ABSTRACT

Title of Dissertation: IMPROVING THE SCALABILITY AND DEPLOYABILITY OF CRYPTOGRAPHIC PROOF SYSTEMS

Kasra Abbaszadeh
Doctor of Philosophy, 2026

Dissertation Directed by: Professor Jonathan Katz
Department of Computer Science

Succinct zero-knowledge arguments (zk-SNARKs) enable a prover to convince a verifier of the truth of a statement via a succinct, efficiently verifiable proof without revealing any additional information about the witness. Despite their powerful capabilities, the practical deployment of zk-SNARKs remains limited to a narrow set of use cases due to high proving and verification costs. In this dissertation, we broaden the applicability of zk-SNARKs by introducing techniques that improve their concrete efficiency and enhance their viability for real-world deployment.

First, we study *proofs of training*—a cryptographic tool that enables a verifier to check whether a (committed) model has been trained on a (committed) dataset as specified—and show how to realize this primitive efficiently for deep neural networks using recursive zk-SNARKs.

Second, we study a further application of recursive zk-SNARKs to *secure aggregation*—a protocol that enables a server to learn the sum of clients' private inputs—and show how recursive zk-SNARKs can reduce the committee cost in state-of-the-art secure aggregation schemes.

Finally, we introduce the notion of *server-aided* zk-SNARKs, which enable a prover (client) to outsource most of its work to an untrusted server while the server learns no information about the witness, the statement, or even the final proof. We propose a technique to achieve practical server-aided proving for several widely deployed zk-SNARKs in a single-server setting.

IMPROVING THE SCALABILITY AND DEPLOYABILITY OF CRYPTOGRAPHIC PROOF SYSTEMS

by

Kasra Abbaszadeh

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2026

Advisory Committee:
Professor Jonathan Katz, Chair/Advisor
Professor Dana Dachman-Soled, Dean's Representative
Professor Ian Miers
Professor Bahar Asgari
Professor David Van Horn

© Copyright by
Kasra Abbaszadeh
2026

Acknowledgments

First and foremost, I thank my advisor, Jonathan Katz, for his insightful guidance, constant support, and for giving me the freedom to pursue my broad research interests in cryptography. I am sincerely grateful to Sarah Meiklejohn for her supervision during my internship at Google. I also thank Ian Miers for welcoming me to UMD and mentoring me during my first year. I am grateful to my coauthors and collaborators—Matteo Maffei, Mariana Raykova, Benedikt Bünz, Dimitrios Papadopoulos, Adrià Gascón, Phillipp Schoppmann, Hossein Hafezi, Christodoulos Pappas, and Lukas Aumayr—for their contributions to the projects that shaped my research. I would also like to thank the other members of my defense committee—Dana Dachman-Soled, Bahar Asgari, and David Van Horn—for their time and feedback. Finally, I would like to thank my family—my parents and my sister—to whom this dissertation is dedicated.

Table of Contents

Acknowledgments **ii**

Table of Contents **iii**

1 Introduction **1**
- 1.1 Zero-Knowledge Proofs of Training for Neural Networks . . . 2
- 1.2 Secure Aggregation with a Lightweight Committee . . . 3
- 1.3 Single-Server Private Outsourcing of zk-SNARKs . . . 4

2 Preliminaries **5**
- 2.1 Notation . . . 5
- 2.2 Argument Systems . . . 7
- 2.3 Proof-Carrying Data . . . 10
- 2.4 Commitment Schemes . . . 13
- 2.5 Error-Correcting Codes . . . 15
- 2.6 Universal Composability Framework . . . 16

3 Zero-Knowledge Proofs of Training for Neural Networks **17**
- 3.1 Related Work . . . 20
- 3.2 Technical Overview . . . 21
  - 3.2.1 Sumcheck-Based Proofs of Gradient Descent . . . 21
  - 3.2.2 Recursive Composition of Sumcheck Proofs . . . 22
  - 3.2.3 Proofs of Training for Neural Networks . . . 25
- 3.3 Additional Preliminaries . . . 25
  - 3.3.1 Neural Networks . . . 25
  - 3.3.2 The GKR Protocol . . . 28
- 3.4 Proofs of Gradient Descent . . . 29
  - 3.4.1 Fixed-Point Operations . . . 29
  - 3.4.2 PoGDs for Neural Networks . . . 32
- 3.5 Foldable Polynomial Commitments . . . 35
  - 3.5.1 Accumulation of Evaluations . . . 38
  - 3.5.2 Accumulation of Commitments . . . 40
- 3.6 Recursive Sumcheck Arguments . . . 44
- 3.7 Zero-Knowledge Proofs of Training . . . 48
  - 3.7.1 Definition . . . 49
  - 3.7.2 zk-PoTs for Neural Networks . . . 50

3.8 Implementation and Evaluation . . . 51
3.8.1 KAIZEN zk-PoT . . . 53
3.8.2 Comparison to Generic Proofs . . . 55

4 Secure Aggregation with a Lightweight Committee 57
4.1 Related Work . . . 61
4.2 Technical Overview . . . 64
4.2.1 Model and Functionality . . . 64
4.2.2 Overview of WILLOWFOLD . . . 66
4.2.3 Required Properties of the PCD Scheme . . . 70
4.2.4 Concrete Instantiation and Optimizations . . . 71
4.3 Additional Preliminaries . . . 73
4.3.1 Encryption Schemes . . . 73
4.3.2 Simulation Extractability . . . 75
4.3.3 Details of KZH and KZH-Fold . . . 78
4.4 WILLOWFOLD Secure Aggregation . . . 81
4.4.1 Setup and Client Registration . . . 81
4.4.2 Client, Server, and Decryptor Roles . . . 81
4.5 Security Analysis of WILLOWFOLD . . . 84
4.5.1 Ciphertext Correlation Intractability . . . 87
4.5.2 Simulation Extractability of PCD Schemes . . . 89
4.5.3 Simulation-Based Security of WILLOWFOLD . . . 92
4.6 Instantiation with Spartan+KZH-Fold . . . 97
4.6.1 Zero-Knowledge Variant of Spartan+KZH . . . 97
4.6.2 Simulation Extractability of Spartan+KZH-Fold . . . 99
4.7 Implementation and Evaluation . . . 106
4.7.1 Optimizations . . . 106
4.7.2 Setup and Registration Costs . . . 108
4.7.3 Client, Server, and Decryptor Costs . . . 108

5 Single-Server Private Outsourcing of zk-SNARKs 112
5.1 Related Work . . . 116
5.2 Additional Preliminaries . . . 118
5.2.1 Arithmetizations . . . 119
5.2.2 Unrolled Protocols of zk-SNARKs . . . 120
5.2.3 Repeat-Accumulate-Accumulate Codes . . . 124
5.2.4 The Learning Parity with Noise Assumption . . . 124
5.3 Concrete Security of Dual-LPN . . . 126
5.3.1 Distance Analysis of RAA Codes . . . 127
5.3.2 Parameter Selection for Dual-LPN . . . 129
5.4 Encrypted Multi-Scalar Multiplication . . . 129
5.4.1 Definition . . . 129
5.4.2 Construction . . . 131
5.4.3 Efficient Instantiation . . . 135
5.4.4 Outsourced KZG Commitments . . . 136

5.5 zk-SNARKs with Server-Aided Proving 136
5.5.1 Definition 137
5.5.2 Server-Aided Proving for Nova 139
5.5.3 Server-Aided Proving for Groth16 141
5.5.4 Server-Aided Proving for Plonk 143
5.6 Implementation and Evaluation 144
5.6.1 EMSM and EKZG Schemes 146
5.6.2 Server-Aided Proving for zk-SNARKs 148

6 Conclusion and Future Work **151**

Bibliography **153**

Chapter 1

Introduction

Succinct non-interactive zero-knowledge arguments of knowledge (zk-SNARKs) enable a prover to convince a verifier of knowledge of a private witness for a given NP statement. They satisfy two properties that enable a broad range of applications. First, proofs are *succinct*; i.e., the proof size and verification time are sublinear in the size of the circuit that checks the NP relation. Second, proofs are *zero-knowledge* and reveal no information about the witness. Recent years have seen numerous practical zk-SNARK constructions [48, 72, 86, 96, 114, 132, 142, 146], a wide range of applications [3, 21, 32, 107, 120, 130, 139], and industrial deployments [8, 12, 118, 150].

Practical adoption of zk-SNARKs, however, is hindered by the high costs of proving and verification. Many applications of zk-SNARKs—such as private smart contracts [32], auditable key transparency [139], and verifiable machine learning [77]—are currently limited to only small instances due to this overhead. In this dissertation, we propose methodologies that improve the efficiency of zk-SNARKs and make them more suited for large-scale, real-world deployment.

Specifically, we first investigate two applications of *recursive* zk-SNARKs—a powerful primitive that enables succinct verification of incremental computations—and propose designs that move these applications toward deployment at scale. We then study *private outsourcing of zk-SNARKs*, where a prover/client delegates most of its proving work to more powerful, *untrusted* servers. We show how to realize this notion efficiently for several widely deployed zk-SNARKs in the *single-server* setting. Below, we provide a high-level overview of each of these.

1.1 Zero-Knowledge Proofs of Training for Neural Networks

A *zero-knowledge proof of training* (zk-PoT) enables an entity to prove that a model has been faithfully trained on a dataset without revealing any additional information about the model or the dataset. An ideal zk-PoT should provide provable security and privacy guarantees, succinct verification, and practical prover efficiency. In Chapter 3, we propose KAIZEN, a zk-PoT for neural networks with a linear-time prover and succinct proof size and verification runtime, both of which are independent of the number of training iterations and the size of the dataset.

Our construction relies on two key building blocks. First, we design a GKR-based proof system for gradient descent with a linear-time, concretely efficient prover. We then show how to recursively compose these proofs across multiple iterations to achieve succinctness. As an independent contribution, we present a framework for recursively composing GKR-based proofs, along with a linear-time accumulation scheme for multilinear polynomial commitments.

Our benchmarks show that KAIZEN can handle the training of complex neural networks such as VGG-11 with 10 million parameters and a batch size of 16. The prover runtime is 15 minutes per iteration, which is $24\times$ faster than state-of-the-art generic recursive proofs while reducing prover memory usage by $27\times$. The proof size is 1.63 megabytes, and the verifier runtime is only 130 milliseconds; note that both are independent of the number of iterations.

This chapter is based on a paper co-authored with Christodoulos Pappas, Jonathan Katz, and Dimitrios Papadopoulos, published in the proceedings of the *ACM SIGSAC Conference on Computer and Communications Security (CCS)* 2024 [3].

1.2 Secure Aggregation with a Lightweight Committee

Secure aggregation enables a server to learn the sum of private inputs of clients while revealing no information beyond the final sum. Recent work, Willow [17], achieves one-shot single-server secure aggregation with dynamic participation. To achieve security under these features, Willow relies on a committee to verify the correctness of the aggregation. Although this verification requires no private information—thereby broadening the set of parties that can serve on the committee—the committee's total work scales linearly with the number of clients, which poses a challenge for large-scale deployments. This linear committee-cost bottleneck is also shared by other state-of-the-art protocols, including Flamingo [112] and OPA [94].

In Chapter 4, we introduce WILLOWFOLD, a secure aggregation scheme based on Willow that enables lightweight verification of the aggregation with only logarithmic committee cost in the number of clients. Our scheme further reduces the server's per-client storage overhead to a single identifier (4 bytes), rather than full contributions as in Willow (1.5 KB). Similar to KAIZEN, WILLOWFOLD also employs recursive zk-SNARKs to achieve succinctness.

We instantiate WILLOWFOLD using KZH-Fold [93] as the underlying recursive proof. As independent contributions, we present a zero-knowledge variant of KZH-Fold with logarithmic randomness, and we further prove that this variant satisfies simulation extractability.

We implement and evaluate WILLOWFOLD, showing that it supports 8 million clients with proof size < 60 KB and verifier time under one second—a 10^5-fold improvement over Willow.

This chapter is based on a paper co-authored with Hossein Hafezi, Adrià Gascón, Philipp Schoppmann, Mariana Raykova, and Benedikt Bünz. The paper is currently under review for *Annual International Cryptology Conference (CRYPTO)* 2026 [87].

1.3 Single-Server Private Outsourcing of zk-SNARKs

A major barrier to the practical deployment of zk-SNARKs is their high proving cost. With this motivation, we study *server-aided* zk-SNARKs, in which a prover (client) outsources most of the proving workload to a *single, untrusted* server, while the server learns nothing about the witness or even the final proof. In Chapter 5, we formalize this notion and show how to realize server-aided proving for widely deployed zk-SNARKs such as Nova, Groth16, and Plonk.

The key building block underlying our designs is a new primitive, *encrypted multi-scalar multiplication* (EMSM), that enables private delegation of multi-scalar multiplications (MSMs). We construct an EMSM from variants of the *learning parity with noise* assumption in which the client does $O(1)$ group operations, while the server's work matches that of the plaintext MSM.

We implement and evaluate our constructions. Compared to local proving, our techniques lower the client's computation by up to $20\times$ and reduce the proving latency by up to $9\times$.

This chapter is based on a paper co-authored with Hossein Hafezi, Jonathan Katz, and Sarah Meiklejohn, accepted to the *IEEE Symposium on Security and Privacy (S&P)* 2026 [2].

Chapter 2

Preliminaries

This chapter introduces the basic notation and primitives used in subsequent chapters.

2.1 Notation

We define $[n] := \{0, 1, \cdots, n-1\}$ for zero-indexed ranges. For other ranges, we explicitly write the (inclusive) endpoints, e.g., $[1, n]$. We write $x := z$ for deterministic assignment, and $x \leftarrow \mathsf{Alg}(z)$ to mean that x is set to the output of running a (possibly randomized) algorithm Alg on input z; we write $\mathsf{Alg}(z; r)$ when the randomness r is explicit. We let $k \xleftarrow{\$} \mathcal{K}$ denote uniform sampling from a set $\mathcal{K}$. We use bold lowercase letters for column vectors (e.g., $\mathbf{a}$) and bold uppercase letters for matrices (e.g., $\mathbf{A}$). We write a_i for the ith entry of $\mathbf{a}$, and $A_{i,j}$ for the entry of $\mathbf{A}$ in the ith row and jth column; $\mathbf{A}_{i,*}$ and $\mathbf{A}_{*,j}$ denote the ith row and the jth column of $\mathbf{A}$, respectively. We use $\mathrm{wt}(\mathbf{a})$ to denote the Hamming weight of $\mathbf{a}$. We let $\odot$ denote the element-wise product. We let λ be the security parameter, and $\mathsf{negl}(\lambda)$ be a negligible function in λ. An algorithm is *efficient* if its runtime is polynomial in λ and in its input length.

Groups and Fields. $\mathbb{F}_q$ denotes the field of integers modulo a prime q, with $\mathbb{F}_q^* = \mathbb{F}_q \setminus \{0\}$. $\mathbb{F}_\mu^{\leq d}[X]$ denotes the set of multivariate polynomials in μ variables with variable degree at most d. We let $\mathbb{G}$ denote a cyclic group of prime order q with generator $g \in \mathbb{G}$, written additively. We use $e \colon \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ to denote an efficiently computable, non-degenerate bilinear pairing map.

Multilinear Extensions. A *multilinear* polynomial is a multivariate polynomial with variable degree 1. A multilinear polynomial $p(X)$ over ℓ variables can be uniquely determined by its 2^ℓ evaluations on the Boolean hypercube $\{0,1\}^\ell$. For $\boldsymbol{w} \in \{0,1\}^\ell$, let $\mathsf{eq}(\boldsymbol{w}, X)$ denote the multilinear Lagrange basis that equals 1 at $X = \boldsymbol{w}$ and 0 elsewhere in $\{0,1\}^\ell$. It holds that

$$\mathsf{eq}(\boldsymbol{w}, X) := \prod_{i=1}^{\ell} \big(X_i w_i + (1 - X_i)(1 - w_i)\big).$$

The *multilinear extension* of a vector $\boldsymbol{v} \in \mathbb{F}^{2^\ell}$ is the unique multilinear polynomial $\widetilde{v}(X)$ such that $\widetilde{v}(\boldsymbol{x}) = v_{\boldsymbol{x}}$ for all $\boldsymbol{x} \in \{0,1\}^\ell$, namely,

$$\widetilde{v}(X) := \sum_{\boldsymbol{w} \in \{0,1\}^\ell} \mathsf{eq}(\boldsymbol{w}, X) \cdot v_{\boldsymbol{w}}.$$

Similarly, the multilinear extension of a matrix $\mathbf{M} \in \mathbb{F}^{2^\ell \times 2^k}$ is the unique multilinear polynomial $\widetilde{\mathbf{M}}(X, Y)$ such that $\widetilde{\mathbf{M}}(\boldsymbol{x}, \boldsymbol{y}) = M_{\boldsymbol{x},\boldsymbol{y}}$ for all $\boldsymbol{x} \in \{0,1\}^\ell$ and $\boldsymbol{y} \in \{0,1\}^k$, that is,

$$\widetilde{\mathbf{M}}(X, Y) := \sum_{\boldsymbol{w} \in \{0,1\}^\ell} \sum_{\boldsymbol{u} \in \{0,1\}^k} \mathsf{eq}(\boldsymbol{w}, X)\, \mathsf{eq}(\boldsymbol{u}, Y) \cdot M_{\boldsymbol{w},\boldsymbol{u}}.$$

Merkle Tree. A Merkle tree [115] is an authenticated data structure with the following syntax.

- Setup: on input the security parameter 1^λ, outputs the specification of a hash function h.
- Build: on input h and a vector $\mathbf{x}$, builds a Merkle tree over $\mathbf{x}$ using h and outputs the root ρ.
- Open: on input the tree and an index i, outputs the leaf x_i and the path p_i from x_i to the root ρ.
- Verify: on input h, ρ, i, x_i, and p_i, verifies the path p_i and outputs 1 (accept) or 0 (reject).

2.2 Argument Systems

A non-interactive *argument* for a relation $\mathcal{R}$ enables a prover $\mathcal{P}$ to convince a verifier $\mathcal{V}$ that there exists a witness w such that $(x, w) \in \mathcal{R}$, where x is the public instance. More formally, it has the following syntax:

- $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$: on input the security parameter 1^λ, outputs public parameters pp.
- $\pi \leftarrow \mathsf{Prove}(\mathsf{pp}, x, w)$: on input pp, an instance x, and witness w, outputs a proof π.
- $\{0, 1\} \leftarrow \mathsf{Verify}(\mathsf{pp}, x, \pi)$: on input pp, x, and π, outputs either 1 (accept) or 0 (reject).

We say that an argument satisfies *completeness* if, in an honest execution, whenever the prover holds a valid witness, the verifier accepts. An argument satisfies *soundness* if—except with negligible probability—no efficient adversary prover $\mathcal{P}^\star$ can produce an accepting proof for an unsatisfiable instance x. *Knowledge soundness* further requires the existence of an efficient *extractor* such that, whenever an efficient $\mathcal{P}^\star$ convinces $\mathcal{V}$ to accept, the extractor can use $\mathcal{P}^\star$ to compute a corresponding witness, except with negligible probability. An argument is *zero knowledge* if there exists an efficient *simulator* that, given only the instance x, can generate a proof π that is indistinguishable from a real proof. Below, we formalize these properties.

Definition 1 (Non-Interactive Zero-Knowledge Arguments of Knowledge)**.** *A non-interactive argument* $\Pi = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ *for a relation* $\mathcal{R}$ *with language* L_R *satisfies completeness, soundness, knowledge soundness, and zero knowledge if it meets the following conditions.*

- *(Perfect) Completeness: For every $(x, w) \in R$,*

$$\Pr\left[\begin{array}{l} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda), \\ \\ \pi \leftarrow \mathsf{Prove}(\mathsf{pp}, x, w) \end{array} : \mathsf{Verify}(\mathsf{pp}, x, \pi) = 1\right] = 1.$$

- *(Adaptive Computational) Soundness: For every efficient adversary $\mathcal{P}^\star$,*

$$\Pr\left[\begin{array}{ll} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda), & x \notin L_R, \\ & \\ (x, \pi) \leftarrow \mathcal{P}^\star(\mathsf{pp}) & \mathsf{Verify}(\mathsf{pp}, x, \pi) = 1 \end{array}\right] \leq \mathsf{negl}(\lambda).$$

- *(Adaptive Computational) Knowledge Soundness: There exists an efficient extractor algorithm $\mathcal{E}$ such that, for every efficient adversary $\mathcal{P}^\star$ and for all randomness ρ,*

$$\Pr\left[\begin{array}{ll} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda), & \\ & (x, w) \notin \mathcal{R}, \\ (x, \pi) \leftarrow \mathcal{P}^\star(\mathsf{pp}; \rho), & \\ & \mathsf{Verify}(\mathsf{pp}, x, \pi) = 1 \\ w \leftarrow \mathcal{E}(\mathsf{pp}, x, \pi; \rho) & \end{array}\right] \leq \mathsf{negl}(\lambda).$$

- *(Adaptive Multi-Theorem Computational) Zero Knowledge: There exists an efficient simulator $\mathcal{S} = (\mathcal{S}_0, \mathcal{S}_1)$ such that for every efficient adversary $\mathcal{A}$,*

$$\Big|\Pr\left[\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda) : \mathcal{A}^{\mathsf{Prove}(\mathsf{pp},\cdot,\cdot)}(\mathsf{pp}) = 1\right] -$$
$$\Pr\left[(\mathsf{pp}, \tau) \leftarrow \mathcal{S}_0(1^\lambda) : \mathcal{A}^{\mathcal{S}_1(\mathsf{pp},\cdot,\cdot)}(\mathsf{pp}) = 1\right]\Big| \leq \mathsf{negl}(\lambda),$$

where the simulator algorithm $\mathcal{S}_1$ ignores its third input argument.

Succinctness requires the proof size to be $o(|x| + |w|)$ and the verifier time to be $|x| + o(|w|)$.

Π_{SUM}

The protocol proceeds in ℓ rounds.

1. In the first round, $\mathcal{P}$ sends the polynomial

$$f_1(x_1) := \sum_{b_2,\ldots,b_\ell \in \{0,1\}} f(x_1, b_2, b_3, \ldots, b_\ell).$$

   $\mathcal{V}$ checks if $H = f_1(0) + f_1(1)$ and sends $r_1 \leftarrow \mathbb{F}$.

2. In the ith round ($2 \leq i \leq \ell - 1$), $\mathcal{P}$ sends the polynomial

$$f_i(x_i) := \sum_{b_{i+1},\ldots,b_\ell \in \{0,1\}} f(r_1, \ldots, r_{i-1}, x_i, b_{i+1}, \ldots, b_\ell).$$

   $\mathcal{V}$ checks if $f_{i-1}(r_{i-1}) = f_i(0) + f_i(1)$ and sends $r_i \leftarrow \mathbb{F}$.

3. In the ℓth round, $\mathcal{P}$ sends the polynomial

$$f_\ell(x_\ell) = f(r_1, r_2, \ldots, x_\ell).$$

   $\mathcal{V}$ checks if $f_{\ell-1}(r_{\ell-1}) = f_\ell(0) + f_\ell(1)$. The verifier samples the final random challenge $r_\ell \in \mathbb{F}$. Given oracle access to f, $\mathcal{V}$ accepts if and only if $f_\ell(r_\ell) = f(r_1, r_2, \ldots, r_\ell)$.

Figure 2.1: Sumcheck protocol.

The Sumcheck Protocol. Fix an ℓ-variate polynomial $f : \mathbb{F}^\ell \to \mathbb{F}$ with variable-degree bound d. The sumcheck protocol [109] (see Figure 2.1) enables a verifier $\mathcal{V}$ to delegate the computation of the sum of f over the binary hypercube, i.e., $H = \sum_{b \in \{0,1\}^\ell} f(b)$, to a prover $\mathcal{P}$. At the end of the protocol, $\mathcal{V}$ needs the ability to learn the evaluation of f at a single random point $r \in \mathbb{F}^\ell$. The prover cost is $O(d^\ell)$, and the verifier cost is $O(d\ell)$. The soundness error is bounded by $O(d\ell/|\mathbb{F}|)$.

The Zerocheck Protocol. Let $f : \mathbb{F}^\ell \to \mathbb{F}$ denote an ℓ-variate polynomial with variable-degree bound d. The Zerocheck protocol enables a verifier to check that $f(\boldsymbol{x}) = 0$ for all $\boldsymbol{x} \in \{0,1\}^\ell$. One possible instantiation runs the sumcheck protocol on $0 = \sum_{\boldsymbol{x} \in \{0,1\}^\ell} f(\boldsymbol{x}) \cdot \mathsf{eq}(\boldsymbol{r}, \boldsymbol{x})$, for a uniform random challenge $\boldsymbol{r} \in \mathbb{F}^\ell$, yielding the same cost as a single sumcheck invocation.

2.3 Proof-Carrying Data

An *incrementally verifiable computation (IVC)* [140] scheme is a recursive argument system that enables verifiable execution of a sequence of computations. Let z_0 be an initial input, and recursively define $z_i := F_i(z_{i-1}, \omega_i)$, where F_i denotes the transition function and ω_i denotes the witness at step i. An IVC scheme produces proofs that each z_i has been computed correctly. Specifically, at step 1, a proof π_1 is generated for the computation $z_1 := F_1(z_0, \omega_1)$. Then, at each subsequent step i, the prover supplies the previous proof π_{i-1} and the witness ω_i. The IVC computes $z_i := F_i(z_{i-1}, \omega_i)$ and outputs a proof π_i attesting to the correctness of the transition computation and the satisfiability of the proof π_{i-1}. By definition, each π_i recursively attests to the correctness of all steps 1 to i. In a *uniform* IVC, all steps use the same transition function.

Proof-carrying data (PCD) [53] generalizes IVC to arbitrary directed acyclic graphs (DAGs) of computation. Each node v in the DAG is associated with a transition function F_v and a local witness ω_v. Suppose v has k incoming edges carrying messages $z_{e_1}, \ldots, z_{e_k}$ and corresponding proofs $\pi_{e_1}, \ldots, \pi_{e_k}$. The PCD scheme computes an outgoing message $z_e := F_v(z_{e_1}, \ldots, z_{e_k}, \omega_v)$ and produces a proof π_e that attests to (i) for all $i \in [1, k]$, the incoming proofs π_{e_i} are satisfiable, and (ii) the local transition is correct. Consequently, each sink output would be accompanied by a proof that recursively implies the correctness of the transition at every node along its ancestry. We capture the validity of each transition via a compliance predicate φ_v such that $\varphi_v(z_e, z_{e_1}, \ldots, z_{e_k}, \omega_v) = 1$ if and only if the transition $z_e = F_v(z_{e_1}, \ldots, z_{e_k}, \omega_v)$ is valid. As before, in a uniform PCD scheme, all nodes share the same transition function, i.e., $F_v = F$ for all v. Below, we provide a formal description of a PCD scheme. To this end, we first need to define *proof-carrying transcripts* and the notion of *compliant* proof-carrying transcripts.

Definition 2 (Proof-Carrying Transcripts)**.** *A proof-carrying transcript $\mathcal{T}$ is a directed acyclic graph in which each vertex $v \in V(\mathcal{T})$ is labeled with a local witness $w_v \in \mathcal{W}$, and each edge $e \in E(\mathcal{T})$ is labeled with an output message $z_e \in \mathcal{Z}$. The output of the transcript $\mathcal{T}$ is denoted by $o(\mathcal{T})$, defined as the output message carried by the lexicographically first sink of $\mathcal{T}$. The fan-in of the transcript $\mathcal{T}$ denotes the maximum number of incoming edges for any vertex.*

Definition 3 (Compliant Proof-Carrying Transcripts)**.** *We denote by $\mathcal{R}$ a class of compliance predicates $\varphi : \mathcal{Z} \times \mathcal{Z}^k \times \mathcal{W} \to \{0,1\}$. Given a proof-carrying transcript $\mathcal{T}$ with fan-in k, we say that a vertex $v \in V(\mathcal{T})$ with witness w_v is φ_v-compliant for a $\varphi_v \in \mathcal{R}$ if, for every outgoing edge $e = (v, u) \in E(\mathcal{T})$ labeled with message z_e, one of the following two conditions holds:*

- *v has no incoming edges.*
- *v has k incoming edges $e_1, \ldots, e_k$, each labeled z_{e_i}, such that $\varphi_v(z_e, z_{e_1}, \ldots, z_{e_k}, w_v) = 1$.*

Define $\boldsymbol{\varphi} := (\varphi_{v_1}, \ldots, \varphi_{v_n})$ such that each φ_{v_i} is the compliance predicate associated with the node $v_i \in V(\mathcal{T})$. Then, we say that $\mathcal{T}$ is $\boldsymbol{\varphi}$-compliant if every node v_i is φ_{v_i}-compliant.

Definition 4 (Proof-Carrying Data)**.** *Let $\mathcal{T}$ be a $\boldsymbol{\varphi}$-compliant proof-carrying transcript. A PCD scheme consists of a set of arguments $\{\Pi_v\}_{v \in V(\mathcal{T})}$, such that each Π_v proves that the subgraph $\mathcal{H}_v$ induced by v and its ancestors is $\boldsymbol{\varphi}_{\mathcal{H}_v}$-compliant, where $\boldsymbol{\varphi}_{\mathcal{H}_v}$ denotes the restriction of $\boldsymbol{\varphi}$ to the subgraph $\mathcal{H}_v$. The PCD scheme is knowledge-sound if each Π_v is an argument of knowledge, and it is zero-knowledge if, for all $v \in V(\mathcal{T})$ with a non-empty local witness, Π_v is zero-knowledge.*

A conventional approach to building IVC/PCD is via recursive composition of SNARKs [26], where the SNARK verifier is encoded within the circuit run at each node. Recent works [40, 42, 96, 97] take a more efficient approach and realize IVC/PCD from accumulation schemes.

Informally, an accumulation scheme takes multiple input proofs—possibly for different predicates—and folds them into a single proof. To avoid trivial constructions, we require that the folded proof be strictly smaller than the collection of input proofs, and that verifying the folded proof—including running the accumulation verifier—be faster than verifying all input proofs individually. To realize an IVC/PCD, each node accumulates its incoming proofs and passes the result to subsequent nodes; rather than verifying each incoming proof, the recursive circuit runs only the accumulation verifier. Below, we give a formal description of accumulation schemes.

