(i)}(j))}(i) = \beta$. In particular, this means that if $j_1 = \big(\pi_q^{(i)} \circ \cdots \circ \pi_1^{(i)}\big)(j_1')$, it must be the case that $i_2' = \hat{\Pi}_2^{(j_1')}(i) = \hat{\Pi}_2^{(j_1)}(i) = i_2$. Since $i_2' = i_t$, this means that $i_2 = i_t$, which complete the proof. □

Theorem 5.42 (No-Concentration in Columns)**.** *Fix integers $t, q \in \mathbb{N}$, where $q = \mathrm{poly}(t)$, and let Π be a permutation over the entries of a t-by-q matrix. Define the following quantities:*

- *Let (Π_1, Π_2, Π_3) be the row-wise random permutation decomposition of Π from Construction 5.40.*
- *Let $\mathbf{B} \in \{0,1\}^{t \times q}$ be an inconsistency matrix with z inconsistencies at indices $(i_1, j_1), \dots, (i_z, j_z) \in [t] \times [q]$. In particular, $\mathbf{B}_{i_\beta, j_\beta} = 1$ for all $\beta \in [z]$.*
- *Let $(i_1', j_1'), \dots, (i_z', j_z')$ be the positions of the inconsistencies in $\Pi_1(\mathbf{B})$.*

Suppose that the following condition holds:

- ***Condition 1:** All of the $i_1, \dots, i_z$ are distinct. Namely, each row of $\mathbf{B}$ contains at most one inconsistency.*
- ***Condition 2:** For every pair of rows $i_1, i_2 \in [t]$, there are at least t indices $j \in [q]$ such that $\Pi(i_1, j)$ is in row i_2.*

For a set $S \subseteq [q]$ of column indices, let n_S denote the number of indices $\beta \in [z]$ where $j_\beta' \in S$. Then, for every constant $c_{\mathsf{col}} > 0$, there exists a constant $s_{\mathsf{col}} > 0$ such that with probability $1 - 2^{-\Omega(t/\log t)}$ (taken over the randomness used to sample Π), the following condition holds: for all sets $S \subseteq [q]$ where $|S| \le s_{\mathsf{col}} \cdot t/\log^2 t$, we have that $n_S < c_{\mathsf{col}} \cdot t/\log t$.

Proof. Let $c_{\mathsf{col}} > 0$ be any constant, and fix a constant $s_{\mathsf{col}} > 0$. Let $S \subseteq [q]$ be a set where $|S| \le s_{\mathsf{col}} \cdot t/\log^2 t$. Suppose Conditions 1 and 2 hold and consider the event where $n_S \ge c_{\mathsf{col}} \cdot t/\log t$. We first show that this event happens with probability $2^{-k \cdot c_{\mathsf{col}} \cdot t/\log t}$ for some constant $k > 0$ (independent of s_{col}), where the probability is taken over the randomness used to sample Π_1 in Construction 5.40.

- First, let $\hat{\Pi}_2$ be the column-wise restricted permutation from Construction 5.40, and for $j \in [q]$, let $\hat{\Pi}_2^{(j)} \colon [t] \to [t]$ be the permutation $\hat{\Pi}_2$ implements on column j. By Condition 2 and the fact that $\hat{\Pi}_1$ and $\hat{\Pi}_3$ are both row-wise restricted, this means that for all pairs of rows $i_1, i_2 \in [t]$, there are at least t indices $j \in [q]$ where $\hat{\Pi}_2^{(j)}(i_1) = i_2$. This means that for each row $i \in [t]$, the sets $\hat{T}_1^{(i)}, \ldots, \hat{T}_t^{(i)}$ in Construction 5.40 all contain at least t elements. Thus, over the randomness used to sample Π_1, for any inconsistency (i_β, j_β), the value of j'_β is uniformly random over a set of size at least t. This means that for all $\beta \in [z]$, $\Pr[j'_\beta \in S] \le |S|/t = O(1/\log^2 t)$ since $|S| = O(t/\log^2 t)$. In particular, for all $\beta \in [z]$ (and sufficiently large t), $\Pr[j'_\beta \in S] \le c_{\mathsf{col}}/(2\log t)$

- For $\beta \in [z]$, let X_β be an indicator random variable for the event $j'_\beta \in S$. From the above analysis, we have that $\Pr[X_\beta = 1] \le c_{\mathsf{col}}/(2\log t)$. Moreover, Π_1 is composed of t independent row permutations, and there is only one inconsistency in each row of $\mathbf{B}$ (Condition 1), so the variables $X_1, \ldots, X_z$ are all independent. By definition, $n_S = \sum_{\beta \in [z]} X_\beta$, so by a Chernoff bound,
$$\Pr[n_S \ge c_{\mathsf{col}} \cdot t/\log t] \le 2^{-k \cdot c_{\mathsf{col}} \cdot (t/\log t)},$$
where $k > 0$ is a constant.

To conclude the proof, we apply a union over all sets $S \subseteq [q]$ of size $s_{\mathsf{col}} \cdot t/\log^2 t$. The number of such sets is bounded by
$$\binom{q}{s_{\mathsf{col}} \cdot t/\log^2 t} \le q^{s_{\mathsf{col}} \cdot t/\log^2 t} \le 2^{k' \cdot s_{\mathsf{col}} \cdot (t/\log t)},$$
for some constant $k' > 0$ (independent of s_{col}) since $q = \mathrm{poly}(t)$. The claim follows by taking $s_{\mathsf{col}} < k/k' \cdot c_{\mathsf{col}}$. $\square$

Theorem 5.43 (No-Concentration in Rows). *Fix positive integers $m, t, q \in \mathbb{N}$ where $t = 2^d$ for some $d \in \mathbb{N}$, and $q = \mathrm{poly}(t)$. Then, define the following quantities:*

- *Let Π be a column-wise restricted permutation over the entries of a t-by-q matrix, and $\Pi_1, \ldots, \Pi_{2d}$ be a randomized 2-local decomposition of Π from Construction 5.13.*

- *Let $\mathbf{B} \in \{0,1\}^{t \times q}$ be an inconsistency matrix with $z = \Omega(t)$ inconsistencies at locations $(i_1, j_1), \ldots, (i_z, j_z) \in [t] \times [q]$. In other words, $\mathbf{B}_{i_1,j_1} = \cdots = \mathbf{B}_{i_z,j_z} = 1$.*

- *For $\ell \in [2d]$, let $(i_1^{(\ell)}, j_1^{(\ell)}), \ldots, (i_z^{(\ell)}, j_z^{(\ell)})$ be the positions $(i_1, j_1), \ldots, (i_z, j_z)$ permuted according to the permutation $\Pi_\ell \circ \cdots \circ \Pi_1$.*

- *For a subset $S \subseteq [t]$ of row indices, let $n_S^{(\ell)} = \left|\{\beta \in [z] : i_\beta^{(\ell)} \in S\}\right|$. In words, $n_S^{(\ell)}$ is the number of inconsistencies in $(\Pi_\ell \circ \cdots \circ \Pi_1)(\mathbf{B})$ that fall into the rows identified by S.*

Suppose that the following conditions hold:

- ***Condition 1:*** *All of the $i_1, \ldots, i_z$ are distinct. Similarly, $i_1^{(2d)}, \ldots, i_z^{(2d)}$ are also all distinct. Namely, each row of $\mathbf{B}$ and $\Pi(\mathbf{B})$ contain at most one inconsistency. In particular, $z \leq t$.*

- ***Condition 2:*** *For every constant $c_{\mathsf{col}} > 0$, there exists a constant $s_{\mathsf{col}} > 0$ such that the following holds: for all sets $S \subseteq [q]$ of column indices where $|S| \leq s_{\mathsf{col}} \cdot t / \log^2 t$, the number of indices $\beta \in [z]$ where $j_\beta \in S$ is less than $c_{\mathsf{col}} \cdot t / \log t$.*

Then, for all constants $c_{\mathsf{row}} > 0$ and $s_{\mathsf{row}} > 0$, with probability $1 - 2^{-\Omega(t/\log^2 t)}$ (taken over the choice of randomness in sampling the 2-local decomposition of Π), the following holds: for all indices $\ell \in [2d]$ and sets $S \subseteq [t]$ where $|S| \leq s_{\mathsf{row}} \cdot t / \log^5 t$, it holds that $n_S^{(\ell)} < c_{\mathsf{row}} \cdot t / \log t$.

Proof. Fix constants $c_{\mathsf{row}}, s_{\mathsf{row}} > 0$, a set $S \subseteq [t]$ where $|S| \leq s_{\mathsf{row}} \cdot t / \log^5 t$ and an index $\ell \in [2d]$. Suppose that Conditions 1 and 2 hold, and consider the event where $n_S^{(\ell)} \geq c_{\mathsf{row}} \cdot t / \log t$. We now show that this event occurs with probability $2^{-\Omega(t/\log^2 t)}$. We first analyze the case where $\ell \leq d$. The $\ell > d$ case is analogous. First, by appealing to Condition 2, we argue that the inconsistencies in $\mathbf{B}$ must be distributed across many columns.

Claim 5.44. *Let $s_{\mathsf{col}} > 0$ be the constant from Condition 2 for the case where $c_{\mathsf{col}} = c_{\mathsf{row}}$. Then, there exists a set $T \subseteq [q]$ of column indices where $|T| \geq (s_{\mathsf{col}}/2) \cdot t / \log^2 t$ satisfying the following properties:*

- *For all $j \in T$, the number of indices $\beta \in [z]$ where $j_\beta = j$ is at most $2/s_{\mathsf{col}} \cdot \log^2 t$. That is, the j^{th} column of $\mathbf{B}$ contains at most $(2/s_{\mathsf{col}}) \cdot \log^2 t$ inconsistencies.*

- *For all $j \in T$, there is at least one $\beta \in [z]$ where $i_\beta^{(\ell)} \in S$ and $j_\beta^{(\ell)} = j$. That is, after applying $\Pi_1, \ldots, \Pi_\ell$ to $\mathbf{B}$, there is an inconsistency in row i and column j of the resulting matrix.*

Proof. Let $\hat{T} = \{ j_\beta^{(\ell)} : \beta \in [z] \text{ and } i_\beta^{(\ell)} \in S \}$ be the set of column indices of the inconsistencies whose row indices fall into set S in layer ℓ. Our argument proceeds in two steps:

- Since $\Pi_1, \ldots, \Pi_\ell$ are column-wise restricted, it follows that $j_\beta^{(\ell)} = j_\beta$ for all $\beta \in [z]$. By assumption, $n_S^{(\ell)} \geq c_{\mathsf{row}} \cdot t / \log t$. This means that $|\hat{T}| > s_{\mathsf{col}} \cdot t / \log^2 t$, since otherwise, $\hat{T}$ is a set with at most $s_{\mathsf{col}} \cdot t / \log^2 t$ indices that contains $c_{\mathsf{row}} \cdot t / \log t = c_{\mathsf{col}} \cdot t / \log t$ indices from the multiset $\{j_1, \ldots, j_z\}$. This violates Condition 2.

- Define $T \subseteq \hat{T}$ to be the set of indices $j \in \hat{T}$ where the j^{th} column of $\mathbf{B}$ contains fewer than $(2/s_{\mathsf{col}}) \cdot \log^2 t$ inconsistencies. By construction, the set T satisfies both of the required properties. It suffices to argue that $|T| \geq (s_{\mathsf{col}}/2) \cdot t / \log^2 t$. Suppose otherwise. From above, we know that $|\hat{T}| > s_{\mathsf{col}} \cdot t / \log^2 t$, and so if $|T| < (s_{\mathsf{col}}/2) \cdot t / \log^2 t$, there are more than $(s_{\mathsf{col}}/2) \cdot t / \log^2 t$ columns in $\mathbf{B}$ that contain $2/s_{\mathsf{col}} \cdot \log^2 t$ inconsistencies, which means that $\mathbf{B}$ contains more than t inconsistencies, which is a contradiction. Thus, $|T| \geq (s_{\mathsf{col}}/2) \cdot t / \log^2 t$, and the claim follows. □

In the following, we take $s_{\mathsf{col}} > 0$ to be the constant from Claim 5.44. Now, since the permutations $\Pi_1, \ldots, \Pi_{2d}$ are generated according to Construction 5.13, the entries in each column of the inconsistency matrix are routed using independent BENEŠ_d networks (each of which implements the permutation Π on its respective column). This means that in layer ℓ, we can partition the t rows of $\mathbf{W}$ into $r = 2^{d-\ell}$ disjoint subsets $R_1, \ldots, R_r$, each containing 2^ℓ rows, such that the permutation $\Pi_\ell \circ \cdots \circ \Pi_1$ factors into the product of r independent permutations, each operating on one of the subsets $R_1, \ldots, R_r$ (Fact 5.7). Let $w = |S|$ and $w_1, \ldots, w_r$ be the number of rows of S that fall into each of the subsets $R_1, \ldots, R_r$, respectively. We now show that there are not many blocks in $\mathbf{B}$ where a large fraction of the rows within those blocks contain an inconsistency.

Claim 5.45. *For any constant $\varepsilon > 0$ (and sufficiently large t), there are at most $(s_{\mathsf{col}}/4) \cdot t/\log^2 t$ rows $i \in [t]$ where $i \in R_k$ for some $k \in [r]$ where $w_k/2^\ell > \varepsilon/\log^2 t$.*

Proof. Suppose there are $(s_{\mathsf{col}}/4) \cdot t/\log^2 t = (s_{\mathsf{col}}/4)(r \cdot 2^\ell)/\log^2 t$ rows i where $i \in R_k$ and $w_k/2^\ell > \varepsilon/\log^2 t$. Since each block contains 2^ℓ rows, this means there are at least $(s_{\mathsf{col}}/4) \cdot r/\log^2 t$ blocks $k \in [r]$ where $w_k/2^\ell > \varepsilon/\log^2 t$. But now,

$$w = \sum_{k \in [r]} w_k > (s_{\mathsf{col}}/4)(r/\log^2 t)(2^\ell \cdot \varepsilon/\log^2 t) = \varepsilon \cdot (s_{\mathsf{col}}/4) \cdot t/\log^4 t = \Theta(t/\log^4 t).$$

This is a contradiction since $w = |S| = O(t/\log^5 t)$. □

Let T be the set of column indices from Claim 5.44 where for all $j \in T$, the jth column in $\mathbf{B}$ contains at most $2/s_{\mathsf{col}} \cdot \log^2 t$ inconsistencies. To complete the proof, we first say that a column $j \in T$ is "good" if for all (i_β, j_β) where $j_\beta = j$, then $i_\beta \in R_k$ for some k where $w_k/2^\ell < s_{\mathsf{col}}/(4\log^2 t)$. Otherwise, we say the column is "bad." By Condition 1, each row has at most one inconsistency, and by Claim 5.45, the number of rows where $i_\beta \in R_k$ for some k where $w_k/2^\ell > s_{\mathsf{col}}/(4\log^2 t)$ is at most $(s_{\mathsf{col}}/4) \cdot t/\log^2 t$. Thus, there can be at most $(s_{\mathsf{col}}/4) \cdot t/\log^2 t$ "bad" columns. Since $|T| \geq (s_{\mathsf{col}}/2) \cdot t/\log^2 t$, we conclude that there are at least $(s_{\mathsf{col}}/4) \cdot t/\log^2 t$ "good" columns in T. We now show the following claim:

Claim 5.46. *If $j \in T$ is a "good" column, then the probability that there exists $\beta \in [z]$ where $j_\beta^{(\ell)} = j$ and $i_\beta^{(\ell)} \in S$ is at most $1/2$. Here, the probability is taken over the randomness used to sample the routing configuration for column j.*

Proof. Take any inconsistency (i_β, j_β) where $j_\beta = j$. Set $k \in [r]$ so that $i_\beta \in R_k$. By Lemma 5.11, over the choice of the randomness used to sample the routing configuration for column j, the distribution of $i_\beta^{(\ell)}$ is uniform over R_k. Therefore, $\Pr[i_\beta^{(\ell)} \in S] = w_k/2^\ell < s_{\mathsf{col}}/(4\log^2 t)$, since j is a "good" column. Finally, since $j \in T$, column j contains at most $(2/s_{\mathsf{col}}) \cdot \log^2 t$ inconsistencies. By a union bound, the probability that there exists $\beta \in [z]$ where $j_\beta = j$ and $i_\beta^{(\ell)} \in S$ is bounded by $1/2$. □

From above, there are at least $(s_{\mathsf{col}}/4) \cdot t/\log^2 t$ "good" columns in T. Moreover, for every $j \in T$, there is a $\beta \in [z]$ where $i_\beta^{(\ell)} \in S$ and $j_\beta^{(\ell)} = j$. Since Construction 5.13 samples the routing configuration for each of the columns of Π randomly and independently, we conclude from Claim 5.46 that

$$\Pr[\forall j \in T : i_\beta^{(\ell)} \in S \text{ and } j_\beta^{(\ell)} = j] \leq 2^{-(s_{\mathsf{col}}/4) \cdot t/\log^2 t} = 2^{-\Omega(t/\log^2 t)},$$

where the probability is taken over the randomness used to sample the decomposition Π. Correspondingly, this implies that the probability that $n_S^{(\ell)} \geq c_{\mathsf{row}} \cdot t/\log t$ is bounded by $2^{-\Omega(t/\log^2 t)}$.

To conclude the proof, we apply a union over all sets $S \subseteq [q]$ of size $s = O(t/\log^5 t)$ and all indices $\ell \in [2d]$. The number of such sets is bounded by $q^s \leq 2^{O(t/\log^4 t)}$ since $q = \mathrm{poly}(t)$. Moreover, $d = \log t$, so by a union bound, the probability that there exists $S \subseteq [t]$ of size $|S| \leq s$ and $\ell \in [2d]$ where $n_S^{(\ell)} \geq c_{\mathsf{row}} \cdot t/\log t$ is $2^{-\Omega(t/\log^2 t)}$. The claim follows. □

Construction 5.47 (Randomized Permutation Decomposition)**.** Fix positive integers $t, q, \gamma \in \mathbb{N}$ where $t = 2^d$ for some $d \in \mathbb{N}$, and $q, \gamma = \mathrm{poly}(t)$. Let Π be a permutation over the entries of a t-by-q matrix. The randomized permutation decomposition of Π is a collection of γ sequences of permutations $\Gamma^{(1)}, \ldots, \Gamma^{(\gamma)}$ where each $\Gamma^{(i)} = \left(\Pi_1^{(1)}, \Pi_{2,1}^{(1)}, \ldots, \Pi_{2,2d}^{(1)}, \Pi_3^{(1)}\right)$ is a sequence of $\alpha = 2d + 2$ 2-locally decomposable permutations on the entries of a t-by-q matrix. We construct each sequence $\Gamma^{(i)}$ as follows:

- For each $i \in [\gamma]$, apply the row-wise random permutation decomposition from Construction 5.40 to Π to obtain permutations $(\Pi_1^{(i)}, \Pi_2^{(i)}, \Pi_3^{(i)})$.
- For each $i \in [\gamma]$, apply the randomized 2-local decomposition from Construction 5.9 to $\Pi_2^{(i)}$ to obtain $\Pi_{2,1}^{(i)}, \ldots, \Pi_{2,2d}^{(i)}$.

By construction, each permutation sequence $\Gamma^{(i)}$ implements Π in the following sense: $\Pi = \Pi_3^{(i)} \circ \Pi_{2,2d}^{(i)} \circ \cdots \circ \Pi_{2,1}^{(i)} \circ \Pi_1^{(i)}$.