Accumulation Schemes. An accumulation scheme has the following syntax:

- $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$: on input the security parameter 1^λ, outputs public parameters pp.
- $(\mathsf{acc}, \pi) \leftarrow \mathsf{Prove}(\mathsf{pp}, (\mathsf{acc}_i)_{i\in[k]})$: on input pp and a collection of accumulators $(\mathsf{acc}_i)_{i\in[k]}$ with $\mathsf{acc}_i = (\mathsf{acc}_i.x, \mathsf{acc}_i.w)$, where $\mathsf{acc}_i.x$ denotes the instance and $\mathsf{acc}_i.w$ denotes the witness, outputs a folded accumulator $\mathsf{acc} = (\mathsf{acc}.x, \mathsf{acc}.w)$ and an accumulation proof π.
- $\mathsf{acc}.x \leftarrow \mathsf{Verify}(\mathsf{pp}, (\mathsf{acc}_i.x)_{i\in[k]}, \pi)$: On input pp and a collection of accumulator instances $(\mathsf{acc}_i.x)_{i\in[k]}$, outputs the instance of the folded accumulator $\mathsf{acc}.x$.
- $\{0,1\} \leftarrow \mathsf{Decide}(\mathsf{pp}, \mathsf{acc})$: on input pp, and acc, outputs either 1 (accept) or 0 (reject).

We say that an accumulation scheme satisfies completeness and knowledge soundness if the accumulation proof π is an argument of knowledge for the following relation:

$$\Big(\mathsf{Decide}(\mathsf{pp}, \mathsf{acc}) = 1 \ \wedge \ \mathsf{acc}.x \leftarrow \mathsf{Verify}(\mathsf{pp}, (\mathsf{acc}_i.x)_i, \pi)\Big) \implies \forall i : \ \mathsf{Decide}(\mathsf{pp}, \mathsf{acc}_i) = 1.$$

In an accumulation-based IVC/PCD, accumulator instances $\mathsf{acc}.x$ carry the computation outputs, and witnesses $\mathsf{acc}.w$ carry proofs of computation. Verify is encoded in the recursive circuit.

2.4 Commitment Schemes

A *commitment scheme* for a message space $\mathcal{M}$ consists of the following algorithms:

- $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$: on input the security parameter λ, outputs public parameters pp.
- $c \leftarrow \mathsf{Commit}(\mathsf{pp}, m; r)$: on input pp, any message $m \in \mathcal{M}$, and randomness $r \in \mathcal{R}$, outputs a commitment c, where $\mathcal{R}$ denotes the randomness space.

A commitment can be opened by sending m and r. A commitment is *hiding* if no efficient adversary can distinguish which of two messages corresponds to a particular commitment. It is *binding* if no efficient adversary can output a commitment that opens to two distinct messages.

Definition 5 (Commitment Schemes)**.** *A commitment scheme* $\Sigma = (\mathsf{Setup}, \mathsf{Commit})$ *for a given message space* $\mathcal{M}$ *satisfies binding and hiding if it meets the following conditions.*

- *(Computational) Binding: For every efficient adversary* $\mathcal{A}$*,*

$$\Pr\left[\begin{array}{ll} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda) & \\ & : \quad m_0 \neq m_1, \\ (m_0, r_0, m_1, r_1) \leftarrow \mathbf{A}(\mathsf{pp}) & \quad \mathsf{Commit}(\mathsf{pp}, m_0; r_0) = \mathsf{Commit}(\mathsf{pp}, m_1; r_1) \end{array}\right] \leq \mathsf{negl}(\lambda).$$

- *(Computational) Hiding: For any* $m_0, m_1 \in \mathcal{M}$ *and every efficient distinguisher* $\mathcal{D}$*,*

$$\left|\Pr\left[\begin{array}{ll} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda), & \\ r_0 \xleftarrow{\$} \mathcal{R}, c_0 \leftarrow \mathsf{Commit}(\mathsf{pp}, m_0; r_0) & : \mathcal{D}(\mathsf{pp}, c_0) = 1 \end{array}\right] - \right.$$

$$\left.\Pr\left[\begin{array}{ll} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda), & \\ r_1 \xleftarrow{\$} \mathcal{R}, c_1 \leftarrow \mathsf{Commit}(\mathsf{pp}, m_1; r_1) & : \mathcal{D}(\mathsf{pp}, c_1) = 1 \end{array}\right]\right| \leq \mathsf{negl}(\lambda).$$

Pedersen Commitment Scheme. Pedersen commitment scheme [125] for the message space $\mathbb{F}_q^n$ uses uniform group generators $(h, g_0, g_1, \cdots, g_{n-1}) \in \mathbb{G}^{n+1}$ as public parameters. Then, the commitment to $\mathbf{a} \in \mathbb{F}_q^n$ with randomness $r \in \mathbb{F}_q$ is defined as $c := rh + \langle \mathbf{a}, \mathbf{g} \rangle$. The commitments are perfectly hiding and computationally binding under the discrete logarithm assumption.

Polynomial Commitments. A polynomial commitment scheme (PCS) is a commitment scheme for the message space $\mathbb{F}_\mu^{\leq d}[X]$ that allows proving the evaluations of committed polynomials. More formally, a polynomial commitment scheme consists of the following algorithms:

- $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$: on input the security parameter λ, outputs public parameters pp.
- $c \leftarrow \mathsf{Commit}(\mathsf{pp}, f(X); r)$: on input pp, a polynomial $f(X) \in \mathbb{F}_\mu^{\leq d}[X]$, and randomness r, outputs a commitment c. (The randomness space depends on the commitment scheme.)
- $(y, \pi) \leftarrow \mathsf{Open}(\mathsf{pp}, c, f(X), x; r)$: on input pp, c, $f(X)$, r, and an evaluation point x, outputs the evaluation $y = f(x)$ and an opening proof π.
- $\{0, 1\} \leftarrow \mathsf{Verify}(\mathsf{pp}, c, x, y, \pi)$: on input pp, c, x, y, and π, outputs either 1 (accept) or 0 (reject).

We say that a polynomial commitment scheme is hiding if its commitments are hiding and, additionally, its opening proofs are zero-knowledge arguments of correct polynomial evaluation. Moreover, we say that a polynomial commitment scheme is extractable if its commitments are binding and, additionally, its opening proofs are arguments of knowledge of correct polynomial evaluation. Below, we recall the details of a widely used polynomial commitment scheme.

KZG Polynomial Commitment Scheme. Let $\mathbb{G}_1, \mathbb{G}_2$ be groups with generators $g_1 \in \mathbb{G}_1$ and $g_2 \in \mathbb{G}_2$, such that there is an associated bilinear pairing map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$. Then, the KZG polynomial commitment scheme [95] for univariate polynomials of degree d works as follows:

- Setup: output $\mathsf{pp} := (g_1, \tau \cdot g_1, \ldots, \tau^d \cdot g_1, \tau \cdot g_2)$, where $\tau \xleftarrow{\$} \mathbb{F}_q$ is a secret trapdoor.

- Commit: on input a polynomial $f(X) = \sum_{i=0}^{d} f_i X^i$, output a polynomial commitment

$$c := f(\tau) \cdot g_1 = \sum_{i=0}^{d} f_i \cdot (\tau^i \cdot g_1).$$

- Open: on input x, compute $y = f(x)$ and $q(X) = \sum_{i=0}^{d-1} q_i X^i = \frac{f(X)-y}{X-x}$, and output y and

$$\pi := q(\tau) \cdot g_1 = \sum_{i=0}^{d-1} q_i \cdot (\tau^i \cdot g_1).$$

- Verify: on input c, x, y, and π, output accept if and only if $e(\pi,\ \tau \cdot g_2 - x \cdot g_2) = e(c - y \cdot g_1,\ g_2)$.

KZG commitments are binding under the q-Strong Diffie–Hellman assumption. In the algebraic group model [70], they are extractable under the same assumption. Note that this variant of KZG is not hiding; however, if needed, we can add hiding using a single additional group element. For the use cases throughout this dissertation, only the non-hiding variant of KZG suffices.

2.5 Error-Correcting Codes

A *linear error-correcting code* $\mathcal{C}$ with block length N and dimension $n < N$ is a set of codewords $\{\mathbf{G}^T\mathbf{x} \mid \mathbf{x} \in \mathbb{F}^n\} \subset \mathbb{F}^N$, where $\mathbf{G} \in \mathbb{F}^{n \times N}$ is a *generator matrix* of $\mathcal{C}$. The *rate* of $\mathcal{C}$ is $R = n/N$. The *distance* of $\mathcal{C}$ is the minimum Hamming distance between any two codewords which, for a linear code, is equal to the minimum weight of a nonzero codeword, since $\mathbf{0}^N$ is a codeword. We define $\delta := \min_{\boldsymbol{v} \in \mathcal{C}} \mathrm{wt}(\boldsymbol{v})/N$ to be the *relative minimum distance* of the code $\mathcal{C}$.

2.6 Universal Composability Framework

The universal composability (UC) framework [46] formalizes security under composition. In particular, the UC framework specifies the security requirements of a protocol Π via an *ideal functionality* $\mathcal{F}$. We say that Π *UC-realizes* $\mathcal{F}$ if, for any efficient adversary $\mathcal{A}$, there exists an efficient *simulator* Sim such that, for any efficient environment $\mathcal{Z}$ with arbitrary auxiliary input z, the output of $\mathcal{Z}$ in the real-world execution—where the parties execute Π and interact with $\mathcal{A}$—is computationally indistinguishable from the output of $\mathcal{Z}$ in the ideal-world execution—where the parties interact with Sim and $\mathcal{F}$. The environment $\mathcal{Z}$ models arbitrary concurrent executions.

Parties in the UC framework are modeled as probabilistic interactive Turing machines with input/output and incoming/outgoing communication tapes. We assume that all communication is authenticated; that is, an adversary can delay but cannot forge or modify messages sent by honest parties. The adversary may be adaptive and corrupt parties at any point during protocol execution. We also consider the $\mathbf{U}$-hybrid model, in which all parties running the protocol, as well as the adversary, simulator, and environment, have oracle access to an auxiliary functionality $\mathbf{U}$.

The environment $\mathcal{Z}$ may read the output tapes of the parties, $\mathcal{A}$, and Sim, and write to their input tapes. The simulator Sim can observe the communication between parties and the ideal functionality and may corrupt parties at any point during the ideal-world execution.

Chapter 3

Zero-Knowledge Proofs of Training for Neural Networks

Machine learning with deep neural networks (DNNs) has received unprecedented attention in recent years. At the same time, their widespread use has raised concerns about the provenance and integrity of DNN models; see, e.g., a recent Google blog post [84] regarding integrity issues in the broader context of machine learning models. Such concerns can arise at any stage of the model-development pipeline. In this chapter, we focus on *proofs of training* (PoTs).

Informally, a PoT enables a prover to convince a verifier that a model $\mathcal{M}$ has been trained correctly using a publicly specified learning algorithm—i.e., with public parameters such as the architecture and batch size—on a particular dataset $\mathcal{D}$. When the model or dataset is private, one may additionally consider *zero-knowledge* PoTs (zk-PoTs), which reveal no information about $\mathcal{M}$ or $\mathcal{D}$ beyond the correctness of training. PoTs can be used to substantiate the provenance of a released model; this can be valuable even when both the model and dataset are public, since training can be far more expensive than verifying that training was performed correctly.

PoTs can further be useful for other real-world applications:

- *Arbitrating Copyright Claims:* Concerns have grown about models being trained on copyrighted data [89]. A model owner may, therefore, want to prove that their model is trained *without* using a particular copyrighted data item. To do so, the owner can commit to the training dataset $\mathcal{D}$, prove—e.g., via a Merkle tree—that the item in question is not contained in $\mathcal{D}$, and then use a PoT to show that the model is trained on $\mathcal{D}$, which excludes the disputed item.

- *Distributed Training:* An entity may distribute training across multiple workers [104], e.g., by sending each worker a portion of the dataset and having them return partial results that are aggregated into a final model. PoTs enable workers to prove that the partial models are correct.

- *Proof of Ownership:* A machine learning model $\mathcal{M}$ may be stolen via extraction attacks [138]. In such cases, the legitimate owner can show that they trained $\mathcal{M}$ by committing to the dataset $\mathcal{D}$ and using a PoT to prove that $\mathcal{M}$ was trained on $\mathcal{D}$. An adversary cannot generate such a proof without access to $\mathcal{D}$; and even with access, they still need to incur the cost of training the model from scratch, undermining the motivation for the attack in the first place.

An ideal zk-PoT would offer strong security guarantees, be succinct, and impose minimal overhead on the prover. As we discuss in Section 3.1, however, existing techniques fall short of achieving at least one of these properties for PoTs of DNNs.

Our Contributions. In this work, we address the above limitation and propose techniques to realize efficient and succinct zk-PoTs for DNNs with provable security guarantees. We also implement and evaluate KAIZEN, a concrete instantiation of our approach. DNNs are typically trained using multiple iterations of the gradient descent algorithm. Our work comprises two main components: an optimized proof system for individual gradient-descent steps and a framework for efficient recursive composition of proofs across multiple iterations.

- *Efficient Proofs of Gradient Descent:* First, we design an efficient *proof of gradient descent* (PoGD)—a scheme for proving the correctness of a gradient-descent step. Our PoGD is built on GKR-based zk-SNARKs [146, 152, 153], which are known for their fast prover time. Moreover, these zk-SNARKs support optimizations [107, 137] that enable sublinear proof-generation time for matrix operations, e.g., convolution, which dominate gradient descent.

- *Recursive Composition of PoGDs:* Although our PoGD yields succinct proofs for individual gradient-descent steps, it is not sufficient to realize a succinct PoT. Training a DNN involves many iterations, typically linear in the size of the dataset. Thus, including a PoGD for every iteration would yield a proof whose size is linear in the dataset size. Alternatively, applying a single GKR-based zk-SNARK to the entire training computation would incur linear prover memory and verifier costs; note, GKR-based verifiers run in time linear in the circuit depth.

  To achieve succinct prover memory, proof size, and verification cost, we employ proof-carrying data (PCD). The challenge is that GKR-based proofs, such as our PoGD, have a relatively large verifier. Therefore, a naive approach that builds PCD directly from PoGDs—by embedding the PoGD verifier circuit into the recursive circuit—incurs substantial recursion cost. To overcome this, we construct an accumulation scheme for GKR-based proofs and then build our PCD via the accumulation approach (see Section 2.3 for details on accumulation-based PCDs).

  A difficulty is that GKR-based proofs use *multivariate* polynomial commitments rather than the *univariate* ones considered in prior work [30, 33]; existing accumulation schemes only handle univariate polynomials, and naively extending them to support multivariate polynomials would result in quasi-linear prover overhead. To mitigate this limitation and as a key building block for our PCD framework, we propose an accumulation scheme for multivariate polynomial commitments with *linear* prover overhead and *logarithmic* verifier complexity.

  To our knowledge, prior work has not considered building PCD from GKR-based proofs. Moreover, we instantiate polynomial commitments using hash-based schemes [83, 147], and our framework yields the first practically efficient PCD based on symmetric assumptions. We believe that our framework—and its core primitives—may be of independent interest.

- *Implementation and Evaluations.* We evaluate KAIZEN on well-known architectures, including the convolutional model VGG-11 with 10 million parameters, 11 layers, and batch size 16. The model is trained on the CIFAR-10 dataset. Our prover time is 15 minutes per iteration and is independent of the number of iterations and the dataset size; the proof size is 1.66 megabytes, and the verifier time is only 130 milliseconds. We compare KAIZEN with generic PCDs [33, 52, 96], achieving **24**× faster prover time and at least **27**× lower memory usage.

3.1 Related Work

Jia et al. [92] proposed a PoT for DNNs. However, their construction does not provide cryptographically strong security guarantees, and subsequent work [64, 65] has identified several concrete attacks against it. Moreover, their approach is neither zero-knowledge nor succinct: it reveals model weights and data items to the verifier, who must partially re-execute training.

More recently, Garg et al. [77] constructed zk-PoTs with provable security. Their scheme, however, is not succinct and therefore supports only simple learning algorithms (e.g., regression); extending the scheme to DNNs would incur prohibitive verification overhead.

In principle, zk-PoTs can be realized generically using zk-SNARKs. However, current systems remain insufficiently scalable to instantiate zk-PoTs for complex models, e.g., DNNs.

Another line of research studies zero-knowledge proofs of inference [13, 101, 107, 143]. These works address a different and easier problem than the one we consider here. Although some of their optimizations may be useful in our setting, they are not sufficient to obtain efficient zk-PoTs for DNNs. In particular, DNN training is inherently iterative, and a single training iteration can be orders of magnitude (e.g., up to 100×) more expensive than inference.

Eisenhofer et al. [61] applied generic SNARKs to verifiable unlearning. Since the targeted unlearning workload is comparatively lightweight, generic proofs can support it efficiently.

Concurrent Work. Bünz et al. [43] propose a PCD construction based on the GKR protocol via the accumulation approach. Due to their use of the ProtoStar compiler [40], their construction has quasi-linear prover time in the iteration size; in contrast, our scheme offers linear-time prover.

Bünz et al. [44] give an accumulation scheme for hash-based vector commitments and then show how to compile it into an IVC/PCD. Their recursion technique is similar to ours but requires commitments to vectors of the same size as the iteration circuit, whereas in our work, only the inputs and outputs of the circuit are committed. Thus, when applied to complex computations such as DNN gradient descent, their scheme incurs a high concrete prover cost.

3.2 Technical Overview

In this section, we provide an overview of our techniques.

3.2.1 Sumcheck-Based Proofs of Gradient Descent

Our PoGD is a GKR-based proof for one iteration of the gradient-descent algorithm. Given a fixed batch of data points and some model weights, PoGD enables the prover to update the weights via a gradient-descent iteration and then convinces the verifier that the update has been executed correctly. For the linear portions of the algorithm, we use optimized sumcheck-based proofs with sublinear proof-generation time. For the non-linear operations, we use bit decompositions and the generic GKR protocol. We next discuss each of the components.

Linear Operations. Linear operations, such as matrix multiplication or convolution, are the most computationally intensive parts of the gradient-descent algorithm. Prior work [107,137] has

shown optimized sumcheck protocols for these operations with proof-generation time sublinear in the runtime of the computation itself. In particular, Thaler [137] proposed a sumcheck proof for the multiplication of two $n \times n$ matrices with $O(n^2)$ prover time; note that the underlying computation takes time $\Theta(n^3)$ if naive matrix multiplication is used, and even the best matrix multiplication algorithms require time $\omega(n^2)$. Recently, Liu et al. [107] presented a sumcheck protocol for the convolution of an $n \times n$ input matrix with $w \times w$ weights, where the prover time is $O(n^2 + w^2)$, asymptotically faster than computing the convolution itself in $O(n^2 \cdot w^2)$ time.

Handling Non-Linear Operations. Sumcheck proofs support arithmetic operations over finite fields. However, neural networks use non-linear operations that cannot be compactly encoded in an arithmetic circuit, even once the underlying values have been quantized. We employ a standard technique to handle these. For example, proofs of comparisons such as $a \geq 0$ can be handled using bit decomposition: the prover provides the binary representation of a (including a sign bit) as an auxiliary input, and the verification circuit checks that the binary representation is consistent. The sign bit is then taken as the result of the comparison. To reduce the size of the circuit and wiring predicates, we express consistency checks directly as sumcheck instances rather than reducing them to an arithmetic circuit. We also handle exponentiations by piece-wise linear approximations, which have been shown to provide sufficient accuracy [47, 62, 119].

3.2.2 Recursive Composition of Sumcheck Proofs

A challenge in applying recursive proof composition to our PoGD is the relatively large size of its verification circuit, which consists of three main components: verifying the opening of polynomial commitments, verifying sumcheck messages, and verifying the hashes used to generate challenges in the Fiat–Shamir transform. We discuss how to improve the efficiency of each

component. In particular, we reduce the overhead of commitment openings using foldable polynomial commitments, the cost of verifying sumcheck messages using an evaluation-reduction technique, and the Fiat–Shamir overhead using sumcheck-friendly hash functions.

Commitment Openings. For a GKR-based proof, the verifier must verify the openings of several polynomial commitments. It is thus desirable to have a polynomial commitment scheme with a small verification overhead. Known constructions fall short: schemes based on pairings [95] have $O(1)$ verification overhead but are not compatible with recursive proof composition [23]; hash-based schemes [20, 83, 153] require the verifier to compute $\Omega(\log^2 N)$ hashes for a polynomial of size N. Schemes based on prime-order groups [39, 142] have verification times $\Omega(\sqrt{N})$, making them impractical for recursive proof composition.

Prior IVC/PCDs [30, 33] addressed this problem via accumulation schemes for polynomial commitments. Roughly speaking, rather than verifying the openings of commitments at each step, the commitments and openings across all steps are accumulated, and the final commitment/opening is verified only at the end. However, the accumulation schemes used in prior work apply only to *univariate* polynomials, whereas applications to GKR-based proofs require commitments to *multivariate* polynomials. Naively extending existing univariate schemes to the multivariate case would incur superlinear, i.e., $\Theta(N \log N)$, prover time.

In this work, we propose an accumulation scheme compatible with multivariate polynomial commitments. For polynomials with N coefficients, our scheme achieves $O(N)$ prover time and $O(\log N)$ proof size and verifier time. Our design is inspired by an evaluation-reduction technique [82], which can be viewed as a special case of accumulation in which multiple evaluations of a single polynomial are combined into one. We generalize this idea to the setting of multiple

polynomials with multiple evaluations and also make the scheme zero-knowledge.

We note that linear-time accumulation in the case of multilinear polynomials has been considered in several prior works for different applications [48, 97]. However, our technique also applies to multivariate polynomials of higher degree.

Sumcheck Messages. The GKR verifier must evaluate the multilinear extensions of the wiring predicates at several random points, and these evaluations have a cost linear in the circuit size. Interactive implementations of GKR mitigate this cost by preprocessing these evaluations [152, 153]. For recursive composition, however, such preprocessing is not feasible.

We avoid this overhead via accumulation. Specifically, predicate evaluations are combined into a single evaluation by a sumcheck run, passed to the next step, and verified only at the end.

Fiat-Shamir Challenges. Due to the Fiat–Shamir transform, the GKR verifier must verify the evaluation of a hash function on many inputs. In particular, $O(d \cdot \log |\mathcal{C}|)$ hash evaluations must be verified, where d denotes the depth of $\mathcal{C}$. We use MiMC [6], a sumcheck-friendly hash function, to reduce this cost. MiMC applies a low-degree round function over multiple rounds. As observed in prior work [19], each round of MiMC can be represented directly as a sumcheck instance, rather than explicitly encoded as an arithmetic circuit, thereby further reducing the size of the recursion circuit.

Polynomially Many Iterations. Our accumulation scheme for polynomial commitments achieves soundness only for $O(1)$ steps; when the number of steps is large, attacks may arise [44]. To address this limitation, we use leaf-decoupled PCDs [26, 122]. Here, the actual gradient descent iterations lie at the leaves of the PCD tree, while each internal node only verifies and accumulates its children. To achieve constant depth, one can set the arity to be polynomially large.

3.2.3 Proofs of Training for Neural Networks

Our zk-PoT for DNNs is realized via recursive composition of our PoGDs across multiple iterations. In doing so, there is one additional complication: we need to ensure that the prover uses a (random) partition of the committed dataset in each epoch. This prevents the prover from biasing the model toward specific data points or omitting parts of the dataset. We ensure this as follows. The prover commits to the dataset using a Merkle tree. In each epoch, the prover partitions the dataset using a public "quasi-random" permutation—e.g., a cubic function—applied to the indices of the n items in the dataset; the key for the permutation is varied across epochs. During each iteration of an epoch, the prover provides the data items contained in the current partition, along with their Merkle proofs. As part of computing the gradient-descent iterations, we additionally check the correctness of the partition and the Merkle proofs.

3.3 Additional Preliminaries

In this section, we provide additional preliminaries used throughout this chapter.

3.3.1 Neural Networks

A neural network maps an input feature to a prediction via a sequence of transformations across L *layers*. The ℓth layer consists of a linear operation parameterized by $\mathbf{W}_\ell$, followed by a non-linear activation function. In a convolutional neural network (CNN), a layer is either *dense* or *convolutional*. In dense layers, the linear operation outputs $\mathbf{T} = \mathbf{W}\mathbf{U}$ given an input $\mathbf{U}$ and weights $\mathbf{W}$; in a convolutional layer, it outputs $\mathbf{T} = \mathbf{W} * \mathbf{U}$, where if $\mathbf{U}$ is of size $u \times u$ and $\mathbf{W}$ is of size $w \times w$, then the matrix $\mathbf{T}$ is of size $(u-w+1) \times (u-w+1)$ with $T_{i,j} = \sum_{a,b=0}^{w-1} U_{i+a,j+b} \cdot W_{a,b}$. We consider three activation functions: $\mathsf{ReLU}(x) = \max(x, 0)$, $\mathsf{Tanh}(x) = (e^{2x} - 1)/(e^{2x} + 1)$,

each applied coordinate-wise, and $\mathsf{Softmax}(x_i) = (\mathbf{e}^{x_i})/(\sum_j e^{x_j})$. A layer may also include a pooling operation that slides a *window* over the input and deterministically extracts one element from each window; for example, it outputs the maximum or average value in each window.

Training DNNs via Gradient Descent. When training a DNN, the architecture—i.e., the number of layers, the type of each layer, and an initial set of weights—is fixed in advance, and the goal is to optimize the weights at each layer to minimize a specified loss function $\mathcal{L}$ over the dataset. The most popular optimization method for DNNs is batch gradient descent [128], which proceeds in a series of *epochs*. In each epoch, the dataset $\mathcal{D}$ is randomly partitioned into batches $\mathbf{B}_1, \ldots, \mathbf{B}_k$, each containing a specified number N of data items. An epoch consists of k iterations. Let $\mathbf{U}_{i,0}$ be the concatenation of the input features of the data items in $\mathbf{B}_i$, and let $\mathbf{U}_{i,\ell}$ (for $\ell > 0$) be the concatenation of the outputs of the ℓth layer under the current weights. In the ith iteration, the weights $\{\mathbf{W}_{i-1,\ell}\}_{\ell=1}^{L}$ are updated to $\{\mathbf{W}_{i,\ell}\}_{\ell=1}^{L}$ via the following steps:

1. *Forward pass:* We feed $\mathbf{U}_{i,0}$ to the model, and for $\ell = 1$ to L, compute $\mathbf{U}_{i,\ell}$.

2. *Backward pass:* Let H_i be the average loss over the batch, and define $\mathbf{G}_{i,\ell} := \partial H_i/\partial \mathbf{W}_{i-1,\ell}$ and $\mathcal{R}_{i,\ell} := \partial H_i/\partial \mathbf{U}_{i,\ell-1}$. These are computed for $\ell = L$ to 1 as:

$$\mathbf{G}_{i,\ell} = \mathcal{R}_{i,\ell+1} \cdot \frac{\partial \mathbf{U}_{i,\ell}}{\partial \mathbf{W}_{i-1,\ell}}, \mathcal{R}_{i,\ell} = \frac{\partial \mathbf{U}_{i,\ell}}{\partial \mathbf{U}_{i,\ell-1}} \cdot \mathcal{R}_{i,\ell+1}. \tag{3.1}$$

   $\mathcal{R}_{i,L+1}$ is evaluated using the labels. $\partial \mathbf{U}_{i,\ell}/\partial \mathbf{W}_{i-1,\ell}$ and $\partial \mathbf{U}_{i,\ell}/\partial \mathbf{U}_{i,\ell-1}$ are evaluated by linear operations and non-linear activations on the inputs $\mathbf{U}_{i,\ell-1}$ and weights $\mathbf{W}_{i-1,\ell}$, respectively.

3. *Update:* Given a learning rate η_ℓ, the weights are updated as $\mathbf{W}_{i,\ell} = \mathbf{W}_{i-1,\ell} - \eta_\ell \cdot \overline{\mathbf{G}}_{i,\ell}$, where $\overline{\mathbf{G}}_{i,\ell}$ indicates the average of $\mathbf{G}_{i,\ell}$ over the data items in the batch.

Input: Weights $\{\mathbf{W}_{i-1,\ell}\}_{j=1}^{L}$ and Batch $\mathbf{B}_{i-1}$

Output: Updated weights $\{\mathbf{W}_{i,\ell}\}_{j=1}^{L}$

Forward pass: Let $\mathbf{U}_{i,0}$ be the concatenation of input features included in $\mathbf{B}_{i-1}$.

for $\ell = 1$ to L **do**
  if the ℓ-th layer is dense **then**
    $\mathbf{T}_{i,\ell} \leftarrow \mathbf{W}_{i-1,\ell}\mathbf{U}_{i,\ell-1}$
    $\mathbf{U}_{i,\ell} \leftarrow \mathsf{act}_\ell(\mathbf{T}_{i,\ell})$ ▷ $\mathsf{act}_\ell \in \{\mathsf{ReLU}, \mathsf{Tanh}, \mathsf{Softmax}\}$
  else if the ℓ-th layer is convolution **then**
    $\mathbf{T}_{i,\ell} \leftarrow \mathbf{W}_{i-1,\ell} * \mathbf{U}_{i,\ell-1}$
    $\mathbf{Q}_{i,\ell} \leftarrow \mathsf{act}_\ell(\mathbf{T}_{i,\ell})$ ▷ $\mathsf{act}_\ell \in \{\mathsf{ReLU}, \mathsf{Tanh}, \mathsf{Softmax}\}$
    $\mathbf{U}_{i,\ell} \leftarrow \mathsf{pool}_\ell(\mathbf{Q}_{i,\ell})$ ▷ $\mathsf{pool}_\ell \in \{\mathsf{MaxPool}, \mathsf{AvgPool}\}$
  end if
end for

Backward pass: Let $\mathcal{R}_{i,L+1} = \partial\mathcal{L}(\mathbf{U}_{i,L}, Y_i)/\partial U_{i,L}$, where Y_i includes labels in $\mathbf{B}_{i-1}$.

for $\ell = L$ to 1 **do**
  if the ℓ-th layer is dense **then**
    $\mathbf{T}'_{i,\ell} = \partial\mathbf{U}_{i,\ell}/\partial\mathbf{T}_{i,\ell} = \partial\mathsf{act}_\ell(\mathbf{T}_{i,\ell})/\partial\mathbf{T}_{i,\ell}$
    $\mathbf{G}_{i,\ell} = (\mathcal{R}_{i,\ell+1} \circ \mathbf{T}'_{i,\ell})\mathbf{U}_{i,\ell-1}^T$
    $\mathcal{R}_{i,\ell} = \mathbf{W}_{i-1,\ell}^T(\mathcal{R}_{i,\ell+1} \circ \mathbf{T}'_{i,\ell})$
  else if the ℓ-th layer is convolution **then**
    $\mathbf{Q}'_{i,\ell} = \partial\mathbf{U}_{i,\ell}/\partial\mathbf{Q}_{i,\ell} = \partial\mathsf{pool}_\ell(\mathbf{Q}_{i,\ell})/\partial\mathbf{Q}_{i,\ell}$
    $\mathbf{T}'_{i,\ell} = \partial\mathbf{U}_{i,\ell}/\partial\mathbf{T}_{i,\ell} = \mathbf{Q}'_{i,\ell} \circ (\partial\mathsf{act}_\ell(\mathbf{T}_{i,\ell})/\partial\mathbf{T}_{i,\ell})$
    $\mathbf{G}_{i,\ell} = (\mathcal{R}_{i,\ell+1} \circ \mathbf{T}'_{i,\ell}) * \mathbf{U}_{i,\ell-1}$
    $\mathcal{R}_{i,\ell} = \mathsf{pad}(\mathbf{W}_{i-1,\ell}^T) * (\mathcal{R}_{i,\ell+1} \circ \mathbf{T}'_{i,\ell})$
  end if
end for

Update weights: Let $\overline{\mathbf{G}}_{i,\ell}$ be the average of the gradients in $\mathbf{G}_{i,\ell}$ over the batch.

for $\ell = 1$ to L **do**
  $\mathbf{W}_{i,\ell} = \mathbf{W}_{i-1,\ell} - \eta\overline{\mathbf{G}}_{i,\ell}$ ▷ η denotes the learning rate
end for

Figure 3.1: Mini-batch gradient-descent algorithm.

A formal description of the algorithm is given in Figure 3.1. $\mathsf{pad}(\mathbf{W}_{i,\ell}^T)$ zero-pads $\mathbf{W}_{i,\ell}^T$ and then rotates it for π. More precisely, for $\mathbf{W}_{i,\ell}$ of size $w \times w$ and $\mathbf{U}_{i,\ell-1}$ of size $n \times n$, $\mathsf{pad}(\mathbf{W}_{i,\ell}^T)$ is of size $(2w-n) \times (2w-n)$. $\overline{\mathbf{G}}_{i,l}$ is the average gradient over the data points in the batch.

3.3.2 The GKR Protocol

Let $\mathcal{C}$ be a circuit of depth d over a field $\mathbb{F}$, with layer 0 as the output layer and layer d as the input layer, where each gate in the ith layer takes two inputs from the $(i+1)$th layer. Let S_i be the number of gates in the ith layer, and $s_i := \lceil \log S_i \rceil$. Define *wiring predicates* $add_i, mult_i : \{0,1\}^{s_i + 2s_{i+1}} \to \{0,1\}$, where add_i (resp., $mult_i$) takes as input a gate label z for layer i, and gate labels x, y for layer $i+1$, then outputs 1 if and only if gate z is an addition (resp., multiplication) gate with inputs x and y. Fixing some input to the circuit, we define $v_i : \{0,1\}^{s_i} \to \mathbb{F}$ so that $v_i(b)$ is the output of the bth gate in the ith layer. Note that

$$\widetilde{v}_i(z) = \sum_{x,y \in \{0,1\}^{s_{i+1}}} \widetilde{add}_i(z,x,y) \cdot (\widetilde{v}_{i+1}(x) + \widetilde{v}_{i+1}(y)) + \widetilde{mult}_i(z,x,y) \cdot \widetilde{v}_{i+1}(x) \cdot \widetilde{v}_{i+1}(y). \tag{3.2}$$

$\mathcal{P}$ can prove the evaluation of $\mathcal{C}$ as follows. Assume $\mathcal{V}$ has oracle access to $\widetilde{v}_0$ and $\widetilde{v}_d$ (i.e., the input and output). $\mathcal{V}$ samples a random challenge r_0 and evaluates $\widetilde{v}_0(r_0)$. Then, $\mathcal{P}, \mathcal{V}$ run the sumcheck protocol on Equation 3.2 for $i = 0$, at the end of which $\mathcal{V}$ asks $\mathcal{P}$ for two evaluations of $\widetilde{v}_1$, say at points $r_{1,0}, r_{1,1}$. Say $\mathcal{P}$ claims $y_0 = \widetilde{v}_1(r_{1,0})$ and $y_1 = \widetilde{v}_1(r_{1,1})$. The verifier then chooses $\gamma_0, \gamma_1 \leftarrow \mathbb{F}$ and the parties run the sumcheck protocol to prove that

$$\begin{aligned}
&\gamma_0 \cdot y_0 + \gamma_1 \cdot y_1 = \sum_{x,y \in \{0,1\}^{s_2}} \\
&(\gamma_0 \cdot \widetilde{add}_1(r_{1,0},x,y) + \gamma_1 \cdot \widetilde{add}_1(r_{1,1},x,y)) \cdot (\widetilde{v}_2(x) + \widetilde{v}_2(y)) \\
&+ (\gamma_0 \cdot \widetilde{mult}_1(r_{1,0},x,y) + \gamma_1 \cdot \widetilde{mult}_1(r_{1,1},x,y)) \cdot \widetilde{v}_2(x) \cdot \widetilde{v}_2(y).
\end{aligned}$$

This reduces the problem to a claim of two evaluations of $\widetilde{v}_2$. The parties continue this procedure for d iterations; at the final iteration, $\mathcal{V}$ can check the evaluations of $\widetilde{v}_d$ by its oracle access. The formal protocol is provided in Figure 3.2. The GKR protocol can be made zero-knowledge by using zero-knowledge sumchecks [31, 146] and zero-knowledge polynomial commitments to provide the oracle access. The protocol is made non-interactive using the Fiat-Shamir transform.

3.4 Proofs of Gradient Descent

We present *proofs of gradient descent* (PoGD), GKR-based proofs of gradient descent.

3.4.1 Fixed-Point Operations

We fix a field size p large enough to ensure no overflow. We quantize a real number r as a q-bit integer ($q \ll p$) with f-bit precision as follows: when $0 \leq r \leq 2^{q-f}$, map r to $s = \lfloor r \cdot 2^f \rceil \in \mathbb{F}_p$, and when $-2^{q-f} \leq r < 0$, map r to $s = p - \lfloor r \cdot 2^f \rceil \in \mathbb{F}_p$.

Integer Arithmetic and Comparison. Two numbers can be added by adding their quantizations. For the multiplication of numbers a and b with quantizations s_a and s_b, respectively, we need to scale $z = s_a s_b$ by a factor of 2^{-f}. For this, the prover provides $z_0, \ldots, z_{q-1}$, the bit-decomposition of z with a sign bit z_q where $z_q = 1$ indicates negativity; The verifier checks whether the provided bit-decomposition consists of binary values consistent with z and outputs the truncated output; i.e., if $z = (-1)^{z_q} \sum_{i=0}^{q-1} z_i \cdot 2^i$ outputs $(-1)^{z_q} \cdot \sum_{i=0}^{q-f-1} z_{i+f} \cdot 2^i$.

Say we need to check whether $z \geq 0$. The prover can provide the bit-decomposition of z, including a sign bit; this can be easily checked by the verifier, and the result of the comparison is then the sign bit. We can also handle divisions. To compute the quotient a/b, where s_a and s_b are the quantizations of a and b, respectively, the prover provides a quotient z and a remainder r.

Π_{GKR}

Let $\mathcal{C} : \mathbb{F}^{s_{in}} \to \mathbb{F}^{s_{out}}$ be a layered arithmetic circuit of depth d. $\mathcal{P}$ wants to convince $\mathcal{V}$ that $\mathcal{C}(\mathbf{in}) = \mathbf{out}$, where $\mathbf{in}$ is the input given by $\mathcal{V}$ and $\mathbf{out}$ is the output sent to $\mathcal{V}$. Without loss of generality, we pad s_{in} and s_{out} to powers of 2. The protocol proceeds as follows.

1. For $z \xleftarrow{\$} \mathbb{F}$, parties compute $\tilde{V}_0(z)$, where $\tilde{V}_0$ is the multilinear extension of array $\mathbf{out}$.
2. $\mathcal{P}$ and $\mathcal{V}$ invoke a sumcheck protocol on

$$\tilde{V}_0(z) = \sum_{x,y\in\{0,1\}^{s_1}} \tilde{add}_0(z,x,y) \cdot \left(\tilde{V}_1(x) + \tilde{V}_1(y)\right) + \tilde{mult}_0(z,x,y) \cdot \tilde{V}_1(x) \cdot \tilde{V}_1(y),$$

   where $\mathcal{V}$ receives two evaluations $\tilde{V}_1(u^{(1)})$, $\tilde{V}_1(v^{(1)})$. $\mathcal{V}$ computes $\tilde{add}_0(z,u^{(1)},v^{(1)})$ and $\tilde{mult}_0(z,u^{(1)},v^{(1)})$ locally and checks that their consistency with $\tilde{V}_1(u^{(1)})$, $\tilde{V}_1(v^{(1)})$.

3. For $i = 1, 2, \ldots, d$:

   - $\mathcal{V}$ sends $\alpha^{(i)} \xleftarrow{\$} \mathbb{F}$ and $\beta^{(i)} \xleftarrow{\$} \mathbb{F}$ to $\mathcal{P}$.
   - $\mathcal{P}$ and $\mathcal{V}$ invoke a sumcheck protocol on

$$\begin{aligned}
&\alpha^{(i)}\tilde{V}_i(u^{(i)}) + \beta^{(i)}\tilde{V}_i(v^{(i)}) \\
&= \sum_{x,y\in\{0,1\}^{s_{i+1}}} (\alpha^{(i)}\tilde{add}_i(u^{(i)},x,y) + \beta^{(i)}\tilde{add}_i(v^{(i)},x,y)) \cdot (\tilde{V}_{i+1}(x) + \tilde{V}_{i+1}(y)) \\
&+ (\alpha^{(i)}\tilde{mult}_i(u^{(i)},x,y) + \beta^{(i)}\tilde{mult}_i(v^{(i)},x,y)) \cdot \tilde{V}_{i+1}(x) \cdot \tilde{V}_{i+1}(y).
\end{aligned}$$

     At the end of the protocol, $\mathcal{V}$ receives two evaluations $\tilde{V}_{i+1}(u^{(i+1)})$, $\tilde{V}_{i+1}(v^{(i+1)})$. $\mathcal{V}$ computes the following and if it does not equal to the last message, returns 0.

$$\begin{aligned}
&\left(\alpha^{(i)}\tilde{add}_i(u^{(i)},u^{(i+1)},v^{(i+1)}) + \beta^{(i)}\tilde{add}_i(v^{(i)},u^{(i+1)},v^{(i+1)})\right) \\
&\qquad \cdot \left(\tilde{V}_{i+1}(u^{(i+1)}) + \tilde{V}_{i+1}(v^{(i+1)})\right) \\
&+ \left(\alpha^{(i)}\tilde{mult}_i(u^{(i)},u^{(i+1)},v^{(i+1)}) + \beta^{(i)}\tilde{mult}_i(v^{(i)},u^{(i+1)},v^{(i+1)})\right) \\
&\qquad \cdot \tilde{V}_{i+1}(u^{(i+1)}) \cdot \tilde{V}_{i+1}(v^{(i+1)}).
\end{aligned}$$

4. At the input layer d, $\mathcal{V}$ has two claims $\tilde{V}_d(u^{(d)})$ and $\tilde{V}_d(v^{(d)})$. As $\mathcal{V}$ has access to the input $\mathbf{in}$, it is possible to directly compute $\tilde{V}_d$ and evaluate it at two points u^d and v^d. If claims are correct, $\mathcal{V}$ returns 1, otherwise returns 0.

Figure 3.2: GKR protocol.

The verifier then checks whether $s_a = z \cdot s_b + r$ and $r < s_b$, and outputs $z \cdot 2^f$ as the result. To further reduce the cost, we defer all scalings to the output of the layer of the DNN model.

Sumcheck for Binary Operations. Scaling and bit-decomposition consistency checks can be handled using generic GKR. However, for complex models, such an approach would blow up the size of the circuit and the wiring predicates. To overcome this, we express these operations directly as sumcheck instances. In particular, let $\mathbf{z}$ be a vector of size n whose bit-decomposition is provided by the prover. Let $\mathbf{Z}_{\mathsf{Bit}}$ be a matrix of size $n \times q$ and let $\mathbf{z}_{\mathsf{Sign}}$ be a vector of size n, where $(\mathbf{Z}_{\mathsf{Bit}})_{i,*}$ is claimed to be the bit-decomposition of z_i, and $(z_{\mathsf{Sign}})_i$ indicates its sign. To check that $\mathbf{Z}_{\mathsf{Bit}}$ and $\mathbf{z}_{\mathsf{Sign}}$ are binary, for random α, r_x, r_y, the parties run sumcheck on:

$$0 = \sum_{x \in \{0,1\}^{\log n}, y \in \{0,1\}^{\log q}} \widetilde{\beta}_x(r_x) \cdot \widetilde{\beta}_y(r_y) \cdot$$
$$(\alpha \cdot \widetilde{\mathbf{z}}_{\mathsf{Sign}}(x) \cdot (1 - \widetilde{\mathbf{z}}_{\mathsf{Sign}}(x)) + \widetilde{\mathbf{Z}}_{\mathsf{Bit}}(x, y) \cdot (1 - \widetilde{\mathbf{Z}}_{\mathsf{Bit}}(x, y))). \tag{3.3}$$

For a challenge r_z, parties verify the consistency of the decompositions by a sumcheck on:

$$\widetilde{\mathbf{z}}(r_z) = \sum_{x \in \{0,1\}^{\log n}, y \in \{0,1\}^{\log q}} \widetilde{\beta}_x(r_z) \cdot \widetilde{\mathbf{Z}}_{\mathsf{Bit}}(x, y) \cdot (1 - 2 \cdot \widetilde{\mathbf{z}}_{\mathsf{Sign}}(x)) \cdot 2^y. \tag{3.4}$$

Scaling can be handled similarly by restricting the ranges of x and y in Equation 3.4.

Exponentiation. Natural exponentiation is used in functions such as Softmax and Tanh. Similar to prior work [77, 119], we employ the following piece-wise approximation for e^x:

$$e^x \approx \begin{cases} 0 & x < -1/2 \\ \frac{1}{2} + x & -1/2 \leq x \leq 1/2 \\ 1 & x > 1/2 \end{cases} \cdot$$

3.4.2 PoGDs for Neural Networks

The prover $\mathcal{P}$ updates $\mathbf{W}_{i-1}$ to $\mathbf{W}_i$ using a data batch $\mathbf{B}_{i-1}$; the PoGD uses sumcheck and GKR to convince a verifier $\mathcal{V}$ that the update is performed correctly. Specifically, $\mathcal{P}$ sends commitments to $\widetilde{\mathbf{W}}_i$, $\widetilde{\mathbf{W}}_{i-1}$, $\widetilde{\mathbf{B}}_{i-1}$, and auxiliary inputs denoted by $\widetilde{\mathbf{AUX}}_i$, which include bit-decompositions, quotients, and remainders; note that $\mathbf{W}_i$, $\mathbf{W}_{i-1}$, and $\mathbf{AUX}_i$ are concatenations of $\mathbf{W}_{i,\ell}$, $\mathbf{W}_{i-1,\ell}$, and $\mathbf{AUX}_{i,\ell}$, respectively, over all layers $\ell = 1$ to L. PoGD then proceeds through several phases. In the first phase, the parties verify the update step; the second phase verifies the backward pass; and the third phase verifies the forward pass. Once the sumcheck messages are generated, the parties combine random evaluations of inputs and outputs received from different phases by running additional sumcheck instances. $\mathcal{V}$ can verify the final evaluations via commitment openings. For linear operations within the forward and backward passes, the parties run sublinear sumcheck protocols [107, 137]. For non-linear operations, we use generic GKR. Below, we outline the four phases of the PoGD protocol.

Phase 1: For $\ell = 1$ to L, $\mathcal{V}$ asks for a random evaluation of $\widetilde{\mathbf{W}}_{i,\ell}$. Having this evaluation, parties run GKR on $\mathbf{W}_{i,\ell} = \mathbf{W}_{i-1,\ell} - \eta_\ell \cdot \overline{\mathbf{G}}_{i,\ell}$, where the run ends with random evaluations of $\widetilde{\mathbf{W}}_{i-1,\ell}$ and $\widetilde{\mathbf{G}}_{i,\ell}$. The former proceeds to Phase 4, and the latter proceeds to Phase 2 to bootstrap sumchecks.

Phase 2: For $\ell = 1$ to L, having the evaluation of $\widetilde{\mathbf{G}}_{i,\ell}$ received from Phase 1 and also the evaluation of $\widetilde{\mathcal{R}}_{i,\ell}$ from the previous iteration (If $\ell \neq 1$) parties verify Equation 3.1. We use the sublinear sumcheck protocols for linear operations, GKR exploiting bit-decomposition of inputs for non-linear operations, and the sumcheck protocol on Equation 3.3 and Equation 3.4 for binary operations of scalings and consistency checks. The run ends with evaluations of $\widetilde{\mathbf{W}}_{i-1,\ell}$, $\widetilde{\mathbf{U}}_{i,\ell-1}$, $\widetilde{\mathcal{R}}_{i+1,\ell}$, and $\widetilde{\mathbf{AUX}}_{i,\ell}$, where $\widetilde{\mathcal{R}}_{i+1,\ell}$ proceeds to the next iteration, $\widetilde{\mathbf{U}}_{i,\ell-1}$ proceeds to Phase 3, and

others proceed to Phase 4. For $\ell = L$, the parties run GKR on the loss calculation, yielding evaluations of $\widetilde{\mathbf{B}}_{i-1}$ and $\widetilde{\mathbf{U}}_{i,L}$, which would proceed to Phase 4 and Phase 3, respectively.

Phase 3: For $\ell = L$ to 1, having evaluations of $\widetilde{\mathbf{U}}_{i,\ell}$ received from Phase 2 and the previous iteration (if $\ell \neq L$), the parties verify the forward pass via the sublinear sumcheck protocols for linear operations, GKR for non-linear operations, and the sumcheck protocol for scalings and consistency checks. The run ends with evaluations of the inputs of the layer, i.e., $\widetilde{\mathbf{W}}_{i-1,\ell}$, $\widetilde{\mathbf{U}}_{i,\ell-1}$, and $\widetilde{\mathbf{AUX}}_{i,\ell}$, where $\widetilde{\mathbf{U}}_{i,\ell-1}$ proceeds to the next iteration, and $\widetilde{\mathbf{W}}_{i-1,\ell}$ and $\widetilde{\mathbf{AUX}}_{i,\ell}$ proceed to Phase 4; we recall that an evaluation of $\widetilde{\mathbf{U}}_{i,0}$ can be viewed as an evaluation of the data batch $\widetilde{\mathbf{B}}_{i-1}$.

Phase 4: Parties combine evaluations of $\widetilde{\mathbf{W}}_{i-1}$, $\widetilde{\mathbf{B}}_{i-1}$, and $\widetilde{\mathbf{AUX}}_i$ received from the previous phases. Consider a generic ℓ-variate multilinear polynomial f and multiple evaluations of the form $\{(x_i, y_i)\}_{i=1}^k$, where $f(x_i) = y_i$. Given challenges $\{\alpha_i\}_{i=1}^k$, we reduce evaluations to a single one by running the sumcheck protocol on:

$$\sum_{i=1}^{k} \alpha_i \cdot y_i = \sum_{z \in \{0,1\}^\ell} f(z) \cdot \sum_{i=1}^{k} \alpha_i \cdot \widetilde{\beta}_z(x_i). \tag{3.5}$$

In Phase 4 of our PoGD, parties run this sumcheck by instantiating f with $\widetilde{\mathbf{W}}_{i-1}$, $\widetilde{\mathbf{B}}_{i-1}$, and $\widetilde{\mathbf{AUX}}_i$. At the end of the run, for each of the polynomials, the verifier receives a single evaluation. Along with the evaluation of $\widetilde{\mathbf{W}}_i$ from Phase 1, the verifier checks these using oracle access, i.e., by requesting evaluation opening proofs.

We give the formal protocol in Figures 3.3 and 3.4. Note that the sumcheck runs on the linear operations in step (4) of phase 2 and step (2) of phase 3 and can be merged together. Parties can combine evaluations $\widetilde{\mathbf{T}}_{i,\ell}(r^{(3)}_{2,\mathbf{T}_{i,\ell}})$ and $\widetilde{\mathbf{T}}_{i,\ell}(r^{(1)}_{3,\mathbf{T}_{i,\ell}})$ and then invoke the sumcheck protocol on the combined evaluation. Similarly, scalings in step (6) of phase 2 and step (2) of phase 3 can

be merged together. The protocol can be made non-interactive by the Fiat-Shamir heuristic and zero-knowledge by zero-knowledge sumchecks [31].

Theorem 3.4.1. Π_{PoGD} *in Figures 3.3 and 3.4 satisfies completeness and knowledge soundness.*

Proof. Following the completeness of GKR, $\mathsf{Sum}_{\mathsf{Mat}}$, and $\mathsf{Sum}_{\mathsf{Conv}}$, and PCS completeness of PoGD is immediate. We first show a negligible bound soundness error with a phase-by-phase analysis. Suppose that the prover sends some malicious parameters $\mathbf{W}^*_{i,\ell} \neq \mathbf{W}_{i,\ell}$.

P1: For $i = 1$ to L, following the Schwartz-Zippel lemma and the soundness of GKR—except with a negligible probability in λ—we have $\widetilde{\mathbf{W}}^*_{i,\ell}(r_{\mathsf{out},\mathbf{W}_{i,\ell}}) \neq \widetilde{\mathbf{W}}_{i,\ell}(r_{\mathsf{out},\mathbf{W}_{i,\ell}})$ unless: (i) $\widetilde{\mathbf{W}}^*_{i-1,\ell}(r_{1,\mathbf{W}_{i-1,\ell}}) \neq \widetilde{\mathbf{W}}_{i-1,\ell}(r_{1,\mathbf{W}_{i-1,\ell}})$, which proceeds to the phase 4; or we have (ii) $\widetilde{\mathbf{G}}^*_{i,\ell}(r_{1,\mathbf{G}_{i,\ell}}) \neq \widetilde{\mathbf{G}}_{i,\ell}(r_{1,\mathbf{G}_{i,\ell}})$, which proceeds to phase 2.

P2: Suppose the case (ii) has occurred in phase 1 and there exists at least one ℓ such that $\widetilde{\mathbf{G}}^*_{i,\ell}(r_{1,\mathbf{G}_{i,\ell}}) \neq \widetilde{\mathbf{G}}_{i,\ell}(r_{1,\mathbf{G}_{i,\ell}})$. Following the soundness of GKR, $\mathsf{Sum}_{\mathsf{Mat}}$, and $\mathsf{Sum}_{\mathsf{Conv}}$, there are three cases: (i) $\widetilde{\mathbf{W}}^*_{i-1,\ell}(r_{2,\mathbf{W}_{i-1,\ell}}) \neq \widetilde{\mathbf{W}}_{i-1,\ell}(r_{2,\mathbf{W}_{i-1,\ell}})$, $\widetilde{\mathbf{AUX}}^*_{i,\ell}(r_{2,\mathbf{AUX}_{i,\ell}}) \neq \widetilde{\mathbf{AUX}}_{i,\ell}(r_{2,\mathbf{AUX}_{i,\ell}})$, or for the data batch $\widetilde{\mathbf{B}}^*_{i,\ell}(r_{2,\mathbf{B}_{i,\ell}}) \neq \widetilde{\mathbf{B}}_{i,\ell}(r_{2,\mathbf{B}_{i,\ell}})$, which proceeds to phase 4; The next case is (ii) $\widetilde{\mathcal{R}}^*_{i,\ell+1}(r_{2,\mathcal{R}_{i,\ell+1}}) \neq \widetilde{\mathcal{R}}_{i,\ell+1}(r_{2,\mathcal{R}_{i,\ell+1}})$, proceeding to the next layer; and finally, we have (iii) $\widetilde{\mathbf{U}}^*_{i,\ell}(r_{2,\mathbf{U}_{i,\ell}}) \neq \widetilde{\mathbf{U}}_{i,\ell}(r_{2,\mathbf{U}_{i,\ell}})$, proceeding to phase 3.

P3: Suppose (iii) occurs in phase 2 and there exists at least one ℓ such that $\widetilde{\mathbf{U}}^*_{i,\ell}(r_{2,\mathbf{U}_{i,\ell}}) \neq \widetilde{\mathbf{U}}_{i,\ell}(r_{2,\mathbf{U}_{i,\ell}})$. Following soundness of GKR, $\mathsf{Sum}_{\mathsf{Mat}}$, and $\mathsf{Sum}_{\mathsf{Conv}}$, there are two possibilities: (i) $\widetilde{\mathbf{W}}^*_{i-1,\ell}(r_{3,\mathbf{W}_{i,\ell}}) \neq \widetilde{\mathbf{W}}_{i-1,\ell}(r_{3,\mathbf{W}_{i,\ell}})$, $\widetilde{\mathbf{B}}^*_{i,\ell}(r_{3,\mathbf{B}_{i,\ell}}) \neq \widetilde{\mathbf{B}}_{i,\ell}(r_{3,\mathbf{B}_{i,\ell}})$, or $\widetilde{\mathbf{AUX}}^*_{i,\ell}(r_{3,\mathbf{AUX}_{i,\ell}}) \neq \widetilde{\mathbf{AUX}}_{i,\ell}(r_{3,\mathbf{AUX}_{i,\ell}})$, which proceeds phase 4; (ii) $\widetilde{\mathbf{U}}^*_{i,\ell-1}(r_{3,\mathbf{U}_{i,\ell-1}}) \neq \widetilde{\mathbf{U}}_{i,\ell-1}(r_{3,\mathbf{U}_{i,\ell-1}})$, which proceeds to the next iteration, i.e., $\ell - 1$-th layer.

P4: Suppose that the case (i) in phases one, two, or three has caused an error, proceeding to this phase. Then, following the soundness of the evaluation reduction sumcheck and also the Schwartz-Zippel lemma, either $\widetilde{\mathbf{W}}^*_{i-1}(r_{\mathsf{in},\mathbf{W}_{i-1}}) \neq \widetilde{\mathbf{W}}_{i-1}(r_{\mathsf{in},\mathbf{W}_{i-1}})$, $\widetilde{\mathbf{B}}^*_i(r_{\mathsf{in},\mathbf{B}_i}) \neq \widetilde{\mathbf{B}}_i(r_{\mathsf{in},\mathbf{B}_i})$, or $\widetilde{\mathbf{AUX}}^*_i(r_{\mathsf{in},\mathbf{AUX}_i}) \neq \widetilde{\mathbf{AUX}}_i(r_{\mathsf{in},\mathbf{AUX}_i})$. Any of these make $\mathcal{V}$ reject based on PCS soundness.

Knowledge soundness follows immediately. As PCS is knowledge sound, the extractor of the PoGD scheme can invoke the extractor of PCS to get witnesses committed to by the prover $\widetilde{\mathbf{W}}^*_i$, $\widetilde{\mathbf{W}}^*_{i-1}$, $\widetilde{\mathbf{B}}^*_{i-1}$, and $\widetilde{\mathbf{AUX}}^*_i$. As PoGD is sound, they are satisfiable. Moreover, as GKR, $\mathsf{Sum}_{\mathsf{Mat}}$, $\mathsf{Sum}_{\mathsf{Conv}}$, and PCS are zero-knowledge, the simulator of PoGD can invoke the simulators of these building blocks to generate proof messages and commitments that are indistinguishable. □

Efficiency. Let $s_{\mathsf{in},\ell}$ be the input size, $s_{\mathsf{out},\ell}$ be the output size of the lth layer, N be the batch size, and q be the quantization bit-length. We use Orion [147] as our polynomial commitment scheme due to its linear-time complexity, concretely efficient prover, and succinct verification. Then, the prover's runtime would be $O(\sum_{i=0}^{L-1} N\, s_{\mathsf{in},\ell}\, s_{\mathsf{out},\ell} + Nqs_{\mathsf{out},\ell})$, and the verifier's runtime would be $O(\sum_{i=0}^{L-1} \log^2(N\, s_{\mathsf{in},\ell}\, s_{\mathsf{out},\ell}) + \log^2(Nqs_{\mathsf{out},\ell}))$, which is dominated by evaluation opening proofs. In Sections 3.5 and 3.6, we discuss how to reduce the verification cost using our accumulation scheme to $O(\sum_{i=0}^{L-1} \log(N\, s_{\mathsf{in},\ell}\, s_{\mathsf{out},\ell}) + \log(Nqs_{\mathsf{out},\ell}))$ when we compose PoGDs recursively.

3.5 Foldable Polynomial Commitments

We present our accumulation scheme for *multivariate* polynomial commitments. Recall the definition of an accumulation scheme given in Section 2.3. Here, each accumulator instance is a tuple (σ_i, x_i, y_i), where σ_i is a commitment to an ℓ-variate polynomial f_i of variable degree at most d. The prover then wants to convince a verifier that, for all i, we have $y_i = f_i(x_i)$, more

Π_{PoGD}

Let GKR be the GKR, $\mathsf{Sum}_{\mathsf{Mat}}$ be the sumcheck for matrix multiplication [137], $\mathsf{Sum}_{\mathsf{Conv}}$ be the sumcheck for convolution [107], and $\mathsf{PCS} = (\mathsf{Setup}, \mathsf{Commit}, \mathsf{Open}, \mathsf{Verify})$ be the polynomial commitment schemes; pp are PCS public parameters . The model has L layers.

Initialization: On input $\mathbf{W}_{i-1} = \{\mathbf{W}_{i-1,\ell}\}_{\ell=1}^{L}$ and $\mathbf{B}_{i-1}$, $\mathcal{P}$ runs gradient descent to evaluate $\mathbf{U}_i = \{\mathbf{U}_{i,\ell}\}_{\ell=1}^{L}$, $\mathbf{G}_i = \{\mathbf{G}_{i,\ell}\}_{\ell=1}^{L}$, and $\mathcal{R}_i = \{\mathcal{R}_{i,\ell}\}_{\ell=1}^{L}$, weights $\mathbf{W}_i = \{\mathbf{W}_{i,\ell}\}_{\ell=1}^{L}$, and auxiliary inputs $\mathbf{AUX}_i = \{\mathbf{AUX}_{i,\ell}\}_{\ell=1}^{L}$. $\mathcal{P}$ sends $\mathcal{V}$, $\sigma_{W_i} \leftarrow \mathsf{PCS.Commit}(\mathsf{pp}, \widetilde{\mathbf{W}}_i, r_{W_i})$, $\sigma_{W_{i-1}} \leftarrow \mathsf{PCS.Commit}(\mathsf{pp}, \widetilde{\mathbf{W}}_{i-1}, r_{W_{i-1}})$, $\sigma_{B_{i-1}} \leftarrow \mathsf{PCS.Commit}(\mathsf{pp}, \widetilde{\mathbf{B}}_{i-1}, r_{B_{i-1}})$, and $\sigma_{\mathbf{AUX}_i} \leftarrow \mathsf{PCS.Commit}(\mathsf{pp}, \widetilde{\mathbf{AUX}}_i, r_{\mathbf{AUX}_i})$. Then, the PoGD protocol proceeds as follows:

Phase 1: Proof of update. For $\ell = 1$ to L do the following steps:

- $\mathcal{V}$ samples $r_{\mathsf{out},\mathbf{W}_{i,\ell}} \xleftarrow{\$} \mathbb{F}$ and queries $\widetilde{\mathbf{W}}_{i,\ell}(r_{\mathsf{out},\mathbf{W}_{i,\ell}})$. Having $\widetilde{\mathbf{W}}_{i,\ell}(r_{\mathsf{out},\mathbf{W}_{i,\ell}})$, parties run GKR on $\mathbf{W}_{i,\ell} = \mathbf{W}_{i-1,\ell} - \eta \cdot \overline{\mathbf{G}}_{i,\ell}$ and receive evaluations $\widetilde{\mathbf{W}}_{i-1,\ell}(r_{1,\mathbf{W}_{i-1,\ell}})$, $\widetilde{\mathbf{G}}_{i,\ell}(r_{1,\mathbf{G}_{i,\ell}})$.