Theorem 5.48 (Randomized Permutation Decomposition)**.** *Fix positive integers $t, q, \gamma \in \mathbb{N}$ where $t = 2^d$ for some $d \in \mathbb{N}$ and $q = \mathrm{poly}(t)$. Then, define the following quantities:*

- *Let Π be a permutation over t-by-q matrices where for every pair of rows $i_1, i_2 \in [t]$, there are at least t indices $j \in [q]$ such that $\Pi(i_1, j)$ is in row i_2.*
- *Let $\Gamma^{(1)}, \ldots, \Gamma^{(\gamma)}$ be the collection of permutation sequences obtained by applying Construction 5.47 to Π. In particular, $\Gamma^{(j)} = \left(\Pi_1^{(j)}, \Pi_{2,1}^{(j)}, \ldots, \Pi_{2,2d}^{(j)}, \Pi_3^{(j)}\right)$ for all $j \in [\gamma]$.*
- *Let $\mathcal{B} \subseteq \{0,1\}^{t \times q}$ be the set of inconsistency patterns $\mathbf{B}$ where $\mathbf{B}$ and $\Pi(\mathbf{B})$ have at most one inconsistency in each row.*

Let $c_1, c_2 > 0$ be arbitrary constants, and let $\kappa_1 = c_1 \cdot t/\log^5 t$ and $\kappa_2 = c_2 \cdot t/\log t$. Then, there exists $\gamma = O(\log^3 t)$ such that with probability $1 - 2^{-\Omega(t)}$ over the choice of randomness in Construction 5.47, *the collection of permutation sequences $\Gamma^{(1)}, \ldots, \Gamma^{(\gamma)}$ implementing Π is (κ_1, κ_2)-non-concentrating for $\mathcal{B}$ (Definition 5.34).*

Proof. We use a union bound. Let $\mathbf{B} \in \mathcal{B}$ be an inconsistency matrix with z inconsistencies at indices $(i_1, k_1), \ldots, (i_z, k_z)$. Consider the probability (over the randomness used by Construction 5.47) that the sequence $\Gamma^{(j)} = \big(\Pi_1^{(j)}, \Pi_{2,1}^{(j)}, \ldots, \Pi_{2,2d}^{(j)}, \Pi_3^{(j)}\big)$ is (κ_1, κ_2)-concentrating for $\mathbf{B}$.

- For $\beta \in [z]$, let $k_{\beta,1}$ denote the column index of (i_β, k_β) after applying the first permutation $\Pi_1^{(j)}$. By Theorem 5.42, for all constants $c_{\mathsf{col}} > 0$, there exists a constant $s_{\mathsf{col}} > 0$ such that with probability $1 - 2^{-\Omega(t/\log t)}$, the following holds: for all sets $S \subseteq [q]$ where $|S| \le s_{\mathsf{col}} \cdot t/\log^2 t$, the number of indices $\beta \in [z]$ where $k_{\beta,1} \in S$ is less than $c_{\mathsf{col}} \cdot t/\log t$.

- For $\beta \in [z]$ and $\ell \in [2d]$, let $i_{\beta,2,\ell}$ denote the row index of (i_β, k_β) after applying the sequence of permutations $\Pi_{2,\ell}^{(j)} \circ \cdots \circ \Pi_{2,1}^{(j)} \circ \Pi_1^{(j)}$. By Theorem 5.43, with probability $1 - 2^{-\Omega(t/\log^2 t)}$, the following holds: for all indices $\ell \in [2d]$ and all sets $S \subseteq [t]$ where $|S| \le c_1 \cdot t/\log^5 t$, the number of indices $\beta \in [z]$ where $i_{\beta,2,\ell} \in S$ is less than $c_2 \cdot t/\log t$. In other words, with probability $1 - 2^{-\Omega(t/\log^2 t)}$, no subset of κ_1 rows in $\Pi_{2,\ell}^{(j)} \circ \cdots \circ \Pi_{2,1}^{(j)} \circ \Pi_1^{(j)}(\mathbf{B})$ contain more than κ_2 inconsistencies.

- By assumption, $\mathbf{B}$ has at most 1 inconsistency in each row. Thus, no subset of κ_1 rows can contain κ_2 inconsistencies (for sufficiently large t). Since $\Pi_1^{(j)}$ is a row-wise restricted permutation, $\Pi_1^{(j)}(\mathbf{B})$ also contains at most 1 inconsistency in each row. Finally, by assumption $\Pi(\mathbf{B})$ contains at most 1 inconsistency in each row, so no subset of κ_1 rows of $\Pi(\mathbf{B})$ can contain κ_2 inconsistencies.

By the above analysis, the probability that $\Gamma^{(j)}$ is *not* (κ_1, κ_2)-non-concentrating for $\mathbf{B}$ is bounded by $2^{-\Omega(t/\log^2 t)}$. Since Construction 5.47 samples all of the permutation sequences $\Gamma^{(j)}$ independently, the probability that $\Gamma^{(j)}$ is *not* (κ_1, κ_2)-non-concentrating for $\mathbf{B}$ for all $j \in [\gamma]$ is bounded by $2^{-\Omega(\gamma t/\log^2 t)}$. Concretely, let $2^{-c_1' \cdot \gamma t/\log^2 t}$ where $c_1' > 0$ is a constant be a bound on the probability that $\Gamma^{(j)}$ is not (κ_1, κ_2)-non-concentrating for $\mathbf{B}$ for all $j \in [\gamma]$

To complete the analysis, we use a union bound to bound the probability that $\Gamma^{(1)}, \ldots, \Gamma^{(\gamma)}$ is not (κ_1, κ_2)-non-concentrating for the set $\mathcal{B}$. First, we have $|\mathcal{B}| \le (q+1)^t$, since $\mathcal{B}$ only contains inconsistency matrices with at most one inconsistency in each row. Since $q = \mathrm{poly}(t)$, this means that $|\mathcal{B}| \le 2^{c_2' \cdot t \log t}$ for some constant $c_2' > 0$. By the union bound, the probability that there exists $\mathbf{B} \in \mathcal{B}$ for which $\Gamma^{(1)}, \ldots, \Gamma^{(\gamma)}$ is not (κ_1, κ_2)-non-concentrating for $\mathbf{B}$ is at most $2^{c_2' \cdot t \log t - c_1' \cdot \gamma t/\log^2 t}$. Setting $\gamma = 2 \cdot c_2'/c_1' \log^3 t = O(\log^3 t)$, this probability becomes $2^{-c_2' \cdot t \log t} = 2^{-\Omega(t)}$. Thus, with probability $1 - 2^{-\Omega(t)}$, the collection of permutation sequences $\Gamma^{(1)}, \ldots, \Gamma^{(\gamma)}$ is (κ_1, κ_2)-non-concentrating for $\mathcal{B}$. □

Remark 5.49 (Padding the Linear PCPs)**.** Theorem 5.48 requires that the permutation Π over t-by-q matrices has the property that for every pair of rows $i_1, i_2 \in [t]$, there are at least $\Omega(t)$ indices $j \in [q]$, where $\Pi(i_1, j)$ is in row i_2. In the quasi-optimal linear MIP construction (Construction 5.35), the permutations Π on which we apply the randomized permutation decomposition may not natively satisfy this property. This problem can be addressed by defining a new permutation Π' that operates on t-by-$(q + t^2)$ matrices as follows:

- For all $i \in [t]$ and $j \leq q$, $\Pi'(i, j) := \Pi(i, j)$.
- For all $i \in [t]$ and $j > q$, $\Pi'(i, j) := (i + j \bmod t, j)$.

This construction ensures that for all pairs of rows $i_1, i_2 \in [t]$, there are at least t indices $j \in [q]$ such that $\Pi'(i_1, j)$ is in row i_2, while preserving the operation of Π in the leftmost t-by-q block of the matrix. In Construction 5.35, we would thus pad the assignment matrices $\mathbf{W}$ accordingly (with a q-by-t^2 block of 0's that are used only for the consistency checks). Padding the assignment matrices in this way increases the query dimension of the linear MIP in Construction 5.35 from $\widetilde{O}(s/t)$ to $\widetilde{O}(s/t) + t^2$, thus increasing the overall prover overhead by an *additive* factor of $\mathrm{poly}(t) = \mathrm{poly}(\lambda)$.

5.3.5 Quasi-Optimal Linear MIP Analysis

In this section, we give the formal security analysis of our linear MIP from Section 5.3.3 (Construction 5.35). First, we state and prove the main theorem on the properties satisfied by Construction 5.35. We conclude by giving the proof of Theorem 5.36.

Theorem 5.50. *Let λ be a security parameter, and let C be an arithmetic circuit over $\mathbb{F}$. Then, define parameters $t, \delta, k, \varepsilon, d, \kappa_1, \kappa_2, \alpha, \gamma, z$ as in Construction 5.35. Suppose $\delta > 0$ is a constant, $\varepsilon \leq 1/\mathrm{poly}(\lambda)$, and $\alpha \cdot \kappa_2 < (1 - \delta)/(8\rho) \cdot t$. Then, Construction 5.35 is an input-oblivious $(k + \alpha\gamma z)$-query linear MIP for the language $\mathcal{L}_C$ of arithmetic circuit satisfiability with $t \cdot (1 + \alpha\gamma z)$ provers and soundness error $2^{-\Omega(t/\rho)} + \alpha \cdot 2^{-\Omega(\kappa_1)}$.*

Proof. We show completeness and soundness of the linear MIP separately.

Completeness. Completeness follows from completeness of the underlying linear PCP systems and completeness of the consistency check procedure (Construction 5.29).

Soundness. Take any $\mathbf{x} \notin \mathcal{L}_C$, and consider the probability (taken over the randomness of query generation) that there exists a proof that the verifier accepts. For $i \in [t]$, let $\mathbf{y}_i = \mathbf{Q}_i^T \boldsymbol{\pi}_i$ denote the responses the verifier obtains from prover P_i on its linear PCP query $\mathbf{Q}_i$. We appeal to the soundness of the underlying linear PCP instances to argue that with probability $1 - 2^{-\Omega(t)}$, if the verifier accepts all of the linear PCP queries responses, then the inconsistency graph $\mathcal{G}_{\mathbf{W},\mathbf{A}}$ contains a matching of size $\Omega(t)$. Recall that $\mathbf{W}$ is the matrix with rows $\mathbf{w}_1, \ldots, \mathbf{w}_t$.

Lemma 5.51. *For any $\mathbf{x} \notin \mathcal{L}_C$, it holds that for all proofs $\boldsymbol{\pi}_1, \ldots, \boldsymbol{\pi}_t \in \mathbb{F}^d$ to the underlying linear PCP instances $(\mathcal{P}_1, \mathcal{V}_1), \ldots, (\mathcal{P}_t, \mathcal{V}_t)$, with probability $1 - 2^{-\Omega(t)}$, one of the following conditions hold:*

- *There is an $i \in [t]$ such that $\mathcal{V}_i$ rejects $\boldsymbol{\pi}_i$.*
- *If for all $i \in [t]$, $\mathcal{V}_i$ accepts $\boldsymbol{\pi}_i = [\mathbf{w}_i, \mathbf{p}_i]$, then the inconsistency graph $\mathcal{G}_{\mathbf{W},\mathbf{A}}$ contains a matching of size $\left(\frac{1-\delta}{2}\right) \cdot \frac{t}{2} = \Omega(t)$.*

Here the probability is taken over the random coins used to generate the queries for the underlying linear PCP instances $(\mathcal{P}_1, \mathcal{V}_1), \dots, (\mathcal{P}_t, \mathcal{V}_t)$. □

Proof. To show the lemma, consider the event where for all $i \in [t]$, $\mathcal{V}_i$ accepts $\boldsymbol{\pi}_i = [\mathbf{w}_i, \mathbf{p}_i]$, but there are fewer than $\left(\frac{1-\delta}{2}\right) \cdot \frac{t}{2}$ mutually disjoint pairs of indices $i, i' \in [t]$ where $\mathbf{w}_i$ and $\mathbf{w}_{i'}$ correspond to inconsistent assignments. We argue that this event occurs with probability $2^{-\Omega(t)}$ over the choice of the verifier's randomness. Let $\mathbf{w}_{i_1}, \dots, \mathbf{w}_{i_\ell}$ be a subset of $(\mathbf{w}_1, \dots, \mathbf{w}_t)$ that represents a consistent assignment to the shared inputs in $C_{i_1}, \dots, C_{i_\ell}$. There are at most $\left(\frac{1-\delta}{2}\right) \cdot \frac{t}{2}$ disjoint pairs of witnesses that are inconsistent, so there must exist a consistent subset of witnesses of size $\ell \geq t - \left(\frac{1-\delta}{2}\right) \cdot t = \left(\frac{1+\delta}{2}\right) \cdot t$. Let $\mathbf{w} \in \mathbb{F}^m$ be an assignment that is consistent with $\mathbf{w}_{i_1}, \dots, \mathbf{w}_{i_\ell}$.

Now, since $\mathbf{x} \notin \mathcal{L}_C$, by δ-robustness of the decomposition $(f_1, \dots, f_t, \mathsf{inp}, \mathsf{wit})$, at most $\delta \cdot t$ of the constraints $f_1(\mathbf{x}, \mathbf{w}), \dots, f_t(\mathbf{x}, \mathbf{w})$ can be satisfied. This means that there are at least $\ell - \delta t \geq \left(\frac{1-\delta}{2}\right) \cdot t$ indices $j \in [\ell]$ where $f_{i_j}(\mathbf{x}, \mathbf{w}) = 0$, or equivalently, $C_{i_j}(\mathbf{x}_{i_j}, \mathbf{w}_{i_j}) = 0$. Since each of the linear PCP systems used to verify C_i are systematic and have knowledge error at most ε, the probability that $\mathcal{V}_i$ accepts $\boldsymbol{\pi}_i$ when $C_i(\mathbf{x}_i, \mathbf{w}_i) = 0$ is at most ε. The linear PCP instances are independent, so

$$\Pr[\forall i \in [t] \colon \mathcal{V}_i^{\boldsymbol{\pi}_i}(\mathbf{x}_i) = 1] \leq \varepsilon^{(1-\delta)/2 \cdot t} = 2^{-\Omega(t)},$$

since there are at least $(1-\delta)/2 \cdot t$ indices i where $C(\mathbf{x}_i, \mathbf{w}_i) = 0$. Thus, with probability $1 - 2^{-\Omega(t)}$, there are at least $\left(\frac{1-\delta}{2}\right) \cdot \frac{t}{2}$ mutually disjoint pairs of indices where $\mathbf{w}_i$ and $\mathbf{w}_{i'}$ are inconsistent. Each disjoint pairs of indices contributes an edge to a matching in $\mathcal{G}_{\mathbf{W},\mathbf{A}}$, and the claim follows. □

Thus, for a false statement $\mathbf{x} \notin \mathcal{L}_C$, with probability $1 - 2^{-\Omega(t)}$, either the verifier rejects the proof or there is a matching of size $\left(\frac{1-\delta}{4}\right) \cdot t$ in the inconsistency graph $\mathcal{G}_{\mathbf{W},\mathbf{A}}$. We now show that if there exists such a matching, then at least one of the consistency checks fails with probability $1 - \alpha \cdot 2^{-\Omega(\kappa_1)} - 2^{-\Omega(t/\rho)}$.

Lemma 5.52. *Suppose there exists a matching of size $\left(\frac{1-\delta}{4}\right) \cdot t$ in $\mathcal{G}_{\mathbf{W},\mathbf{A}}$. Then, at least one of the consistency checks fails with probability $1 - \alpha \cdot 2^{-\Omega(\kappa_1)} - 2^{-\Omega(t/\rho)}$.*

Proof. Suppose there exists a matching of size $\left(\frac{1-\delta}{4}\right) \cdot t$ in $\mathcal{G}_{\mathbf{W},\mathbf{A}}$. Since $(\Pi_1, \dots, \Pi_z)$ is a collection of ρ-regularity-inducing permutations, there exists some $\beta \in [z]$ where the inconsistency graph of $\mathbf{W}_\beta = \Pi_\beta(\mathbf{W})$ with respect to $\mathbf{A}_\beta = \Pi_\beta(\mathbf{A})$ contains a *regular* matching of size $s = \left(\frac{1-\delta}{4\rho}\right) \cdot t$. Let M be the regular matching of size s in $\mathcal{G}_{\mathbf{W}_\beta,\mathbf{A}_\beta}$. For each edge $(i_1, i_2) \in M$, we associate with it two indices $j_1, j_2 \in [q]$ where $\mathbf{W}_\beta[i_1, j_1] \neq \mathbf{W}_\beta[i_2, j_2]$ but $\mathbf{A}_\beta[i_1, j_1] = \mathbf{A}_\beta[i_2, j_2]$. Note that j_1, j_2 always exists by definition of $\mathcal{G}_{\mathbf{W}_\beta,\mathbf{A}_\beta}$. Define the inconsistency matrix $\mathbf{B}_\beta \in \{0,1\}^{t \times q}$ where for each

$(i_1, i_2) \in M$, $\mathbf{B}_\beta[i_1, j_1] = 1 = \mathbf{B}_\beta[i_2, j_2]$. All other values in $\mathbf{B}_\beta$ are set to 0. Let $\mathbf{B} = \Pi_\beta^{-1}(\mathbf{B}_\beta)$. By construction, $\mathbf{B}_\beta$ has at most a single 1 in each row, and moreover, since each edge in M corresponds to an edge in a matching of the inconsistency graph $\mathcal{G}_{\mathbf{W},\mathbf{A}}$, matrix $\mathbf{B}$ also has at most a single 1 in each row. Note that even through $\mathbf{W}$ and $\mathbf{W}_\beta$ may have more inconsistent assignments than those indicated in $\mathbf{B}$ and $\mathbf{B}_\beta$, it is not necessary to consider them in the analysis.

By construction, $\mathbf{B}$ and $\Pi_\beta(\mathbf{B})$ have at most one inconsistency in each row (where an inconsistency is an entry with value 1). This means that $\mathbf{B} \in \mathcal{B}_\beta$. Next, since $\Gamma_\beta^{(1)}, \ldots, \Gamma_\beta^{(\gamma)}$ is a collection of permutation sequences that is non-concentrating for $\mathcal{B}_\beta$, there exists some $j \in [\gamma]$ where $\Gamma_\beta^{(j)} = \left(\Pi_{\beta,1}^{(j)}, \ldots, \Pi_{\beta,\alpha}^{(j)}\right)$ is non-concentrating for $\mathbf{B}$. For all $\ell \in [\alpha]$, let $\mathbf{B}_\ell = \Pi_{\beta,\ell}^{(j)}(\mathbf{B}_{\ell-1})$ where $\mathbf{B}_0 = \mathbf{B}$. The non-concentration property states that no subset of κ_1 rows of $\mathbf{B}_\ell$ contains κ_2 inconsistencies.