Phase 2: Proof of backward pass. For $\ell = 1$ to L do the following steps:

- Having $\widetilde{\mathbf{G}}_{i,\ell}(r_{1,\mathbf{G}_{i,\ell}})$ from P1, parties run $\mathsf{Sum}_{\mathsf{Conv}}$ on $\mathbf{G}_{i,\ell} = (\mathcal{R}_{i,\ell+1} \odot \mathbf{T}'_{i,\ell}) * \mathbf{U}_{i,\ell-1}$ if the ℓth layer is convolutional or $\mathsf{Sum}_{\mathsf{Mat}}$ on $\mathbf{G}_{i,\ell} = (\mathcal{R}_{i,\ell+1} \odot \mathbf{T}'_{i,\ell}) \cdot \mathbf{U}_{i,\ell-1}^{T}$ otherwise. At the end, parties receive evaluations $\widetilde{\mathbf{T}'}_{i,\ell+1}(r^{(1)}_{2,\mathbf{T}'_{i,\ell+1}})$, $\widetilde{\mathbf{U}}_{i,\ell}(r^{(1)}_{2,\mathbf{U}_{i,\ell}})$, and $\widetilde{\mathcal{R}}_{i,\ell+1}(r^{(1)}_{2,\mathcal{R}_{i,\ell+1}})$.

- For $\ell \neq L$, having $\widetilde{\mathcal{R}}_{i,\ell}(r_{2,\mathcal{R}_{i,\ell}})$, parties run $\mathsf{Sum}_{\mathsf{Conv}}$ on $\mathcal{R}_{i,\ell} = \mathbf{W}_{i-1,\ell}^{T} * (\mathcal{R}_{i,\ell+1} \odot \mathbf{T}'_{i,\ell})$ if convolutional and $\mathsf{Sum}_{\mathsf{Mat}}$ on $\mathcal{R}_{i,\ell} = \mathbf{W}_{i-1,\ell}^{T} \cdot (\mathcal{R}_{i,\ell+1} \odot \mathbf{T}'_{i,\ell})$ otherwise. Then, parties receive these evaluations: $\widetilde{\mathcal{R}}_{i,\ell+1}(r^{(2)}_{2,\mathcal{R}_{i,\ell+1}})$, $\widetilde{\mathbf{W}}_{i-1,\ell}(r^{(2)}_{2,\mathbf{W}_{i-1,\ell}})$, and $\widetilde{\mathbf{T}'}_{i,\ell+1}(r^{(2)}_{2,\mathbf{T}'_{i,\ell+1}})$.

- Parties combine $\widetilde{\mathbf{T}'}_{i,\ell+1}(r^{(1)}_{2,\mathcal{R}_{i,\ell+1}})$ and $\widetilde{\mathbf{T}'}_{i,\ell}(r^{(2)}_{2,\mathbf{T}'_{i,\ell}})$ by evaluation reduction. Parties run GKR on $\mathbf{T}'_{i,\ell} = \mathbf{Q}'_{i,\ell} \odot (\partial \mathsf{act}_\ell(\mathbf{T}_{i,\ell})/\partial \mathbf{T}_{i,\ell})$, then, $\mathbf{Q}'_{i,\ell} = \partial \mathsf{pool}_\ell(\mathsf{act}_\ell(\mathbf{T}_{i,\ell}))/\partial \mathsf{act}_\ell(\mathbf{T}_{i,\ell})$ if the layer is convolutional or on $\mathbf{T}'_{i,\ell} = \partial \mathbf{U}_{i,\ell}/\partial \mathbf{T}_{i,\ell} = \partial \mathsf{act}_\ell(\mathbf{T}_{i,\ell})/\partial \mathbf{T}_{i,\ell}$ otherwise. At the end, parties receive evaluations $\widetilde{\mathbf{T}}_{i,\ell}(r^{(3)}_{2,\mathbf{T}_{i,\ell}})$ and $\widetilde{\mathbf{AUX}}_{i,\ell}(r^{(3)}_{2,\mathbf{AUX}_{i,\ell}})$.

- Having $\widetilde{\mathbf{T}}_{i,\ell}(r^{(3)}_{2,\mathbf{T}_{i,\ell}})$, parties run $\mathsf{Sum}_{\mathsf{Conv}}$ on $\mathbf{T}_{i,\ell} = \mathbf{W}_{i-1,\ell} * \mathbf{U}_{i,\ell-1}$ if convolution or $\mathsf{Sum}_{\mathsf{Mat}}$ on $\mathbf{T}_{i,\ell} = \mathbf{W}_{i-1,\ell} \cdot \mathbf{U}_{i,\ell-1}$ otherwise. At the end, parties receive $\widetilde{\mathbf{U}}_{i,\ell-1}(r^{(4)}_{2,\mathbf{U}_{i,\ell-1}})$ and $\widetilde{\mathbf{W}}_{i-1,\ell}(r^{(4)}_{2,\mathbf{W}_{i-1,\ell}})$. For $\ell = 1$, $\widetilde{\mathbf{U}}_{i,0}(r^{(4)}_{2,\mathbf{U}_{i,0}})$ is viewed as $\widetilde{\mathbf{B}}_{i-1}(r^{(4)}_{2,\mathbf{B}_{i-1}})$.

Figure 3.3: PoGD protocol, part a.

Π_{PoGD}

- For $\ell = L$, parties combine $\widetilde{\mathcal{R}}_{i,L+1}(r^{(1)}_{2,\mathcal{R}_{i,L+1}})$ and $\widetilde{\mathcal{R}}_{i,L+1}(r^{(2)}_{2,\mathcal{R}_{i,L+1}})$ by an evaluation reduction. Parties run GKR on $\mathcal{R}_{i,L+1} = \partial\mathcal{L}(\mathbf{U}_{i,L}, Y_i)/\partial\mathbf{U}_{i,L}$. At the end, parties receive evaluations $\widetilde{\mathbf{B}}_{i-1}(r^{(5)}_{2,\mathbf{B}_{i-1}})$, $\widetilde{\mathbf{U}}_{i,L}(r^{(5)}_{2,\mathbf{U}_{i,L}})$, and $\widetilde{\mathbf{AUX}}_{i,\ell}(r^{(5)}_{2,\mathbf{AUX}_{i,\ell}})$.

- Consistency checks and scalings are proved by running sumchecks on Equations 3.3 and 3.4, which ends with several evaluations of auxiliary inputs $\widetilde{\mathbf{AUX}}_{i,\ell}$. The evaluations are combined together into a single evaluation $\widetilde{\mathbf{AUX}}_{i,\ell}(r^{(6)}_{2,\mathbf{AUX}_{i,\ell}})$.

- For $\ell = L$, all evaluations of $\widetilde{\mathbf{W}}_{i-1,\ell}$, $\widetilde{\mathbf{B}}_{i-1}$, $\widetilde{\mathbf{U}}_{i,\ell}$, $\widetilde{\mathbf{AUX}}_{i,\ell}$, from the previous steps, are combined into $\widetilde{\mathbf{W}}_{i-1,\ell}(r_{2,\mathbf{W}_{i-1,\ell}})$, $\widetilde{\mathbf{U}}_{i,\ell}(r_{2,\mathbf{U}_{i,\ell}})$, $\widetilde{\mathbf{B}}_{i-1}(r_{2,\mathbf{B}_{i-1}})$, $\widetilde{\mathbf{AUX}}_{i,\ell}(r_{2,\mathbf{AUX}_{i,\ell}})$.

Phase 3: Proof of forward pass. For $\ell = L$ to 1 do the following steps:

1. Let $\widetilde{\mathbf{U}}_{i,L}(r_{3,\mathbf{U}_{i,L}}) = \widetilde{\mathbf{U}}_{i,L}(r_{2,\mathbf{U}_{i,L}})$. Parties combine $\widetilde{\mathbf{U}}_{i,\ell}(r_{3,\mathbf{U}_{i,\ell}})$ (if $\ell \neq L$) and $\widetilde{\mathbf{U}}_{i,\ell-1}(r_{2,\mathbf{U}_{i,\ell-1}})$ from P2. Having this, parties run GKR on $\mathbf{U}_{i,\ell} = \mathsf{pool}_\ell(\mathbf{Q}_{i,\ell})$ followed by another run on $\mathbf{Q}_{i,\ell} = \mathsf{act}_\ell(\mathbf{T}_{i,\ell})$ if the layer is convolutional or on $\mathbf{U}_{i,\ell} = \mathsf{act}_\ell(\mathbf{T}_{i,\ell})$ otherwise. At the end, parties receive evaluations $\widetilde{\mathbf{T}}_{i,\ell}(r^{(1)}_{3,\mathbf{T}_{i,\ell}})$ and $\widetilde{\mathbf{AUX}}_{i,\ell}(r^{(1)}_{3,\mathbf{AUX}_{i,\ell}})$.

2. Having $\widetilde{\mathbf{T}}_{i,\ell}(r^{(1)}_{3,\mathbf{T}_{i,\ell}})$, parties run $\mathsf{Sum}_{\mathsf{Conv}}$ on $\mathbf{T}_{i,\ell} = \mathbf{W}_{i-1,\ell} * \mathbf{U}_{i,\ell-1}$ or $\mathsf{Sum}_{\mathsf{Mat}}$ on $\mathbf{T}_{i,\ell} = \mathbf{W}_{i-1,\ell}\mathbf{U}_{i,\ell-1}$ and receive $\widetilde{\mathbf{W}}_{i-1,\ell}(r_{3,\mathbf{W}_{i,\ell}})$ and $\widetilde{\mathbf{U}}_{i,\ell-1}(r_{3,\mathbf{U}_{i,\ell-1}})$. If $i = 1$, $\widetilde{\mathbf{U}}_{i,0}(r_{3,\mathbf{U}_{i,0}})$ is viewed as $\widetilde{\mathbf{B}}_{i-1,\ell}(r_{3,\mathbf{B}_{i-1,\ell}})$. For consistency checks and scalings, parties run sumchecks on Equations 3.3 and 3.4 and and receive $\widetilde{\mathbf{AUX}}_{i,\ell}(r^{(1)}_{3,\mathbf{AUX}_{i,\ell}})$ into $\widetilde{\mathbf{AUX}}_{i,\ell}(r_{3,\mathbf{AUX}_{i,\ell}})$.

Phase 4: Proof of evaluation reduction.

1. Parties combine received all evaluations of $\widetilde{\mathbf{W}}_{i-1,\ell}$ into a single evaluation $\widetilde{\mathbf{W}}_{i-1}(r_{\mathsf{in},\mathbf{W}_{i-1}})$, evaluations of $\widetilde{\mathbf{B}}_{i-1,\ell}$ into an evaluation $\widetilde{\mathbf{B}}_{i-1}(r_{\mathsf{in},\mathbf{B}_{i-1}})$ and evaluations of $\widetilde{\mathbf{AUX}}_{i,\ell}$ into an evaluation $\widetilde{\mathbf{AUX}}_i(r_{\mathsf{in},\mathbf{AUX}_i})$. The final combined evaluations can be verified by invoking the evaluation opening algorithm of the commitment scheme.

Figure 3.4: PoGD protocol, part b.

efficiently than by giving an opening proof for each instance. To do so, the prover accumulates the instances into a single tuple (σ^*, x^*, y^*) such that σ^* is a commitment to a polynomial f^* of size at most that of each input polynomial f_i and such that, with high probability, $f^*(x^*) = y^*$ if

and only if $\forall i : f_i(x_i) = y_i$; the decider then receives an opening proof for $f^*(x^*) = y^*$.

3.5.1 Accumulation of Evaluations

We begin with an idea for accumulating multiple evaluations of a single polynomial [82].

Single Polynomial, Multiple Points. First, consider the basic case where $f_1 = \cdots = f_k = f$ and $\sigma_1 = \cdots = \sigma_k = \sigma$, but $\{x_i\}$ are distinct. The prover and verifier interpolate a degree-$(k-1)$ univariate polynomial $L : \mathbb{F} \to \mathbb{F}^\ell$ with $L(i) = x_i$ for $i \in \{1, \ldots, k\} \subseteq \mathbb{F}$. The prover then sends the verifier a polynomial $g = f \circ L : \mathbb{F} \to \mathbb{F}$ of degree $O(dk\ell)$. The verifier checks that $\forall i : g(i) = y_i$, then, for a random $r \in \mathbb{F}$, the parties set $x^* = L(r)$ and $y^* = g(r)$; in this case, the output polynomial and commitments are $f^* = f$ and $\sigma^* = \sigma$.

We claim that $f(x^*) = y^*$ iff $\forall i : f(x_i) = y_i$ with high probability. Completeness is immediate. Soundness can be argued as follows. Without loss of generality, say $f(x_1) \neq y_1$. If $g = f \circ L$, the verifier rejects since $g(1) \neq y_1$. If $g \neq f \circ L$, then g and $f \circ L$ agree on at most $\deg(g) = O(dk\ell)$ points, and we have $f(x^*) = y^*$ with a probability $O(dk\ell/|\mathbb{F}|)$.

Multiple Polynomials, Multiple Points. We now extend the above idea to the general case of (possibly) distinct polynomials $\{f_i\}$ and points $\{x_i\}$. As before, the parties define $L : \mathbb{F} \to \mathbb{F}^\ell$ such that $\forall i : L(i) = x_i$. The prover then sends the k univariate polynomials $g_i = f_i \circ L$, and the verifier checks whether $\forall i : g_i(i) = y_i$. Then, for random challenges $\{\alpha_i\}_{i=1}^k$ and r, the parties set $x^* = L(r)$, $g = \sum_i \alpha_i g_i$, and $y^* = g(r)$. The output polynomial is $f^* = \sum_i \alpha_i f_i$; we defer to Section 3.5.2 for the details of how the parties can generate the commitment σ^*.

We claim that $f(x^*) = y^*$ iff $\forall i : f(x_i) = y_i$ with high probability. Completeness holds as

$$f^*(x^*) = (f^* \circ L)(r) = \textstyle\sum_i \alpha_i (f_i \circ L)(r) = \sum_i \alpha_i g_i(r) = g(r).$$

For soundness, if there exists some $f_i(x_i) \neq y_i$, the prover must send $g_i \neq f_i \circ L$ to pass verification. But with high probability over $\{\alpha_i\}, r$, we have $y = g(r) \neq (f^* \circ L)(r) = f^*(x^*)$.

Linear-Time Prover. Naive interpolation of each g_i would impose prover cost $O(dk\ell \cdot 2^\ell)$ as it requires evaluating a polynomial f_i of size $O(2^\ell)$ at $O(dk\ell)$ points. Fortunately, we can achieve a linear-time prover using a recursive interpolation approach. Define $P_j : \mathbb{F} \to \mathbb{F}$ of degree $O(dk\ell)$ such that $P_j(i)$ outputs the jth variable of x_i for all $i \in \{1, \ldots, k\}$. We observe that

$$g_i = (1 - P_1)(f_i(0, x_2, ..., x_\ell) \circ L) + P_1(f_i(1, x_2, \ldots, x_\ell) \circ L);$$

the prover reduces the interpolation of g_i to the interpolation of the polynomials $f_i(0, x_2, ..., x_\ell) \circ L$ and $f_i(1, x_2, ..., x_\ell) \circ L$, which can be done in $O(2^{\ell-1})$ time. Then, g_i can be evaluated in $O(dk\ell)$ additional time. This process can be applied recursively to the reduced interpolations to yield a linear runtime (i.e., $O(2^\ell)$) overall overhead for interpolation of each polynomial g_i.

Zero Knowledge. We make the scheme zero-knowledge using masking polynomials. For all i, the prover randomly samples an ℓ-variate polynomial h_i and sends a commitment σ_i' to h_i along with an evaluation $v_i = h_i(x_i)$. Given challenges $\{\beta_i\}$, the prover sends $g_i = (f_i + \beta_i h_i) \circ L$, and the verifier checks that $g(i) = y_i + \beta_i v_i$. The rest of the protocol is similar to above.

Naively, one may choose each h_i above as a random ℓ-variate polynomial of variable-degree at most d. The number of coefficients in each h_i would then be $O(d^\ell)$ and large, making it inefficient in practice to commit to the $\{h_i\}$. We later show it suffices to sample each h_i as $h_i = \sum_{j=1}^{\ell} h_{i,j}$, where each $h_{i,j}$ is a random *univariate* polynomial (in the jth variable) of degree $d\ell$. The size of each h_i is then $O(d\ell^2)$, and the cost of zero knowledge is $O(kd\ell^2)$.

3.5.2 Accumulation of Commitments

What remains is to show how parties can compute σ^*, a commitment to $f^* = \sum_i \alpha_i f_i$ given $\{\sigma_i\}$, commitments to the $\{f_i\}$; the procedure naturally extends to the case of the zero-knowledge scheme, where $f^* = \sum_i \alpha_i f_i + \alpha_i \beta_i h_i$, and parties also hold $\{\sigma'_i\}$. Of course, if the underlying commitment scheme is homomorphic, then commitment accumulation is trivial. However, practical homomorphic commitment schemes [39, 95, 142] are known only from group assumptions and cannot be used efficiently for recursive proof composition. Indeed, in our application we use Orion [147], a hash-based commitment scheme that is not homomorphic. We describe how it is possible to commit to the linear combination of input polynomials for that.

Orion. First, we briefly review Orion. In that scheme, the coefficients of a polynomial are mapped to the coefficients of a square matrix $\mathbf{M}$, say of size $n \times n$. Then, a linear error-correcting code $E : \mathbb{F}^n \to \mathbb{F}^p$ is applied to $\mathbf{M}$ twice. First, we apply the code to each row of $\mathbf{M}$, yielding an $n \times p$ matrix $\mathbf{C}_1$. Then, we apply the code to each column of $\mathbf{C}_1$, yielding a $p \times p$ matrix $\mathbf{C}_2$. The prover computes a Merkle root for each column of $\mathbf{C}_2$, and then a Merkle root computed over those p roots is returned as a commitment to the polynomial. The verifier checks the well-formedness of a commitment by sending a random vector $\mathbf{r} \in \mathbb{F}^n$ along with $t = \Theta(\lambda)$ random indices in $[p]$. The prover responds with a vector $\mathbf{v} = \mathbf{r}^T \cdot \mathbf{M}$ along with the corresponding t columns of $\mathbf{C}_1$ and the associated Merkle paths for the encodings of those columns. The verifier (i) evaluates the codeword $\mathbf{w} = E(\mathbf{v})$; (ii) for each challenge index i, the returned column $\mathbf{C}_1[:, i]$ satisfies $\mathbf{w}_i = \mathbf{r}^T \cdot \mathbf{C}_1[:, i]$; and (iii) the Merkle paths are all valid. Evaluation proofs can be generated similarly, but we omit details. Orion achieves zero knowledge and succinctness by randomized encoding and composition with a generic succinct zero-knowledge proof [153].

Accumulation Scheme for Orion. Assume the verifier holds commitments $\{\sigma_i\}$ to polynomials $\{f_i\}$ known to the prover, and let $\mathbf{M}_i, \mathbf{C}_{i,1}, \mathbf{C}_{i,2}$ be the corresponding matrices that were generated by the prover when committing to f_i. The prover generates a commitment σ^* to $f^* = \sum \alpha_i f_i$ and sends it to the verifier; let $\mathbf{M}^*, \mathbf{C}_1^*, \mathbf{C}_2^*$ be the corresponding matrices generated during this process. For an honest prover $\mathbf{M}^* = \sum_i \alpha_i \mathbf{M}_i$, $\mathbf{C}_1^* = \sum_i \alpha_i \mathbf{C}_{i,1}$, and $\mathbf{C}_2^* = \sum_i \alpha_i \mathbf{C}_{i,2}$.

On the other hand, if $\mathbf{M}^* \neq \sum_i \alpha_i \mathbf{M}_i$, then letting δ be the minimum distance of E, the matrices $\mathbf{C}_2^*$ and $\sum_i \alpha_i \mathbf{C}_{i,2}^*$ must differ in at least δ^2 entries. The key observation is that, in addition to checking the well-formedness of σ^*, the verifier can check that the correct linear relationship holds between random entries of $\mathbf{C}_2^*$ and the corresponding entries of $\{\mathbf{C}_{i,2}\}$; if E has a constant relative distance, it suffices to check that the linear combination relationship holds for $\Theta(\lambda)$ randomly opened entries of each $\mathbf{C}_2^*$ and $\{\mathbf{C}_{i,2}\}$. Formal details are given in Figure 3.5.

Theorem 3.5.1. Π_{ACC} *in Figure 3.5 is complete, knowledge sound, and zero-knowledge.*

Proof. Completeness is followed immediately. The soundness proof proceeds in two steps. First, we must prove the soundness with respect to commitment accumulation, i.e., if we can prove the well-formedness of σ^*, the verifier is convinced that each σ_i is well-formed, and the linear combination relation holds. Second, we should prove the soundness with respect to evaluation accumulation, i.e., if (σ^*, x^*, y^*) is satisfied, then $\forall i : (\sigma_i, x_i, y_i)$ is satisfied.

Consider the case of commitment accumulation. Let δ be the distance of the underlying code. First, suppose that the linear combination relation holds. Assume there exists some σ_i or σ_i', which is not well-formed, i.e., there exists a cth column of $\mathbf{C}_{i,2}$ or $\mathbf{C}'_{i,2}$, which is not a valid codeword. Then, over the choice of random challenges α_i and β_i, the $\mathbf{c}$th column of $\mathbf{C}_2$ is not a valid codeword with overwhelming probability [83, Claim 1], which yields σ^* to be not

Π_{ACC}

Let $\mathsf{PCS} = (\mathsf{Setup}, \mathsf{Commit}, \mathsf{Open}, \mathsf{Verify})$ denote the Orion [147] commitment with public parameters pp. Let $f_1, f_2, \ldots, f_k$ be ℓ-variate polynomials with variable-degree d; each has an evaluation $y_i = f_i(x_i)$ and is committed to by σ_i with randomness r_i. $\mathbf{M}_i$ denotes coefficients of f_i parsed as a square matrix, and $\mathbf{C}_{i,1}, \mathbf{C}_{i,2}$ denotes the associated encoded matrices generated during the commitment phase (see Section 3.5.2 for details).

Phase 1: Evaluations

- $\mathcal{P}$ and $\mathcal{V}$ interpolate a polynomial $L : \mathbb{F} \to \mathbb{F}^\ell$ such that $L(i) = x_i$.
- $\forall i \in [1, k]$, $\mathcal{P}$ samples $h_{i,1} \ldots, h_{i,\ell} \xleftarrow{\$} \mathbb{F}^{<d\ell}[X]$, sets $h_i(z_1, \ldots, z_\ell) := \sum_{j\in[1,\ell]} h_{i,j}(z_j)$, and sends $v_i := h_i(x_i)$ and $\sigma'_i \leftarrow \mathsf{PCS.Commit}(h_i, r'_i, \mathsf{pp})$ to $\mathcal{V}$. Let $\mathbf{M}'_i$ denote the coefficient matrix of h_i and $\mathbf{C}'_{i,1}, \mathbf{C}'_{i,2}$ denote the associated encoded matrices.
- $\forall i \in [1, k]$, $\mathcal{V}$ sends $\beta_i \xleftarrow{\$} \mathbb{F}$, and $\mathcal{P}$ sends the polynomial $g_i \leftarrow (f_i + \beta_i h_i) \circ L$.
- $\forall i \in [1, k]$, $\mathcal{V}$ checks whether $g_i(i) = y_i + \beta_i v_i$; if so, sends $\alpha_i \xleftarrow{\$} \mathbb{F}$ and $r \xleftarrow{\$} \mathbb{F}$.
- Parties compute $g \leftarrow \sum_{i=1}^{k} \alpha_i g_i$ and output $x^* = L(r)$, and $y^* = g(r)$.
- $\mathcal{P}$ computes the output witness polynomial $f^* = \sum_{i=1}^{k} \alpha_i f_i + \alpha_i \beta_i h_i$.

Phase 2: Commitments

- $\mathcal{P}$ sends $\sigma^* \leftarrow \mathsf{PCS.Commit}(f^*, r^*, \mathsf{pp})$. Let $\mathbf{M}^*$ denote the coefficient matrix of f^* and $\mathbf{C}^*_1, \mathbf{C}^*_2$ denote the associated encoded matrices.
- $\mathcal{V}$ randomly samples and sends a subset of indices I of size $t \in \Theta(\lambda)$.
- $\forall i \in [1, k]$, $\forall \mathsf{idx} \in I$, $\mathcal{P}$ sends $s^* = \mathbf{C}^*_2[\mathsf{idx}]$, $s_i = \mathbf{C}_{i,2}[\mathsf{idx}]$, and $s'_i = \mathbf{C}'_{i,2}[\mathsf{idx}]$ with their corresponding Merkle proofs.
- $\forall i \in [1, k]$, $\forall \mathsf{idx} \in I$, $\mathcal{V}$ checks $s^* = \sum_{i=1}^{k} \alpha_i s_i + \alpha_i \beta_i s'_i$ and the Merkle proofs are valid.
- The verifier returns (σ^*, x^*, y^*) as the accumulated instances, the prover returns $(\sigma^*, x^*, y^*), (f^*, r^*)$ as the accumulated instance–witness pair.

Figure 3.5: Accumulation scheme for Orion polynomial commitments.

well-formed. Similarly, if there exist an rth row of $\mathbf{C}_{i,1}$ or $\mathbf{C}'_{i,1}$, which is not code-word, it would cause the rth row of $\mathbf{C}_1$ to be an invalid code-word, yielding the verifier to return reject during the well-formedness check of σ^*. We should then show that the verifier returns reject if the linear

combination does not hold. Suppose that this case occurs, and $\mathbf{M}^* = \mathbf{M} + \sum_i \alpha_i \mathbf{M}_i + \alpha_i \beta_i \mathbf{M}'_i$, where $\mathbf{M}$ is a non-zero matrix, i.e., there exist an index $(\mathsf{idx}_r, \mathsf{idx}_c)$ such that $\mathbf{M}[\mathsf{idx}_r, \mathsf{idx}_c] \neq 0$. If they apply the row-wise encoding, we have $\mathbf{C}_1^* = \mathbf{C}_1 + \sum_i \alpha_i \mathbf{C}_{i,1} + \alpha_i \beta_i \mathbf{C}'_{i,1}$, where the row $\mathbf{C}_1[\mathsf{idx}_r, :]$ has at least δ non-zero entries. Then, if we apply the column-wise encoding, we have $\mathbf{C}_2^* = \mathbf{C}_2 + \sum_i \alpha_i \mathbf{C}_{i,2} + \alpha_i \beta_i \mathbf{C}'_{i,2}$, where $\mathbf{C}_2$ includes δ^2 non-zero entries. In other words, the two-step application of the encoding scheme propagates the inconsistency through the matrix. Let $\gamma = \frac{\delta}{p}$, be the relative distance of the code, where p is the codeword size. By opening $\Theta(\lambda)$ entries, all opened values are zero with a negligible probability of at most $(1-\gamma^2)^{\Theta(\lambda)}$.

Then, for the case of evaluation accumulation, suppose that we have $\exists i : f_i(x_i) \neq y_i$. If the prover sends a correct $g_i = (f_i + \beta_i h_i) \circ L$, then with overwhelming probability over the choice of the β_i, we should have $y_i + \beta_i v_i \neq g_i(i)$, and the verifier aborts. Therefore, the prover must send some $g_i \neq (f_i + \beta_i h_i) \circ L$ to pass the verification. However, with overwhelming probability,

$$g = \sum_i \alpha_i g_i \neq \sum_i \alpha_i (f_i + \beta_i h_i) \circ L = f \circ L$$

Then, the Schwartz-Zippel lemma implies $(f \circ L)(r) \neq g(r)$ except with negligible probability.

Knowledge soundness follows immediately from the above. Our accumulation scheme requires the prover to give the verifier oracle access to matrices $\mathbf{C}_2^*$, $\mathbf{C}_{i,2}$, and $\mathbf{C}'_{i,2}$ by their Merkle root. It is well known that for such Merkle-based oracle accesses, one can build a straight-line extractor to extract the witness committed to by its Merkle root [22, 116]. Once matrices are extracted, we can recover coefficients by running the decoding procedure.

Zero knowledge holds since each $h_{i,j}$ is uniformly sampled from the set of polynomials in $\mathbb{F}[X]$ of degree $d\ell$. The simulator Sim first samples random polynomials $\{h^*_{i,j}\}_{i=1,j=1}^{k,\ell}$, and sets

$h_i^*(z_1, \ldots, z_\ell) = h_{i,1}^*(z_1) + \cdots + h_{i,\ell}^*(z_\ell)$. Then, Sim commits to each h_i^* and the sends $\sigma_i'^*$ and the evaluation $v_i^* = h_i^*(x)$ to $\mathcal{A}$. As $h_{i,j}^*$ in the ideal world and $h_{i,j}$ in the real world both are sampled uniformly from the same distribution, they are indistinguishable, and hence, polynomials h_i^* and h_i and the evaluations v_i and v_i^* are indistinguishable. Moreover, as PCS is zero-knowledge σ_i' and $\sigma_i'^*$ are indistinguishable. Sim then samples input polynomials $\{f_i^*\}_{i=1}^k$ such that $f_i^*(x_i) = y_i$ and sends σ_i^*, a commitment to f_i^* to the verifier, which is indistinguishable from σ_i. Sim also sends polynomials $g_i^* = (f_i^* + \beta_i h_i^*) \circ L$. We note that each $f_i \circ L$ is a univariate polynomial from the set $\mathbb{F}[X]$ of degree at most $dk\ell$, and $h_i \circ L$ is also a univariate polynomial over the field with the same degree. Therefore, each coefficient of the polynomial $f_i \circ L$, and hence g_i, is masked by $h_i \circ L$. Then, as h_i^* and h_i are indistinguishable, g_i^* and g_i are also indistinguishable.

□

Efficiency. The prover is dominated by generating the commitment σ^*, which costs $O(d^\ell)$, and evaluating the $\{g_i\}$, which costs $O(kd^\ell)$. The proof size is dominated by the Merkle paths of size $O(kd\ell)$; this is also the verifier cost. The scheme can be made non-interactive via Fiat-Shamir.

3.6 Recursive Sumcheck Arguments

In this section, we present our GKR-based PCD framework. In what follows, we discuss our techniques for minimizing the recursion overhead in this framework.

Accumulation of Polynomial Commitments. In GKR-based proof systems (e.g., our PoGD), the prover commits to the input and/or output layers of the circuit and opens these commitments at several random evaluation points. Verifying these openings inside the recursion circuit would blow up the recursion overhead. To mitigate this, the prover uses an accumulation scheme for the

commitments to fold all openings into a single instance. The recursion circuit then verifies only the accumulation messages (which is more efficient than verifying each individual opening) and passes the accumulated instance to the next step.

More formally, each PCD node v receives two commitment–evaluation instances through each incoming edge e_i: (i) $(\sigma^*_{e_i}, x^*_{e_i}, y^*_{e_i})$, which is the accumulated instance carried by e_i; this includes a commitment $\sigma^*_{e_i}$ and an evaluation tuple $(x^*_{e_i}, y^*_{e_i})$; and (ii) $(\sigma_{e_i}, x_{e_i}, y_{e_i})$, which is the commitment–evaluation instance for the execution at the predecessor node along edge e_i. Note that, for simplicity, we assume that only the input layers of circuits are committed, and the output layer is given in clear to the verifier. These instances are accumulated via an accumulation scheme, and the result $(\sigma^*_e, x^*_e, y^*_e)$ is passed to the successor nodes.

Handling Wiring Predicates. The GKR verifier must also evaluate the wiring predicates of the circuit at several random points to validate the sumcheck messages. Hard-coding all these predicates in the recursion circuit blows up the recursion cost. In our framework, we delegate these evaluations to the prover. Specifically, we use evaluation-reduction sumcheck, i.e., the sumcheck protocol on Equation 3.5, to combine all wiring-predicate evaluations into a single one, which is then passed to the next step. The final verifier receives and validates the accumulated evaluation of the wiring predicates.

Sumcheck-Friendly Hash Functions. Due to the Fiat–Shamir transform, the GKR verifier must compute a hash function on many inputs to obtain the Fiat–Shamir challenges. Furthermore, the verifier must invoke the hash function multiple times to verify the Merkle paths included in accumulation messages (cf. Section 3.5). To reduce this cost, we instantiate the hash function with MiMC-p/p, a sponge construction whose ith round function is $F_i(x) = (x + k_i)^3 \bmod p$ for

a round key k_i. Suppose that there are n hash invocations, and let d denote the number of rounds of the hash function. We can prove each round by running a single sumcheck protocol. Let $\mathbf{v}_0$ be the vector of hash outputs, $\mathbf{v}_d$ the vector of inputs, and $\mathbf{v}_i := F_i(\mathbf{v}_{i-1})$ for all $i \in [d]$. Given a challenge r_0, the prover applies sumcheck to

$$\widetilde{\mathbf{v}}_0(r_0) = \sum_{z \in \{0,1\}^{\log n}} \widetilde{\beta}_z(r_0) \cdot (\widetilde{\mathbf{v}}_1(z) + k_1)^3. \tag{3.6}$$

This ends with an evaluation $\widetilde{\mathbf{v}}_1(r_1)$ and proceeds to the next round's sumcheck invocation.

Putting Everything Together. Let v denote a PCD node with local witness w_v and incoming edges $e_1, \ldots, e_k$. Each edge e_i carries a message z_{e_i} and a proof π_{e_i}, such that π_{e_i} includes proof messages (i.e., sumcheck messages and accumulation messages), a commitment accumulator acc_{e_i}, a wiring-predicate accumulator wp_{e_i}, and a commitment–evaluation tuple $(\sigma_{e_i}, x_{e_i}, y_{e_i})$. The node v verifies the proof messages, accumulates $\{\mathsf{acc}_{e_i}\}_i$ and $\{(\sigma_{e_i}, x_{e_i}, y_{e_i})\}_i$ to an updated commitment accumulator acc_e, and accumulates $\{\mathsf{wp}_{e_i}\}_i$ to an updated wiring-predicate accumulator wp_e. Finally, v outputs acc_e, wp_e, and $z_e := F_v(z_{e_1}, \ldots, z_{e_k}, w_v)$, where F_v denotes the transition function. We assume all predecessor nodes of v share the same transition function.

Once the execution at node v is completed, the prover applies hash sumchecks to the MiMC-p/p hashes and the baseline argument to the other components. The prover also generates accumulation messages for the next step. In the final step, the prover must additionally provide opening proofs for the final commitment accumulator. We provide formal details in Figure 3.6.

Theorem 3.6.1. Π_{PCD} *in Figure 3.6 is complete, knowledge sound, and zero-knowledge.*

Proof. First, we consider completeness. For a PCD node v, let $\{(z_{e_i}, \pi_{e_i})\}_{e_i}$ denote the incoming

$\underline{\Pi_{\mathsf{PCD}}}$

Let PCS denote a polynomial commitment scheme; let ACC denote an accumulation scheme for PCS; and let GKR denote a GKR-based baseline argument. Let $\mathcal{F}_v$, ω_v, and Π_v denote the transition function, local witness, and PCD argument for node v, respectively. $\Pi_v.\mathsf{Setup}$ runs the setup algorithm of PCS. Below, we describe $\Pi_v.\mathsf{Prove}$ and $\Pi_v.\mathsf{Verify}$.

$\underline{(z_e, \pi_e) \leftarrow \Pi_v.\mathsf{Prove}(\mathsf{pp}, \{(z_{e_i}, \pi_{e_i})\}_{e_i}, \omega_v) :}$

- Parse π_{e_i} as a tuple $(\pi_{e_i,G}, \pi_{e_i,A}, \pi_{e_i,E}, \mathsf{acc}_{e_i}, \mathsf{wp}_{e_i}, (\sigma_{e_i}, x_{e_i}, y_{e_i}))$, where $\pi_{e_i,G}$ denotes the sumcheck messages, $\pi_{e_i,A}$ denotes the accumulation messages, $\pi_{e_i,E}$ denotes the opening messages, acc_{e_i} denotes the commitment accumulator, wp_{e_i} denotes the wiring-predicate accumulator, and $(\sigma_{e_i}, x_{e_i}, y_{e_i})$ denotes a commitment–evaluation tuple.
- $z_e \leftarrow F_v^A(\mathsf{pp}, \{(z_{e_i}, \pi_{e_i})\}_{e_i}, \omega_v)$:
  1. Verify the Fiat–Shamir and Merkle hashes in $\pi_{e_i,G}$ and $\pi_{e_i,A}$.
  2. Verify the sumcheck messages $\pi_{e_i,G}$, which must terminate in $(\sigma_{e_i}, x_{e_i}, y_{e_i})$ and wp_{e_i}.
  3. Verify the accumulation messages $\pi_{e_i,A}$, which must terminate in acc_{e_i}.
  4. If verification succeeds, compute $z_e := F_v(z_{e_1}, \ldots, z_{e_k}, \omega_v)$.
- Let $\mathbf{u}_e$ be the inputs to the current invocation of F_v^A. Set $\sigma_e \leftarrow \mathsf{PCS.Commit}(\mathsf{pp}, \widetilde{\mathbf{u}}_e; r_e)$.
- Run hash sumcheck for step (1) and $\mathsf{GKR.Prove}$ on the other steps of the current F_v^A.
- Combine all wiring-predicate accumulators into wp_e and the evaluations of $\widetilde{\mathbf{u}}_e$ into (x_e, y_e) via an evaluation-reduction sumcheck. Include the sumcheck messages in $\pi_{e,G}$.
- Run $\mathsf{ACC.Prove}$ on input $\{\mathsf{acc}_{e_i}\}_{e_i}$ and $\{(\sigma_{e_i}, x_{e_i}, y_{e_i})\}_{e_i}$ to accumulate all received commitment–evaluation instances into acc_e. Include the aggregation messages in $\pi_{e,A}$.
- If this is the final step, include in $\pi_{e,E}$ the opening proofs for $\mathsf{acc}_e.x$ and (σ_e, x_e, y_e), and set $\mathsf{acc}_e.w = \perp$. Otherwise, set $\pi_{e,E} := \mathbf{u}_e$.
- Let $\pi_e := (\pi_{e,G}, \pi_{e,A}, \pi_{e,E}, \mathsf{acc}_e, \mathsf{wp}_e, (\sigma_e, x_e, y_e))$, and return (z_e, π_e).

$\underline{\{0,1\} \leftarrow \mathcal{V}(\mathsf{pp}, z_e, \pi_e) :}$

- Parse $\pi_i := (\pi_{i,G}, \pi_{i,A}, \pi_{i,E}, \mathsf{acc}_e, \mathsf{wp}_e, (\sigma_e, x_e, y_e))$.
- Verify $\pi_{i,G}$, $\pi_{i,A}$, and $\pi_{i,E}$, and their consistency with z_e, acc_e, wp_e, and (σ_e, x_e, y_e).

Figure 3.6: GKR-based PCD framework.

message–proof tuples and ω_v denote the local witness. We must show that if all (z_{e_i}, π_{e_i}) are satisfiable, then (z_e, π_e) returned by the node v is also satisfiable. Since for each (z_e, π_e) we have

$\Pi_v(\mathsf{pp}, z_{e_i}, \pi_{e_i}) = 1$, the function F_v^A completes successfully, and z_e results from a correct transition. π_e consists of sumcheck messages $\pi_{e,G}$, accumulation messages $\pi_{e,A}$, opening messages $\pi_{e,E}$, a commitment accumulator acc_e, a wiring-predicate accumulator wp_e, and a commitment–evaluation instance (σ_e, x_e, y_e). Due to the completeness of GKR, $\pi_{e,G}$ is satisfiable and consistent with wp_e; due to the completeness of ACC, $\pi_{e,A}$ is satisfiable and consistent with acc_e; and due to the completeness of PCS, $\pi_{e,E}$ is satisfiable and consistent with acc_e and (σ_e, x_e, y_e).

Second, we consider knowledge soundness. Let ξ^{PCS} be the extractor of the commitment scheme PCS. Let (z_e, π_e) be a satisfiable message–proof tuple returned by a node v. Since $\pi_{e,E}$ is satisfiable, we can run ξ^{PCS} on input (σ_e, x_e, y_e) to extract $\mathbf{u}_e$, the input to the function F_v^A. The input $\mathbf{u}_e$ includes the local witness ω_v and all incoming message–proof tuples $\{(z_{e_i}, \pi_{e_i})\}_{e_i}$. The PCD extractor can then recursively invoke the extractor for each node, from sink to source, to extract the witness. By the soundness of PCS, GKR, and ACC, the extraction succeeds.

Finally, we consider zero knowledge. We build a simulator Sim that produces proof messages and commitments indistinguishable from the real proofs and commitments. In particular, Sim runs the simulator of GKR to simulate sumcheck messages, the simulator of ACC to simulate accumulation messages, and the simulator of PCS to simulate opening messages. □

Efficiency. Consider an instantiation with Orion [147] as the commitment scheme and the GKR protocol as the baseline proof; at each node v, the prover time is $O(|F_v| + |\omega_v| + \sum_{e_i} |z_{e_i}|)$. The proof size/verifier time is $O(d \cdot \log |F_v| + \log^2(|\omega_v| + \sum_{e_i} |z_{e_i}|))$, where d is the depth of F_v.

3.7 Zero-Knowledge Proofs of Training

In this section, we define zk-PoTs and present a concrete instantiation for DNNs, KAIZEN.

3.7.1 Definition

Garg et al. [77] introduced a monolithic definition of zk-PoT that proves the correct execution of a universal circuit representing the entire training procedure. Such a definition is not suitable for iterative training algorithms, such as gradient descent, since it requires fixing the number of training iterations in advance; moreover, the prover must complete all iterations to generate a satisfiable proof. In many applications, a model owner incrementally updates the model, and it is therefore desirable to have a succinct zk-PoT at any stage of the training process, attesting to the correctness of the iterations executed so far. To address this, we propose an incremental definition of zk-PoT that generates a zk-PoT at the end of each iteration.

A consideration in an incremental definition is ensuring data consistency: each iteration is given only a batch of data items claimed to be randomly sampled from the dataset. To verify this, we assume that there exists a fixed (pseudo)random permutation of the dataset for each epoch and that the dataset is committed to by a *position-binding* commitment, e.g., a Merkle tree. At each iteration, the prover uses the batch consistent with the permutation and provides Merkle paths for each data item in the batch. Before running gradient descent, these paths are first validated.

More precisely, let $\mathcal{C}$ be the iteration circuit in which the permutations are hard-coded. Let $\mathbf{W}_0$ be some initial model and $\mathcal{D}$ the dataset committed to by ρ. In the ith iteration, $\mathcal{C}$ takes as input the iteration counter i, the dataset commitment ρ, the previous model weights $\mathbf{W}_{i-1}$, a batch of data samples $\mathbf{B}_{i-1} \subseteq \mathcal{D}$, and the batch opening $\mathbf{p}_{B_{i-1}}$. The circuit first validates the opening and the consistency of $\mathbf{B}_{i-1}$ with the hard-coded permutations. If verification passes, it applies the gradient descent step and outputs weights $\mathbf{W}_i$.

A zk-PoT enables a prover to commit to a dataset and initial model weights, sample a

batch in each iteration, and execute $\mathcal{C}$. After completing each iteration, the prover can generate a commitment to the updated weights together with a succinct proof that all iterations have been executed correctly. For this primitive to be practical, we require both the proof size and the verifier runtime to be $O(|C|)$ and independent of the number of iterations, while the prover cost should be $O(|C| \log |C|)$ per iteration. A zk-PoT satisfies soundness if no adversary can produce a valid proof for inconsistent counters and commitments. Knowledge soundness ensures that any prover producing a valid proof must know the corresponding dataset and satisfying weights. Zero knowledge guarantees that no additional information about the model or dataset is revealed.

3.7.2 zk-PoTs for Neural Networks

KAIZEN commits to the dataset via a Merkle tree, and the opening of a particular data batch includes the Merkle paths for each data item in the batch. The weights are committed to via a polynomial commitment to their multilinear extension.

Let PRP be a pseudorandom permutation, e.g., a cubic function, with the initial key s_0. The key can be updated for the eth epoch as $s_e = \mathcal{H}(e + s_0)$, given a collision-resistant hash function $\mathcal{H}$; this ensures that each epoch trains the model on a randomly sampled partition. Let N be the batch size. The ith iteration of the eth epoch executes $\mathcal{C}$ as follows:

1. $s_e \leftarrow \mathcal{H}(e + s_0)$, and for all $k \in [N]$, set $v_k \leftarrow \mathsf{PRP}(s_e, (i-1)N + k)$.

2. Validate that $\forall k \in [N] : \mathsf{MT.Verify}(v_k, \rho_{\mathcal{D}}, \mathbf{B}_{i-1}[k], \mathbf{p}_{B_i}[k]) = 1$.

3. Return $\mathbf{W}_i = \mathsf{GD}(\mathbf{W}_{i-1}, \mathbf{B}_{i-1})$, where GD denotes the gradient descent algorithm.

KAIZEN is a concrete instantiation of our PCD framework, outlined in Figure 3.6. Specifically, KAIZEN is a leaf-decoupled PCD [122] in which the transition function at the leaf nodes

is $\mathcal{C}$, as defined above. Internal nodes only accumulate their children and do not execute any transition function. To prove the correct execution of $\mathcal{C}$ at the leaf nodes, the prover runs the GKR protocol for step (1), hash sumcheck (cf. Section 3.6) for step (2), and PoGD for step (3). The polynomial commitment scheme is instantiated with Orion, and the accumulation scheme is instantiated with our construction outlined in Figure 3.5.

Efficiency. Let $s_{\mathsf{in},\ell}$ be the input size and $s_{\mathsf{out},\ell}$ the output size of the ℓth layer, let N denote the batch size, and let q denote the bit-length of quantization. The prover cost of KAIZEN is $O(\sum_{\ell=0}^{L-1} N s_{\mathsf{in},\ell} s_{\mathsf{out},\ell} + N q s_{\mathsf{out},\ell})$ for both leaf and internal nodes. The proof size and verifier cost are $O(\sum_{\ell=0}^{L-1} \log^2(N\, s_{\mathsf{in},\ell}\, s_{\mathsf{out},\ell}) + \log^2(N q s_{\mathsf{out},\ell}))$, independent of the number of iterations.

3.8 Implementation and Evaluation

We implement KAIZEN and evaluate its performance. We use open-source libraries [107, 152] for the sumcheck and GKR protocols. Our implementation does not support the zero-knowledge variants of the sumcheck protocol, the Orion commitment scheme, or the accumulation scheme. However, since it suffices to apply the zero-knowledge extensions only to the final iteration, the amortized cost would be insignificant. Our code is publicly available at `https://github.com/ChristodoulosPappas/zkPoT`. Our implementation could be improved through parallelization, which we did not attempt. We ran experiments on a Linux VM with 8 physical CPUs (2.80 GHz Intel(R) Xeon(R) Platinum 8370) and 512GB of RAM.

Protocol Instantiation. Our construction works over any finite field. In our implementation, we choose the extension field $\mathbb{F}_{p^2}$ for the prime $p = 2^{61} - 1$. We use MiMC-p^2/p^2 [6] with 80 rounds and SHA-256 as our hash functions. The pseudorandom permutation PRP for shuffling

the dataset is instantiated by a cubic function over a field whose size matches that of the dataset. We set $q = 64$ and $F = 32$ in the quantization scheme to avoid overflow. We use a PCD tree with arity $m = 12$ and depth $d = 4$; hence, our implementation supports $12^4 = 20736$ iterations. Our parameters are chosen to achieve $\lambda = 100$ bits of security. We instantiate our polynomial commitments with Orion [147]. A difficulty with the Orion protocol is the small relative distance of its code, which results in a significant number of Merkle-tree openings in the accumulation scheme. To reduce this number, instead of parsing the coefficients of polynomials as square matrices, we parse them as matrices with a small constant number of columns and linearly many rows. Although this yields quasi-linear opening time, the commitment and accumulation times remain linear. As in KAIZEN, openings are invoked only once at the final iteration.

Furthermore, we replace MiMC-p^2/p^2 with SHA-256 in up to the bottom 4 layers of the Merkle trees to reduce commitment-generation time.

Gradient Descent API. We allow users to select specifications such as the model architecture, batch size, and learning rate. One can also add any number of convolutional or dense layers to the architecture and specify input, output, and weight sizes. KAIZEN supports a wide range of activation functions: ReLU, Tanh, Sigmoid, Softmax, and both average and max pooling. Our implementation further supports an arbitrary number of input and output channels for convolutional layers. In particular, when a convolutional layer has multiple channels, one convolution is required for each input/output channel pair. These convolutions are then turned into multiple FFTs of the same size. By the linearity of the FFT, we can reduce all these FFTs to a single FFT via a random linear combination—a technique proposed in prior work [13].

Models and Datasets. We evaluate the performance of KAIZEN on three well-known model

architectures: LeNet [100], AlexNet [99], and VGG-11 [133]. For AlexNet and VGG-11, we reduce the number of parameters by decreasing the number of channels and the output sizes of the layers. LeNet has 3 convolutional layers, 2 average-pooling layers, 2 dense layers, and a total of 61,706 trainable parameters. AlexNet has 5 convolutional layers, 3 average-pooling layers, 2 dense layers, and a total of 4.2 million parameters. Finally, VGG-11 has 8 convolutional layers, 5 average-pooling layers, 3 dense layers, and a total of 10.1 million parameters. We use Softmax as the activation function for the output layers and ReLU for all other layers. We use the MNIST dataset [60], consisting of 60,000 grayscale 32×32 images, for LeNet, and CIFAR-10 [98], consisting of 60,000 RGB 64×64 images, for AlexNet and VGG-11.

3.8.1 KAIZEN zk-PoT

In this section, we report the results of our benchmarks for KAIZEN.

Prover Cost of PoGD. We report the prover time of PoGD, broken down into its four phases: the update step, the backward pass, the forward pass, and evaluation reduction. The batch size ranges from $N = 4$ to 16. The results are reported and summarized in Table 3.1; note that the prover cost of polynomial commitments is excluded. As shown in the table, generating the sumcheck messages for one gradient-descent iteration of LeNet with $N = 16$ takes only 13.44 seconds. For AlexNet and VGG-11, it takes 57.91 and 179.4 seconds, respectively.

KAIZEN Performance. The prover commits to the dataset via a Merkle tree. This takes 18 seconds for MNIST and 218 seconds for CIFAR-10, where the Merkle tree is built using the MiMC-p^2/p^2 hash function. The prover then executes the training iterations and computes the PCD tree. For leaf nodes, the cost is dominated by PoGD. Each internal node only accumulates its children and proves the correctness of the this accumulation.

	LeNet			AlexNet			VGG-11		
	$N=4$	$N=8$	$N=16$	$N=4$	$N=8$	$N=16$	$N=4$	$N=8$	$N=16$
Phase 1: Update	0.058	0.095	0.164	8.069	11.62	17.07	11.81	17.91	28.96
Phase 2: Backward	2.900	3.851	5.920	5.948	8.546	13.13	28.91	33.80	48.75
Phase 3: Forward	1.688	2.482	4.042	13.96	17.08	22.71	27.51	40.70	65.94
Phase 4: Evaluation	1.032	1.816	3.312	4.425	4.499	4.992	15.93	22.89	35.78
Total Prover Runtime (s)	5.676	8.244	13.44	32.42	41.72	57.91	84.15	115.3	179.4

Table 3.1: PoGD prover time as a function of the batch size N.

	LeNet			AlexNet			VGG-11		
	$N=4$	$N=8$	$N=16$	$N=4$	$N=8$	$N=16$	$N=4$	$N=8$	$N=16$
Commitments	70.04	93.46	106.6	94.72	139.7	233.7	140.3	233.5	452.1
Proof of Agg.	9.877	13.15	19.98	30.33	45.45	75.65	45.76	76.44	142.1
Proof of $\mathcal{V}_b$	26.61	26.63	52.76	104.8	104.5	104.7	106.2	106.8	105.6
Proof of $\mathcal{C}$	5.836	8.563	14.08	33.04	42.95	60.39	84.77	116.5	182.2
Total Prover Runtime (s)	112.4	141.8	193.4	262.9	332.6	474.4	377.0	533.2	882.0
Sumcheck Proofs	553.4	575.7	608.9	739.1	790.9	819.0	1045	1117	1154
Comm. Openings	366.8	434.2	412.2	388.9	395.9	436.3	424.4	448.3	473.3
Total Proof Size (KB)	920.2	1009	1021	1128	1187	1255	1469	1565	1627
Sumcheck Verifier	0.042	0.044	0.050	0.056	0.060	0.063	0.084	0.090	0.105
Comm. Verifier	0.020	0.021	0.023	0.021	0.022	0.023	0.022	0.023	0.025
Total Verifier Runtime (s)	0.062	0.065	0.073	0.077	0.082	0.086	0.108	0.113	0.130

Table 3.2: KAIZEN prover time, proof size, and verifier time for batch size N.

The prover time is reported as the *amortized* runtime of building the PCD tree per gradient descent iteration. We further break this down and report the individual costs of commitment, accumulation, the verifier circuit, and PoGD. We consider batch sizes N ranging from 4 to 16. The results are summarized in Table 3.2. As shown in the table, KAIZEN achieves a prover runtime of 193 seconds per iteration for LeNet, 474 seconds for AlexNet, and 882 seconds for VGG-11 at batch size $N = 16$. The maximum observed prover memory usage across all our experiments is 134.4GB for LeNet, 269.1GB for AlexNet, and 466.3GB for VGG-11.

We report the proof size and verifier time, with a breakdown into the sumcheck cost and the polynomial commitment cost. The verifier cost is independent of the number of iterations. For $N = 16$, the proof size is 1.021MB for LeNet, 1.255MB for AlexNet, and 1.627MB for VGG-11. The verifier time is 73ms for LeNet, 86ms for AlexNet, and 130ms for VGG-11.

	Prover (s)			Proof Size (KB)			Verifier (s)		
	LeNet	AlexNet	VGG-11	LeNet	AlexNet	VGG-11	LeNet	AlexNet	VGG-11
Fractal [52]	326,568	1,306,397	5,225,712	243	269	291	0.022	0.026	0.029
Halo [33]	50,850	203,399	813,595	4.98	5.21	5.49	3,970	15,882	63,528
Nova [96]	1,868	7,020	20,880	9.80	10.1	10.3	1,677	6,300	18,735
KAIZEN	193	474	882	1021	1255	1627	0.073	0.086	0.130

Table 3.3: KAIZEN vs. generic IVC schemes given batch size $N = 16$.

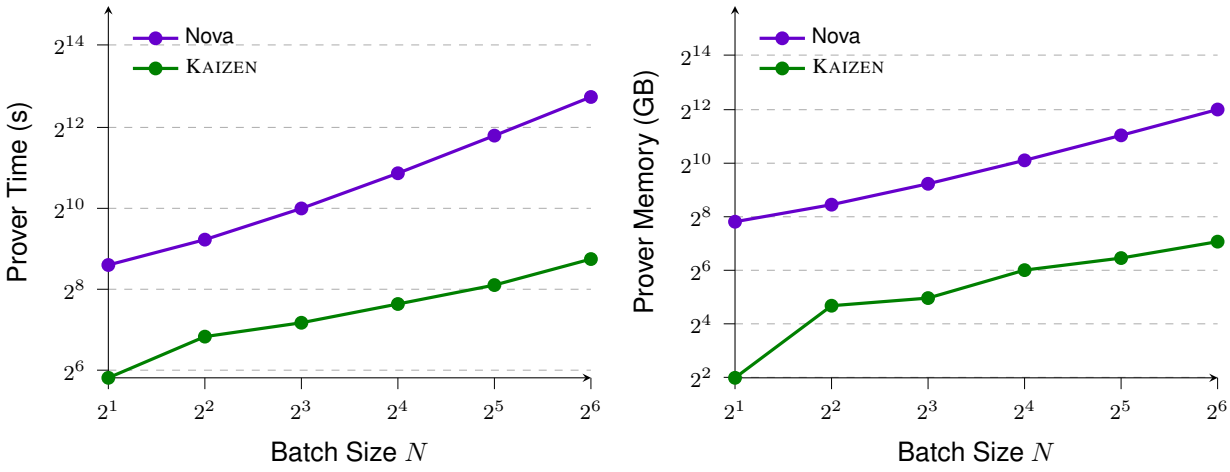


Figure 3.7: Prover time and memory usage of KAIZEN and Nova for LeNet.

3.8.2 Comparison to Generic Proofs

We compare the performance of KAIZEN with the generic IVC/PCD constructions Fractal [52], Halo [33], and Nova [96]. These schemes require the computation to be modeled as an arithmetic circuit; specifically, they rely on the *rank-1 constraint system* (R1CS) representation. We made an effort to minimize the size of the R1CS representation of the iteration circuit $\mathcal{C}$ when measuring the performance of these schemes. We handled matrix multiplications using Frievalds' algorithm and FFTs using the Cooley–Tukey algorithm, rather than the naive algorithms used in existing implementations [1]. Non-linear operations were handled using bit-decompositions. For $N = 16$, $\mathcal{C}$ has 429M gates for LeNet, 1.6B gates for AlexNet, and 5.1B gates for VGG-11.

The comparisons are summarized in Table 3.3. As shown, for VGG-11, our prover is $23.7\times$ faster than the state of the art, namely Nova. Our verifier is also significantly faster than those

of Halo and Nova. Our proof size is relatively large; however, in practical applications, it may be considered negligible. We extrapolate the performance of the generic schemes based on their asymptotic complexities. We also achieve $27\times$ lower memory usage.

We give a more detailed comparison of KAIZEN and Nova in terms of prover runtime and memory usage for LeNet with batch sizes $N = 2$ to 64. The results are reported in Figure 3.7. As shown, our prover is 6.9–$16.1\times$ faster, and we expect it to require 30.7–$56.2\times$ less memory.

Chapter 4

Secure Aggregation with a Lightweight Committee

Secure aggregation enables one or more servers to learn the sum of clients' private inputs, while revealing no information beyond the final sum. It underpins a wide range of applications, including federated learning [29, 54], and has recently seen real-world deployments [91, 144].

Secure aggregation can be realized via a generic secure multi-party computation; however, such an approach typically requires multiple rounds of interaction between the servers and the clients. In most applications, clients are end-user devices that voluntarily provide input to the servers. Hence, lowering the requirements for clients is a key design goal. The gold standard is *one-shot* protocols [17, 94], in which clients send a single message to the server(s). Another standard notion is *dynamic client participation*, in which the set of participating clients need not be known in advance, and clients may join the aggregation session at arbitrary points.

Single-server protocols face a separate challenge known as *residual attacks*, in which a malicious server colluding with a group of clients tries to learn the honest parties' inputs. To prevent this, a recent line of research [17, 94, 103, 112] relies on a committee of parties that assists the server and ensures that it acts honestly. Unfortunately, in all prior works, the committee's total work and communication scales at least linearly with the number of clients. This is a significant limitation since the committee is typically run by volunteers or other altruistic parties.

As a concrete example in Willow [17], with $n = 10^7$ number of clients, the committee needs to receive and verify ≈ 15 GB of proofs, taking ≈ 630 hours *in total* to verify them.

A natural approach to reduce the committee's cost is to delegate the verification task to the server via SNARKs. Specifically, the server proves that it has correctly executed the committee's verification procedure and produced the aggregate ciphertext. This can significantly reduce the committee's work and communication requirements, keeping them constant or logarithmic in the number of clients. Unfortunately, the verification statement is large and linear in the number of clients. This renders the use of *monolithic* SNARKs impractical, as they incur high memory costs at the server and do not support dynamic client participation or streaming aggregation.

In this work, we instead use PCD. This approach naturally supports streaming aggregation and dynamic participation. On the downside, recursive constructions introduce several additional challenges: (i) The verifier must ensure there are *no duplicate client contributions* included—a property that is non-trivial to encode in recursive statements; (ii) prior systems such as Willow, rely on non-succinct proofs to verify the aggregation; however, non-succinct proofs are inherently incompatible with recursive proof composition; and (iii) as we show, the security of aggregation systems requires the proof system to satisfy simulation extractability, a strong non-malleability notion. This requirement has largely been overlooked, and most prior efficient recursive proof systems have not been shown to satisfy simulation extractability. We address these challenges and provide a PCD-based secure aggregation scheme that requires minimal client cost and total committee overhead while maintaining reasonable server performance and efficiency.