Consider now the sequence of consistency checks the verifier performs for the permutations $\Pi_{\beta,1}^{(j)}, \ldots, \Pi_{\beta,\alpha}^{(j)}$. By construction of the consistency check queries, the verifier's behavior precisely corresponds to performing the approximate consistency check procedure in Construction 5.29 to verify the following relations:

$$\mathbf{W}_{\beta,1}^{(j)} \approx \Pi_{\beta,1}^{(j)}(\mathbf{W}) \quad \text{and} \quad \mathbf{W}_{\beta,\ell}^{(j)} \approx \Pi_{\beta,\ell}^{(j)}(\mathbf{W}_{\beta,\ell-1}^{(j)}) \text{ for all } 1 < \ell \le \alpha.$$

Consider the first relation. By construction, the inconsistency matrix $\mathbf{B}$ encodes the positions of s pairs of inconsistent assignments in $\mathbf{W}$, and the matrix $\mathbf{B}_1$ encodes the (permuted) positions of the same s pairs of inconsistent assignments in $\Pi_{\beta,1}^{(j)}(\mathbf{W})$. We now argue that $\mathbf{W}_{\beta,1}^{(j)}$ contains at least $s - \kappa_2$ pairs of inconsistent assignments, except with probability $2^{-\Omega(\kappa_1)}$. This follows immediately from the assumption that $\Gamma_\beta^{(j)}$ is (κ_1, κ_2)-non-concentrating (Definition 5.34) and soundness of the consistency check (Lemma 5.30). In particular, by soundness of the consistency check, with probability $1 - 2^{-\Omega(\kappa_1)}$ (over the randomness of the query generation algorithm), matrices $\mathbf{W}_{\beta,1}^{(j)}$ and $\Pi_{\beta,1}^{(j)}(\mathbf{W})$ can differ on at most κ_1 rows. But since $\Gamma_\beta^{(j)}$ is (κ_1, κ_2)-non-concentrating, no subset of κ_1 rows of $\mathbf{B}_1$ contains κ_2 inconsistencies. We conclude that $\mathbf{W}_{\beta,1}^{(j)}$ must contain at least $s - \kappa_2$ pairs of inconsistent rows. Applying this argument α times, once for each permutation $\Pi_{\beta,\ell}^{(j)}$ for $\ell \in [\alpha]$, we conclude that with probability at least $1 - \alpha \cdot 2^{-\Omega(\kappa_1)}$, the number of pairs of inconsistent rows in the final matrix $\mathbf{W}_{\beta,\alpha}^{(j)}$ is at least

$$s - \alpha \cdot \kappa_2 = \left(\frac{1-\delta}{4\rho}\right) \cdot t - \alpha \cdot \kappa_2 \le \left(\frac{1-\delta}{8\rho}\right) \cdot t.$$

Finally, $\mathbf{B}_\alpha = \Pi_\beta(\mathbf{B})$ is the inconsistency matrix derived from a *regular* matching. Thus, if the final matrix $\mathbf{W}_{\beta,\alpha}^{(j)}$ contains at least $\left(\frac{1-\delta}{8\rho}\right) \cdot t$ pairs of inconsistent rows according to the inconsistency pattern $\mathbf{B}_\alpha$, then $\Pi_{\beta'}(\mathbf{W}_{\beta,\alpha}^{(j)})$ and $\mathbf{W}_{\beta,\alpha}^{(j)}$ differs on at least $\left(\frac{1-\delta}{8\rho}\right) \cdot t$ pairs of *adjacent* rows. But then, by Corollary 5.32, the probability that the verifier accepts is at most $2^{-\Omega(t/\rho)}$. Putting everything together, the probability that the verifier accepts is bounded by $\alpha \cdot 2^{-\Omega(\kappa_1)} + 2^{-\Omega(t/\rho)}$, and the claim follows. □

Combining Lemmas 5.51 and 5.52, we conclude that the verifier accepts a proof of a false statement

$\mathbf{x} \notin \mathcal{L}_C$ with probability at most $\alpha \cdot 2^{-\Omega(\kappa_1)} + 2^{-\Omega(t/\rho)}$. $\square$

Proof of Theorem 5.36. First, we describe how we instantiate each of the primitives in Construction 5.35:

- We instantiate the (t, δ)-robust decomposition using the construction from Corollary 5.21, where $t = \Theta(\lambda)$, and $\delta > 0$ is a constant. Let $(f_1, \ldots, f_t, \mathsf{inp}, \mathsf{wit})$ be the robust decomposition of C, and let $C_1, \ldots, C_t$ be the arithmetic circuits that compute $f_1, \ldots, f_t$, respectively. Each of the circuits C_i can be computed by an arithmetic circuit of size $\widetilde{O}(s/t) + \mathrm{poly}(t, \log s)$.
- We use the k-query linear PCP from Fact 5.24 to instantiate each of the linear PCP $(\mathcal{P}_i, \mathcal{V}_i)$ instances for C_i for all $i \in [t]$. In this case, $\varepsilon = 1/\mathrm{poly}(\lambda)$, $k = O(1)$, and the query length is $d = \widetilde{O}(s/t) + \mathrm{poly}(t, \log s)$.
- We use Construction 5.38 to instantiate the regularity-inducing permutations. In this case, $\rho = O(1)$ and $z = O(1)$.
- We use Construction 5.47 to instantiate the non-concentrating sequence of permutations, where we set $\kappa_1 = t/\log^5 t$ and $\kappa_2 = c \cdot t/\log t$, where the constant c is chosen so that $\kappa_2 \cdot (\log t + 2) < \frac{1-\delta}{8\rho} \cdot t$. In this case, $\alpha = \log t + 2 = \Theta(\log t)$ and $\gamma = O(\log^3 t)$.

Note that in order to argue that the sequences of permutations output by Construction 5.47 is non-concentrating (Theorem 5.48), we may additionally need to pad the query (and proof) vectors with an extra $t^2 = O(\lambda^2)$ components (Remark 5.49). Putting everything together then, we have the following:

- The number of provers in the linear MIP system is $t \cdot (1 + \alpha\gamma z) = t \cdot \mathrm{polylog}(t) = \widetilde{O}(\lambda)$.
- The query length is determined by the query length d of the underlying linear PCP instances (and any extra padding from Remark 5.49). Thus, the query length is $\widetilde{O}(s/\lambda) + \mathrm{poly}(\lambda, \log s)$.
- The total number of queries is $k + \alpha\gamma z$. Since $k = O(1)$, $\alpha = \Theta(\log t)$, $\gamma = O(\log^3 t)$, and $z = O(1)$, the total number of queries is $k + O(\log^4 t) = \mathrm{polylog}(\lambda)$.
- The prover's computation can be broken down as follows. First, the robust encoding $\mathbf{x} \leftarrow \mathsf{inp}(\mathbf{x}')$ and $\mathbf{w} \leftarrow \mathsf{wit}(\mathbf{x}', \mathbf{w}')$ can be computed by an arithmetic circuit of size $\widetilde{O}(s) + \mathrm{poly}(t, \log s)$. From Fact 5.24, each of the underlying linear PCP proofs can be computed by a circuit of size $\widetilde{O}(s/t) + \mathrm{poly}(t, \log s)$. Thus, all t proofs for each of the underlying linear PCP instances can be constructed by a circuit of size $\widetilde{O}(s) + \mathrm{poly}(t, \log s)$. Finally, permuting the entries in an assignment matrix (of size $\widetilde{O}(s) + t^2$) can be performed also by a circuit of size $\widetilde{O}(s) + \mathrm{poly}(t)$. There are a total of $\alpha\gamma z = \mathrm{polylog}(t)$ such permutations, which adds another polylogarithmic overhead to the overall prover complexity. Summing together the different contributions and

noting that $t = \Theta(\lambda)$, we conclude that the prover's algorithm can be computed by a circuit of size $\widetilde{O}(s) + \text{poly}(\lambda, \log s)$.

- The query-generation procedure can be broken down as follows. From Fact 5.24, generating the queries for the underlying linear PCP instance requires a circuit of size $\widetilde{O}(s/t)$. There are t instances, so generating all of the queries requires a circuit of size $\widetilde{O}(s)$. To perform the consistency checks, the query-generation algorithm additionally generates $\alpha\gamma z = \text{polylog}(t)$ random matrices, each of size $\widetilde{O}(s) + \text{poly}(t)$. Thus, the overall algorithm can be modeled by a circuit of size $\widetilde{O}(s) + \text{poly}(\lambda, \log s)$.
- The verifier's decision algorithm consists of checking t independent linear PCP instances, which can be computed by a circuit of size at most $O(tn)$ (Fact 5.24). In addition, the decision algorithm needs to perform $O(\alpha\gamma z) = \text{polylog}(t)$ consistency checks (Construction 5.29), each of which requires computing t linear relations. This incurs an additive cost of $\widetilde{O}(t)$. Thus, the overall cost is bounded by $\widetilde{O}(\lambda n)$.
- Finally, by Theorem 5.50, for this particular choice of parameters, the overall construction achieves soundness error

$$\alpha \cdot 2^{-\Omega(\kappa_1)} + 2^{-\Omega(t/\rho)} = (\log t + 2) \cdot 2^{-\Omega(t/\log^5 t)} + 2^{-\Omega(t)} = 2^{-\Omega(\lambda/\text{polylog}(\lambda))}.$$

We can amplify the soundness to $2^{-\lambda}$ by parallel repetition. Since we only require polylog(λ) parallel instances, this introduces an additional polylog(λ) overhead to the prover complexity and the proof size. Thus, the resulting construction remains quasi-optimal. □

5.4 Quasi-Optimal SNARGs

In this section, we formally introduce the notion of a quasi-optimal SNARG. Next, in Section 5.4.3, we show how to compile a linear MIP into a designated-verifier SNARG in the preprocessing model using a generalization of linear-only vector encryption (Section 5.4.2) to rings. Combined with our quasi-optimal linear MIP from Section 5.3, this yields a quasi-optimal designated-verifier SNARG for Boolean circuit satisfiability in the preprocessing model.

5.4.1 Defining Quasi-Optimality

In this section, we formally define our notion of a quasi-optimal SNARG. In Remark 5.55, we also describe a heuristic approach for instantiating quasi-optimal SNARGs.

Definition 5.53 (Quasi-Optimal SNARG)**.** Let $\Pi_{\mathsf{SNARG}} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ be a SNARG (Definition 4.1) for a family of Boolean circuits $\mathcal{C} = \{C_n\}_{n \in \mathbb{N}}$. Then, Π_{SNARG} is *quasi-optimal* if it achieves $2^{-\lambda}$ soundness error against provers of size 2^{λ} and satisfies the following properties:

- **Prover Complexity:** The running time of Prove is $\widetilde{O}(|C_n|) + \mathrm{poly}(\lambda, \log |C_n|)$.
- **Succinctness:** The length of the proof output by Prove is $\widetilde{O}(\lambda)$.

Next, in Lemma 5.54, we show that our notion of quasi-optimality is tight in the following sense: assuming NP does not have succinct *proofs*, any argument system for NP that provides soundness error $2^{-\lambda}$ must have proofs of length $\Omega(\lambda)$. .

Lemma 5.54. *Let $\mathcal{C} = \{C_n\}_{n \in \mathbb{N}}$ be a family of Boolean circuits for some language $\mathcal{L} = \bigcup_{n \in \mathbb{N}} \mathcal{L}_{C_n}$, where $C_n \colon \{0,1\}^n \times \{0,1\}^{m(n)} \to \{0,1\}$ for all $n \in \mathbb{N}$. Fix a soundness parameter ρ and a security parameter λ. Let $\Pi_{\mathsf{SNARG}} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ be a SNARG for $\mathcal{C}$ with soundness $2^{-\rho}$ against provers of size* $\mathrm{poly}(\lambda)$. *If $\mathcal{L}_{C_n} \not\subseteq \mathbf{DTIME}(2^{o(n)})$, then the length $\ell(\rho)$ of an argument in Π_{SNARG} is $\Omega(\rho)$.*

Proof. Let $\Pi_{\mathsf{SNARG}} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ be a SNARG for $\mathcal{L}$ with soundness error $2^{-\rho}$ against provers of size $\mathrm{poly}(\lambda)$ and argument length $\ell(\rho) \leq c\rho$ for all constants $c > 0$. Let $1/2^{\delta}$ denote the probability that there exists a statement $\mathbf{x} \notin \mathcal{L}_{\mathcal{C}_n}$ and a proof $\boldsymbol{\pi}$ such that $\mathsf{Verify}(\tau, \mathbf{x}, \boldsymbol{\pi}) = 1$, where $(\sigma, \tau) \leftarrow \mathsf{Setup}(1^{\lambda}, 1^n)$, and the probability is taken over the coins of the Setup algorithm. Since Π_{SNARG} has soundness error $2^{-\rho}$, it follows that $2^{-(\delta+\ell)} \leq 2^{-\rho}$. Otherwise, a prover with $\mathbf{x}$ hard-wired inside it can guess $\boldsymbol{\pi}$ and break soundness with probability $2^{-(\delta+\ell)}$. Equivalently, this means that $\delta + \ell \geq \rho$. Since $\ell \leq c\rho$ for all $c > 0$, this means that $\delta = \Omega(\rho)$. We use Π_{SNARG} to construct a *proof* system for $\mathcal{L}$ by concatenating $2 \cdot n/\delta$ instances of Π_{SNARG}. The length of the proofs in this new system is then $2 \cdot n\ell/\delta = o(n)$, which is succinct. Moreover, for any false statement $\mathbf{x} \in \{0,1\}^n$, the probability that there exists a proof $\boldsymbol{\pi}$ that causes the verifier to accept is now $(1/2^{\delta})^{(2n/\delta)} = 1/2^{2n}$. Taking a union bound over all 2^n possible statements, the probability that there exists any false statement with a proof that convinces the verifier is at most 2^{-n}. This yields a succinct proof system for $\mathcal{L}$ with soundness error 2^{-n}, which contradicts the assumption that $\mathcal{L}$ does not have succinct proofs. Thus, there must exist some constant $c > 0$ such that $\ell > c\rho$, from which we conclude that $\ell = \Omega(\rho)$. $\square$

Remark 5.55 (Heuristic Construction of Quasi-Optimal SNARGs)**.** One approach for constructing a quasi-optimal SNARG is to compose a SNARG that provides quasi-optimal prover complexity with one that is quasi-optimally succinct. To prove that $C(\mathbf{x}, \mathbf{w}) = 1$, the prover first constructs a proof $\boldsymbol{\pi}$ for the statement using the "inner" SNARG that provides quasi-optimal prover complexity. Then, the prover uses the "outer" SNARG that is quasi-optimally succinct to prove that it knows a proof $\boldsymbol{\pi}$ of the statement $\mathbf{x}$ under the inner SNARG. The additional cost of generating this second proof is proportional to the size of the verifier for the inner proof system, which is polylogarithmic in the size of C. Thus, this composition is quasi-optimal. Note that to show soundness of the composition, we additionally require that both SNARGs satisfies a knowledge property (namely, that they are SNARKs). However, to our knowledge, the only candidate SNARK with quasi-optimal prover efficiency is Micali's CS proofs [Mic00] in the random oracle model. Thus, composing CS proofs with a quasi-optimally succinct SNARK (e.g., the constructions from [GGPR13, BCI$^+$13] or the one from Section 4.4.2) can only yield a construction whose security is heuristic. This is because the prover in the outer SNARK needs access to the circuit description of the verification algorithm of the inner SNARK (in order to prove knowledge of an accepting proof $\boldsymbol{\pi}$), but the verification algorithm in the inner SNARK necessarily makes random oracle queries.

5.4.2 Linear-Only Vector Encryption over Rings

In Chapter 4, we showed how to compile linear PCPs into preprocessing SNARGs (Construction 4.14) using linear-only vector encryption (Definition 4.10). In this chapter, we further refine this methodology and show how to compile linear MIPs into preprocessing SNARGs using *linear-only vector encryption over rings*. Specifically, a vector encryption encryption over a ring R is an encryption scheme where the message space R^k is a vector of ring elements, where k here denotes the dimension of the vector. In our constructions, the ring $R = \mathbb{F}_p[x]/\Phi_m(x)$ is a polynomial ring (where $\Phi_m(x)$ here denote the m^{th} cyclotomic polynomial), and the parameters m and p are chosen so that R splits into $\ell = \varphi(m)$ isomorphic copies of $\mathbb{F}_p$. We now introduce the schema adapted from Definition 4.9:

Definition 5.56 (Vector Encryption Scheme over R)**.** Fix a ring R. A secret-key vector encryption scheme over R^k is a tuple of algorithms $\Pi_{\mathsf{venc}} = (\mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt})$ with the following properties:

- $\mathsf{KeyGen}(1^\lambda, 1^k) \to \mathsf{sk}$: On input the security parameter λ and the dimension k of the message space, the key-generation algorithm outputs a secret key sk.
- $\mathsf{Encrypt}(\mathsf{sk}, \mathbf{v}) \to \mathsf{ct}$: On input the secret key sk and a vector $\mathbf{v} \in R^k$, the encryption algorithm outputs a ciphertext ct.
- $\mathsf{Decrypt}(\mathsf{sk}, \mathsf{ct}) \to R^k \cup \{\perp\}$: On input the secret key sk and a ciphertext ct, the decryption algorithm either outputs a vector $\mathbf{v} \in R^k$ or a special symbol $\perp$ (to denote an invalid ciphertext).

We can view Definition 4.9 of vector encryption over $\mathbb{Z}_p^*$ as a special case of vector encryption over rings where $R = \mathbb{Z}_p^*$. We can define correctness, CPA-security (Definition 2.1), linear targeted malleability (Definition 4.11), and linear-only (Definition 4.10) for vector encryption schemes over rings analogously.

Vector encryption over polynomial rings. We now describe a candidate linear-only vector encryption over a polynomial ring R. One such candidate is the natural generalization of Construction 4.23 to the ring LWE setting. Ring LWE analogs of Regev encryption have been used previously for optimizing FHE [BGV12, GHS12, GHS12]. We describe the construction for encrypting vectors below, as well as the conjecture we require to instantiate our quasi-optimal SNARG candidate (Conjecture 5.58). We note that Conjecture 5.58 is the natural generalization of Conjecture 4.24.

Construction 5.57 (Linear-Only Vector Encryption over Rings)**.** Fix a security parameter λ, lattice parameters $m, q = \text{poly}(\lambda)$, an error distribution χ, and a cyclotomic polynomial ring $R = \mathbb{Z}[x]/\Phi_m(x)$. Let k be the plaintext dimension, and let $R_p = \mathbb{Z}_p[x]/\Phi_m(x)$ be the plaintext ring. The ciphertext ring is then $R_q = \mathbb{Z}_q[x]/\Phi_m(x)$.

- $\mathsf{Setup}(1^\lambda, 1^k) \to \mathsf{sk}$: Choose $\bar{\mathbf{a}} \xleftarrow{\text{R}} R_q$, $\mathbf{s} \xleftarrow{\text{R}} R_q^k$, and $\bar{\mathbf{e}} \leftarrow \chi^k$. Define $\mathbf{a} \in R_q^{(1+k)}$ as follows:
$$\mathbf{a} = \begin{bmatrix} \bar{\mathbf{a}} \\ \bar{\mathbf{a}}\mathbf{s} + \bar{\mathbf{e}} \end{bmatrix},$$
Output the secret key $\mathsf{sk} = (\mathbf{a}, \mathbf{s})$.
- $\mathsf{Encrypt}(\mathsf{sk}, \mathbf{v}) \to \mathbf{c}$: To encrypt a vector $\mathbf{v} \in R_p^k$, choose $\mathbf{r} \xleftarrow{\text{R}} R_2$ and output the ciphertext $\mathbf{c} \in R_q^{(1+k)}$ where
$$\mathbf{c} = \mathbf{a}\mathbf{r} + \begin{bmatrix} 0 \\ \lfloor q/p \rceil \cdot \mathbf{v} \end{bmatrix}.$$
- $\mathsf{Decrypt}(\mathsf{sk}, \mathbf{c}) \to \mathbf{v}$: Let c_1 denote the first component of $\mathbf{c}$ and let $\bar{\mathbf{c}} \in R_q^k$ be the last k components of $\mathbf{c}$. Compute and output $[[\bar{\mathbf{c}} - c_1\mathbf{s}]_q]_p$.

Conjecture 5.58 (Linear Targeted Malleability of Construction 5.57)**.** The vector encryption scheme Π_{venc} from Construction 5.57 satisfies exponentially-strong linear targeted malleability (Definition 4.11).

5.4.3 Quasi-Optimal SNARGs from Quasi-Optimal Linear MIPs

In this section, we show how to combine a linear MIPs with linear-only vector encryption over rings to obtain a quasi-optimal SNARG. We give our construction and state our security analysis below. The security proofs follow by the same argument as that used in [BCI+13, §6] and Section 4.4.2.

Construction 5.59 (SNARG from Linear MIP)**.** Fix a prime p and let $\mathcal{C} = \{C_n\}_{n\in\mathbb{N}}$ be a family of arithmetic circuits over $\mathbb{F}_p$. Let $\mathcal{R}_\mathcal{C}$ be the relation associated with $\mathcal{C}$. Let $(\mathcal{P}, \mathcal{V})$ be an input-oblivious k-query linear MIP with ℓ provers and query length d for the relation $\mathcal{R}_\mathcal{C}$. Write the verifier algorithm as $\mathcal{V} = (\mathcal{Q}, \mathcal{D})$, where $\mathcal{Q}$ is a query-generation algorithm and $\mathcal{D}$ is a decision algorithm. Let $\Pi_{\mathsf{venc}} = (\mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt})$ be a secret-key vector encryption scheme over R^k where $R \cong \mathbb{F}_p^\ell$. Our single-theorem, designated-verifier SNARG $\Pi_{\mathsf{SNARG}} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ in the preprocessing model for $\mathcal{R}_\mathcal{C}$ is given below:

- $\mathsf{Setup}(1^\lambda, 1^n) \to (\sigma, \tau)$: On input the security parameter λ and the circuit family parameter n, the setup algorithm does the following:

1. Invoke the query-generation algorithm $\mathcal{Q}$ for the linear MIP to obtain a tuple of query matrices $\mathbf{Q}_1, \ldots, \mathbf{Q}_\ell \in \mathbb{F}_p^{d \times k}$ and state information st.
2. Generate a secret key $\mathsf{sk} \leftarrow \mathsf{KeyGen}(1^\lambda, 1^\ell)$ for the vector encryption scheme.
3. Pack the ℓ query matrices $\mathbf{Q}_1, \ldots, \mathbf{Q}_\ell$ into a single query matrix $\mathbf{Q} \in R^{d \times k}$ (recall that the ring R splits into ℓ isomorphic copies of $\mathbb{F}_p$).
4. Encrypt each row of $\mathbf{Q}$ (an element of R^k) using the vector encryption scheme. In other words, for $i \in [d]$, let $\mathbf{q}_i \in R^d$ be the i^{th} row of $\mathbf{Q}$. In this step, the setup algorithm computes ciphertexts $\mathsf{ct}_i \leftarrow \mathsf{Encrypt}(\mathsf{sk}, \mathbf{q}_i)$.
5. Output the common reference string $\sigma = (\mathsf{ct}_1, \ldots, \mathsf{ct}_d)$ and the verification state $\tau = (\mathsf{sk}, \mathsf{st})$.