Our contributions. We introduce WILLOWFOLD, a secure aggregation scheme with lightweight committees. WILLOWFOLD builds on Willow and reduces the verification cost to *logarithmic* in the number of clients, which effectively removes the need for a separate verifier role; *any* party—including the decryptors—can efficiently verify the correctness of the aggregation via a

succinct, efficiently verifiable proof. Moreover, WILLOWFOLD supports *streaming* aggregation and reduces per-client server storage by orders of magnitude compared to Willow. Specifically, Willow requires the server to *persistently* store all per-client contributions (1.5 KB), whereas in WILLOWFOLD the server stores a single ID (4 Bytes) for each client and can discard client contributions after they are aggregated. This facilitates the distributed execution of the server's computation. Below, we summarize our contributions.

- *Tightening Willow's Security Analysis.* We identify a gap in Willow's security analysis and show that security against *correlated ciphertext* attacks requires the underlying zero-knowledge proofs to satisfy *simulation extractability* (SE). Although the proof system used in the Willow protocol satisfies this property, the formalization has been omitted from the security proof.

- *Lightweight Secure Aggregation.* WILLOWFOLD uses PCD to offload the verifier's work in Willow to the server and reduce the committee's cost. The server collects per-client proofs and, using PCD, accumulates them into a succinct proof that certifies the correctness of the aggregation. Along the way, we need to overcome multiple challenges:

  – We need to mitigate *replay* attacks, in which an adversary may duplicate client contributions. To this end, the PCD statement maintains a binary vector indicating which clients have been aggregated and which have not. The size of this list is linear in the total number of clients; thus, representing it within the PCD statement exceeds our efficiency goals. To circumvent this, the server commits to one-hot vectors using a homomorphic, constant-size polynomial commitment scheme and distributes these commitments to clients as IDs. Then, in each aggregation session, the server sums the IDs and produces a succinct proof that the aggregated ID commits to a $\{0, 1\}$-vector, which implies that no ID is duplicated.

– As we discuss later, security against correlated ciphertext attacks requires the PCD scheme to satisfy SE. Despite recent efforts [4], SE remains relatively understudied in the context of PCDs. In our work, we use PCD schemes in a leaf-decoupled mode [122]. Accordingly, we tailor the definition of SE to leaf-decoupled PCDs, which we refer to as leaf-SE.

– Willow requires encoding a ring operation in its circuit. A naive implementation incurs a circuit of size $O(d^2)$. Instead, we use a probabilistic verification procedure with cost $O(d)$. Furthermore, these operations are non-native to the field used by group-based PCD, causing the circuit to exceed 6 million R1CS constraints. We implement several other optimizations that reduce the circuit size to 1.4 million constraints while keeping the witness sparse.

- *Instantiation.* We present an instantiation of WILLOWFOLD with Spartan+KZH-Fold [93], which is a PCD scheme with a succinct proof size and verification time. The challenge is that Spartan+KZH-Fold is neither zero-knowledge nor simulation extractable. In this work, we propose a zero-knowledge variant of Spartan+KZH-Fold that uses only a sublinear amount of randomness. We then show that this construction satisfies simulation extractability.

  At the core of our designs, we revisit prior work on zero-knowledge sumcheck [49, 146] and introduce a zero-knowledge sumcheck that uses only a logarithmic amount of randomness and, unlike prior work, can be instantiated with any multilinear polynomial commitment scheme.

- *Implementation and Evaluation.* We benchmark and evaluate WILLOWFOLD, demonstrating its practicality for real-world deployments, particularly by eliminating reliance on a committee whose cost scales linearly with the number of clients. For 2^{23} clients with input length $\ell = 10^7$, WILLOWFOLD has a proof size of 57 KB and committee computation time of less than one second—$250000\times$ and $2000000\times$ improvements over Willow, respectively.

Client computation takes approximately 3.3 seconds, and the server's computation per client is ≈ 4.3 seconds—representing $3.5\times$ and $22\times$ increases compared to Willow, respectively. However, server storage in WILLOWFOLD is only 4 bytes per client, compared to 1.5 KB in Willow—a $370\times$ improvement—thereby enabling more effective streaming aggregation.

4.1 Related Work

We compare WILLOWFOLD with Flamingo [112], LERNA [103], and OPA [94], as well as Willow [17]. Asymptotic costs are reported in Table 4.1. As shown, a common limitation of prior work is that the committee cost scales linearly with the number of clients, which is logarithmic in WILLOWFOLD. A more detailed discussion appears below.

Flamingo. This scheme relies on *pairwise masking* to realize one-shot secure aggregation. In addition to the committee's cost, Flamingo has two major limitations. First, it requires a PKI available for all clients. Further, although Flamingo achieves one-shot communication, the subset of clients participating in each session must be fixed and known in advance; thus, it doesn't support dynamic client participation. On the positive side, Flamingo supports changing the committee members between sessions while re-using the setup via key resharing.

LERNA. Similar to Willow and WILLOWFOLD LERNA achieves one-shot clients and dynamic client participation using homomorphic encryption. Its main limitation is concrete communication cost—e.g., with 20K clients and input length 50K, LERNA requires 2GB of communication per client and 4.4GB per committee member. LERNA also requires a stateful committee.

OPA. This scheme supports one-shot clients, dynamic participation, and a stateless committee. Compared to WILLOWFOLD, OPA requires a large committee size. Specifically, to achieve a log-

arithmic per-member committee cost, OPA requires $O(n)$ committee members; WILLOWFOLD uses a constant-size committee. This poses a challenge for large-scale deployments. Therefore, OPA is better suited to settings with multiple iterations and a small number of clients.

Willow. Built on Willow, our scheme inherits many of its properties—one-shot clients, dynamic participation, and a stateful committee. The main difference is the committee cost: it improves from linear in Willow to logarithmic in WILLOWFOLD, at the cost of slightly higher server and client overhead (see Section 4.7.3 for a concrete comparison).

Other Related Work. We do not directly compare our work with single-server secure aggregation schemes that require multiple synchronization rounds between clients [16, 29]. These protocols do not support scenarios with low client availability and incur high latency. Secure aggregation has also been extensively studied in the multi-server setting [5, 55]. While this setting can offer better concrete efficiency than single-server protocols, each server incurs communication and computation costs proportional to the number of clients, and the servers must be non-colluding, which introduces logistical constraints. An orthogonal line of work studies input validation in secure aggregation [15]. We do not consider this aspect; however, we believe such techniques are compatible with WILLOWFOLD.

Concurrent Work. TACITA [113] is a one-shot secure aggregation scheme that supports dynamic participation and further achieves security against "Sybil attacks" where an adversary who controls a large majority of clients, e.g., $n - 1$, can recover honest client inputs. To do this, TACITA employs homomorphic threshold signatures. Their technique, however, incurs a high concrete cost. In particular, while client and committee communication costs are constant, each client's computation scales linearly with the number of clients. For example, with only 1000

Protocol	Client	Server	Committee
Flamingo [112]	$\ell + \log n$	$n(m + \log n + \ell)$	$n \log n$
LERNA [103]	ℓ	$n\ell$	$n + \ell$
OPA [94]	$\ell + m$	$n(\ell + m)$	n/M
Willow [17]	ℓ	$n\ell$	$n/M + m$
WILLOWFOLD	$\ell + \log n$	$n(\log n + \ell)$	$\log n + m$

Table 4.1: n is the number of clients, ℓ the client input size, M the number of subcommittees, and m the size of each subcommittee; m is a constant determined by the security parameter and corruption threshold. For OPA and Willow, it additionally grows with $\log M$. In both protocols, there is a trade-off between M and n, since each committee member's communication and computation costs scale with $O\left(\frac{n}{M}\right)$. Communication and computation have the same asymptotics, except for Flamingo's committee, which has communication $O(n \log n)$ and computation $O(n^2)$.

clients, client's work takes 32 seconds per aggregation session, while WILLOWFOLD has logarithmic client cost and can efficiently handle even millions of clients with about 3 seconds of computation per client. Furthermore, unlike WILLOWFOLD, TACITA requires PKI between clients and supports only small input domains. For WILLOWFOLD, potential defenses against Sybil attacks include enforcing a high minimum threshold on the number of client contributions or applying differential privacy techniques [17, Appendix I].

Heli [102] is a concurrent work that builds on techniques from OPA. It proposes a two-server aggregation scheme in which the *aggregator* server performs the bulk of the computation, while the *decryptor* server assists only with registration and decryption and incurs smaller overhead. Although Heli achieves improved concrete efficiency compared to other multi-server schemes, the decryptor server still has linear overhead in the number of clients: it must register with all clients, and its final decryption computation scales linearly with the number of users who drop out. Finally, as noted above, from a logistical standpoint, it is often more practical to assume that a committee of users does not collude with the central server than to rely on the existence of two non-colluding servers. It is unclear how to realize the decryptor as a lightweight committee.

4.2 Technical Overview

In this section, we overview our results. Specifically, we first introduce the threat model and define the ideal functionality for secure aggregation. We then overview WILLOWFOLD.

4.2.1 Model and Functionality

In the single-server setting for secure aggregation, a server S aims to compute the sum of n client-held vectors $\mathbf{x}_i \in \mathbb{F}^\ell$. Communication follows a *star topology* with S at the center; i.e., each client communicates with S over a private, point-to-point channel. As in Willow, our model supports *one-shot* clients with *dynamic participation*, meaning each client sends a single message to S; the client set does not need to be known and fixed in advance, and clients may join the aggregation session as it progresses. To prevent *replay* attacks, S assigns each client a random identifier id, and the client attaches to its contribution a proof of possession of id. This identifier is session-specific and can be issued during the client–server handshake. Furthermore, S interacts with a decryptor role D (which can be instantiated as a decryptor committee) whose public key is known to S and the clients. Below, we provide an overview of the different roles.

- Clients $\{\mathsf{C}_i\}_{i\in[n]}$: Clients provide aggregation inputs, and the goal is to ensure their inputs' privacy. The adversary may actively corrupt up to a threshold number of clients. We do not assume a PKI between clients, but they need to know the committee's public key.

- Server S : The server is the entity that receives the final sum but learns no additional information about the clients' inputs. It performs the majority of the work and acts as the coordinator Coord for the committee members. We consider both semi-honest and malicious servers.

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

Parameters: Input domain $\mathbb{F}^{\ell}$; Number of clients n; Minimum threshold of number of clients $\mathsf{min_t}$.

Parties: Server S ; Clients $\{\mathsf{C}_i\}_{i\in[n]}$ each C_i holding input $\mathbf{x}_i \in \mathbb{F}^{\ell}$; $\mathcal{A}$ corrupts $\mathcal{C} \in \{\mathsf{C}_i\}_{i\in[n]} \cup \{\mathsf{S}\}$.

Functionality:

- $\forall i \in [n]$, $\mathcal{F}_{\mathsf{Agg}}$ receives $\mathbf{x}_i$ from C_i; if C_i is corrupted, i.e., $\mathsf{C}_i \in \mathcal{C}$, then $\mathbf{x}_i$ is chosen by $\mathcal{A}$.
- If S is corrupted, $\mathcal{A}$ sends the set of participants $S \subset [n]$ to $\mathcal{F}_{\mathsf{Agg}}$. Otherwise, we set $S := [n]$.
- $\mathcal{A}$ sends $\mathsf{msg} \in \{\mathsf{continue}, \mathsf{abort}\}$ and S to $\mathcal{F}_{\mathsf{Agg}}$.
- If $|S| \geq \mathsf{min_t}$ and $\mathsf{msg} = \mathsf{continue}$, $\mathcal{F}_{\mathsf{Agg}}$ sends $\sum_{i\in S} \mathbf{x}_i$ to S. Otherwise, $\mathcal{F}_{\mathsf{Agg}}$ returns $\perp$ to S.

Figure 4.1: Secure aggregation ideal functionality.

- Committee $\{\mathsf{D}_j\}_{j\in[k]}$: Committee members verify the correctness of aggregation and assist the server in decryption. We assume that a majority of committee members are honest and do not collude with the server. Moreover, we require their communication and computation costs to be sublinear in both the number of clients n and the input size ℓ. Committee members are stateful because they participate in distributed key generation and must store their shares. We assume PKI among the members and that clients can obtain their aggregated public key.

Secure Aggregation Functionality. The functionality $\mathcal{F}_{\mathsf{Agg}}$ in Figure 4.1 formalizes our security guarantees. It ensures output delivery whenever the number of contributions exceeds $\mathsf{min_t}$ (i.e., can also be set to be 0), and the adversary $\mathcal{A}$ does not abort. By design, $\mathcal{F}_{\mathsf{Agg}}$ permits non-selective aborts: $\mathcal{A}$ must decide to abort before the server learns the output. As in Willow, our protocol can be extended to provide output delivery under selective aborts via an extra round of distributed key generation [17, Section 6.7].

Correctness and Output Delivery. Our scheme supports dynamic participation, i.e., clients may drop out without affecting correctness or liveness. Our model is robust to committee-member

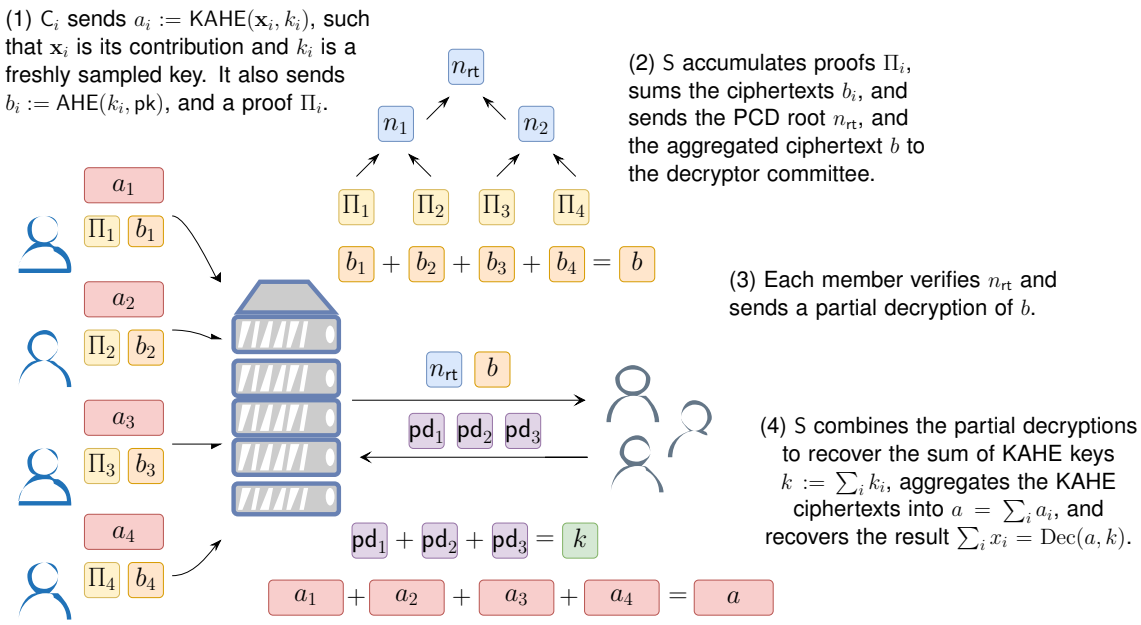


Figure 4.2: An overview of WILLOWFOLD. Each client sends an encryption of its contribution and attaches a proof. The server accumulates client proofs and sends the final proof to the committee. The committee checks the proof and sends the partial decryption of *the sum of the symmetric keys* $k = \sum_i k_i$, enabling the server to recover k and, in turn, the sum of client contributions $\sum_i \mathbf{x}_i$.

dropouts when a majority remain available, as in Willow. Output delivery is guaranteed when the server is semi-honest, and committee members are non-adaptive.

Input Validation. We believe that existing techniques from prior work [15, 54, 110] can be integrated into WILLOWFOLD to ensure well-formed inputs (e.g., low-norm inputs). Input validation is not our focus, and we do not explore it here.

4.2.2 Overview of WILLOWFOLD

We first review the Willow protocol and its verification bottleneck. We then present our framework, which overcomes this barrier. Figure 4.2 illustrates our design.

Overview of Willow. Willow operates in two phases: offline and online. In the offline phase, the committee and the server (as coordinator) run a distributed key generation that outputs a public

key pk for an additively homomorphic encryption scheme AHE and secret shares the decryption key sk among the committee. In the online phase, each C_i submits a homomorphic encryption of its input $\mathbf{x}_i$, and the committee assists the server in decrypting $\sum_i \mathbf{x}_i$. In particular, C_i encrypts $\mathbf{x}_i$ under a symmetric, key and message-homomorphic encryption scheme KAHE and sends

$$a_i = \mathsf{KAHE.Enc}(\mathbf{x}_i, k_i) \quad \text{and} \quad b_i = \mathsf{AHE.Enc}(k_i, \mathsf{pk}).$$

The committee decrypts $\sum_i b_i$ and sends $k = \sum_i k_i$ to S, which then recovers $\sum_i \mathbf{x}_i$.

Correlated Ciphertext Attacks. The scheme described above is insecure when the adversary actively controls clients and observes the server's view. Suppose the adversary $\mathcal{A}$ corrupts a single client C_{n-1}. After clients $\mathsf{C}_1, \ldots, \mathsf{C}_{n-2}$ submit their contributions, $\mathcal{A}$ sends $b_{n-1} = -\sum_{i \in [n-2]} b_i$ and $a_{n-1} = -\sum_{i \in [n-2]} a_i$. When C_n submits its contribution, the committee decrypts

$$\sum_i b_i = \mathsf{AHE.Enc}(\sum_i k_i, \mathsf{pk}) = \mathsf{AHE.Enc}(k_n, \mathsf{pk}),$$

which allows S to recovers $\sum_i \mathbf{x}_i = \mathbf{x}_n$, the input of C_n. To resolve this, Willow requires each C_i to attach π_i, a *zero-knowledge proof of knowledge* of k_i, which ensures that it is sampled independently. This observation, however, must be formalized, which is omitted from Willow's security proof. To fill this gap, we define security against ciphertext correlation attacks and show that if the proofs π_i satisfy simulation extractability (SE)—a strong non-malleability notion—the scheme is secure against such attacks. The intuition is that an adversary can produce a proof only for a ciphertext whose encryption key and randomness it knows, and this must hold even *after* the adversary observes honest proofs for other ciphertexts. Further details appear in Section 4.5.1.

Lightweight Secure Aggregation via PCD. In Willow, there is a verifier role that verifies each client's proof π_i individually and performs the ciphertext summation. This requires a linear cost in the number of clients. In WILLOWFOLD, the server instead directly computes the combined ciphertext and generates a succinct proof that the decryptor committee can efficiently verify. To realize this, we use PCD. Specifically, clients send the server their encryption, along with proofs of contribution as in Willow. The server collects all per-client proofs and, via PCD, accumulates them into a single proof. The same proof shows that the ciphertexts are correctly summed. The server sends the aggregated ciphertext and the final proof to the committee—whereas Willow forwards all ciphertexts and clients' proofs. The final proof certifies all clients' proofs.

This approach also enables streaming aggregation. It reduces the server's storage. Once a contribution arrives, the server immediately accumulates it into the running proof and ciphertext and can discard the raw input without waiting for other contributions. Even with PCD, streaming requires careful treatment of client IDs to prevent replay attacks. Our solution requires the server to track only which clients have participated by storing a small per-client ID. This contrasts with Willow, which requires storing all client proofs and ciphertexts until the end of the protocol.

To further enable parallelism and improve performance, we use leaf-decoupled PCD [122]. Leaf-decoupling reduces the circuit size at the leaf and internal nodes by eliminating unnecessary computations. The root node then recursively proves the correctness of the entire tree.

ID Registration and Replay Attacks. A challenge is that an adversarial server may duplicate client contributions and run *replay attacks*. To avoid this, Willow assigns each client a unique random ID, which the client includes in its contribution. Then, the verifier committee checks that no duplicate IDs appear in the contributions. However, in WILLOWFOLD the issue is more

involved; we cannot send IDs to the committee, since this incurs a linear cost for committee members. Additionally, enforcing the uniqueness of client IDs directly within the PCD circuit would require a circuit whose size scales linearly with the number of clients.

Instead, we apply the following technique. Let $\mathsf{commit}(\cdot)$ be a homomorphic commitment such that $\mathsf{commit}(\boldsymbol{a} + \boldsymbol{b}) = \mathsf{commit}(\boldsymbol{a}) \cdot \mathsf{commit}(\boldsymbol{b})$. The server assigns each C_i a unique ID $g_i = \mathsf{commit}(\mathbf{e}_i)$, where $\mathbf{e}_i$ encodes a one-hot vector of length n (total number of clients), i.e., its ith entry is 1, and others are 0. The server maintains a participation vector pv initialized as $\mathsf{pv} := 0$ with commitment $\mathsf{commit}(\mathsf{pv}) = 1$. When C_i submits its contribution, the server updates $\mathsf{pv} \leftarrow \mathsf{pv} + \mathbf{e}_i$ and $\mathsf{commit}(\mathsf{pv}) \leftarrow \mathsf{commit}(\mathsf{pv}) \cdot g_i$. This vector is stored efficiently by simply storing the set of client IDs. The PCD circuit verifies that these updates are correct, where adding two commitments is a constant-size operation. Finally, the server proves to the committee that $\mathsf{commit}(\mathsf{pv})$ commits to a $\{0, 1\}$-valued vector, ensuring there is no duplication. Such a proof can be generated via a *zerocheck*, with proof size and verification time $O(\log n)$.

What remains is to certify that IDs are well-formed and that each ID indeed commits to a one-hot vector. To ensure this, we run a one-time setup to generate a public Merkle tree $\mathcal{T}$ whose leaves contain all valid IDs; both the IDs and the tree are publicly verifiable. The Merkle root serves as a commitment to the set of IDs. When the server assigns a client an ID g_i, it also provides the corresponding Merkle path. The client then includes g_i and its Merkle path in its contribution, and the PCD circuit checks its consistency with the Merkle root. Note, the same tree $\mathcal{T}$ can be reused across many aggregation sessions, and IDs are unique within a given session.

Extraction from PCD Proofs. For simulation security, we require the simulator to extract the IDs of honest clients who participated in the aggregation—this enables the simulator to learn the

sum of the honest clients' inputs and to reprogram the random oracle accordingly. In Willow, such extraction is straightforward: the server forwards all per-client proofs, and the simulator runs the extractor of the proof system to obtain the IDs. In WillowFold, the server forwards only the PCD root. However, the definition of leaf-SE for PCDs guarantees an efficient extractor that, given the root, can recover the entire PCD transcript, including the leaf nodes and their witnesses. Hence, a simulation-based security proof analogous to Willow also applies here.

Distributed and Parallel Aggregation. An advantage of WILLOWFOLD is that the server's work is amenable to a high degree of parallelism and distributed execution across many machines. The server workflow has three phases: (i) a one-time setup to $\mathcal{T}$; (ii) PCD computation; and (iii) a zerocheck proving ID uniqueness. Computing a Merkle tree and performing PCD aggregation can be naturally distributed. However, the *zerocheck* in principle is difficult to distribute. Nevertheless, in practice, it can be executed effectively by a single server. The support for parallel and distributed computation comes from our use of PCD. If one instead applies a monolithic SNARK to Willow's verification statement, it is unclear how to parallelize or distribute the computation, since the non-duplication check does not naturally decompose into independent chunks.

4.2.3 Required Properties of the PCD Scheme

In this section, we outline the properties required of the underlying PCD scheme and provide an overview of our notion of SE for leaf-decoupled PCDs.

PCD Requirements. For the security of WILLOWFOLD, we require the PCD scheme to satisfy two properties: (i) support accumulation over zero-knowledge proofs—that is, in WILLOWFOLD, the leaves of the PCD tree are per-client proofs, which must hide the keys used to encrypt client inputs; and (ii) be leaf-SE to ensure security against correlated ciphertext attacks. For efficiency,

we also require the PCD to be *sublinear* [93], with proof size and verification time sublinear in the size of the statement. This ensures that clients' and committee members' communication, as well as the server's computational overhead for verifying client proofs, remains concretely small.

PCD Simulation Extractability. SE for a zero-knowledge proof system is typically defined by giving the adversary access to an oracle that produces simulated proofs for arbitrary queried statements; SE then guarantees that the adversary cannot forge a satisfiable proof—that was not queried to the simulator—for any statement without *knowing* the corresponding witness; in particular, there is an efficient extractor algorithm that can extract the witness from the adversary.

Defining SE for PCDs, however, requires additional care. An adversary may query the simulator for a proof π and then run the proving algorithm to extend it into a fresh PCD proof π^* that has not been queried, yet π^* may not be extractable since it builds on the simulated proof π. Such attacks do not arise in non-recursive proofs. A concurrent work [4] introduces a notion of SE for PCDs, in which the extractor is required to recover the witnesses for non-simulated nodes.

In our leaf-decoupled PCD architecture, leaf nodes must be zero-knowledge and simulation extractable. In contrast, internal nodes are deterministic and need not be simulation extractable, since there may be no simulator for them. Motivated by this observation, we relax the notion of SE for PCDs from [4] by requiring the simulator to answer only leaf queries. We then show that this notion is achieved when the leaf nodes are SE and the internal nodes are knowledge sound.

4.2.4 Concrete Instantiation and Optimizations

We instantiate WILLOWFOLD with Spartan+KZH-Fold [93], a sublinear PCD scheme.

KZH-Fold. Recent work [93] built KZH, a polynomial commitment scheme with succinct openings, and constructed KZH-Fold, a *sublinear accumulation scheme* for the KZH verifier (see

Section 4.3.3). Using a standard compilation [50], they instantiated Spartan [132] with KZH to obtain a SNARK, which we refer to as Spartan+KZH. They applied KZH-Fold to obtain a PCD, called Spartan+KZH-Fold. Due to the sublinearity of KZH and KZH-Fold, this PCD scheme has sublinear proof size and verification time. Despite this sublinearity, Spartan+KZH-Fold further achieves prover time comparable to state-of-the-art PCD schemes such as Nova [96]. These features motivate our choice of Spartan+KZH-Fold as the PCD backend for WILLOWFOLD.

However, Spartan+KZH-Fold is neither zero-knowledge nor simulation extractable. To address this gap, we first present a zero-knowledge variant of Spartan+KZH and then show that it satisfies SE, which in turn implies that Spartan+KZH-Fold is a leaf-SE PCD scheme.

To realize zero knowledge for Spartan+KZH, we note that its two primary components are: Spartan and KZH commitments. Prior work [88] shows how to achieve zero-knowledge for KZH commitments. Fortunately, this transformation introduces only *minimal* changes to the KZH verification algorithm, and the original accumulation scheme (i.e., KZH-Fold) can still be used—with minimal modifications—to accumulate zk-KZH. The Spartan polynomial IOP can also be made zero-knowledge via known zero-knowledge sumcheck schemes [146] with only logarithmic randomness. Putting together these yields a zero-knowledge variant of Spartan+KZH, which we call zk-Spartan+zk-KZH. However, there is a mismatch: techniques for zero-knowledge sumcheck require commitments to multivariate polynomials with individual degrees > 1. KZH supports only multilinear polynomials. To address this issue, we propose a method for committing to higher-degree sparse polynomials using multilinear polynomial commitments, while keeping the zero-knowledge overhead logarithmic. Further details appear in Section 4.6.1.

We then show that zk-Spartan+zk-KZH satisfies SE. Prior work [74] identifies sufficient conditions: (i) *knowledge soundness*, (ii) *unique response*, and (iii) *trapdoor-less zero-knowledge*.

We prove that zk-Spartan+zk-KZH satisfies all these properties. Since KZH-Fold is knowledge sound [93] and zk-Spartan+zk-KZH satisfies SE, the PCD built from these satisfies leaf-SE.

We note that WILLOWFOLD can be generically instantiated with any leaf-SE PCD scheme. As discussed above, sublinearity of PCD is crucial for both communication and computation costs. Thus, this observation suggests future work on identifying alternative PCDs beyond Spartan+KZH-Fold, potentially improving efficiency.

Arithmetization of Ring Operations. Similar to Willow, WILLOWFOLD uses an AHE scheme based on RLWE. Spartan+KZH-Fold (which is based on R1CS) does not natively support ring operations, which would incur high arithmetization overhead. More concretely, the client's circuit includes a product of two ring elements, e.g., $\mathbf{w} = \mathbf{u} \cdot \mathbf{v}$. Each ring element is a polynomial of degree d; thus, this product relation adds $O(d^2)$ constraints. To avoid this, we verify each product via a probabilistic check: evaluate the underlying polynomials at a random challenge r and show that $w(r) = u(r) \cdot v(r)$, which requires only $O(d)$ constraints. A subtle issue is efficiently deriving the challenge r; to do so, we use the *commit-and-prove* paradigm [45].

4.3 Additional Preliminaries

In this section, we provide additional preliminaries used throughout this chapter.

4.3.1 Encryption Schemes

Similar to Willow, our construction employs two different homomorphic encryption schemes: (i) KAHE, a symmetric-key, key-and-message additive homomorphic encryption scheme, and (ii) a public-key, threshold homomorphic encryption scheme AHE. Below, we discuss each.

KAHE. For our KAHE scheme, we require that, for any pair of messages m_1, m_2 and keys k_1, k_2,

given ciphertexts ct_1 and ct_2 that encrypt m_1 and m_2 under keys k_1 and k_2, respectively, $\mathsf{ct}_1 + \mathsf{ct}_2$ is an encryption of $m_1 + m_2$ under the key $k_1 + k_2$. We further require security to hold even if the adversary learns the sum of the keys. Such a KAHE scheme can be instantiated under the RLWE assumption [17]. The syntax of KAHE is as follows:

- $\mathsf{Setup}(1^\lambda) \to \mathsf{pp}$: on input λ, outputs the public parameters pp.
- $\mathsf{KeyGen}(\mathsf{pp}) \to k$: on input pp, outputs a random secret key k.
- $\mathsf{Enc}(\mathsf{pp}, m, k) \to \mathsf{ct}$: on input pp, a message m, and a key k, outputs a ciphertext ct.
- $\mathsf{Dec}(\mathsf{pp}, \mathsf{ct}, k)$: on input pp, a ciphertext ct, and the secret key k, outputs the plaintext m.