- $\mathsf{Prove}(\sigma, \mathbf{x}, \mathbf{w}) \to \boldsymbol{\pi}$. On input the common reference string $\sigma = (\mathsf{ct}_1, \ldots, \mathsf{ct}_d)$, a statement $\mathbf{x}$, and a witness $\mathbf{w}$, the prover's algorithm works as follows:

  1. For each $i \in [\ell]$, invoke the linear MIP prover algorithm P_i on input $\mathbf{x}$ and $\mathbf{w}$ to obtain a proof $\boldsymbol{\pi}_i \leftarrow P_i(\mathbf{x}, \mathbf{w}) \in \mathbb{F}_p^d$.
  2. Pack the ℓ proof vectors $\boldsymbol{\pi}_1, \ldots, \boldsymbol{\pi}_\ell \in \mathbb{F}_p^d$ into a single proof vector $\boldsymbol{\pi} \in R^d$. Then, viewing the ciphertexts $\mathsf{ct}_1, \ldots, \mathsf{ct}_m$ as vector encryptions of the rows of the query matrix $\mathbf{Q} \in R^{d \times k}$, homomorphically compute an encryption of the matrix-vector product $\mathbf{Q}^T \boldsymbol{\pi} \in R^k$. In particular, the prover homomorphically computes the sum $\mathsf{ct}' = \sum_{i \in d} \boldsymbol{\pi}_i \cdot \mathsf{ct}_i$.
  3. Output the proof ct'.

- $\mathsf{Verify}(\tau, \mathbf{x}, \boldsymbol{\pi}) \to \{0, 1\}$: On input the verification state $\tau = (\mathsf{sk}, \mathsf{st})$, the statement $\mathbf{x}$, and the proof $\pi = \mathsf{ct}'$, the verifier does the following:

  1. Decrypt the proof ct' using the secret key sk to obtain the prover's responses $\mathbf{y} \leftarrow \mathsf{Decrypt}(\mathsf{sk}, \mathsf{ct}')$. If $\mathbf{y} = \perp$, the verifier terminates with output 0.
  2. The verifier decomposes $\mathbf{y} \in R^k$ into vectors $\mathbf{y}_1, \ldots, \mathbf{y}_\ell \in \mathbb{F}_p^k$. It then invokes the linear MIP decision algorithm $\mathcal{D}$ on the statement $\mathbf{x}$, the responses $\mathbf{y}_1, \ldots, \mathbf{y}_\ell$, and the verification state st and outputs $\mathcal{D}(\mathsf{st}, \mathbf{x}, \mathbf{y}_1, \ldots, \mathbf{y}_\ell)$.

Theorem 5.60 ([BCI+13, Lemma 6.3, adapted])**.** *Fix a security parameter λ and a prime p. Let $\mathcal{C} = \{C_n\}_{n \in \mathbb{N}}$ be a family of arithmetic circuits over $\mathbb{F}_p$, $\mathcal{R}_\mathcal{C}$ be the relation associated with $\mathcal{C}$, and $(\mathcal{P}, \mathcal{V})$ be a k-query linear MIP with ℓ provers, query length d, and soundness error $\varepsilon(\lambda)$ against affine provers for the relation $\mathcal{R}_\mathcal{C}$. Let $\Pi_{\mathsf{venc}} = (\mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt})$ be a vector encryption scheme over a ring $R \cong \mathbb{F}_p^\ell$ with linear targeted malleability (Definition 4.11). Then, applying Construction 5.59 to $(\mathcal{P}, \mathcal{V})$ and Π_{venc} yields a non-adaptive designated-verifier preprocessing SNARG with soundness error $2 \cdot \varepsilon(\lambda) + \operatorname{negl}(\lambda)$.*

Theorem 5.61 ([BCI$^+$13, Lemma 6.2, adapted]). *Fix a security parameter λ and a prime p. Let $\mathcal{C} = \{C_n\}_{n \in \mathbb{N}}$ be a family of arithmetic circuits over $\mathbb{F}_p$, $\mathcal{R}_{\mathcal{C}}$ be the relation associated with $\mathcal{C}$, and $(\mathcal{P}, \mathcal{V})$ be a k-query linear MIP with ℓ provers, query length d, and soundness error $\varepsilon(\lambda)$ against affine provers for the relation $\mathcal{R}_{\mathcal{C}}$. Let $\Pi_{\mathsf{venc}} = (\mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt})$ be a linear-only vector encryption scheme (Definition 4.10). Then, applying Construction 5.59 to $(\mathcal{P}, \mathcal{V})$ and Π_{venc} yields an adaptive designated-verifier preprocessing SNARG with soundness error $\varepsilon(\lambda) + \mathrm{negl}(\lambda)$.*

Instantiating the construction. To conclude this section, we show that combining the candidate vector encryption scheme Π_{venc} over polynomial rings R^k, where $R \cong \mathbb{F}_p^\ell$ (Construction 5.57) with our quasi-optimal linear MIP construction from Theorem 5.36 yields a quasi-optimal SNARG from linear-only vector encryption. We first note that the vector encryption scheme Π_{venc} from Construction 5.57 has the following properties:

- When $k = \mathrm{polylog}(\lambda)$, $\ell = \widetilde{O}(\lambda)$, and $|\mathbb{F}| = \mathrm{poly}(\lambda)$, each ciphertext encrypting an element of R^k has length $\widetilde{O}(\lambda)$.
- Scalar multiplication and homomorphic addition of two ciphertexts can be performed in time $\widetilde{O}(\lambda)$.

When we apply Construction 5.59 to the linear MIP from Theorem 5.36 and Π_{venc}, the prover complexity and proof sizes are then as follows (targeting soundness error $2^{-\lambda}$):

- **Prover complexity:** The SNARG prover first invokes the underlying linear MIP prover to obtain proofs $\boldsymbol{\pi}_1, \ldots, \boldsymbol{\pi}_\ell$ for each of the $\ell = \widetilde{O}(\lambda)$ provers. From Theorem 5.36, this step requires time $\widetilde{O}(s) + \mathrm{poly}(\lambda, \log s)$, where s is the size of the circuit. To construct the proof, the prover has to perform d homomorphic operations, where $d = \widetilde{O}(s/\lambda) + \mathrm{poly}(\lambda, \log s)$ is the query length of the construction from Theorem 5.36. Since each homomorphic operation can be computed in $\widetilde{O}(\lambda)$ time, the overall prover complexity is $\widetilde{O}(s) + \mathrm{poly}(\lambda, \log s)$.
- **Proof size:** The proof in Construction 5.59 consists of a single ciphertext, which for our parameter settings, have length $\widetilde{O}(\lambda)$.

From this analysis, we obtain the following quasi-optimal SNARG instantiation:

Corollary 5.62 (Quasi-Optimal SNARGs). *Let $\mathcal{C} = \{C_n\}_{n \in \mathbb{N}}$ be a family of Boolean circuits. Under Conjecture 5.58, there exists a non-adaptive designated-verifier quasi-optimal SNARG (Definition 5.53) for the relation $\mathcal{R}_{\mathcal{C}}$ in the preprocessing model.*

Remark 5.63 (Adaptively-Secure Quasi-Optimal SNARGs). We can apply the same sparsification technique from Remark 4.29 to Construction 5.57 to obtain a vector encryption scheme that plausibly satisfies the stronger property of linear-only vector encryption (Definition 4.10). In conjunction with Construction 5.59 and Theorem 4.17, this yields an adaptive designated-verifier quasi-optimal SNARG for Boolean circuit satisfiability in the preprocessing model.

Construction 5.59 gives a construction of a *single-theorem* SNARG from any linear MIP system. In the following remark, we highlight some of the challenges in extending our construction to provide multi-theorem security.

Remark 5.64 (Difficulties in Constructing Multi-Theorem Quasi-Optimal SNARGs)**.** Construction 5.59 gives a construction of a *single-theorem* SNARG from any linear MIP system. The work of Bitansky et al. [BCI+13] as well as our construction from Chapter 4 (Theorem 4.22) show how to construct multi-theorem designated-verifier SNARGs by relying on a stronger notion of soundness at the linear PCP level coupled with a stronger interactive linear-only encryption assumption. While we could rely on the same type of cryptographic assumption as in Chapter 4, our linear MIP from Section 5.3 does not satisfy the notion of "reusable" or "strong" soundness from Definition 4.4. To recall, strong soundness says that for all proofs, the probability that the verifier accepts or that it rejects is negligibly close to 1 (where the probability is taken over the randomness used to generate the queries). In particular, whether the verifier decides to accept or reject should be *uncorrelated* with the randomness associated with its secret verification state. In our linear MIP model, we operate over a polynomial-size field, so a prover making a local change will cause the verifier's decision procedure to change with noticeable probability. This reveals information about the secret verification state, which can enable the malicious prover to break soundness. We leave it as an open problem to construct a quasi-optimal linear MIP that provides strong soundness. Such a primitive would be useful in constructing a quasi-optimal multi-theorem SNARGs.

5.5 Chapter Summary

In this chapter, we showed how to extend the framework from Chapter 4 to obtain the first candidate construction of a quasi-optimal SNARG from linear-only vector encryption over rings. To conclude, we provide a concrete comparison of our new lattice-based SNARG candidates from both Sections 4.5 and 5.4 to existing SNARGs for Boolean circuit satisfiability. The same results extend to SNARGs for arithmetic circuit satisfiability over polynomial-size fields. Among SNARGs with quasi-optimal succinctness (proof size $\widetilde{O}(\lambda)$), our construction based on standard lattices (Construction 4.14) achieves the same prover efficiency as the current state-of-the-art (GGPR [GGPR13] and BCIOP [BCI+13]). Our construction based on ideal lattices (Construction 5.59) is the first candidate SNARG from a concrete hardness assumption that achieves quasi-optimality. Moreover, both of our candidates are lattice-based, and thus, plausibly resist quantum attacks. We conclude with two interesting open problems.

Multi-theorem quasi-optimal SNARG. Can we build a multi-theorem quasi-optimal SNARG? In the case of Construction 4.14 from Chapter 4, we can achieve multi-theorem security by making a stronger cryptographic assumption (Section 4.4.3, Theorem 4.22). We currently do not know how to

Construction	**Type***	**Prover Complexity**	**Proof Size**	**Assumption**
CS Proofs [Mic00]	PV	$\widetilde{O}(s + \lambda^2)$	$\widetilde{O}(\lambda^2)$	Random Oracle
Groth [Gro10]	PV	$\widetilde{O}(s^2\lambda + s\lambda^2)$	$\widetilde{O}(\lambda)$	Knowledge of Exponent
GGPR [GGPR13]	PV	$\widetilde{O}(s\lambda)$	$\widetilde{O}(\lambda)$	
BCIOP [BCI$^+$13]† (Paillier)	DV	$\widetilde{O}(s\lambda^3)$	$\widetilde{O}(\lambda^3)$	Linear-Only Encryption
BCIOP [BCI$^+$13]† (Pairing)	PV	$\widetilde{O}(s\lambda)$	$\widetilde{O}(\lambda)$	
BCIOP [BCI$^+$13]‡ (Regev)‡	DV	$\widetilde{O}(s\lambda^2)$	$\widetilde{O}(\lambda^2)$	
Const. 4.14 (Corollary 4.25)	DV	$\widetilde{O}(s\lambda)$	$\widetilde{O}(\lambda)$	Linear-Only Vector Encryption
Const. 5.59 (Corollary 5.62)	DV	$\widetilde{O}(s)$	$\widetilde{O}(\lambda)$	

*We write "PV" to denote public verifiability and "DV" for designated verifiability.
†Instantiated using a linear PCP based on quadratic span programs [GGPR13].
‡Based on a direct instantiation of [BCI$^+$13] using Regev-based encryption.

Table 5.1: Asymptotic performance of different SNARG systems for Boolean circuit satisfiability. Here, s is the size of the circuit and λ is a security parameter guaranteeing $2^{-\lambda}$ soundness error against provers of size 2^λ. All of the schemes can be converted into an *argument of knowledge* (i.e., a SNARK)—in some cases, this requires a stronger cryptographic assumption.

construct a multi-theorem quasi-optimal SNARG (Remark 5.64).

Publicly-verifiable SNARGs from lattices. Our new lattice-based SNARG candidates from Chapter 4 as well as this chapter are all secure in the designated-verifier model. Can we build publicly-verifiable SNARGs from lattices? Note that this question is interesting even independently of the (orthogonal) goal of obtaining quasi-optimality.

Zero-knowledge SNARGs from lattices. Can we build zero-knowledge SNARGs from lattices with the same (or better) asymptotic complexity as the pairing-based candidates? While we can apply the general techniques from Bitansky et al. [BCI$^+$13] to our SNARG constructions to obtain a lattice-based zero-knowledge SNARG, the resulting SNARGs are much longer. In particular, to apply the Bitansky et al. approach, we need a circuit-private homomorphic encryption scheme. While we can use Gentry's "noise flooding" [Gen09a] technique to achieve this, doing so increases the length of the proofs by a multiplicative factor proportional to the statistical security parameter. As a result, the resulting SNARGs are much longer, and in particular, *not* quasi-optimally succinct.

Chapter 6

The Power of Optimally-Laconic Arguments

In this chapter, we explore what happens when we push succinctness to the limit in the context of argument systems. In particular, we consider the notion of an *optimally-succinct* argument, and show an intriguing connection between optimally-succinct arguments and powerful forms of encryption (i.e., witness encryption [GGSW13]). In fact, our results extends to the setting of two-round *laconic* arguments (i.e., two-round argument systems where the total communication from the prover to the verifier is succinct—the communication from the verifier to the prover can be long).

Optimally-laconic arguments and SNARGs. In Chapters 4 and 5, we constructed SNARGs that achieved soundness error $2^{-\lambda}$ against provers of size 2^{λ}, where λ was a concrete security parameter. But in general, it is not essential to tie the soundness error to the security parameter, and we can ask the question of what is the minimal proof length needed to achieve soundness error $2^{-\rho}$ (against $2^{-\lambda}$ bounded provers). Here, ρ is a separate soundness parameter.

Lemma 5.54 shows that assuming NP does not have succinct proof systems, achieving $2^{-\rho}$ soundness error requires proofs of length $\Omega(\rho)$. Thus, when $\rho = \Omega(\lambda)$, many existing SNARG candidates (c.f., Table 5.1) are quasi-optimally succinct. However, if we are satisfied with soundness error $\rho = o(\lambda)$ against 2^{λ}-bounded provers, the size of the proof for all of the construction in Table 5.1 remains $\Omega(\lambda)$—this is needed to ensure security against an adversary of size 2^{λ}. A natural question to ask is whether there exist SNARGs where the proof length achieves the lower bound of $\Omega(\rho)$ for providing ρ bits of soundness. Taken to the extreme, we ask whether there exists a 1-bit SNARG with soundness error $1/2 + \mathrm{negl}(\lambda)$. Observe that a 1-bit SNARG implies an *optimally-succinct* SNARG for all soundness parameters ρ: namely, to build a SNARG with soundness error $2^{-\rho}$, concatenate ρ independent instances of a 1-bit SNARG. This question of constructing optimally-succinct SNARGs is interesting even independently of the goal of minimizing prover complexity.

More broadly, we can view a quasi-optimally succinct SNARG in the preprocessing model as a two-round *interactive* argument system with a maximally *laconic* prover. In the setting of (two-round) laconic argument systems, the verifier is allowed to send an arbitrarily long string (namely, the CRS) that can depend on the statement being proved, and the goal is to minimize the number of bits the prover communicates to the verifier. An optimally-succinct SNARG implies a an optimally-succinct two-round laconic argument. Similarly, a quasi-optimal SNARG implies a quasi-optimal two-round laconic argument (for an analogous notion of quasi-optimality).

Constructing laconic arguments. In Section 6.1, we show that the designated-verifier analog of the Sahai-Waters [SW14] construction of NIZK proofs from indistinguishability obfuscation [BGI+01, GGH+13] and one-way functions is a 1-bit SNARG. Then, in Section 6.1.3, we show that in the *interactive* two-round setting, we can construct 1-bit laconic arguments from witness encryption [GGSW13]. We do not know how to build 1-bit SNARGs and 1-bit laconic arguments for general NP languages from weaker assumptions,[1] and leave this as an open problem.

The power of optimally-laconic arguments. Finally, we show an intriguing connection between 1-bit laconic arguments and a variant of witness encryption. Briefly, a witness encryption scheme [GGSW13] allows anyone to encrypt a message m with respect to a statement $\mathbf{x}$ in an NP language; then, anyone who holds a witness $\mathbf{w}$ for $\mathbf{x}$ is able to decrypt the ciphertext. In Section 6.2, we show that a 1-bit laconic argument (or SNARG) for a cryptographically-hard[2] language $\mathcal{L}$ implies a relaxed form of witness encryption for $\mathcal{L}$ where semantic security holds for messages encrypted to a *random* false instance (as opposed to an arbitrary false instance in the standard definition). While this is a relaxation of the usual notion of witness encryption, it already suffices to realize some of the powerful applications of witness encryption described in [GGSW13]. This implication thus demonstrates the power of optimally-laconic arguments, as well as some of the potential challenges in constructing them from simple assumptions.

Our construction of witness encryption from 1-bit arguments relies on the observation that for a (random) false statement $\mathbf{x}$, any computationally-bounded prover can only produce a valid proof $\pi \in \{0,1\}$ with probability that is negligibly close to $1/2$. Thus, the proof π can be used to hide the message m in a witness encryption scheme (when encrypting to the statement $\mathbf{x}$). Here, we implicitly assume that a (random) statement $\mathbf{x}$ has exactly one accepting proof—this assumption holds for any cryptographically-hard language. Essentially, our construction shows how to leverage the soundness property of a proof system to obtain a secrecy property in an encryption scheme. Previously, Applebaum et al. [AIK10] showed how to leverage secrecy to obtain soundness, so in

[1]Note that for some special languages such as graph non-isomorphism, we do have 1-bit laconic arguments [Gol01].

[2]Here, we say a language is cryptographically-hard if there exists a distribution over YES instances that is computationally indistinguishable from a distribution of NO instances for the language.

some sense, we can view our construction as a dual of their secrecy-to-soundness construction. The recent work of Berman et al. [BDRV17] also showed how to obtain public-key encryption from laconic *zero-knowledge* arguments. While their construction relies on the additional assumption of zero-knowledge, their construction does not require the argument system be optimally laconic.

We can also view a 1-bit argument for a cryptographically-hard language as a "predictable argument" (c.f., [FNV17]). A predictable argument is one where there is exactly one accepting proof for any statement. Faonio et al. [FNV17] show that any predictable argument gives a witness encryption scheme. In this work, we show that soundness *alone* suffices for this transformation, provided we make suitable restrictions on the underlying language. We discuss this in greater detail in Remark 6.15.