AHE. The AHE scheme is required to be additive threshold homomorphic: given ciphertexts ct_1 and ct_2 that encrypt messages m_1 and m_2 under the same public key pk, $\mathsf{ct}_1 + \mathsf{ct}_2$ is a valid ciphertext encrypting $m_1 + m_2$ under pk. To prevent *rogue-key attacks*, each party, when publishing its partial public key, must also provide a proof of knowledge of that partial secret key. To further protect against correlated ciphertext attacks, every encrypting party additionally supplies a *proof of plaintext knowledge* corresponding to the ciphertext it produces. Our concrete instantiation of AHE (which follows Willow) is realized under RLWE. The syntax is as follows:

- $\mathsf{Setup}(1^\lambda) \to \mathsf{pp}$: on input λ, outputs public parameters pp.
- $\mathsf{VerifiableKeyGen}(\mathsf{pp}) \to (\mathsf{pk}_i, \mathsf{sk}_i, \pi_i)$: on input pp, each party outputs a partial public key pk_i, a partial secret key sk_i and a proof of knowledge of the secret key π_i.
- $\mathsf{VerifyKeyGen}(\mathsf{pp}, \mathsf{pk}_i, \pi_i) \to \{0, 1\}$: on input pp, pk_i and π_i, accepts or rejects.
- $\mathsf{AggregateKey}(\mathsf{pp}, \{\mathsf{pk}_i\}_i) \to \mathsf{pk}$: on input pp and $\{\mathsf{pk}_i\}_i$, outputs an aggregated public key pk.

- $\mathsf{VerifiableEnc}(\mathsf{pp}, \mathsf{pk}, m) \rightarrow (\mathsf{ct}, \pi)$: on input pp, pk, a message m, outputs a randomized ciphertext ct and a proof of knowledge of the plaintext π.

- $\mathsf{VerifyEnc}(\mathsf{pp}, \mathsf{pk}, \mathsf{ct}, \pi) \rightarrow \{0, 1\}$: on input pp, pk, ct, and π, accepts or rejects.

- $\mathsf{ParDec}(\mathsf{pp}, \mathsf{ct}, \mathsf{sk}_i) \rightarrow \mathsf{pd}_i$: on input pp, ct, and sk_i, outputs a partial decryption pd_i.

- $\mathsf{Recover}(\mathsf{pp}, \mathsf{ct}, \{\mathsf{pd}_i\}_i) \rightarrow m$: on input pp, ct, and $\{\mathsf{pd}_i\}_i$, outputs a the plaintext m.

Our concrete instantiations of these two encryption schemes follow Willow. Details can be found in [17, Appendices B and C]. One minor detail concerns the choice of the underlying proofs of knowledge. For the AHE encryption scheme, part of the encryption procedure is executed inside the PCD circuit, which serves as a proof of knowledge of the underlying message. For AHE key generation—run by the committee members—the associated proof of knowledge can be chosen arbitrarily, e.g., the Bulletproofs-based instantiation proposed in Willow.

4.3.2 Simulation Extractability

We first recall the definition of simulation extractability for non-interactive arguments.

Definition 6 (Simulation Extractability)**.** *Let* $\Pi := \langle \mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify} \rangle$ *be a non-interactive zero-knowledge argument, and* Sim *be a zero-knowledge simulator for* Π*. Then, we say that* Π *is simulation extractable with respect to the simulator* Sim *if there exists an efficient extractor* $\mathcal{E}$ *such that for every efficient adversary* $\mathcal{A}$*,*

$$\mathsf{Sim\text{-}Ext}_{0,\Pi}^{\mathsf{Sim},\mathcal{A}}(1^\lambda) \approx_c \mathsf{Sim\text{-}Ext}_{1,\Pi,R}^{\mathcal{E},\mathsf{Sim},\mathcal{A}}(1^\lambda).$$

Below, we recall the definitions of special soundness, trapdoor-less zero-knowledge, and

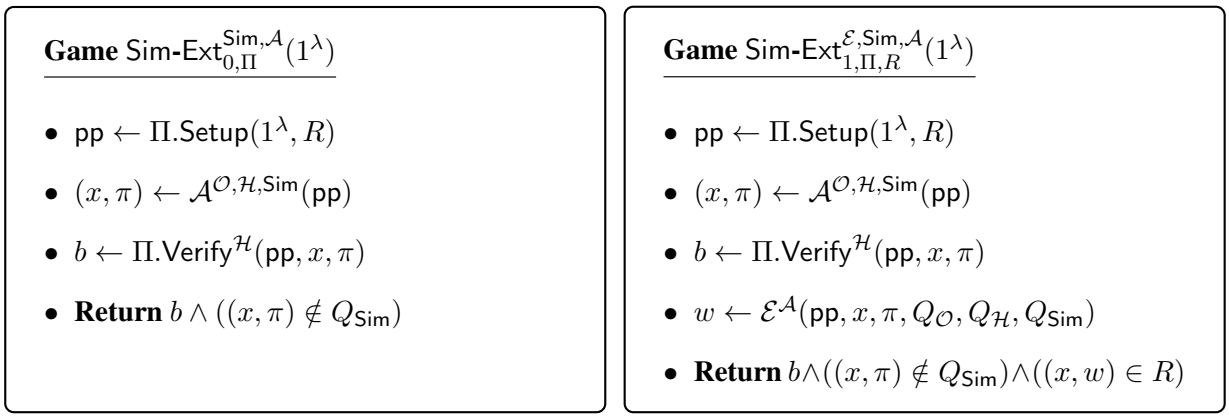


Figure 4.3: Simulation extractability security games. $\mathcal{H}$ denotes the random oracle, $\mathcal{O}$ denotes the srs update oracle, and Sim is a simulator oracle for Π. Sim takes as input x and outputs a proof π and may reprogram $\mathcal{H}$. $Q_{\mathcal{O}}$, $Q_{\mathcal{H}}$, and Q_{Sim} record queries to $\mathcal{O}$, $\mathcal{H}$, and Sim oracles, respectively. $\mathcal{E}$ is given black-box access to $\mathcal{A}$, can rewind $\mathcal{A}$, and implements $\mathcal{H}$ and Sim for $\mathcal{A}$.

unique response—three properties that we require for an argument to prove its SE. To define special soundness, we first need to introduce the notion of a tree of accepting transcripts.

Definition 7 (Tree of Accepting Transcripts)**.** *An $(n_1, \ldots, n_r)$-tree of accepting transcripts is a tree in which each node at depth $i \in [r+1]$ is the ith prover message of an accepting transcript. Edges are labeled by challenges such that no two edges at the same depth share the same label. Each node at depth i has $n_i - 1$ siblings and n_{i+1} children. The number of branches is $N = \prod_i n_i$.*

Definition 8 (Special Soundness)**.** *Let $\Pi := \langle \mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify} \rangle$ denote a $(2r+1)$-message and FS-transformed non-interactive zero-knowledge argument. For any $\boldsymbol{n} = (n_1, \ldots, n_r) \in \mathbb{N}^r$, We say that Π satisfies $\boldsymbol{n}$-special soundness if there exists an efficient tree-extraction algorithm $\mathcal{TE}$, such that for every efficient adversary $\mathcal{A}$,*

$$\mathsf{Special}_{0,\Pi}^{\mathcal{A}}(1^\lambda) \approx_c \mathsf{Special}_{1,\Pi,\boldsymbol{n}}^{\mathcal{TE},\mathcal{A}}(1^\lambda).$$

Game $\mathsf{Special}^{\mathcal{A}}_{0,\Pi}(1^\lambda)$

- $\mathsf{pp} \leftarrow \Pi.\mathsf{Setup}(1^\lambda, R)$
- $(x, \pi) \leftarrow \mathcal{A}^{\mathcal{O},\mathcal{H}}(\mathsf{pp})$
- $b \leftarrow \Pi.\mathsf{Verify}^{\mathcal{H}}(\mathsf{pp}, x, \pi)$
- **Return** b

Game $\mathsf{Special}^{\mathcal{TE},\mathcal{A}}_{1,\Pi,\boldsymbol{n}}(1^\lambda)$

- $\mathsf{pp} \leftarrow \Pi.\mathsf{Setup}(1^\lambda, R)$
- $(x, \pi) \leftarrow \mathcal{A}^{\mathcal{O},\mathcal{H}}(\mathsf{pp})$
- $b_0 \leftarrow \Pi.\mathsf{Verify}^{\mathcal{H}}(\mathsf{pp}, x, \pi)$
- $\mathcal{T}, w \leftarrow \mathcal{TE}^{\mathcal{A}}(\mathsf{pp}, x, \pi, Q_{\mathcal{O}}, Q_{\mathcal{H}})$
- $b_1 \leftarrow \mathsf{IsAccepting}^{\mathcal{H}}(\mathsf{pp}, x, \boldsymbol{n}, \mathcal{T})$
- **Return** $b_0 \wedge b_1 \wedge ((x, w) \in R)$

Figure 4.4: Special soundness security games. IsAccepting is an efficient algorithm that checks whether $\mathcal{T}$ is an $\boldsymbol{n}$-tree of accepting transcripts with respect to the argument Π and instance x.

Definition 9 (k-Trapdoor-Less Zero Knowledge)**.** *Assume that* $\Pi := \langle \mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify} \rangle$ *is a* $(2r+1)$*-message and FS-transformed non-interactive zero-knowledge argument. We say that* Π *satisfies* k*-trapdoor-less zero-knowledge for a* $k \in [1, r]$*, if there exists a efficient simulator* $\mathsf{Sim}^{k\text{-}\mathsf{TLZK}}$ *such that for every efficient adversary* $\mathcal{A}$*,*

$$k\text{-}\mathsf{TLZK}^{\mathcal{P},\mathcal{A}}_{0,\Pi}(1^\lambda) \approx k\text{-}\mathsf{TLZK}^{\mathsf{Sim},\mathcal{A}}_{1,\Pi}(1^\lambda).$$

Definition 10 (k-Unique Response)**.** *Let* $\Pi := \langle \mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify} \rangle$ *be a* $(2r+1)$*-message and FS-transformed non-interactive zero-knowledge argument. Then, we say that* Π *satisfies* k*-unique response for a* $k \in [1, r]$*, if for every efficient adversary* $\mathcal{A}$*,*

$$\Pr[k\text{-}\mathsf{UR}^{\mathcal{A}}_{\Pi}(1^\lambda)] \leq \mathsf{negl}(\lambda).$$

The following lemma shows that simulation extractability of a FS-transformed non-interactive

Game k-$\mathsf{TLZK}^{\mathcal{P},\mathcal{A}}_{0,\Pi}(1^\lambda)$

- $\mathsf{pp} \leftarrow \Pi.\mathsf{Setup}(1^\lambda, R)$
- $b \leftarrow \mathcal{A}^{\mathcal{O},\mathcal{H},\mathcal{P}}(\mathsf{pp})$
- **Return** b

$\mathcal{P}(\mathsf{pp}, x, w)$

- **If** $(x, w) \notin R$ **abort**
- **Else return** $\Pi.\mathsf{Prove}(\mathsf{pp}, x, w)$

Game k-$\mathsf{TLZK}^{\mathsf{Sim},\mathcal{A}}_{1,\Pi}(1^\lambda)$

- $\mathsf{pp} \leftarrow \Pi.\mathsf{Setup}(1^\lambda, R)$
- $b \leftarrow \mathcal{A}^{\mathcal{O},\mathcal{H},\mathsf{Sim}}(\mathsf{pp})$
- **Return** b

$\mathsf{Sim}(\mathsf{pp}, x, w)$

- **If** $(x, w) \notin R$ **abort**
- **Else return** $\mathsf{Sim}^{k\text{-}\mathsf{TLZK}}(\mathsf{pp}, x)$

Figure 4.5: k-trapdoor-less zero-knowledge security games. $\mathsf{Sim}^{k\text{-}\mathsf{TLZK}}$ is a simulator for Π such that for each query (x, π), it reprograms $\mathcal{H}$ only once in round k.

Game k-$\mathsf{UR}^{\mathcal{A}}_{\Pi}(1^\lambda)$

- $\mathsf{pp} \leftarrow \Pi.\mathsf{Setup}(1^\lambda, R)$
- $(x, \pi_0, \pi_1, c) \leftarrow \mathcal{A}^{\mathcal{O},\mathcal{H}}(\mathsf{pp})$
- $b_0 \leftarrow \Pi.\mathsf{Verify}^{\mathcal{H}[(\mathsf{pp},x,\pi_0|_k)\mapsto c]}(\mathsf{pp}, x, \pi_0)$
- $b_1 \leftarrow \Pi.\mathsf{Verify}^{\mathcal{H}[(\mathsf{pp},x,\pi_1|_k)\mapsto c]}(\mathsf{pp}, x, \pi_1)$
- **Return** $b_0 \wedge b_1 \wedge (\pi_0 \neq \pi_1) \wedge (\pi_0|_k = \pi_1|_k)$

Figure 4.6: k-unique response security game. $\mathcal{H}[(\mathsf{pp}, x, \pi_b|_k) \mapsto c]$ indicates that $\mathcal{H}$ has been reprogrammed to c on the input $(\mathsf{pp}, x, \pi_b|_k)$, where $\pi_b|_k$ denotes proof messages for rounds 1 to k.

argument follows from special soundness, trapdoor-less zero-knowledge, and unique response.

Lemma 4.3.1. *[74] Let* $\Pi := \langle \mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify} \rangle$ *be a* $(2r+1)$*-message and FS-transformed non-interactive zero-knowledge argument.* Π *is simulation extractable if it has (updatable-SRS) special soundness,* k*-unique response, and* k*-trapdoor-less zero knowledge for the same* k*.*

4.3.3 Details of KZH and KZH-Fold

We provide the details of KZH and KZH-Fold.

Notation. We use the same notation as in [93]. We denote the space of n dimensional tensors with dimensions $(d_1, d_2, \ldots, d_n)$ on field $\mathbb{F}$ with $\mathbb{F}^{d_1 \times d_2 \times \cdots \times d_n}$. For $\mathbf{T} \in \mathbf{F}^{d_1 \times d_2 \times \cdots \times d_k}$ and $\boldsymbol{x}_1 \in \mathbf{F}^{d_1}$, we denote tensor inner product as

$$\langle \mathbf{T},\, \boldsymbol{x}_1 \rangle = \langle \mathbf{T}_1, \boldsymbol{x}_1 \rangle \otimes \mathbf{T}_2 \otimes \ldots \otimes \mathbf{T}_k.$$

Note that the result is a tensor in $\mathbf{F}^{d_2 \times \cdots \times d_k}$. Similarly for $\boldsymbol{x}_1 \in \mathbf{F}^{d_1}$ and $\boldsymbol{x}_2 \in \mathbf{F}^{d_2}$, it holds that $\langle \mathbf{T}, \boldsymbol{x}_1 \otimes \boldsymbol{x}_2 \rangle = \langle \langle \mathbf{T}, \boldsymbol{x}_1 \rangle, \boldsymbol{x}_2 \rangle \in \mathbf{F}^{d_3 \times \cdots \times d_k}$.

KZH Protocol. KZH-k is a commitment scheme for such tensors. In particular, $\mathbf{T}$ represents the polynomial that can be opened at evaluation point $(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k)$ for $\boldsymbol{x}_j \in \mathbf{F}^{d_j}$ as:

$$\langle \langle \langle \langle \mathbf{T}, \boldsymbol{x}_1 \rangle, \boldsymbol{x}_2 \rangle, \ldots, \boldsymbol{x}_{k-1} \rangle, \boldsymbol{x}_k \rangle = y \in \mathbf{F}.$$

The scheme admits an efficient accumulation scheme and supports both multilinear and univariate polynomial commitments. The core insight is that we commit to $\mathsf{com} = \langle \mathbf{T},\, \boldsymbol{\mu}_1 \otimes \boldsymbol{\mu}_2 \cdots \otimes \boldsymbol{\mu}_k \rangle \times \mathsf{G}$, for secret elements $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_k$ and group generator G, and open $[\mathsf{com}_i]_{i=1}^k$, commitments to all $k-1$ dimensional slices of $\mathbf{T}$. The verifier can check that these slices are correct using a pairing with $\boldsymbol{\mu}_1$ and compute $\mathsf{com}' = \langle (\mathsf{com}_1, \ldots, \mathsf{com}_k), \boldsymbol{x}_1, \rangle$. com' is now a commitment to the $k-1$ dimensional tensor $\mathbf{T}_1 = \langle \mathbf{T}, \boldsymbol{x}_1 \rangle$ and we can recursively apply the scheme. We present the full protocol for degree $[d]^k := (d,\, d,\, \ldots,\, d)$ tensor in Figure 4.7.

KZH-Fold protocol description. KZH-Fold is an accumulation scheme for the KZH-k commitment scheme, with accumulator size and the decider time being $O(k \cdot n^{\frac{1}{k}})$. The accumulation verifier performs $O(k)$ $\mathbb{G}_1$ operations. For details, we refer the reader to [93, Appendix C].

KZH-k.Setup:

1. Sample $\mathsf{G} \leftarrow \mathbb{G}_1, \mathsf{V} \leftarrow \mathbb{G}_2$ and $[[\mu_{i,j}]_{i=1}^{d}]_{j=1}^{k} \leftarrow \mathbb{F}$
2. $\mathbf{H}_1 = \{\mathsf{H}_{i_1,\ldots,i_k} \leftarrow (\prod_{j=1}^{k} \mu_{i_s,j}) \times \mathsf{G} : \forall i_1, \ldots, i_k \in [d]\}$
3. $\mathbf{H}_2 = \{\mathsf{H}_{i_2,\ldots,i_k} \leftarrow (\prod_{j=2}^{k} \mu_{i_j,j}) \times \mathsf{G} : \forall i_2, \ldots, i_k \in [d]\}$

   ...
4. $\mathbf{H}_k = \{\mathsf{H}_{i_k} \leftarrow \mu_{i_k,k} \times \mathsf{G} : \forall i_k \in [d]\}$
5. $\mathbf{V} = \{\mathsf{V}_{i,j} \leftarrow \mu_{i,j} \times \mathsf{V} : \forall i \in [d],\ j \in [k]\}$
6. Output $(\mathsf{G}, \mathsf{V}, \mathbf{H}_1, \mathbf{H}_2, \ldots, \mathbf{H}_k, \mathbf{V})$

KZH-k.Commit: For $\mathbf{T} \in \mathbb{F}^{d \times d \times \cdots \times d}$, compute the commitment as it follows:

1. Output $\mathsf{com} \leftarrow \langle \mathbf{T}, \mathbf{H}_1 \rangle = \sum_{(i_1,i_2,\ldots,i_k) \in [d]^k} \mathbf{T}_{i_1,i_2,\ldots,i_k} \times \mathsf{H}_{i_1,i_2,\ldots,i_k}$

KZH-k.Open: Given commitment $\mathsf{com} \in \mathbb{G}_1$, tensor $\mathbf{T} \in \mathbb{F}^{d \times d \times \cdots \times d}$ and inputs $\boldsymbol{x}_j \in \mathbb{F}^d$:

1. Let $\mathbf{T}_1 = \mathbf{T}$
2. For $j = 1, \ldots, k-1$:
   (a) Set $\mathbf{T}_{j+1} \leftarrow \langle \mathbf{T}_j, \boldsymbol{x_j} \rangle$
   (b) Compute vector $\mathbf{D}_j$ e.g. $\mathsf{D}_j[i] \leftarrow \langle \langle i \rangle \otimes \mathbf{T}_j, \mathbf{H}_{j+1} \rangle$ for all $i \in [d]$.
3. Output $\pi \leftarrow \{[\mathbf{D}_j]_{j \in [k-1]}, \mathbf{T}_k\}$

KZH-k.Verify: Given commitment $\mathsf{com} \in \mathbb{G}_1$, inputs $\boldsymbol{x}_j \in \mathbb{F}^d$, output $y \in \mathbb{F}$ and opening proof π:

1. Parse $\{[\mathbf{D}_j]_{j \in [k-1]}, \mathbf{T}_k\} \leftarrow \pi$
2. Set $\mathsf{com}_0 = \mathsf{com}$
3. For $j \in 1, \ldots, k-1$
   (a) Check that $e(\mathsf{com}_{j-1}, \mathsf{V}) = \sum_{i=0}^{d} e(\mathsf{D}_j[i], \mathsf{V}_{i,j})$
   (b) Compute $\mathsf{com}_j \leftarrow \langle \boldsymbol{x}_j, \mathbf{D}_j \rangle$
4. Check that $\mathsf{com}_{k-1} = \langle \mathbf{T}_k, \mathbf{H}_k \rangle$
5. Check that $\langle \mathbf{T}_k, \boldsymbol{x}_k \rangle = y$

Figure 4.7: KZH-k description.

4.4 WILLOWFOLD Secure Aggregation

In this section, we present WILLOWFOLD a framework for secure aggregation.

4.4.1 Setup and Client Registration

In this section, we give details of the setup phase and client registration.

Setup. The setup phase outputs: (i) public parameters of PCS, PCD, KAHE, and AHE and (ii) a Merkle tree $\mathcal{T}$ built over the client IDs. During this phase, the committee, with the assistance of the server, also executes the distributed key generation (DKG) protocol, which distributes shares of the AHE secret key among the committee members. The setup protocol is illustrated in Figure 4.8. If the setup requires no secret randomness, it can be executed by the server (and by the committee members). Otherwise, if it requires a trusted setup—such as our instantiation of PCD with Spartan+KZH-Fold—it must be run by a trusted entity.

Registration. Whenever a client wishes to contribute, it first performs a handshake with the server to obtain an identifier g_i together with a Merkle proof of membership π_i attesting that g_i belongs to $\mathcal{T}$. Importantly, the server does not need to maintain any mapping between clients and their assigned g_i. The particular assignment is immaterial, provided that the client can demonstrate the authenticity of g_i via the Merkle proof. The server only keeps track of which generators have already been issued. The registration protocol is also illustrated in Figure 4.8.

4.4.2 Client, Server, and Decryptor Roles

Below, we outline the protocols executed by each role in WILLOWFOLD.

Client Role. Each client C_i samples a seed s_i and masks its input $\boldsymbol{x}_i$ with $\mathsf{PRG.Expand}(s_i)$, where

Setup:

1. Run $\mathsf{PCS.Setup}(1^\lambda, N) \to \mathsf{pp_{PCS}}$. Then, define the set of client IDs $(g_0, g_1, \ldots, g_{N-1})$ such that $g_i = \mathsf{PCS.Commit}(\mathsf{pp_{PCS}}, \tilde{e}_i)$ where $\tilde{e}_i$ is the multilinear extension of the ith unit vector.
2. Sample H from a hash function family $\mathcal{H}$ and compute a Merkle tree $\mathcal{T}$ over $(g_0, g_1, \ldots, g_{N-1})$ using H. Let rt be the Merkle root, which will be used for membership verification.
3. Run $\mathsf{PCD.Setup}(1^\lambda) \to \mathsf{pp_{PCD}}$ to generate the public parameters of PCD. This setup may be circuit-specific and take the step circuit as input or be universal as in Spartan+KZH-Fold.
4. Run $\mathsf{KAHE.Setup}(1^\lambda) \to \mathsf{pp_{KAHE}}$, $\mathsf{AHE.Setup}(1^\lambda) \to \mathsf{pp_{AHE}}$, $\mathsf{E.Setup}(1^\lambda) \to \mathsf{pp_E}$ to generate the public parameters of encryption schemes.
5. Run DKG as described in Figure 4.11 and forward the resulting keys pk and $\mathsf{pk_{AUX}}$ to the server.
6. Output the public parameters $\mathsf{pp} := (\mathsf{pp_{PCS}}, \mathsf{pp_{PCD}}, \mathsf{rt}, \mathsf{pp_{KAHE}}, \mathsf{pp_{AHE}}, \mathsf{pp_E}, \mathsf{pk}, \mathsf{pk_{AUX}})$.

Register:

1. Retrieve rt corresponding to the Merkle tree $\mathcal{T}$ over the set of IDs $(g_0, \ldots, g_{N-1})$.
2. Select an unassigned ID g_i from the set of IDs and mark it as assigned.
3. Let π_i denote the Merkle authentication path from the leaf g_i to the root rt.
4. Output $(\mathsf{rt}, g_i, \pi_i)$.

Figure 4.8: Setup and client registration subroutines.

PRG is a pseudorandom generator built from a random oracle. This step is necessary since the secure aggregation functionality is impossible to realize in the plain model [17, Theorem 1], and we rely on the random oracle to circumvent this barrier. C_i sends to the server an encryption of the seed s_i under the committee's auxiliary public key $\mathsf{pk_{AUX}}$. It also sends an encryption of its masked input $\boldsymbol{y}_i = \boldsymbol{x}_i + \mathsf{PRG.Expand}(s_i)$ under KAHE, together with an AHE encryption of the KAHE key k_i under the committee's main public key pk. Finally, the client provides a PCD leaf proof that the AHE encryption of k_i is a valid ciphertext, and it owns a correct ID by including the ID and its Merkle tree membership proof in the statement. The protocol is shown in Figure 4.9. A subtle aspect of the client's statement within the PCD circuit is that, due to the lattice-based

Each client C_i holds an input $\boldsymbol{x}_i \in \mathbb{F}^\ell$ and proceeds as follows:

1. **Receive the ID and the public keys:**
   (a) Receive $(\mathcal{D}, \Sigma, \mathsf{pk}, \mathsf{pk}_{\mathsf{aux}})$ from the server.
   (b) Check that $|\Sigma| \geq t$ and signatures in Σ are valid signatures on $(\mathsf{pk}, \mathsf{pk}_{\mathsf{aux}})$; otherwise abort.
   (c) Call functionality $\mathsf{Register}$ and receive the ID tuple $(\mathsf{rt}, g_i, \pi_i)$ from the server.
2. **Mask the input and compute ciphertexts:**
   (a) Run $\mathsf{KAHE.KeyGen}(\mathsf{pp}_{\mathsf{KAHE}}) \to k_i$ and $\mathsf{PRG.KeyGen}(1^\lambda) \to s_i$.
   (b) Compute the masked input $\boldsymbol{y}_i := \mathsf{PRG.Expand}(s_i, \ell) + \boldsymbol{x}_i$.
   (c) Run $\mathsf{KAHE.Enc}(\mathsf{pp}_{\mathsf{KAHE}}, \boldsymbol{y}_i, k_i) \to \mathsf{ct}_i^{\mathsf{masked}}$.
   (d) Run $\mathsf{AHE.Enc}(\mathsf{pp}_{\mathsf{AHE}}, \mathsf{pk}, k_i) \to \mathsf{ct}_i^{\mathsf{key}} := (\mathsf{ct}_0, \mathsf{ct}_1)$.
   (e) Run $\mathsf{E.Enc}(\mathsf{pp}_{\mathsf{E}}, \mathsf{pk}_{\mathbf{AUX}}, s_i) \to \mathsf{ct}_i^{\mathsf{seed}}$.
3. **Generate a proof of correct contribution:**
   (a) Construct a zero-knowledge PCD base proof Π_i^{PCD} for a relation $\mathcal{R}_i^{\text{mr}} \wedge \mathcal{R}_i^{\text{Enc}}$, where $\mathcal{R}_i^{\text{mr}}$ and $\mathcal{R}_i^{\text{Enc}}$ are defined as follows. Let $\mathcal{X}_i^{\text{PCD}} := (\mathsf{rt}, g_i, \mathbf{u}, \mathsf{ct}_1)$ be the instance for this relation.
   - Membership proof: The client includes its ID g_i as input to the circuit and proves that g_i is a valid element in the server's Merkle tree $\mathcal{T}$ with root rt, i.e.,
   $$\mathcal{R}_{\text{mr}} := \{((g_i, \text{root}), \pi_i) : \mathsf{MR.Verify}(\text{root}, g_i, \pi_i) = 1\}.$$
   - Encryption with constraints: The client encrypts k_i using AHE. Within the circuit, the client proves that the ciphertext is valid, i.e. the key and the ciphertext randomness satisfy the following constraints.
   $$\mathcal{R}_{\mathsf{Enc}} := \left\{((\mathbf{u}, \mathsf{ct}_1); (\mathbf{v}, e^1)) \middle| \mathsf{ct}_1 = \mathbf{u} \cdot \mathbf{v} + e^1 \bmod q_2, \|\mathbf{v}\| \leq B_s, \|e^1\| \leq B_e\right\}.$$
4. **Send ciphertexts and proofs:**
   (a) C_i sends $((\mathsf{ct}_i^{\mathsf{masked}}, \mathsf{ct}_i^{\mathsf{key}}, \mathsf{ct}_i^{\mathsf{seed}}), (\mathcal{X}_i^{\text{PCD}}, \Pi_i^{\text{PCD}}))$ to the server.

Figure 4.9: Client role protocol.

construction of AHE, it entails a single ring multiplication and multiple range proofs.

Server Role. The server plays a central role in our protocol and is responsible for the following tasks: (i) It assists in the setup phase by *possibly* generating the system's public parameters and supports the decryptors in executing the DKG; and (ii) it receives the clients' contributions

and aggregates them into a single ciphertext and PCD proof. The server then generates a proof of uniqueness of IDs, ensuring that no contribution is duplicated, and sends this proof to the decryptors. Upon receiving the partial decryptions, the server can recover the sum of client inputs. The protocol is formally illustrated in Figure 4.10.

Committee Member Role. The committee members operate in two phases. First, they execute DKG to generate a public key. Later, they assist the server in the decryption process.

- *Distributed Key Generation.* DKG is run by the committee with the server as the coordinator. Each committee member generates its AHE and auxiliary keys and shares the underlying secret keys with a verifiable secret sharing scheme. These are then aggregated into global public keys. The committee verifies the validity of the aggregated keys and signs them to certify correctness.

- *Decryption.* The role of the decryptor in WILLOWFOLD is similar to that in Willow, with one key difference. In Willow, decryptors receive a signature from an external verifier attesting to the correctness of the aggregation, and then decrypt the sum of the keys. In WILLOWFOLD, each decryptor instead verifies the aggregation PCD proof on its own; the proof is succinct and efficient to verify. This removes reliance on an external verifier. During decryption, parties use secret shares to reconstruct the keys of any dropped-out members. Once a threshold number of valid partial decryptions is collected, the server can complete the decryption procedure.

The formal protocol is provided in Figure 4.11.

4.5 Security Analysis of WILLOWFOLD

In this section, we analyze the security of WILLOWFOLD. Specifically, we first justify the simulation-extractability requirement. We then discuss simulation extractability in the context of

The server S proceeds as follows during setup and aggregation.

1. **Get the public parameters and the committee's public keys:**

   (a) Run setup (see Figure 4.8). Let $\mathcal{K} = (\mathcal{D}, \Sigma, \mathsf{pk}, \mathsf{pk}_{\text{aux}})$, where $\mathcal{D}$ is the set of decryptors participated in the DKG with $|\mathcal{D}| > t$, and Σ is the set of their signatures on $(\mathsf{pk}, \mathsf{pk}_{\text{aux}})$.