6.1 Optimally-Succinct SNARGs and Laconic Arguments

Recall that a "1-bit SNARG" is a SNARG that achieves soundness error $1/2 + \text{negl}(\lambda)$ with just a *single* bit of proof. In this section, we show that a variant of the Sahai-Waters NIZK construction from indistinguishability obfuscation (and one-way functions) [SW14] can be used to construct a 1-bit SNARG. As discussed at the beginning of this chapter, we can also view a 1-bit SNARG as a 1-bit laconic interactive argument system. Thus, our results also gives the first 1-bit laconic argument system for NP assuming indistinguishability obfuscation and one-way functions (Section 6.1.2). Then, in Section 6.1.3, we show that in the interactive setting, we can also build 1-bit laconic arguments from witness encryption.

6.1.1 Indistinguishability Obfuscation and Puncturable PRFs

In this section, we review the definitions of indistinguishability obfuscation and puncturable PRFs.

Definition 6.1 (Indistinguishability Obfuscation [BGI+01, GGH+13])**.** An indistinguishability obfuscator $i\mathcal{O}$ for a circuit family $\mathcal{C} = \{\mathcal{C}_\lambda\}_{\lambda \in \mathbb{N}}$ is a uniform and efficient algorithm satisfying the following requirements:

- **Correctness.** For all $\lambda \in \mathbb{N}$, all circuits $C \in \mathcal{C}_\lambda$, and all inputs x, we have that

$$\Pr[C' \leftarrow i\mathcal{O}(C) : C'(x) = C(x)] = 1.$$

- **Indistinguishability.** For all $\lambda \in \mathbb{N}$, and any two circuits $C_0, C_1 \in \mathcal{C}_\lambda$, if $C_0(x) = C_1(x)$ for all inputs x, then for all efficient adversaries $\mathcal{A}$, we have that

$$|\Pr[\mathcal{A}(i\mathcal{O}(C_0)) = 1] - \Pr[\mathcal{A}(i\mathcal{O}(C_1)) = 1]| = \text{negl}(\lambda).$$

Definition 6.2 (Puncturable PRFs [BW13, KPTZ13, BGI14])**.** A puncturable pseudorandom function with key-space $\mathcal{K}$, domain $\mathcal{X}$, and range $\mathcal{Y}$ is an efficiently computable function $\mathsf{F}\colon \mathcal{K}\times\mathcal{X} \to \mathcal{Y}$ with three additional algorithms $(\mathsf{F.Setup}, \mathsf{F.Puncture}, \mathsf{F.Eval})$ defined as follows:

- $\mathsf{F.Setup}(1^\lambda) \to k$: On input the security parameter λ, the setup algorithm outputs a PRF key $k \in \mathcal{K}$.
- $\mathsf{F.Puncture}(k, x^*) \to k_{x^*}$: On input the PRF key $k \in \mathcal{K}$ and a point $x^* \in \mathcal{X}$, the puncture algorithm outputs a punctured key k_{x^*}.
- $\mathsf{F.Eval}(k_{x^*}, x) \to y$: On input a punctured key k_{x^*}, the evaluation algorithm outputs a value $y \in \mathcal{Y} \cup \{\perp\}$.

Moreover F satisfies the following properties:

- **Correctness:** For all $x^* \in \mathcal{X}$ and $x \neq x^*$, and letting $k \leftarrow \mathsf{F.Setup}(1^\lambda)$, $k_{x^*} \leftarrow \mathsf{F.Puncture}(k, x^*)$, we have that
$$\Pr[\mathsf{F.Eval}(k_{x^*}, x) = \mathsf{F}(k, x)] = 1.$$
- **Pseudorandom at punctured points:** For all efficient adversaries $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, and letting $k \leftarrow \mathsf{F.Setup}(1^\lambda)$, $(\mathsf{st}, x^*) \leftarrow \mathcal{A}_1^{\mathsf{F}(k,\cdot)}(1^\lambda)$, $k_{x^*} \leftarrow \mathsf{F.Puncture}(k, x^*)$, and $y \xleftarrow{\text{R}} \mathcal{Y}$, we have that
$$\left|\Pr\left[\mathcal{A}_2^{\mathsf{F}(k,\cdot)}(\mathsf{st}, k_{x^*}, \mathsf{F}(k, x^*))\right] - \Pr\left[\mathcal{A}_2^{\mathsf{F}(k,\cdot)}(\mathsf{st}, k_{x^*}, y)\right]\right| = \operatorname{negl}(\lambda),$$
provided that $\mathcal{A}_1$ and $\mathcal{A}_2$ do not query $\mathsf{F}(k,\cdot)$ on the challenge point x^*.

6.1.2 1-Bit SNARGs from Indistinguishability Obfuscation

In this section, we show how to construct 1-bit SNARGs from indistinguishability obfuscation ($i\mathcal{O}$). Our construction is essentially the Sahai-Water NIZK [SW14] specialized to the designated-verifier setting. The CRS is an obfuscated program that takes as input a statement $\mathbf{x}$ and a witness $\mathbf{w}$, and outputs a 1-bit PRF on $\mathbf{x}$ if $C(\mathbf{x}, \mathbf{w}) = 1$, and $\perp$ otherwise. The PRF key is hard-coded inside the obfuscated program. The secret verification key is the PRF key. We can essentially view the CRS as an obfuscated program that checks whether $(\mathbf{x}, \mathbf{w})$ is a satisfying assignment, and if so, outputs a 1-bit message authenticated code (MAC) on the statement $\mathbf{x}$. We now give the construction and its security analysis.

Construction 6.3 (1-Bit SNARG from $i\mathcal{O}$)**.** Let $\mathcal{C} = \{C_n\}_{n\in\mathbb{N}}$ be a family of Boolean circuits, where each $C_n\colon \{0,1\}^n \times \{0,1\}^{m(n)} \to \{0,1\}$ for all $n \in \mathbb{N}$. Let $\mathcal{R}_\mathcal{C}$ be the associated relation and $\mathcal{L}_\mathcal{C}$ be the associated language. Let $\mathsf{F}_n\colon \mathcal{K}_n \times \{0,1\}^n \to \{0,1\}$ be a puncturable PRF family (indexed by a parameter n). We construct a 1-bit designated-verifier SNARG $\Pi_{\mathsf{SNARG}} = (\mathsf{KeyGen}, \mathsf{Prove}, \mathsf{Verify})$ in the preprocessing model for $\mathcal{R}_\mathcal{C}$ as follows:

- $\mathsf{Setup}(1^\lambda, 1^n)$: The setup algorithm samples a puncturable PRF key $k \leftarrow \mathsf{F}_n.\mathsf{Setup}(1^\lambda)$, and constructs the obfuscated program $P \leftarrow i\mathcal{O}(\mathsf{Prove}[C_n, k])$,[3] where the program $\mathsf{Prove}[C_n, k]$ is defined as follows:

Constants: a circuit $C_n \colon \{0,1\}^n \times \{0,1\}^{m(n)} \to \{0,1\}$ and a key k for F_n

On input $\mathbf{x} \in \{0,1\}^n, \mathbf{w} \in \{0,1\}^{m(n)}$:

1. If $C(\mathbf{x}, \mathbf{w}) = 1$, then output $\mathsf{F}_n(k, \mathbf{x})$. Otherwise, output $\perp$.

Figure 6.1: The program $\mathsf{Prove}[C_n, k]$.

The setup algorithm outputs the common reference string $\sigma = P$ and the verification state $\tau = k$.

- $\mathsf{Prove}(\sigma, \mathbf{x}, \mathbf{w}) \to \pi$: On input the common reference string $\sigma = P$ a statement $\mathbf{x} \in \{0,1\}^n$ and a witness $\mathbf{w} \in \{0,1\}^m$, the prover runs P on $(\mathbf{x}, \mathbf{w})$ to obtain a proof $\pi \leftarrow P(\mathbf{x}, \mathbf{w})$ and outputs $\pi \in \{0,1\}$.

- $\mathsf{Verify}(\tau, \mathbf{x}, \pi) \to \{0,1\}$: On input the secret verification state $\tau = k$, a statement $\mathbf{x} \in \{0,1\}^n$, and a proof $\pi \in \{0,1\}$, the verifier outputs 1 if $\pi = \mathsf{F}_n.\mathsf{Eval}(k, \mathbf{x})$, and 0 otherwise.

Theorem 6.4. *Suppose F_n is a family of puncturable PRFs and $i\mathcal{O}$ is an indistinguishability obfuscator. Then, Construction 6.3 is a non-adaptive designated-verifier 1-bit SNARG for $\mathcal{C} = \{C_n\}_{n\in\mathbb{N}}$ in the preprocessing model. In particular, Π_{SNARG} achieves soundness error $1/2 + \mathrm{negl}(\lambda)$ against polynomial-time bounded provers.*[4]

Proof. Completeness of the construction follows immediately by correctness of the indistinguishability obfuscator, so it suffices to show soundness. Take any statement $\mathbf{x}^* \notin \mathcal{L}_{C_n}$. We show that no efficient prover can produce a proof $\pi^* \in \{0,1\}$ where $\mathsf{Verify}(\tau, \mathbf{x}^*, \pi^*) = 1$ except with probability $1/2 + \mathrm{negl}(\lambda)$ over the randomness used to sample $(\sigma, \tau) \leftarrow \mathsf{Setup}(1^\lambda, 1^n)$. Our proof follows the same structure as the corresponding proof in [SW14, Theorem 9]. In particular, we define a sequence of hybrid arguments:

- Hyb_0: This is the real game, where the challenger generates $(\sigma, \tau) \leftarrow \mathsf{Setup}(1^\lambda, 1^n)$, and gives σ to the prover. The prover outputs $\pi^* \in \{0,1\}$. The output of the experiment is 1 if $\mathsf{Verify}(\tau, \mathbf{x}^*, \boldsymbol{\pi}^*) = 1$, and 0 otherwise.

[3]Note that we pad the program $\mathsf{Prove}[C_n, k]$ to the maximum size of any program that appears in the proof of Theorem 6.4.

[4]If we make the stronger assumption that the underlying primitives (the indistinguishability obfuscator and the puncturable PRF family) are secure against subexponential-time adversaries, then we correspondingly achieve soundness error $1/2 + \mathrm{negl}(\lambda)$ against all subexponential-time provers.

- Hyb_1: This is the same game as Hyb_0, except during the setup algorithm, the challenger first computes $k_{\mathbf{x}^*} \leftarrow \mathsf{F}_n.\mathsf{Puncture}(k, \mathbf{x}^*)$. When constructing the obfuscated program P, the challenger replaces the invocation $\mathsf{F}_n(k, \mathbf{x})$ with $\mathsf{F}_n.\mathsf{Eval}(k_{\mathbf{x}^*}, \mathbf{x})$.
- Hyb_2: This is the same game as Hyb_1, except in the verification procedure, the challenger samples $b \xleftarrow{\text{R}} \{0,1\}$ and accept if $\pi^* = b$.

We now argue that each pair of consecutive hybrid experiments is computationally indistinguishable.

- Hybrids Hyb_0 and Hyb_1 are computationally indistinguishable by security of $i\mathcal{O}$ and correctness of the puncturable PRF family. In particular, by correctness of the puncturable PRF, $\mathsf{F}_n(k, \cdot)$ and $\mathsf{F}_n.\mathsf{Eval}(k_{\mathbf{x}^*}, \cdot)$ agree on all inputs $\mathbf{x} \in \{0,1\}^n$ where $\mathbf{x} \neq \mathbf{x}^*$. Moreover, the programs P in Hyb_0 and Hyb_1 never needs to evaluate the PRF at $\mathbf{x}^*$, since by assumption, for all $\mathbf{w} \in \{0,1\}^m$, $C(\mathbf{x}^*, \mathbf{w}) = 0$. This means that the outputs of P in Hyb_0 and Hyb_1 are identical on all inputs $(\mathbf{x}, \mathbf{w})$. Indistinguishability then follows by security of $i\mathcal{O}$.
- Hybrids Hyb_1 and Hyb_2 are computationally indistinguishable by security of the punctured PRF. Concretely, suppose there is an adversary $\mathcal{A}$ that can distinguish the outputs of Hyb_2 and Hyb_3. We can build an adversary $\mathcal{B}$ that breaks the puncturing security of F_n as follows. Algorithm $\mathcal{B}$ submits $\mathbf{x}^*$ as the punctured point to the puncturing security challenger and receives a punctured key $k_{\mathbf{x}^*}$ and a challenge value $y \in \{0,1\}$. Algorithm $\mathcal{B}$ constructs the obfuscated program P as in Hyb_1 and Hyb_2 using the punctured key $k_{\mathbf{x}^*}$, and gives $\sigma = P$ to adversary $\mathcal{A}$. At the end of the experiment, after $\mathcal{A}$ output $\pi^* \in \{0,1\}$, $\mathcal{B}$ outputs 1 if $y = \pi^*$ and 0 otherwise. By construction, $\mathcal{B}$ perfectly simulates the output distribution of Hyb_1 if $y = \mathsf{F}_n(k, \mathbf{x}^*)$ is pseudorandom and the output distribution of Hyb_2 if y is truly random. We conclude that by puncturing security of F_n, hybrids Hyb_1 and Hyb_2 are computationally indistinguishable.

To conclude the proof, we note that the output distribution of Hyb_2 is 1 with probability $1/2$ (since y is uniform and independent of the prover's view). Since Hyb_0 and Hyb_2 are computationally indistinguishable, this means that the output of Hyb_0 is 1 with probability at most $1/2 + \text{negl}(\lambda)$. We conclude that Π_{SNARG} provides soundness error $1/2 + \text{negl}(\lambda)$. $\square$

Remark 6.5 (Additional Properties)**.** By the same argument in [SW14, Theorem 8], the 1-bit SNARG in Construction 6.3 is perfect zero-knowledge. Moreover, by a standard hybrid argument, we can show that it is non-adaptively reusable in the following sense. For any set of statements $\mathbf{x}_1, \ldots, \mathbf{x}_k \notin \mathcal{L}_{C_n}$, the probability that a malicious prover can produce proofs $\boldsymbol{\pi}_1, \ldots, \boldsymbol{\pi}_k$ such that $\mathcal{V}(\tau, \mathbf{x}_i, \boldsymbol{\pi}_i) = 1$ for all $i \in [k]$ and $(\sigma, \tau) \leftarrow \mathsf{Setup}(1^\lambda, 1^n)$ is bounded by $1/2^k + \text{negl}(\lambda)$.

Remark 6.6 (Adaptivity via VBB Obfuscation)**.** If we replace the indistinguishability obfuscator in Construction 6.3 with a VBB obfuscator [BGI+01], then it is straightforward to prove that the resulting construction is adaptively sound. We leave it as an open problem to construct adaptively sound 1-bit SNARGs without relying on such strong forms of obfuscation.

6.1.3 1-Bit Laconic Arguments and Witness Encryption

A 1-bit SNARG immediately implies a 2-round laconic argument where the prover communicates just a *single* bit. Namely, in the first round, the verifier runs the setup algorithm for the 1-bit SNARG, and sends the CRS to the prover. The prover's response consists of the 1-bit SNARG proof for the statement. Thus, Theorem 6.4 shows that assuming the existence of indistinguishability obfuscation and one-way functions, there exists a 1-bit laconic argument for NP.

While we do not know of alternative constructions of 1-bit SNARGs from weaker assumptions, such constructions are possible in the interactive setting. Importantly, in the interactive setting, the verifier's initial message (i.e., the Setup algorithm) can depend on the statement $\mathbf{x}$, while in the standard non-interactive setting, the setup algorithm (that generates the CRS) is independent of the statement. We now show how to construct a 2-round laconic argument for NP from any semantically-secure witness encryption scheme for NP [GGSW13]. Recall that in a witness encryption scheme for an NP relation $\mathcal{R}$ (and associated language $\mathcal{L}$), the encrypter can encrypt a message m with respect to a statement $\mathbf{x}$. Then, anyone who knows a witness $\mathbf{w}$ such that $(\mathbf{x}, \mathbf{w}) \in \mathcal{R}$ is able to decrypt. The security guarantee states that ciphertexts encrypted to a statement $\mathbf{x} \notin \mathcal{L}$ are semantically secure. We review the definition of witness encryption below, and then describe our 2-round laconic argument construction from witness encryption.

Definition 6.7 (Witness Encryption [GGSW13])**.** A witness encryption for an NP language $\mathcal{L}$ (with corresponding NP relation $\mathcal{R}$) is a tuple of algorithms $\Pi_{\mathsf{WE}} = (\mathsf{Encrypt}, \mathsf{Decrypt})$ with the following properties:

- $\mathsf{Encrypt}(1^\lambda, \mathbf{x}, m) \to \mathsf{ct}$: On input the security parameter λ, a statement $\mathbf{x}$ and a message $m \in \{0,1\}$, the encryption algorithm outputs a ciphertext ct.
- $\mathsf{Decrypt}(\mathsf{ct}, \mathbf{w}) \to m'$: On input a ciphertext ct and a witness $\mathbf{w}$, the decryption algorithm outputs a message $m' \in \{0,1\} \cup \{\bot\}$.

Moreover, Π_{WE} must satisfy the following properties:

- **Correctness:** For all messages $m \in \{0,1\}$, and any statement-witness pair $(\mathbf{x}, \mathbf{w})$ where $R(\mathbf{x}, \mathbf{w}) = 1$, it follows that

$$\Pr[\mathsf{Decrypt}(\mathsf{Encrypt}(1^\lambda, \mathbf{x}, m), \mathbf{w}) = m] = 1.$$

- **Semantic Security:** For all efficient adversaries $\mathcal{A}$, and all statements $\mathbf{x} \notin \mathcal{L}$,

$$\left|\Pr[\mathcal{A}(\mathsf{Encrypt}(1^\lambda, \mathbf{x}, 0)) = 1] - \Pr[\mathcal{A}(\mathsf{Encrypt}(1^\lambda, \mathbf{x}, 1)) = 1]\right| = \operatorname{negl}(\lambda). \tag{6.1}$$

Remark 6.8 (Equivalent Security Notion)**.** In our analysis, it will often be easier to work with the following equivalent notion of security for witness encryption:

- **Unguessable:** For all efficient adversaries $\mathcal{A}$, and all statements $\mathbf{x} \notin \mathcal{L}$,

$$\left| \Pr[m \xleftarrow{\text{R}} \{0,1\} : \mathcal{A}(\mathsf{Encrypt}(1^\lambda, \mathbf{x}, m)) = m] - \frac{1}{2} \right| = \text{negl}(\lambda). \tag{6.2}$$

To see the equivalence, take any adversary $\mathcal{A}$. Without loss of generality, assume that $\mathcal{A}$ always outputs a bit. Let $p_0 = \Pr[\mathcal{A}(\mathsf{Encrypt}(1^\lambda, \mathbf{x}, 0)) = 0]$ and $p_1 = \Pr[\mathcal{A}(\mathsf{Encrypt}(1^\lambda, \mathbf{x}, 1)) = 1]$. Then, the guessing advantage (Eq. (6.2)) of $\mathcal{A}$ is $|(1 - p_0 - p_1)/2|$, and the distinguishing advantage (Eq. (6.1)) of $\mathcal{A}$ is $|1 - p_0 - p_1|$. In particular, this means that if the guessing advantage of $\mathcal{A}$ is ε, then the distinguishing advantage of $\mathcal{A}$ is 2ε.

Construction 6.9 (1-Bit Laconic Argument for NP from Witness Encryption)**.** Let $\mathcal{L}$ be an NP language, and let $\Pi_{\mathsf{WE}} = (\mathsf{Encrypt}, \mathsf{Decrypt})$ be a witness encryption scheme for $\mathcal{L}$. We construct an interactive 1-bit laconic argument system $\Pi_{\mathsf{arg}} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ for $\mathcal{L}$ as follows:[5]

- $\mathsf{Setup}(1^\lambda, \mathbf{x}) \to (\sigma_{\mathbf{x}}, \tau_{\mathbf{x}})$: On input the security parameter λ and a statement $\mathbf{x}$, the setup algorithm chooses a random message $m \xleftarrow{\text{R}} \{0,1\}$ and computes the ciphertext $\mathsf{ct} \leftarrow \mathsf{Encrypt}(1^\lambda, \mathbf{x}, m)$. It outputs the initial message $\sigma_{\mathbf{x}} = \mathsf{ct}$ and the verification state $\tau_{\mathbf{x}} = m$.
- $\mathsf{Prove}(\sigma_{\mathbf{x}}, \mathbf{w}) \to \pi$: On input the verifier's initial message $\sigma_{\mathbf{x}} = \mathsf{ct}$ and a witness $\mathbf{w}$, the prover computes $m' \leftarrow \mathsf{Decrypt}(\mathsf{ct}, \mathbf{w})$, and outputs the proof $\pi = m'$.
- $\mathsf{Verify}(\tau_{\mathbf{x}}, \pi) \to \{0,1\}$: On input the verification state $\tau = m$ and the proof $\pi = m'$, the verifier outputs 1 if $m = m'$, and 0 otherwise.

Theorem 6.10 (Laconic Arguments from Witness Encryption)**.** *Let $\mathcal{L}$ be an NP language. If Π_{WE} is a witness encryption scheme for $\mathcal{L}$, then Construction 6.9 is a 1-bit laconic argument for $\mathcal{L}$.*

Proof. Completeness follows from correctness of the witness encryption scheme. Soundness follows from security of the witness encryption scheme. Namely, if $\mathbf{x} \notin \mathcal{L}$, then ct is a semantically-secure encryption of the message $m \in \{0,1\}$. Since m is sampled uniformly at random, the probability that the adversary outputs m' where $m' = m$ is at most $1/2 + \text{negl}(\lambda)$. □

This construction shows that any witness encryption for a language $\mathcal{L}$ yields a 1-bit laconic interactive argument system for the same language $\mathcal{L}$. It is unclear how to leverage this construction to construct a 1-bit preprocessing SNARG (critically, the verifier's message is *not* oblivious, and depends on the underlying statement). We leave it as an open construction to construct 1-bit SNARGs from witness encryption or other weaker assumptions. Along those same lines, it is also interesting to construct 1-bit laconic interactive arguments from weaker assumptions.

[5]We use the same schema as a SNARG (Definition 4.1) to describe our 2-round interactive argument. The main difference is that the verifier's first message (i.e., the output of the Setup algorithm) can depend on the statement $\mathbf{x}$.

6.2 Witness Encryption from 1-Bit Laconic Arguments

In Section 6.1.3, we showed how to use witness encryption for NP to build a 1-bit laconic argument for NP. In this section, we show that a variant of the converse holds: namely, a 1-bit argument system for a "cryptographically-hard" language implies a relaxed notion of witness encryption for the same language. While the notion of witness encryption we obtain is weaker than the traditional one, we show that it still suffices for instantiating some of the main applications of witness encryption (c.f., Section 6.2.1). In our weaker variant of witness encryption (Definition 6.11), semantic security is only required to hold when encrypting to a *randomly* sampled statement $\mathbf{x} \notin \mathcal{L}$ rather than any statement $\mathbf{x} \notin \mathcal{L}$.

Our results in this section are conceptually similar to those of Faonio et al. [FNV17], who previously showed how to construct witness encryption from any *predictable* argument system. In our setting, we do not impose any additional restriction on the underlying argument system. Instead, we show that for a class of "cryptographically-hard" languages, soundness of an optimally laconic argument alone already implies a "predictability" property, which suffices to give our relaxed variant of witness encryption We discuss the connection between our 1-bit arguments for cryptographically-hard languages and the notion of predictable arguments from [FNV17] in greater detail in Remark 6.15. We now formally introduce our "distributional notion" of witness encryption and then show how to realize it from any 1-bit laconic argument system.

Definition 6.11 (Distributional Witness Encryption)**.** Fix a parameter $n \in \mathbb{N}$. Let $\mathcal{L} \subseteq \{0,1\}^n$ be an NP language, and let $\mathcal{D}$ be a probability distribution over $\{0,1\}^n \setminus \mathcal{L}$. A distributional witness encryption scheme for $\mathcal{L}$ with respect to $\mathcal{D}$ is a tuple of algorithms $\Pi_{\mathsf{WE}} = (\mathsf{Encrypt}, \mathsf{Decrypt})$ with the same properties and requirements as Definition 6.7, except the semantic security requirement is replaced by a weaker $\mathcal{D}$-semantic security requirement:

- **$\mathcal{D}$-Semantic Security:** For all efficient adversaries $\mathcal{A}$ and $\mathbf{x} \leftarrow \mathcal{D}$

$$\left|\Pr[\mathcal{A}(\mathsf{Encrypt}(1^\lambda, \mathbf{x}, 0)) = 1] - \Pr[\mathcal{A}(\mathsf{Encrypt}(1^\lambda, \mathbf{x}, 1)) = 1]\right| = \mathrm{negl}(\lambda),$$

  where the probability is taken over the randomness of sampling $\mathbf{x}$, the encryption randomness, as well as the adversary's randomness.

As described in Remark 6.8, we can replace $\mathcal{D}$-semantic security with an equivalent notion of $\mathcal{D}$-unguessability.

Construction 6.12 (Distributional Witness Encryption from Laconic Argument System)**.** Fix a parameter n. Let $\mathcal{L} \subseteq \{0,1\}^n$ be an NP language and let $\mathcal{D}$ be a distribution over $\{0,1\}^n \setminus \mathcal{L}$. Let $\Pi_{\mathsf{arg}} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ be a 1-bit laconic argument for $\mathcal{L}$. We construct a distributional witness encryption scheme $\Pi_{\mathsf{WE}} = (\mathsf{Encrypt}, \mathsf{Decrypt})$ for $\mathcal{L}$ with respect to $\mathcal{D}$ as follows:

- $\mathsf{Encrypt}(1^\lambda, \mathbf{x}, m)$: On input the security parameter λ, a statement $\mathbf{x}$, and a message $m \in \{0,1\}$, the encryption algorithm samples parameters $(\sigma_\mathbf{x}, \tau_\mathbf{x}) \leftarrow \mathsf{Setup}(1^\lambda, \mathbf{x})$, and computes $\alpha_0 \leftarrow \mathsf{Verify}(\tau_\mathbf{x}, 0)$ and $\alpha_1 \leftarrow \mathsf{Verify}(\tau_\mathbf{x}, 1)$. Then, it does the following:
  - If $\alpha_0 = 1 = \alpha_1$, then the encryption algorithm outputs the message m in the clear.
  - If $\alpha_0 = 0 = \alpha_1$, then the encryption algorithm outputs $\perp$.
  - Otherwise, the encryption algorithm outputs $(\sigma_\mathbf{x}, m \oplus b)$ where $b \in \{0,1\}$ is such that $\alpha_b = 1$.
- $\mathsf{Decrypt}(\mathsf{ct}, \mathbf{w}) \to \{0,1\} \cup \perp$. If $\mathsf{ct} = 0$, $\mathsf{ct} = 1$, or $\mathsf{ct} = \perp$, then the decryption algorithm outputs ct. Otherwise, it parses $\mathsf{ct} = (\sigma_\mathbf{x}, \beta)$, computes $b \leftarrow \mathsf{Prove}(\sigma_\mathbf{x}, \mathbf{w})$, and outputs $\beta \oplus b$.

Next, we show that Construction 6.12 gives a distributional witness encryption scheme for any language that is "cryptographically-hard." Intuitively, we say that an NP language $\mathcal{L}$ is cryptographically-hard if there exists a distribution $\mathcal{D}_{\text{YES}}$ over YES instances that is computationally indistinguishable from a distribution $\mathcal{D}_{\text{NO}}$ of NO instances.

Definition 6.13 (Cryptographically-Hard Language)**.** Let $\mathcal{L} \subseteq \{0,1\}^n$ be an NP language. We say that $\mathcal{L}$ is a *cryptographically-hard* language if there exists distributions $\mathcal{D}_{\text{YES}}$ over $\mathcal{L}$ and $\mathcal{D}_{\text{NO}}$ over $\{0,1\}^n \setminus \mathcal{L}$ such that $\mathcal{D}_{\text{YES}} \overset{c}{\approx} \mathcal{D}_{\text{NO}}$.

Theorem 6.14 (Distributional Witness Encryption from Laconic Argument System)**.** *Fix a security parameter λ and let $\mathcal{L} \subseteq \{0,1\}^{n(\lambda)}$ be an NP language, and suppose that $\mathcal{L}$ is cryptographically-hard (Definition 6.13). Let $\mathcal{D}_{\text{YES}}$ and $\mathcal{D}_{\text{NO}}$ be the distributions over* YES *instances and* NO *instances, respectively, for $\mathcal{L}$ from Definition 6.13. Assume moreover that Π_{arg} is a 1-bit laconic argument for $\mathcal{L}$. Then, Π_{WE} from Construction 6.12 is a distributional witness encryption scheme for $\mathcal{L}$ with respect to $\mathcal{D}_{\text{NO}}$.*

Proof. We show correctness and security separately.

Correctness. Take any statement $\mathbf{x} \in \mathcal{L}$ and any witness $\mathbf{w}$ where $R(\mathbf{x}, \mathbf{w}) = 1$. We show that

$$\Pr[\mathsf{Decrypt}(\mathsf{Encrypt}(1^\lambda, \mathbf{x}, m), \mathbf{w}) = m] = 1.$$

Let $\mathsf{ct} \leftarrow \mathsf{Encrypt}(1^\lambda, \mathbf{x}, m)$, and $(\sigma_\mathbf{x}, \tau_\mathbf{x})$ be the parameters sampled by the encryption algorithm. By perfect completeness of Π_{arg}, if $b \leftarrow \mathsf{Prove}(\sigma_\mathbf{x}, \mathbf{x}, \mathbf{w})$, then $\mathsf{Verify}(\tau_\mathbf{x}, b) = 1$. We consider two possible scenarios:

- Suppose $\mathsf{Verify}(\tau_\mathbf{x}, 1-b) = 1$. In this case, $\mathsf{ct} = m$, and the decryption algorithm also outputs m. Correctness holds.
- Suppose $\mathsf{Verify}(\tau_\mathbf{x}, 1-b) = 0$. In this case, $\mathsf{ct} = (\sigma_\mathbf{x}, m \oplus b)$, and the decryption algorithm outputs $(m \oplus b) \oplus b = m$.

Security. By assumption, $\mathcal{D}_{\text{YES}} \overset{c}{\approx} \mathcal{D}_{\text{NO}}$. Now, we show that

$$\Pr[\mathbf{x} \leftarrow \mathcal{D}_{\text{NO}}; (\sigma_{\mathbf{x}}, \tau_{\mathbf{x}}) \leftarrow \mathsf{Setup}(1^\lambda, \mathbf{x}) : \mathsf{Verify}(\tau_{\mathbf{x}}, 0) = \mathsf{Verify}(\tau_{\mathbf{x}}, 1)] = \operatorname{negl}(\lambda).$$

We consider the two possibilities separately:

- Suppose $\mathsf{Verify}(\tau_{\mathbf{x}}, 0) = 0 = \mathsf{Verify}(\tau_{\mathbf{x}}, 1)$ with probability ε. This implies that $\mathcal{D}_{\text{YES}}$ and $\mathcal{D}_{\text{NO}}$ are distinguishable with the same advantage ε. Specifically, on input an instance $\mathbf{x}$, the distinguisher samples $(\sigma_{\mathbf{x}}, \tau_{\mathbf{x}}) \leftarrow \mathsf{Setup}(1^\lambda, \mathbf{x})$ and outputs 1 if $\mathsf{Verify}(\tau_{\mathbf{x}}, 0) = 0 = \mathsf{Verify}(\tau_{\mathbf{x}}, 1)$. If $\mathbf{x} \leftarrow \mathcal{D}_{\text{YES}}$, then $\mathbf{x} \in \mathcal{L}$ and by perfect completeness of Π_{arg}, the distinguisher outputs 1 with probability 0. Conversely, if $\mathbf{x} \leftarrow \mathcal{D}_{\text{NO}}$, then by assumption, the distinguisher outputs 1 with probability ε.

- Suppose $\mathsf{Verify}(\mathbf{x}, 0) = 1 = \mathsf{Verify}(\mathbf{x}, 1)$ with probability ε. Then, we can construct an adversary that breaks soundness of Π_{arg} with advantage $1/2 + \varepsilon/2 - \operatorname{negl}(\lambda)$. Consider the adversary that samples a statement $\mathbf{x} \leftarrow \mathcal{D}_{\text{NO}}$ and outputs a random bit $b \xleftarrow{\text{R}} \{0,1\}$ as its proof. We compute the probability that $\mathsf{Verify}(\tau_{\mathbf{x}}, b) = 1$, where $(\sigma_{\mathbf{x}}, \tau_{\mathbf{x}}) \leftarrow \mathsf{Setup}(1^\lambda, \mathbf{x})$. From the first case, we have that $\mathsf{Verify}(\tau_{\mathbf{x}}, 0) = 0 = \mathsf{Verify}(\tau_{\mathbf{x}}, 1)$ with negligible probability. Thus, with probability $1 - \operatorname{negl}(\lambda)$, at least one of $b \in \{0,1\}$ is a valid proof for $\mathbf{x}$. The probability that the guessing adversary succeeds in breaking soundness is then

$$\Pr[\mathsf{Verify}(\tau_{\mathbf{x}}, b) = 1] \geq \varepsilon + \frac{1}{2}(1 - \varepsilon - \operatorname{negl}(\lambda)) = \frac{1}{2} + \frac{\varepsilon}{2} - \operatorname{negl}(\lambda).$$

  This contradicts soundness of Π_{arg}.

We conclude that with overwhelming probability over the choice of $\mathbf{x}$ and the Setup randomness, there is exactly one proof $b \in \{0,1\}$ such that $\mathsf{Verify}(\tau_{\mathbf{x}}, b) = 1$. To show the claim, suppose there exists an efficient adversary $\mathcal{A}$ whose guessing advantage (Eq. (6.2)) is ε. Without loss of generality, suppose that given an encryption of $m \xleftarrow{\text{R}} \{0,1\}$, adversary $\mathcal{A}$ outputs m with probability at least $1/2 + \varepsilon$ (if $\mathcal{A}$ outputs m with probability less than $1/2 - \varepsilon$, we can consider an adversary that runs $\mathcal{A}$ and outputs the complement of $\mathcal{A}$'s output). We use $\mathcal{A}$ to construct an adversary $\mathcal{B}$ that breaks soundness of Π_{arg} with probability $1/2 + \varepsilon$. Algorithm $\mathcal{B}$ works as follows:

1. At the beginning of the game, algorithm $\mathcal{B}$ samples a statement $\mathbf{x} \leftarrow \mathcal{D}_{\text{NO}}$ and gives $\mathbf{x}$ to the challenger for the soundness game. It receives a common reference string $\sigma_{\mathbf{x}}$ from the soundness challenger.

2. Algorithm $\mathcal{B}$ samples a random bit $\beta \xleftarrow{\text{R}} \{0,1\}$ and sends $(\sigma_{\mathbf{x}}, \beta)$ to the guessing adversary $\mathcal{A}$.

3. When $\mathcal{A}$ outputs a guess $m \in \{0,1\}$, $\mathcal{B}$ submits $m \oplus \beta$ as its proof.

First, we argue that $\mathcal{B}$ correctly simulates the unguessability game for adversary $\mathcal{A}$. From above, we have that with overwhelming probability (over the choice of $\mathbf{x}$ and the randomness in the Setup

algorithm), $\mathsf{Verify}(\tau_{\mathbf{x}}, 0) \neq \mathsf{Verify}(\tau_{\mathbf{x}}, 1)$. Let $b \in \{0, 1\}$ be such that $\mathsf{Verify}(\tau_{\mathbf{x}}, b) = 1$. In this case, a valid ciphertext for a message m consists of the tuple $(\sigma_{\mathbf{x}}, b \oplus m)$. In the unguessability game, the message m is sampled uniformly at random, and so the bit $b \oplus m$ is also uniformly random. This is precisely the distribution $\mathcal{B}$ simulates in the reduction.

By assumption $\mathcal{A}$ is able to guess the message with probability at least $1/2 + \varepsilon$. In particular, this means that the bit m output by $\mathcal{A}$ satisfies $m = \beta \oplus b$ where $\mathsf{Verify}(\tau_{\mathbf{x}}, b) = 1$. But in this case, $b = m \oplus \beta$, and algorithm $\mathcal{B}$ has produced an accepting proof for the statement $\mathbf{x}$. We conclude that if $\mathcal{A}$ has guessing advantage ε, then $\mathcal{B}$ breaks soundness with advantage $1/2 + \varepsilon$. $\square$

Remark 6.15 (1-Bit Arguments and Predictable Arguments)**.** We can interpret the first step in our soundness proof of Theorem 6.14 as showing that a 1-bit argument for a cryptographically-hard language is essentially a predictable argument (c.f., [FNV17]). Specifically, we show that for a randomly-sampled statement $\mathbf{x}$, there is exactly *one* proof that the verifier accepts. Previously, Faonio et al. [FNV17] showed that any predictable argument for a language $\mathcal{L}$ implies a witness encryption for the same language. Since our arguments are predictable (for a randomly-sampled instance) when the underlying language is cryptographically-hard, we obtain the distributional variant of witness encryption for cryptographically-hard languages.

6.2.1 Distributional Witness Encryption to Public-Key Encryption

Although 1-bit laconic arguments only suffice for constructing a weaker distributional variant of witness encryption, this variant still suffices to instantiate some of the applications of witness encryption from [GGSW13]. In this section, we recall the construction of public-key encryption from witness encryption from [GGSW13, §4.1] and show how we can instantiate it using a distributional witness encryption scheme for the same language. In particular, this means that a 1-bit SNARG, and more generally, a 1-bit laconic argument implies a public-key encryption scheme where the complexity of the key-generation algorithm is *independent* of the complexity of the underlying argument system. Key-generation in this scheme only requires a single evaluation of a pseudorandom generator. The only public-key encryption schemes that have this property rely on witness encryption (or stronger assumptions). This provides some evidence on the difficulty of realizing optimally laconic arguments from simpler assumptions.

Construction 6.16 (Public Key Encryption from Witness Encryption [GGSW13, §4.1])**.** Fix a security parameter λ and let $G\colon \{0,1\}^{\lambda} \to \{0,1\}^{2\lambda}$ be a length-doubling PRG. Define the language $\mathcal{L} \subset \{0,1\}^{2\lambda}$ as $\mathcal{L} = \left\{y \in \{0,1\}^{2\lambda} : y = G(x) \text{ for some } x \in \{0,1\}^{\lambda}\right\}$. Let $\Pi_{\mathsf{WE}} = (\mathsf{WE.Encrypt}, \mathsf{WE.Decrypt})$ be a witness encryption scheme for $\mathcal{L}$. We define the public key encryption scheme $\Pi_{\mathsf{PKE}} = (\mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt})$ as follows:

- $\mathsf{KeyGen}(1^{\lambda}) \to (\mathsf{pk}, \mathsf{sk})$: On input the security parameter λ, the key-generation algorithm samples a seed $s \xleftarrow{\text{R}} \{0,1\}^{\lambda}$, computes $t \leftarrow G(s)$, and outputs $\mathsf{pk} = (\lambda, t)$ and $\mathsf{sk} = s$.

- $\mathsf{Encrypt}(\mathsf{pk}, m)$: On input the public key $\mathsf{pk} = (\lambda, t)$ and a message $m \in \{0,1\}$, the encryption algorithm outputs the ciphertext $\mathsf{ct} \leftarrow \mathsf{WE.Encrypt}(1^\lambda, t, m)$.
- $\mathsf{Decrypt}(\mathsf{sk}, \mathsf{ct})$: On input the secret key $\mathsf{sk} = s$ and a ciphertext ct, the decryption algorithm outputs $\mathsf{WE.Decrypt}(\mathsf{ct}, s)$.

Theorem 6.17. *Fix a security parameter λ and let $G \colon \{0,1\}^\lambda \to \{0,1\}^{2\lambda}$ be a length-doubling PRG. Define the language $\mathcal{L} \subset \{0,1\}^{2\lambda}$ as in Construction 6.16. Let $\mathcal{D}_{\text{NO}}$ be the uniform distribution over $\{0,1\}^{2\lambda} \setminus \mathcal{L}$. If Π_{WE} is a witness encryption scheme for $\mathcal{L}$ with respect to $\mathcal{D}_{\text{NO}}$ and G is a secure PRG, then Π_{PKE} from Construction 6.16 is a semantically-secure PKE scheme.*

Proof. The proof is essentially identical to the corresponding proof in [GGSW13, Appendix A.1]. We give the formal hybrid argument below:

- Hyb_0: This is the semantic security game where the challenger samples $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$ and responds to the adversary's query with $\mathsf{Encrypt}(\mathsf{pk}, m_0)$.
- Hyb_1: Same as Hyb_0 except instead of setting $\mathsf{pk} = (\lambda, t)$ where $t = G(s)$ and $s \xleftarrow{\text{R}} \{0,1\}^\lambda$, the challenger samples $t \xleftarrow{\text{R}} \{0,1\}^{2\lambda}$. Hybrids Hyb_0 and Hyb_1 are computationally indistinguishable by PRG security of G.
- Hyb_2: Same as Hyb_1, except the challenger samples $t \xleftarrow{\text{R}} \{0,1\}^{2\lambda} \setminus \mathcal{L}$. Hyb_1 and Hyb_2 are statistically indistinguishable.
- Hyb_3: Same as Hyb_2 except the challenger encrypts message m_1 when responding to the adversary's challenge. Hybrids Hyb_2 and Hyb_3 are computationally indistinguishable by distributional semantic security of Π_{WE}.
- Hyb_4: Same as Hyb_3 except the challenger samples $t \xleftarrow{\text{R}} \{0,1\}^{2\lambda}$. Hybrids Hyb_3 and Hyb_4 are statistically indistinguishable.
- Hyb_5: Same as Hyb_4 except the challenger sets $t = G(s)$ where $s \xleftarrow{\text{R}} \{0,1\}^\lambda$. This is the semantic security game where the challenger samples $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$ and respond to the adversary's query with $\mathsf{Encrypt}(\mathsf{pk}, m_1)$. Hybrids Hyb_4 and Hyb_5 are computationally indistinguishable by PRG security of G. □

To instantiate Construction 6.16 from a 1-bit laconic argument for NP (or more specifically, for the language $\mathcal{L}$ in Construction 6.16), it suffices to show that there exists a distribution $\mathcal{D}_{\text{YES}}$ over $\mathcal{L}$ such that $\mathcal{D}_{\text{YES}} \overset{c}{\approx} \mathcal{D}_{\text{NO}}$, where $\mathcal{D}_{\text{NO}}$ is the distribution from Construction 6.16. This follows from PRG security. Specifically, let $\mathcal{D}_{\text{YES}}$ be the distribution $\{s \xleftarrow{\text{R}} \{0,1\}^\lambda : G(s)\}$. By PRG security, $\mathcal{D}_{\text{YES}}$ is computationally indistinguishable from the uniform distribution over $\{0,1\}^{2\lambda}$. Finally, the uniform distribution over $\{0,1\}^{2\lambda}$ is statistically indistinguishable from $\mathcal{D}_{\text{NO}}$ and the claim follows. Thus, a 1-bit laconic argument for NP implies a public-key encryption scheme where the complexity of the key-generation algorithm is independent of the complexity of the witness encryption scheme.

6.3 Chapter Summary

In this chapter, we introduced and studied 1-bit laconic arguments, and more specifically, 1-bit SNARGs (namely, SNARGs that provide soundness error $1/2 + \mathrm{negl}(\lambda)$ with 1-bit of proof). We then showed how to construct 1-bit SNARGs from indistinguishability obfuscation by adapting the Sahai-Waters NIZK construction from indistinguishability obfuscation [SW14]. Furthermore, in the interactive setting, we showed how to obtain a 1-bit laconic argument from the (plausibly weaker) primitive of witness encryption. In fact, we demonstrated that this is (almost) a two-way implication as 1-bit laconic arguments imply a relaxed form of witness encryption. This latter connection suggests that building 1-bit laconic arguments from simpler assumptions is likely to be difficult. Nonetheless, we leave this as an interesting open problem and another potential route for realizing witness encryption from standard assumptions.

Chapter 7

Conclusions

Proof systems play an important role in the construction of numerous cryptographic protocols. Beyond that, they are also a fundamental notion in theoretical computer science. In this thesis, we studied two important properties of (non-interactive) proof systems: zero-knowledge and succinctness. Although numerous constructions of non-interactive proofs systems satisfying zero-knowledge or succinctness (or even both) have been proposed since the seminal work of Goldwasser, Micali, and Rackoff [GMR85], the prior construction all relied on the random oracle heuristic, number-theoretic and group-theoretic assumptions (e.g., factoring, pairings), or heavyweight tools like indistinguishability obfuscation. A major class of cryptographic assumptions missing from this list is the class of lattice-based assumptions.

In this thesis, we filled in some of these gaps by constructing the first non-interactive zero-knowledge argument in the preprocessing model as well as the first (quasi-optimal) succinct non-interactive argument from lattice-based assumptions. To briefly recall, we showed how to build multi-theorem NIZK arguments (and proofs) for all of NP in a preprocessing model using context-hiding homomorphic signatures (which can in turn be instantiated from standard lattice assumptions). In the preprocessing model, there is a trusted setup that generates proving and verification keys (used to construct and verify proofs, respectively). The main challenge in the preprocessing model was to construct a scheme that is multi-theorem secure: namely, the proving and verification keys could be *reused* to generate or to verify multiple proofs without compromising either soundness or zero-knowledge. While our lattice-based NIZK argument system operates in the weaker preprocessing model (rather than the more traditional CRS model), we are hopeful that our techniques will provide a useful stepping stone towards resolving the open problem of constructing NIZK proofs (or arguments) for NP from standard lattice assumptions.

With respect to succinct arguments, this thesis built upon and extended the general framework of Bitansky et al. [BCI$^+$13] to obtain the first candidate SNARGs from lattice-based assumptions. Our techniques gave the first quasi-optimally succinct SNARG from lattice assumptions, and further extension of our techniques yielded the first *quasi-optimal* SNARG from concrete assumptions over ideal lattices—namely, a SNARG that simultaneously minimizes the prover complexity as well as the proof size. Finally, we explored what happens when we push succinctness to the limit and consider a "1-bit SNARG." Here, we demonstrated an intriguing connection between 1-bit SNARGs and a relaxed variant of witness encryption.

Our work on constructing lattice-based SNARGs leaves open several direction for future research. First, all of the lattice-based SNARG candidates we introduced in this work are designated-verifier (i.e., a secret verification key is needed to verify the proofs). Can we build a *publicly-verifiable* SNARG from lattice assumptions? Can we build *zero-knowledge* SNARGs while preserving quasi-optimal succinctness from lattice assumptions? Finally, what is the concrete efficiency of these new lattice-based SNARGs, and how do they compare against their pairing-based counterparts [PHGR13, BCG+13]?[1]

[1]A preliminary implementation of our lattice-based SNARG candidate from Chapter 4 is currently available here: `https://github.com/dwu4/lattice-snarg`.

Bibliography

[ABB10a] Shweta Agrawal, Dan Boneh, and Xavier Boyen. Efficient lattice (H)IBE in the standard model. In *EUROCRYPT*, 2010.

[ABB10b] Shweta Agrawal, Dan Boneh, and Xavier Boyen. Lattice basis delegation in fixed dimension and shorter-ciphertext hierarchical IBE. In *CRYPTO*, 2010.

[ABC$^+$07] Giuseppe Ateniese, Randal C. Burns, Reza Curtmola, Joseph Herring, Lea Kissner, Zachary N. J. Peterson, and Dawn Xiaodong Song. Provable data possession at untrusted stores. In *ACM CCS*, 2007.

[ABC$^+$15] Jae Hyun Ahn, Dan Boneh, Jan Camenisch, Susan Hohenberger, Abhi Shelat, and Brent Waters. Computing on authenticated data. *J. Cryptology*, 28(2), 2015.

[Abe01] Masayuki Abe. A secure three-move blind signature scheme for polynomially many signatures. In *EUROCRYPT*, 2001.

[ACPS09] Benny Applebaum, David Cash, Chris Peikert, and Amit Sahai. Fast cryptographic primitives and circular-secure encryption based on hard learning problems. In *CRYPTO*, 2009.

[AFG$^+$10] Masayuki Abe, Georg Fuchsbauer, Jens Groth, Kristiyan Haralambiev, and Miyako Ohkubo. Structure-preserving signatures and commitments to group elements. In *CRYPTO*, 2010.

[AHO10] Masayuki Abe, Kristiyan Haralambiev, and Miyako Ohkubo. Signing on elements in bilinear groups for modular protocol design. *IACR Cryptology ePrint Archive*, 2010, 2010.

[AIK10] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. From secrecy to soundness: Efficient verification via secure computation. In *ICALP*, 2010.

[AJL$^+$12] Gilad Asharov, Abhishek Jain, Adriana López-Alt, Eran Tromer, Vinod Vaikuntanathan, and Daniel Wichs. Multiparty computation with low communication, computation and interaction via threshold FHE. In *EUROCRYPT*, 2012.

[Ajt96] Miklós Ajtai. Generating hard instances of lattice problems. In *STOC*, 1996.

[Ajt99] Miklós Ajtai. Generating hard instances of the short basis problem. In *ICALP*, 1999.

[AKK09] Giuseppe Ateniese, Seny Kamara, and Jonathan Katz. Proofs of storage from homomorphic identification protocols. In *ASIACRYPT*, 2009.

[AL11] Nuttapong Attrapadung and Benoît Libert. Homomorphic network coding signatures in the standard model. In *PKC*, 2011.

[ALM$^+$92] Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan, and Mario Szegedy. Proof verification and the hardness of approximation problems. In *FOCS*, 1992.

[AO09] Masayuki Abe and Miyako Ohkubo. A framework for universally composable non-committing blind signatures. In *ASIACRYPT*, 2009.

[AP09] Joël Alwen and Chris Peikert. Generating shorter bases for hard random lattices. In *STACS*, 2009.

[AP14] Jacob Alperin-Sheriff and Chris Peikert. Faster bootstrapping with polynomial error. In *CRYPTO*, 2014.

[APSD18] Navid Alamati, Chris Peikert, and Noah Stephens-Davidowitz. New (and old) proof systems for lattice problems. In *PKC*, 2018.

[BBDQ18] Fabrice Benhamouda, Olivier Blazy, Léo Ducas, and Willy Quach. Hash proof systems over lattices revisited. In *PKC*, 2018.

[BC12] Nir Bitansky and Alessandro Chiesa. Succinct arguments from multi-prover interactive proofs and their efficiency benefits. In *CRYPTO*, 2012.

[BCC88] Gilles Brassard, David Chaum, and Claude Crépeau. Minimum disclosure proofs of knowledge. *J. Comput. Syst. Sci.*, 37(2), 1988.

[BCC$^+$17] Nir Bitansky, Ran Canetti, Alessandro Chiesa, Shafi Goldwasser, Huijia Lin, Aviad Rubinstein, and Eran Tromer. The hunting of the SNARK. *J. Cryptology*, 30(4), 2017.

[BCCT12] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In *ITCS*, 2012.

[BCD$^+$16] Joppe Bos, Craig Costello, Léo Ducas, Ilya Mironov, Michael Naehrig, Valeria Nikolaenko, Ananth Raghunathan, and Douglas Stebila. Frodo: Take off the ring! Practical, quantum-secure key exchange from LWE. *IACR Cryptology ePrint Archive*, 2016, 2016.

[BCG$^+$13] Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, Eran Tromer, and Madars Virza. SNARKs for C: verifying program executions succinctly and in zero knowledge. In *CRYPTO*, 2013.

[BCG$^+$14] Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *IEEE SP*, 2014.

[BCI$^+$13] Nir Bitansky, Alessandro Chiesa, Yuval Ishai, Rafail Ostrovsky, and Omer Paneth. Succinct non-interactive arguments via linear interactive proofs. In *TCC*, 2013.

[BCPR14] Nir Bitansky, Ran Canetti, Omer Paneth, and Alon Rosen. On the existence of extractable one-way functions. In *STOC*, 2014.

[BCTV14] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. Succinct non-interactive zero knowledge for a von Neumann architecture. In *USENIX Security Symposium*, 2014.

[BDMP91] Manuel Blum, Alfredo De Santis, Silvio Micali, and Giuseppe Persiano. Noninteractive zero-knowledge. *SIAM J. Comput.*, 20(6), 1991.

[BDRV17] Itay Berman, Akshay Degwekar, Ron Rothblum, and Prashant Nalini Vasudevan. From laconic zero-knowledge to public-key cryptography. *Electronic Colloquium on Computational Complexity (ECCC)*, 2017, 2017.

[Bea91] Donald Beaver. Efficient multiparty protocols using circuit randomization. In *CRYPTO*, 1991.

[Ben64] Václad E Beneš. Optimal rearrangeable multistage connecting networks. *Bell Labs Technical Journal*, 43(4), 1964.

[BF01] Dan Boneh and Matthew K. Franklin. Identity-based encryption from the Weil pairing. In *CRYPTO*, 2001.

[BF11a] Dan Boneh and David Mandell Freeman. Homomorphic signatures for polynomial functions. In *EUROCRYPT*, 2011.

[BF11b] Dan Boneh and David Mandell Freeman. Linearly homomorphic signatures over binary fields and new tools for lattice-based signatures. In *PKC*, 2011.

[BFF$^+$09] Christina Brzuska, Marc Fischlin, Tobias Freudenreich, Anja Lehmann, Marcus Page, Jakob Schelbert, Dominique Schröder, and Florian Volk. Security of sanitizable signatures revisited. In *PKC*, 2009.

[BFKW09] Dan Boneh, David Mandell Freeman, Jonathan Katz, and Brent Waters. Signing a linear subspace: Signature schemes for network coding. In *PKC*, 2009.

[BFLS91] László Babai, Lance Fortnow, Leonid A. Levin, and Mario Szegedy. Checking computations in polylogarithmic time. In *STOC*, 1991.

[BFM88] Manuel Blum, Paul Feldman, and Silvio Micali. Non-interactive zero-knowledge and its applications. In *STOC*, 1988.

[BFR13a] Michael Backes, Dario Fiore, and Raphael M. Reischuk. Verifiable delegation of computation on outsourced data. In *ACM CCS*, 2013.

[BFR$^+$13b] Benjamin Braun, Ariel J. Feldman, Zuocheng Ren, Srinath T. V. Setty, Andrew J. Blumberg, and Michael Walfish. Verifying computations with state. In *ACM SIGOPS 24th Symposium on Operating Systems Principles, SOSP '13, Farmington, PA, USA, November 3-6, 2013*, 2013.

[BGG$^+$14] Dan Boneh, Craig Gentry, Sergey Gorbunov, Shai Halevi, Valeria Nikolaenko, Gil Segev, Vinod Vaikuntanathan, and Dhinakaran Vinayagamurthy. Fully key-homomorphic encryption, arithmetic circuit ABE and compact garbled circuits. In *EUROCRYPT*, 2014.

[BGI$^+$01] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. In *CRYPTO*, 2001.

[BGI14] Elette Boyle, Shafi Goldwasser, and Ioana Ivan. Functional signatures and pseudorandom functions. In *PKC*, 2014.

[BGI16] Elette Boyle, Niv Gilboa, and Yuval Ishai. Breaking the circuit size barrier for secure computation under DDH. In *CRYPTO*, 2016.

[BGV12] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (Leveled) fully homomorphic encryption without bootstrapping. In *ITCS*, 2012.

[BHZ87] Ravi B. Boppana, Johan Håstad, and Stathis Zachos. Does co-np have short interactive proofs? *Inf. Process. Lett.*, 25(2), 1987.

[BISW17] Dan Boneh, Yuval Ishai, Amit Sahai, and David J. Wu. Lattice-based SNARGs and their application to more efficient obfuscation. In *EUROCRYPT*, 2017.

[BISW18] Dan Boneh, Yuval Ishai, Amit Sahai, and David J. Wu. Quasi-optimal SNARGs via linear multi-prover interactive proofs. In *EUROCRYPT*, 2018.

[BKM17] Dan Boneh, Sam Kim, and Hart William Montgomery. Private puncturable prfs from standard lattice assumptions. In *EUROCRYPT*, 2017.

[BKW00] Avrim Blum, Adam Kalai, and Hal Wasserman. Noise-tolerant learning, the parity problem, and the statistical query model. In *STOC*, 2000.

[BKW17] Dan Boneh, Sam Kim, and David J. Wu. Constrained keys for invertible pseudorandom functions. In *TCC*, 2017.

[BL13] Foteini Baldimtsi and Anna Lysyanskaya. Anonymous credentials light. In *ACM CCS*, 2013.

[BLP$^+$13] Zvika Brakerski, Adeline Langlois, Chris Peikert, Oded Regev, and Damien Stehlé. Classical hardness of learning with errors. In *STOC*, 2013.

[BNPS03] Mihir Bellare, Chanathip Namprempre, David Pointcheval, and Michael Semanko. The one-more-rsa-inversion problems and the security of chaum's blind signature scheme. *J. Cryptology*, 16(3), 2003.

[Bol03] Alexandra Boldyreva. Threshold signatures, multisignatures and blind signatures based on the gap-diffie-hellman-group signature scheme. In *PKC*, 2003.

[BP04a] Boaz Barak and Rafael Pass. On the possibility of one-message weak zero-knowledge. In *TCC*, 2004.

[BP04b] Mihir Bellare and Adriana Palacio. The knowledge-of-exponent assumptions and 3-round zero-knowledge protocols. In *CRYPTO*, 2004.

[BP04c] Mihir Bellare and Adriana Palacio. Towards plaintext-aware public-key encryption without random oracles. In *ASIACRYPT*, 2004.

[Bra00] Stefan A. Brands. *Rethinking public key infrastructures and digital certificates: building in privacy.* MIT Press, 2000.

[BS08] Eli Ben-Sasson and Madhu Sudan. Short PCPs with polylog query complexity. *SIAM J. Comput.*, 38(2), 2008.

[BSW12] Dan Boneh, Gil Segev, and Brent Waters. Targeted malleability: homomorphic encryption for restricted computations. In *ITCS*, 2012.

[BTVW17] Zvika Brakerski, Rotem Tsabary, Vinod Vaikuntanathan, and Hoeteck Wee. Private constrained prfs (and more) from LWE. In *TCC*, 2017.

[BV14] Zvika Brakerski and Vinod Vaikuntanathan. Lattice-based FHE as secure as PKE. In *ITCS*, 2014.

[BV15] Zvika Brakerski and Vinod Vaikuntanathan. Constrained key-homomorphic prfs from standard lattice assumptions - or: How to secretly embed a circuit in your PRF. In *TCC*, 2015.

[BW13] Dan Boneh and Brent Waters. Constrained pseudorandom functions and their applications. In *ASIACRYPT*, 2013.

[Can00] Ran Canetti. Security and composition of multiparty cryptographic protocols. *J. Cryptology*, 13(1), 2000.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS*, 2001.

[Can04] Ran Canetti. Universally composable signature, certification, and authentication. In *CSFW*, 2004.

[Cat14] Dario Catalano. Homomorphic signatures and message authentication codes. In *SCN*, 2014.

[CC17a] Ran Canetti and Yilei Chen. Constraint-hiding constrained prfs for nc^1 from LWE. In *EUROCRYPT*, 2017.

[CC17b] Pyrros Chaidos and Geoffroy Couteau. Efficient designated-verifier non-interactive zero-knowledge proofs of knowledge. *IACR Cryptology ePrint Archive*, 2017, 2017.

[CCRR18] Ran Canetti, Yilei Chen, Leonid Reyzin, and Ron D. Rothblum. Fiat-shamir and correlation intractability from strong KDM-secure encryption. In *EUROCRYPT*, 2018.

[CD04] Ronald Cramer and Ivan Damgård. Secret-key zero-knowlegde and non-interactive verifiable exponentiation. In *TCC*, 2004.

[CF13] Dario Catalano and Dario Fiore. Practical homomorphic macs for arithmetic circuits. In *EUROCRYPT*, 2013.

[CFGN14] Dario Catalano, Dario Fiore, Rosario Gennaro, and Luca Nizzardo. Generalizing homomorphic macs for arithmetic circuits. In *PKC*, 2014.

[CFH$^+$15] Craig Costello, Cédric Fournet, Jon Howell, Markulf Kohlweiss, Benjamin Kreuter, Michael Naehrig, Bryan Parno, and Samee Zahur. Geppetto: Versatile verifiable computation. In *IEEE SP*, 2015.

[CFW12] Dario Catalano, Dario Fiore, and Bogdan Warinschi. Efficient network coding signatures in the standard model. In *PKC*, 2012.

[CFW14] Dario Catalano, Dario Fiore, and Bogdan Warinschi. Homomorphic signatures with efficient verification for polynomial functions. In *CRYPTO*, 2014.

[CG15] Pyrros Chaidos and Jens Groth. Making sigma-protocols non-interactive without random oracles. In *PKC*, 2015.

[Cha82] David Chaum. Blind signatures for untraceable payments. In *CRYPTO*, 1982.

[CHKP10] David Cash, Dennis Hofheinz, Eike Kiltz, and Chris Peikert. Bonsai trees, or how to delegate a lattice basis. In *EUROCRYPT*, 2010.

[CKW04] Jan Camenisch, Maciej Koprowski, and Bogdan Warinschi. Efficient blind signatures without random oracles. In *SCN*, 2004.

[CLOS02] Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. Universally composable two-party and multi-party secure computation. In *STOC*, 2002.

[CMS99] Christian Cachin, Silvio Micali, and Markus Stadler. Computationally private information retrieval with polylogarithmic communication. In *EUROCRYPT*, 1999.

[CMT12] Graham Cormode, Michael Mitzenmacher, and Justin Thaler. Practical verified computation with streaming interactive proofs. In *ITCS*, 2012.

[CN11] Yuanmi Chen and Phong Q. Nguyen. BKZ 2.0: Better lattice security estimates. In *ASIACRYPT*, 2011.

[CR03] Ran Canetti and Tal Rabin. Universal composition with joint state. In *CRYPTO*, 2003.

[CS02] Ronald Cramer and Victor Shoup. Universal hash proofs and a paradigm for adaptive chosen ciphertext secure public-key encryption. In *EUROCRYPT*, 2002.

[Dam91] Ivan Damgård. Towards practical public key systems secure against chosen ciphertext attacks. In *CRYPTO*, 1991.

[Dam92] Ivan Damgård. Non-interactive circuit based proofs and non-interactive perfect zero-knowledge with proprocessing. In *EUROCRYPT*, 1992.

[DDO$^+$01] Alfredo De Santis, Giovanni Di Crescenzo, Rafail Ostrovsky, Giuseppe Persiano, and Amit Sahai. Robust non-interactive zero knowledge. In *CRYPTO*, 2001.

[DFGK14] George Danezis, Cédric Fournet, Jens Groth, and Markulf Kohlweiss. Square span programs with applications to succinct NIZK arguments. In *ASIACRYPT*, 2014.

[DFH12] Ivan Damgård, Sebastian Faust, and Carmit Hazay. Secure two-party computation with low communication. In *TCC*, 2012.

[DFN06] Ivan Damgård, Nelly Fazio, and Antonio Nicolosi. Non-interactive zero-knowledge from homomorphic encryption. In *TCC*, 2006.

[DIK10] Ivan Damgård, Yuval Ishai, and Mikkel Krøigaard. Perfectly secure multiparty computation and the computational overhead of cryptography. In *EUROCRYPT*, 2010.

[Din06] Irit Dinur. The PCP theorem by gap amplification. In *STOC*, 2006.

[DMP87] Alfredo De Santis, Silvio Micali, and Giuseppe Persiano. Non-interactive zero-knowledge proof systems. In *CRYPTO*, 1987.

[DMP88] Alfredo De Santis, Silvio Micali, and Giuseppe Persiano. Non-interactive zero-knowledge with preprocessing. In *CRYPTO*, 1988.

[DPSZ12] Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In *CRYPTO*, 2012.

[DVW09] Yevgeniy Dodis, Salil P. Vadhan, and Daniel Wichs. Proofs of retrievability via hardness amplification. In *TCC*, 2009.

[FGL$^+$91] Uriel Feige, Shafi Goldwasser, László Lovász, Shmuel Safra, and Mario Szegedy. Approximating clique is almost np-complete (preliminary version). In *FOCS*, 1991.

[FHKS16] Georg Fuchsbauer, Christian Hanser, Chethan Kamath, and Daniel Slamanig. Practical round-optimal blind signatures in the standard model from weaker assumptions. In *SCN*, 2016.

[FHS15] Georg Fuchsbauer, Christian Hanser, and Daniel Slamanig. Practical round-optimal blind signatures in the standard model. In *CRYPTO*, 2015.

[Fis06] Marc Fischlin. Round-optimal composable blind signatures in the common reference string model. In *CRYPTO*, 2006.

[FLS90] Uriel Feige, Dror Lapidot, and Adi Shamir. Multiple non-interactive zero knowledge proofs based on a single random string. In *FOCS*, 1990.

[FMNP16] Dario Fiore, Aikaterini Mitrokotsa, Luca Nizzardo, and Elena Pagnin. Multi-key homomorphic authenticators. In *ASIACRYPT*, 2016.

[FNV17] Antonio Faonio, Jesper Buus Nielsen, and Daniele Venturi. Predictable arguments of knowledge. In *PKC*, 2017.

[Fre12] David Mandell Freeman. Improved security for linearly homomorphic signatures: A generic framework. In *PKC*, 2012.

[FS86] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *CRYPTO*, 1986.

[Fuc09] Georg Fuchsbauer. Automorphic signatures in bilinear groups and an application to round-optimal blind signatures. *IACR Cryptology ePrint Archive*, 2009, 2009.

[Gen09a] Craig Gentry. *A fully homomorphic encryption scheme*. PhD thesis, Stanford University, 2009. `crypto.stanford.edu/craig`.

[Gen09b] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *STOC*, 2009.

[GGH$^+$13] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *FOCS*, 2013.

[GGI$^+$15] Craig Gentry, Jens Groth, Yuval Ishai, Chris Peikert, Amit Sahai, and Adam D. Smith. Using fully homomorphic hybrid encryption to minimize non-interative zero-knowledge proofs. *J. Cryptology*, 28(4), 2015.

[GGM84] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. In *FOCS*, 1984.

[GGPR13] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct nizks without PCPs. In *EUROCRYPT*, 2013.

[GGSW13] Sanjam Garg, Craig Gentry, Amit Sahai, and Brent Waters. Witness encryption and its applications. In *STOC*, 2013.

[GH98] Oded Goldreich and Johan Håstad. On the complexity of interactive proofs with bounded communication. *Inf. Process. Lett.*, 67(4), 1998.

[GHS12] Craig Gentry, Shai Halevi, and Nigel P. Smart. Fully homomorphic encryption with polylog overhead. In *EUROCRYPT*, 2012.

[GKKR10] Rosario Gennaro, Jonathan Katz, Hugo Krawczyk, and Tal Rabin. Secure network coding over the integers. In *PKC*, 2010.

[GKR08] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. Delegating computation: interactive proofs for muggles. In *STOC*, 2008.

[GKW18] Rishab Goyal, Venkata Koppula, and Brent Waters. Collusion resistant traitor tracing from learning with errors. In *STOC*, 2018.

[GMR85] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof-systems (extended abstract). In *STOC*, 1985.

[GMW86] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to prove all np-statements in zero-knowledge, and a methodology of cryptographic protocol design. In *CRYPTO*, 1986.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In *STOC*, 1987.

[GO94] Oded Goldreich and Yair Oren. Definitions and properties of zero-knowledge proof systems. *J. Cryptology*, 7(1), 1994.

[Gol01] Oded Goldreich. *The Foundations of Cryptography - Volume 1, Basic Techniques*. Cambridge University Press, 2001.

[Gol04] Oded Goldreich. *The Foundations of Cryptography - Volume 2, Basic Applications*. Cambridge University Press, 2004.

[GOS06] Jens Groth, Rafail Ostrovsky, and Amit Sahai. Perfect non-interactive zero knowledge for NP. In *EUROCRYPT*, 2006.

[GOS12] Jens Groth, Rafail Ostrovsky, and Amit Sahai. New techniques for noninteractive zero-knowledge. *J. ACM*, 59(3), 2012.

[GPV08] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In *STOC*, 2008.

[Gro06] Jens Groth. Simulation-sound NIZK proofs for a practical language and constant size group signatures. In *ASIACRYPT*, 2006.

[Gro09] Jens Groth. Linear algebra with sub-linear zero-knowledge arguments. In *CRYPTO*, 2009.

[Gro10] Jens Groth. Short pairing-based non-interactive zero-knowledge arguments. In *ASIACRYPT*, 2010.

[Gro16] Jens Groth. On the size of pairing-based non-interactive arguments. In *EUROCRYPT*, 2016.

[GRS$^+$11] Sanjam Garg, Vanishree Rao, Amit Sahai, Dominique Schröder, and Dominique Unruh. Round optimal blind signatures. In *CRYPTO*, 2011.

[GS08] Jens Groth and Amit Sahai. Efficient non-interactive proof systems for bilinear groups. In *EUROCRYPT*, 2008.

[GS12] Essam Ghadafi and Nigel P. Smart. Efficient two-move blind signatures in the common reference string model. In *ISC*, 2012.

[GS18] Sanjam Garg and Akshayaram Srinivasan. Two-round multiparty secure computation from minimal assumptions. In *EUROCRYPT*, 2018.

[GSW13] Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In *CRYPTO*, 2013.

[GV15] Sergey Gorbunov and Dhinakaran Vinayagamurthy. Riding on asymmetry: Efficient ABE for branching programs. In *ASIACRYPT*, 2015.

[GVW01] Oded Goldreich, Salil P. Vadhan, and Avi Wigderson. On interactive proofs with a laconic prover. In *ICALP*, 2001.

[GVW13] Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Attribute-based encryption for circuits. In *STOC*, 2013.

[GVW15a] Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Predicate encryption for circuits from LWE. In *CRYPTO*, 2015.

[GVW15b] Sergey Gorbunov, Vinod Vaikuntanathan, and Daniel Wichs. Leveled fully homomorphic signatures from standard lattices. In *STOC*, 2015.

[GW11] Craig Gentry and Daniel Wichs. Separating succinct non-interactive arguments from all falsifiable assumptions. In *STOC*, 2011.

[GW13] Rosario Gennaro and Daniel Wichs. Fully homomorphic message authenticators. In *ASIACRYPT*, 2013.

[HK16] Lucjan Hanzlik and Kamil Kluczniak. A short paper on blind signatures from knowledge assumptions. In *Financial Cryptography*, 2016.

[IKO07] Yuval Ishai, Eyal Kushilevitz, and Rafail Ostrovsky. Efficient arguments without short PCPs. In *CCC*, 2007.

[IKOS07] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Zero-knowledge from secure multiparty computation. In *STOC*, 2007.

[IPS09] Yuval Ishai, Manoj Prabhakaran, and Amit Sahai. Secure arithmetic computation with no honest majority. In *TCC*, 2009.

[Jou00] Antoine Joux. A one round protocol for tripartite Diffie-Hellman. In *ANTS*, 2000.

[KF15] Paul Kirchner and Pierre-Alain Fouque. An improved BKW algorithm for LWE with applications to cryptography and lattices. In *CRYPTO*, 2015.

[Kil92] Joe Kilian. A note on efficient zero-knowledge proofs and arguments. In *STOC*, 1992.

[KMO89] Joe Kilian, Silvio Micali, and Rafail Ostrovsky. Minimum resource zero-knowledge proofs. In *CRYPTO*, 1989.

[KPTZ13] Aggelos Kiayias, Stavros Papadopoulos, Nikos Triandopoulos, and Thomas Zacharias. Delegatable pseudorandom functions and applications. In *ACM CCS*, 2013.

[KR06] Yael Tauman Kalai and Ran Raz. Succinct non-interactive zero-knowledge proofs with preprocessing for LOGSNP. In *FOCS*, 2006.

[KV09] Jonathan Katz and Vinod Vaikuntanathan. Smooth projective hashing and password-based authenticated key exchange from lattices. In *ASIACRYPT*, 2009.

[KW17] Sam Kim and David J. Wu. Watermarking cryptographic functionalities from standard lattice assumptions. In *CRYPTO*, 2017.

[KW18] Sam Kim and David J. Wu. Multi-theorem preprocessing NIZKs from lattices. In *CRYPTO*, 2018.

[KZ06] Aggelos Kiayias and Hong-Sheng Zhou. Concurrent blind signatures without random oracles. In *SCN*, 2006.

[LFKN90] Carsten Lund, Lance Fortnow, Howard J. Karloff, and Noam Nisan. Algebraic methods for interactive proof systems. In *FOCS*, 1990.

[Lip12] Helger Lipmaa. Progression-free sets and sublinear pairing-based non-interactive zero-knowledge arguments. In *TCC*, 2012.

[Lip13] Helger Lipmaa. Succinct non-interactive zero knowledge arguments from span programs and linear error-correcting codes. In *ASIACRYPT*, 2013.

[Lip16] Helger Lipmaa. Prover-efficient commit-and-prove zero-knowledge SNARKs. In *AFRICACRYPT*, 2016.

[LNSW13] San Ling, Khoa Nguyen, Damien Stehlé, and Huaxiong Wang. Improved zero-knowledge proofs of knowledge for the ISIS problem, and applications. In *PKC*, 2013.

[LP07] Yehuda Lindell and Benny Pinkas. An efficient protocol for secure two-party computation in the presence of malicious adversaries. In *EUROCRYPT*, 2007.

[LP11] Richard Lindner and Chris Peikert. Better key sizes (and attacks) for LWE-based encryption. In *CT-RSA*, 2011.

[LPSS14] San Ling, Duong Hieu Phan, Damien Stehlé, and Ron Steinfeld. Hardness of k-lwe and applications in traitor tracing. In *CRYPTO*, 2014.

[LS90] Dror Lapidot and Adi Shamir. Publicly verifiable non-interactive zero-knowledge proofs. In *CRYPTO*, 1990.

[LW15] Vadim Lyubashevsky and Daniel Wichs. Simple lattice trapdoor sampling from a broad class of distributions. In *PKC*, 2015.

[Mic00] Silvio Micali. Computationally sound proofs. *SIAM J. Comput.*, 30(4), 2000.

[Mic04] Daniele Micciancio. Almost perfect lattices, the covering radius problem, and applications to ajtai's connection factor. *SIAM J. Comput.*, 34(1), 2004.

[Mie08] Thilo Mie. Polylogarithmic two-round argument systems. *J. Mathematical Cryptology*, 2(4), 2008.

[MM11] Daniele Micciancio and Petros Mol. Pseudorandom knapsacks and the sample complexity of LWE search-to-decision reductions. In *CRYPTO*, 2011.

[MP12] Daniele Micciancio and Chris Peikert. Trapdoors for lattices: Simpler, tighter, faster, smaller. In *EUROCRYPT*, 2012.

[MP13] Daniele Micciancio and Chris Peikert. Hardness of SIS and LWE with small parameters. In *CRYPTO*, 2013.

[MR07] Daniele Micciancio and Oded Regev. Worst-case to average-case reductions based on gaussian measures. *SIAM J. Comput.*, 37(1), 2007.

[MR09] Daniele Micciancio and Oded Regev. Lattice-based cryptography. In *Post-Quantum Cryptography*, 2009.

[MW16] Pratyay Mukherjee and Daniel Wichs. Two round multiparty computation via multi-key FHE. In *EUROCRYPT*, 2016.

[Nao03] Moni Naor. On cryptographic assumptions and challenges. In *CRYPTO*, 2003.

[NR97] Moni Naor and Omer Reingold. Number-theoretic constructions of efficient pseudo-random functions. In *FOCS*, 1997.

[OTW71] D.C. Opferman and N.T. Tsao-Wu. On a class of rearrangeable switching networks part I: Control algorithm. *Bell Labs Technical Journal*, 50(5), 1971.

[Pai99] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *EUROCRYPT*, 1999.

[Pei09] Chris Peikert. Public-key cryptosystems from the worst-case shortest vector problem. In *STOC*, 2009.

[Pei16] Chris Peikert. A decade of lattice cryptography. *Foundations and Trends in Theoretical Computer Science*, 10(4), 2016.

[PHGR13] Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. Pinocchio: Nearly practical verifiable computation. In *IEEE SP*, 2013.

[PS96] David Pointcheval and Jacques Stern. Security proofs for signature schemes. In *EUROCRYPT*, 1996.

[PS00] David Pointcheval and Jacques Stern. Security arguments for digital signatures and blind signatures. *J. Cryptology*, 13(3), 2000.

[PS18] Chris Peikert and Sina Shiehian. Privately constraining and programming prfs, the LWE way. In *PKC*, 2018.

[PV08] Chris Peikert and Vinod Vaikuntanathan. Noninteractive statistical zero-knowledge proofs for lattice problems. In *CRYPTO*, 2008.

[PVW08] Chris Peikert, Vinod Vaikuntanathan, and Brent Waters. A framework for efficient and composable oblivious transfer. In *CRYPTO*, 2008.

[Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In *STOC*, 2005.

[RSS18] Ron D. Rothblum, Adam Sealfon, and Katerina Sotiraki. Towards non-interactive zero-knowledge for NP from LWE. *IACR Cryptology ePrint Archive*, 2018, 2018.

[Rüc10] Markus Rückert. Lattice-based blind signatures. In *ASIACRYPT*, 2010.

[Sch80] Jacob T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *J. ACM*, 27(4), 1980.

[Sch89] Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In *CRYPTO*, 1989.

[SE94] Claus-Peter Schnorr and M. Euchner. Lattice basis reduction: Improved practical algorithms and solving subset sum problems. *Math. Program.*, 66, 1994.

[Sha90] Adi Shamir. IP=PSPACE. In *FOCS*, 1990.

[Sho94] Peter W. Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *FOCS*, 1994.

[SMBW12] Srinath T. V. Setty, Richard McPherson, Andrew J. Blumberg, and Michael Walfish. Making argument systems for outsourced computation practical (sometimes). In *NDSS*, 2012.

[SP92] Alfredo De Santis and Giuseppe Persiano. Zero-knowledge proofs of knowledge without interaction. In *FOCS*, 1992.

[SVP$^+$12] Srinath T. V. Setty, Victor Vu, Nikhil Panpalia, Benjamin Braun, Andrew J. Blumberg, and Michael Walfish. Taking proof-based verified computation a few steps closer to practicality. In *USENIX Security Symposium*, 2012.

[SW08] Hovav Shacham and Brent Waters. Compact proofs of retrievability. In *ASIACRYPT*, 2008.

[SW14] Amit Sahai and Brent Waters. How to use indistinguishability obfuscation: deniable encryption, and more. In *STOC*, 2014.

[Tha13] Justin Thaler. Time-optimal interactive proofs for circuit evaluation. In *CRYPTO*, 2013.

[TRMP12] Justin Thaler, Mike Roberts, Michael Mitzenmacher, and Hanspeter Pfister. Verifiable computation with massively parallel interactive proofs. In *HotCloud*, 2012.

[VSBW13] Victor Vu, Srinath T. V. Setty, Andrew J. Blumberg, and Michael Walfish. A hybrid architecture for interactive verifiable computation. In *IEEE SP*, 2013.

[Wak68] Abraham Waksman. A permutation network. *Journal of the ACM (JACM)*, 15(1), 1968.

[WB15] Michael Walfish and Andrew J. Blumberg. Verifying computations without reexecuting them. *Commun. ACM*, 58(2), 2015.

[Wee05] Hoeteck Wee. On round-efficient argument systems. In *ICALP*, 2005.

[WSR$^+$15] Riad S. Wahby, Srinath T. V. Setty, Zuocheng Ren, Andrew J. Blumberg, and Michael Walfish. Efficient RAM and control flow in verifiable outsourced computation. In *NDSS*, 2015.

[XXW13] Xiang Xie, Rui Xue, and Minqian Wang. Zero knowledge proofs from ring-lwe. In *CANS*, 2013.

[Zip79] Richard Zippel. Probabilistic algorithms for sparse polynomials. In *EUROSAM*, 1979.

[ZY17] Jiang Zhang and Yu Yu. Two-round PAKE from approximate SPH and instantiations from lattices. In *ASIACRYPT*, 2017.