   (b) Initialize encSum, encKey, $\mathsf{encSeeds}$, proofs, and IDs.

2. **Collect client contributions and build the PCD tree:**

   (a) For each i representing a different contribution do the following:

      i. Send $\mathcal{K}$ along with a fresh registration ID $(\mathsf{rt}, g_i, \pi_i)$ to client C_i (see Figure 4.8)

      ii. Receive the contribution $((\mathsf{ct}_i^{\text{masked}}, \mathsf{ct}_i^{\text{key}}, \mathsf{ct}_i^{\text{seed}}), (\mathcal{X}_i^{\text{PCD}}, \Pi_i^{\text{PCD}}))$ from $\mathcal{C}_i$.

      iii. Check the contribution's validity: (i) parse $\mathsf{ct}_i^{\text{key}}$ as $(\mathsf{ct}_0, \mathsf{ct}_1)$ and $\mathcal{X}_i^{\text{PCD}} := (\mathsf{rt}, g_i, u, \mathsf{ct}_1)$ and check ct_1 is consistent; and (ii) Check g_i is a fresh ID and the root rt is correct. Then, append g_i to IDs. (iii) Check Π_i^{PCD} is a valid proof.

      iv. If valid, then update the current state as below:

$$\mathsf{encKey} \mathrel{+}= \mathsf{ct}_i^{\text{key}}, \quad \mathsf{encSum} \mathrel{+}= \mathsf{ct}_i^{\text{masked}}$$

$$\mathsf{encSeeds.append}(\mathsf{ct}_i^{\text{seed}}), \quad \mathsf{proofs.append}((\mathcal{X}_i^{\text{PCD}}, \Pi_i^{\text{PCD}}))$$

   (b) Run PCD to aggregate contributions into a single instance; the base proofs $(\mathcal{X}_i^{\text{PCD}}, \Pi_i^{\text{PCD}})$ appear at the leaves, and each internal node with children $(\mathcal{X}_b, \Pi_b)_{b\in\{0,1\}}$:

      i. Verify the PCD proof for each instance.

      ii. Extract $\{\mathsf{ct}_1^{(b)}\}_{b\in\{0,1\}}$ and $\{g^{(b)}\}_{b\in\{0,1\}}$ from the contributions, and then output

$$g^{\text{sum}} \leftarrow g^{(0)} \cdot g^{(1)}, \quad \mathsf{ct}_1^{\text{sum}} \leftarrow \mathsf{ct}_1^{(0)} + \mathsf{ct}_1^{(1)}.$$

   Let $(\mathcal{X}, \Pi)$ denote the result of accumulating all base PCD proofs via the leaf-decoupled architecture. $\mathcal{X}$ includes the sum of IDs, $\mathsf{com}_{\mathsf{pv}} := g^{\text{sum}}$, and the sum of ciphertexts $\mathsf{ct}_1^{\text{sum}}$.

3. **Prove uniqueness of IDs:** Let $\mathsf{pv}(\boldsymbol{x})$ be the vector committed by $\mathsf{com}_{\mathsf{pv}}$. To prove ID uniqueness of IDs, define $p(\boldsymbol{x}) := \mathsf{pv}(\boldsymbol{x}) \cdot (1 - \mathsf{pv}(\boldsymbol{x}))$ and run a zerocheck on $p(\boldsymbol{x})$.

4. **Prove minimum participation:** To ensure that at least $\mathsf{min_t}$ clients have contributed, run the sumcheck protocol on $\sum_{\boldsymbol{x}\in\{0,1\}^N} \mathsf{pv}(\boldsymbol{x}) > \min_t$, where $\mathsf{pv}(\boldsymbol{x})$ has been committed to by $\mathsf{com}_{\mathsf{pv}}$.

5. **Decrypt the sum of the contributions:** Parse encKey as $(\mathsf{ct}^0, \mathsf{ct}^1)$.

   (a) Run decryption protocol with $\mathcal{D}$ as described in Figure 4.11: Send ciphertext ct^1, along with proofs (Π, Π_{ID}) Receive k and $\mathsf{sk}_{\mathsf{aux}}$ from $\mathcal{D}$.

   (b) Recover masked sum $\mathsf{maskSum} := \sum_{\mathsf{ct}_i^{\text{seed}}\in\mathsf{encSeeds}} \mathsf{PRG.Expand}(\mathsf{E.Dec}(\mathsf{sk}_{\mathsf{aux}}, \mathsf{ct}_i^{\text{seed}}), \ell)$

   (c) Output $\mathsf{KAHE.Dec}(\mathsf{pp}_{\mathsf{KAHE}}, \mathsf{encSum}, k) - \mathsf{maskSum}$.

Figure 4.10: Server role protocol.

Each decryptor (committee member) participates in two phases—DKG and decryption—as follows.

Distributed Key Generation

1. **Generate keys and secret-share them:**
   (a) Each committee member D_j proceeds as follows:
      i. Sample randomness r_j and r_j^{aux}. Run $(\mathsf{pk}_j, \mathsf{sk}_j, \pi_j) := \mathsf{AHE.VerifiableKeyGen}(r_j)$ and $(\mathsf{pk}_j^{\text{aux}}\mathsf{sk}_j^{\text{aux}}, \pi_i^{\text{aux}}) := \mathsf{E.VerifiableKeyGen}(r_j^{\text{aux}})$.
      ii. Secret-shares r_j and r_j^{aux} within the committee with threshold t.
      iii. Send $(\mathsf{pk}_j, \pi_j, \mathsf{pk}_j^{\text{aux}}, \pi_j^{\text{aux}})$ along with a valid signature σ_j to Coord.
   (b) Let $\mathcal{D}$ be the set of members provided valid proofs/signatures. If $|\mathcal{D}| < t$, abort. Otherwise:

$$\mathsf{pk} := \sum_{j\in[m]} \mathsf{pk}_j, \quad \mathsf{pk}^{\text{aux}} := \sum_{j\in[m]} \mathsf{pk}_j^{\text{aux}}.$$

2. **Verify the keys and sign them:**
   (a) Coord broadcasts $S := \{(\mathsf{pk}_j, \pi_j, \mathsf{pk}_j^{\text{aux}}, \pi_j^{\text{aux}}), \sigma_j\}_{j\in\mathcal{D}}$ within $\mathcal{D}$.
   (b) Each decryptor $\mathsf{D}_j \in \mathcal{D}$:
      i. Check $|S| \geq t$ and proofs π_j and π_j^{aux} and signatures σ_j are valid. Otherwise aborts.
      ii. Send a signature on $(\mathsf{pk}, \mathsf{pk}^{\text{aux}})$ to Coord.
   (c) Coord defines Σ as the resulting set of signatures.
   (d) If signatures in Σ are valid and $|\Sigma| \geq t$, Coord sends $(\mathcal{D}, \Sigma, \mathsf{pk}, \mathsf{pk}^{\text{aux}})$ to $\mathcal{D}$.

Verification and Decryption

1. **Verify the proofs, decrypt the ciphertexts, and forward reconstruction shares:**
   (a) Receive input $(\mathsf{ct}^1, (\mathcal{X}^{\mathsf{PCD}}, \Pi^{\text{PCD}}), (\mathcal{X}^{\text{ID}}, \Pi^{\text{ID}}))$ from the server. Each decryptor $\mathsf{D}_j \in \mathcal{D}$:
      i. Verify proofs Π^{PCD} and Π^{ID}, and ensure their consistency with ct^1.
      ii. Send the partial decryption $\mathsf{pd}_j := \mathsf{AHE.PartialDec}(\mathsf{ct}^1, \mathsf{sk}_j)$.
      iii. Send shares of r_i^{aux} received in DKG to reconstruct the secret key sk^{aux}.
   (b) Coord reconstructs sk^{aux}. If it doesn't match the pk^{aux} from DKG, abort.

2. **Construct secret keys and partial decryptions for inactive decryptors:**
   (a) Coord sends P to every decryptor in P.
   (b) Each $\mathsf{D}_j \in P$, aborts if $\mathsf{D}_j \notin P$. Otherwise sends shares received each $\mathsf{D}_k \notin P$ to Coord.
   (c) Coord reconstructs sk_k for each $\mathsf{D}_k \notin P$. If it does not match pk_k, abort.
   (d) Coord sends $p_j := \sum_{j\in\mathcal{D}} p_j$ and sk^{aux} to the server.

Figure 4.11: Decryptor role protocol.

PCD schemes. Finally, we provide the simulation-security proof of WILLOWFOLD.

4.5.1 Ciphertext Correlation Intractability

Each client C_i must prove knowledge of the key k_i, i.e., the plaintext underlying $\mathsf{ct}_i^{\mathsf{key}}$, to rule out correlated ciphertext attacks. However, knowledge soundness alone is insufficient: we require the proofs to be non-malleable. For example, if per-client proofs were additively homomorphic, an adversary could sum the ciphertexts $\mathsf{ct}_i^{\mathsf{key}}$ together with their proofs and thereby mount the attack. To formalize this intuition, we introduce *ciphertext correlation intractability*.

This property ensures that no adversary, given a set $(c_i, \pi_i)_i$ of AHE ciphertexts—each accompanied by a proof of knowledge of its underlying *random* plaintext m_i—can produce a new ciphertext c^* together with a valid proof of knowledge of its plaintext m^* such that m^* is correlated with the original messages m_{ii}. In other words, we require that $(m^*, \{m_i\}_i) \approx_c (m^*, \{b_i\}_i)$ for uniformly random messages $\{b_i\}_i$, where $\approx_c$ denotes *computational indistinguishability*.

Definition 11 (Ciphertext Correlation Intractability)**.** *Let* AHE *be a threshold additively homomorphic encryption scheme with message space* $\mathcal{M}$ *and nonce space* $\mathcal{R}$*. Let* Π *be an argument for the relation*

$$\mathcal{R}_{\mathsf{e}} = \{(\mathsf{ct}, (m, r)) : \mathsf{ct} = \mathsf{AHE.Encrypt}(m, \mathsf{pk}; r)\}.$$

For any $n, \ell \in \mathbb{N}$*, and a message distribution* $\mathcal{K}$ *over* $\mathcal{M}$*, we say that* $\Phi := (\mathsf{AHE}, \Pi)$ *satisfies* $(\mathcal{K}, n, \ell)$*-ciphertext correlation intractability (see Figure 4.12) if for every efficient adversary* $\mathcal{A}$ *and distinguisher* $\mathcal{D}$*,*

$$\mathsf{CCI}_{0,\Phi}^{\mathcal{D},\mathcal{A}}(\lambda) \approx_c \mathsf{CCI}_{1,\Phi}^{\mathcal{D},\mathcal{A}}(\lambda).$$

Below, we show that ciphertext correlation intractability holds when AHE is $\mathsf{IND\text{-}CPA}$ se-

Game $\mathsf{CCI}^{\mathcal{D},\mathcal{A}}_{b,(\mathsf{AHE},\Pi)}(1^\lambda)$

1. Run $\mathsf{pp} \leftarrow \mathsf{AHE.Setup}(1^\lambda)$.
2. Run $(\mathsf{pk}, \{\mathsf{sk}_j\}_{j\in[k]}) \leftarrow \mathsf{AHE.KeyGen}(\mathsf{pp})$.
3. Set $\forall i \in [n], \forall c \in \{0,1\} : m_{i,c} \leftarrow \mathcal{K}, r_{i,c} \leftarrow \mathcal{R}$.
4. Set $\forall i \in [n] : (\mathsf{ct}_i, \pi_i) \leftarrow \mathsf{AHE.VerifiableEnc}(\mathsf{pp}, \mathsf{pk}, m_{i,b}; r_{i,b})$.
5. Run $\{(\mathsf{ct}^*_i, \pi^*_i)\}_{i\in[\ell]} \leftarrow \mathcal{A}(\mathsf{pp}, \{(\mathsf{ct}_i, \pi_i)\}_{i\in[n]}, \mathsf{pk})$.
6. **For** $i = 1, 2, \ldots, \ell$:
   (a) **If** $(\mathsf{ct}^*_i, \pi^*_i) \in \{(\mathsf{ct}_i, \pi_i)\}_{i\in[n]}$ **set** $m^*_i = \perp$.
   (b) **Else set** $m^*_i \leftarrow \mathsf{VDec}_{\mathsf{pp}_\Pi}(\mathsf{pp}, \mathsf{ct}^*_i, \pi^*_i, \mathsf{pk}, \{\mathsf{sk}_j\}_j)$
   (c) Let $M \subseteq \{m^*_i\}_{i\in[\ell]}$ with all message $m^*_i \neq \perp$.
7. **Return** $\mathcal{D}(\{m_{i,0}, m_{i,1}\}_{i\in[n]}, M)$

$\mathsf{VDec}_{\mathsf{pp}_\Pi}(\mathsf{pp}, \mathsf{ct}, \pi, \mathsf{pk}, \{\mathsf{sk}_j\}_{j\in[k]})$

1. **If** $\mathsf{AHE.VerifyEnc.}(\mathsf{pp}, \mathsf{pk}, \mathsf{ct}, \pi) = 0$ **return** $\perp$
2. $\forall j \in [k] : \mathsf{pd}_j \leftarrow \mathsf{AHE.ParDec}(\mathsf{pp}, \mathsf{ct}, \mathsf{sk}_j)$
3. $m \leftarrow \mathsf{AHE.Recover}(\mathsf{pp}, \mathsf{ct}, \{\mathsf{pd}_j\}_{j\in[k]})$
4. **Return** m^*

Figure 4.12: Ciphertext-correlation intractability game.

cure and the proofs of plaintext knowledge are simulation extractable.

Theorem 4.5.1. *Assume* AHE *is an* IND-CPA *threshold additively homomorphic encryption scheme, and let* Π *be the underlying argument system used for proofs of plaintext knowledge such that* Π *satisfies simulation extractability. Then, for any* $n, \ell \in \mathbb{N}$ *and any distribution* $\mathcal{K}$ *over the plaintext domain,* AHE *satisfies* $(\mathcal{K}, n, \ell)$*-ciphertext correlation intractability.*

Proof. Let $(\mathcal{A}, \mathcal{D})$ denote efficient algorithms against CCI game with a non-negligible advantage ϵ_{CCI}. Then, there exists an efficient adversary $\mathcal{B}$ against the IND-CPA security of AHE. In particu-

lar, let $\mathcal{C}$ be the challenger in the IND-CPA security game of AHE, and Sim and $\mathcal{E}$ be the simulator and extractor in the Sim-Ext security of Π. Then, the reduction works as follows:

- Run $\mathsf{pp}_\Pi \leftarrow \Pi.\mathsf{Setup}(\lambda)$.
- Receive $(\mathsf{pp}, \mathsf{pk})$ from $\mathcal{C}$.
- $\forall i \in [n], \forall c \in \{0,1\} : m_{i,c} \leftarrow \mathcal{K}$.
- Send $\{m_{i,c}\}_{i,c}$ to $\mathcal{C}$ and receive $\{\mathsf{ct}_i\}_{i\in[n]}$.
- $\forall i \in [n]$: Run Sim on input $(\mathsf{pp}, \mathsf{ct}_i, \mathsf{pk})$ to simulate a proof π_i. Record $(\mathsf{pp}, \mathsf{ct}_i, \mathsf{pk}, \pi_i)$ in Q_{Sim}.
- Run $\{(\mathsf{ct}_i^*, \pi_i^*)\}_{i\in[\ell]} \leftarrow \mathcal{A}(\mathsf{pp}, \{(\mathsf{ct}_i, \pi_i)\}_{i\in[n]}, \mathsf{pk})$.
- For $i = 1, 2, \ldots, \ell$:
  - If $(\mathsf{ct}_i^*, \pi_i^*) \in \{(\mathsf{ct}_i, \pi_i)\}_{i\in[n]}$ set $m_i^* = \perp$.
  - If π_i^* is not a valid proof for ct_i^* set $m_i^* = \perp$.
  - Else run $\mathcal{E}^{\mathcal{A}}$ on input $(\mathsf{ct}_i^*, \pi_i^*)$ to extract m_i^*.
  - Let $M \subseteq \{m_i^*\}_{i\in[\ell]}$ with all message $m_i^* \neq \perp$.
- Return $b^* \leftarrow \mathcal{D}(\{m_{i,0}, m_{i,1}\}_{i\in[n]}, M)$.

The simulated proofs π_i are indistinguishable from real ones, since Π is zero-knowledge. $\mathcal{E}^{\mathcal{A}}$ extracts m_i^* with overwhelming probability $\epsilon_{\mathsf{Sim\text{-}Ext}}$. Thus, $\mathcal{B}$ has advantage $\epsilon_{\mathsf{CCI}} - \epsilon_{\mathsf{Sim\text{-}Ext}}$. □

4.5.2 Simulation Extractability of PCD Schemes

As discussed before, clients' proofs must satisfy simulation extractability to rule out correlated ciphertext attacks. In Willow, these proofs are instantiated with Bulletproofs [39], which is

simulation extractable [58]. In WILLOWFOLD, the situation is more subtle, since clients' proofs are aggregated into a single instance via a leaf-decoupled PCD scheme. The standard definition of simulation extractability does not extend directly to PCD schemes. For example, an adversary may query the simulator for a PCD proof and then locally run the next PCD step to obtain a fresh proof. Since this proof is built on a simulated proof, the PCD transcript may not be extractable.

A concurrent work [4] addressed this issue and introduced a notion of simulation extractability for PCD in which the extractor extracts witnesses only from non-simulated PCD nodes, and stops once it reaches a simulated proof. They show that a PCD scheme built from a NARK and an accumulation scheme for the NARK verifier satisfies SE, provided that the NARK is simulation extractable and the accumulation scheme is zero-knowledge and knowledge sound.

WILLOWFOLD uses leaf-decoupled PCD in which the argument and accumulation schemes at the internal nodes are not zero-knowledge. Below, we tailor the notion of SE to leaf-decoupled PCDs where the simulator answers only base proof (leaf) queries. We call this notion *leaf-SE*, and show that it holds when base proofs are SE, and the accumulation scheme is knowledge sound. To state the definition, we first introduce the notion of partially compliant PCD transcripts. A PCD transcript is partially compliant if all nodes are either compliant or simulated.

Definition 12 (Simulation Extractability of Leaf-Decoupled PCD)**.** *Let* PCD *be a leaf-decoupled PCD scheme with a leaf simulator* $\mathsf{Sim}^{\mathsf{leaf}}$*. We say that* PCD *satisfies simulation extractability with respect to* $\mathsf{Sim}^{\mathsf{leaf}}$ *if there exists an extractor* $\mathcal{E}$ *such that, for every efficient adversary* $\mathcal{A}$*,*

$$\mathsf{Sim\text{-}Ext\text{-}PCD}_{0,\Pi}^{\mathsf{Sim}^{\mathsf{leaf}},\mathcal{A}}(1^\lambda) \approx_c \mathsf{Sim\text{-}Ext\text{-}PCD}_{1,\Pi,R}^{\mathcal{E},\mathsf{Sim}^{\mathsf{leaf}},\mathcal{A}}(1^\lambda).$$

The following lemma shows that a leaf-decoupled PCD built from a simulation extractable

Game $\mathsf{Sim\text{-}Ext\text{-}PCD}^{\mathsf{Sim}^{\mathsf{leaf}},\mathcal{A}}_{0,\Pi}(1^\lambda)$

- $\mathsf{pp} \leftarrow \Pi.\mathsf{Setup}(1^\lambda, R)$
- $(z, \pi) \leftarrow \mathcal{A}^{\mathcal{O},\mathcal{H},\mathsf{Sim}^{\mathsf{leaf}}}(\mathsf{pp})$
- $b \leftarrow \mathcal{V}^{\mathcal{H}}(\mathsf{pp}, z, \pi)$
- **Return** $b \wedge ((z, \pi) \notin Q_{\mathsf{Sim}^{\mathsf{leaf}}})$

$\underline{\mathcal{V}^{\mathcal{H}}(\mathsf{pp}, z, \pi)}$

- **If** $\mathsf{isBase}(z) = 1$ **return** $\mathsf{V}^{\mathcal{H}}_{\mathsf{leaf}}(\mathsf{pp}, z, \pi)$
- **Else return** $\mathsf{V}^{\mathcal{H}}_{\mathsf{internal}}(\mathsf{pp}, z, \pi)$

Game $\mathsf{Sim\text{-}Ext\text{-}PCD}^{\mathcal{E},\mathsf{Sim}^{\mathsf{leaf}},\mathcal{A}}_{1,\Pi,R}(1^\lambda)$

- $\mathsf{pp} \leftarrow \Pi.\mathsf{Setup}(1^\lambda, R)$
- $(z, \pi) \leftarrow \mathcal{A}^{\mathcal{O},\mathcal{H},\mathsf{Sim}^{\mathsf{leaf}}}(\mathsf{pp})$
- $b \leftarrow \mathcal{V}^{\mathcal{H}}(\mathsf{pp}, z, \pi)$
- $\mathsf{PCDT} \leftarrow \mathcal{E}^{\mathcal{A}}(\mathsf{pp}, z, \pi, Q_{\mathcal{O}}, Q_{\mathcal{H}}, Q_{\mathsf{Sim}^{\mathsf{leaf}}})$
- $b_{\mathsf{out}} \leftarrow (z = \mathsf{PCDT.Output})$
- $b_{\mathsf{comp}} \leftarrow \mathsf{IsComp}(\mathsf{PCDT}, Q_{\mathsf{Sim}^{\mathsf{leaf}}})$
- **Return** $b \wedge b_{\mathsf{out}} \wedge b_{\mathsf{comp}} \wedge ((z, \pi) \notin Q_{\mathsf{Sim}^{\mathsf{leaf}}})$

$\underline{\mathcal{V}^{\mathcal{H}}(\mathsf{pp}, z, \pi)}$

- **If** $\mathsf{isBase}(z) = 1$ **return** $\mathsf{V}^{\mathcal{H}}_{\mathsf{leaf}}(\mathsf{pp}, z, \pi)$
- **Else return** $\mathsf{V}^{\mathcal{H}}_{\mathsf{internal}}(\mathsf{pp}, z, \pi)$

Figure 4.13: Leaf-decoupled PCD simulation extractability games. $\mathsf{Sim}^{\mathsf{leaf}}$ denotes the simulator of leaf nodes. IsComp is an efficient algorithm that checks whether PCDT is $(\mathcal{V}^{\mathcal{H}}, Q_{\mathsf{Sim}})$-compliant.

argument and a knowledge sound accumulation scheme satisfies simulation extractability.

Lemma 4.5.2. *Let* Π_{leaf} *be a simulation extractable and* Π_{internal} *be a knowledge sound non-interactive argument. Let* $\mathsf{ACC}_{\mathsf{leaf}}$ *and* $\mathsf{ACC}_{\mathsf{internal}}$ *be knowledge sound accumulation schemes for the verifier statements of* Π_{leaf} *and* Π_{internal}*, respectively. The Leaf-decoupled PCD built from* $(\Pi_{\mathsf{leaf}}, \mathsf{ACC}_{\mathsf{leaf}}, \Pi_{\mathsf{internal}}, \mathsf{ACC}_{\mathsf{internal}})$*—where* Π_{leaf} *is used at the leaves and* Π_{internal} *at internal nodes—satisfies leaf-SE.*

Proof. Our proof follows the approach of Abdolmaleki et al. [4], with the key difference that our simulator is weaker and only answers leaf queries. For each layer $i \in [d]$ of the PCD transcript, we construct an extractor $\mathcal{E}_i$ that extracts a PCD transcript of depth i. We show how to build $\mathcal{E}_{i+1}$ from $\mathcal{E}_i$ for each $i \in [d-2]$ using the extractor of the accumulation scheme. Finally, we combine

$\mathcal{E}_{d-1}$ with the extractor of the NARK to obtain $\mathcal{E}_d$, which outputs witnesses for all non-simulated leaf nodes of the PCD tree. The extractor $\mathcal{E}_0$ is simply the adversary, which outputs a tuple (z, π). Let $\mathsf{PCDT}_0 := (z, \pi)$. Then, for each $i \in [d-3]$, $\mathcal{E}_{i+1}$ proceeds as follows:

- Run $\mathcal{E}_i$ to obtain PCDT_i, a PCD transcript of depth i.
- For each node u at depth i of PCDT_i, run $\mathcal{E}^{\mathsf{ACC}_{\mathsf{internal}}}$, the extractor for $\mathsf{ACC}_{\mathsf{internal}}$, to extract children of u. Since $\mathsf{ACC}_{\mathsf{internal}}$ is knowledge sound, this succeeds with high probability.
- Extend PCDT_i with the extracted children nodes to obtain PCDT_{i+1}. Output PCDT_{i+1}.

The extractor $\mathcal{E}_{d-1}$ proceeds as above; however, instead of invoking $\mathcal{E}^{\mathsf{ACC}_{\mathsf{internal}}}$, it invokes $\mathcal{E}^{\mathsf{ACC}_{\mathsf{leaf}}}$ to extract the leaf nodes and add them to the PCD transcript. Finally, $\mathcal{E}_d$ first runs $\mathcal{E}_{d-1}$ to obtain PCDT_{d-1} and then runs $\mathcal{E}^{\Pi_{\mathsf{leaf}}}$ on each leaf node to extract a witness. Each extractor $\mathcal{E}_i$ invokes $\mathsf{Sim}^{\mathsf{leaf}}$ to answer the adversary's simulator queries. Since Π_{leaf} is simulation extractable, the above extraction succeeds with overwhelming probability. □

4.5.3 Simulation-Based Security of WILLOWFOLD

We show security of WILLOWFOLD in the standard ideal/real simulation paradigm with respect to the secure aggregation functionality in Figure 4.1. We first outline the simulator in both malicious and semi-honest cases. Let Sim denote the simulator, $\mathcal{A}$ denote the adversary, and H and C be the sets of honest and corrupted clients, respectively.

Malicious Security. For DKG, Sim runs the DKG protocol on behalf of the honest committee members. For aggregation, Sim uses input 0^ℓ and sends contributions on behalf of all honest clients. The adversary $\mathcal{A}$ controls the server and forwards a PCD instance $(\mathcal{X}'^{\mathsf{PCD}}, \Pi^{\mathsf{PCD}})$ that includes a ciphertext $\mathsf{ct}^{\mathsf{sum}}$ and a commitment $\mathsf{com}_{\mathsf{pv}}$. Once Sim receives this message, it re-

programs the random oracle so that the sum of the honest clients' contributions *that the server included in the aggregation* matches the real-world sum. To this end, Sim runs the PCD extractor on $(\mathcal{X}^{\mathsf{PCD}}, \Pi^{\mathsf{PCD}})$ to recover the full PCD transcript and the witnesses for leaf nodes and, using this information, Sim computes the set S_H of the honest clients that were included by the server in the final aggregate. Sim then submits S_H to $\mathcal{F}_{\mathsf{Agg}}$ to obtain the sum of the relevant honest their inputs. Finally, Sim reprograms the random oracle at the points used by the PRG of one of the honest client so that the simulated contributions (with input 0^ℓ) match the real-world contributions in sum. Moreover, in the above simulation, Sim also ensures that IDs are unique and verifies the PCD proof, therefore playing the role of the verifier.

Semi-Honest Security. Consider a semi-honest server that follows the protocol, but whose memory is observed by the adversary. For setup and DKG, Sim runs the protocol on behalf of the server and the committee members. For aggregation, as in the previous case, Sim uses inputs 0^ℓ to simulate the honest clients and additionally runs the server protocol. In this setting, however, Sim cannot query $\mathcal{F}_{\mathsf{Agg}}$ for the sum of only the honest clients' inputs; instead, it must provide $\mathcal{F}_{\mathsf{Agg}}$ with the full set of participating clients, including both honest and corrupted ones. To reprogram the random oracle, Sim therefore needs to learn the sum of the corrupted clients' inputs and subtract it from the output returned by $\mathcal{F}_{\mathsf{Agg}}$. This can be done by running the extractor for the PCD base proofs received from $\mathcal{A}$; it outputs witnesses that include the inputs chosen by $\mathcal{A}$. Sim also checks that the IDs are unique, which can be verified directly from the proofs sent by the adversary.

To provide the hybrid argument, we first recall the notion of key-sum leakage resistance—a property we require for the KAHE scheme.

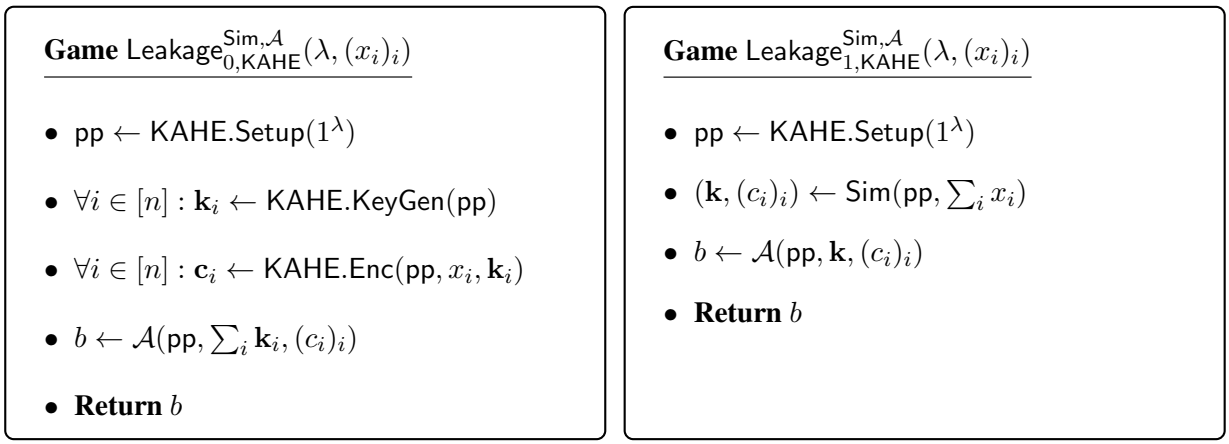


Figure 4.14: Key sum leakage resistance game.

Definition 13 (Key-Sum Leakage Resistance)**.** *[17] For any $n \in \mathbb{N}$, an KAHE scheme is n-semantic secure under leakage of key sum (Figure 4.14) if there exists an efficient simulator* Sim *s.t. for any inputs $(x_i)_{i\in[n]}$ and every efficient $\mathcal{A}$,*

$$\mathsf{Leakage}^{\mathsf{Sim},\mathcal{A}}_{0,\mathsf{KAHE}}(\lambda,(x_i)_i) \approx_c \mathsf{Leakage}^{\mathsf{Sim},\mathcal{A}}_{1,\mathsf{KAHE}}(\lambda,(x_i)_i).$$

Theorem 4.5.3. *Assume* KAHE *is a key- and message-homomorphic encryption scheme with key-sum leakage resistance,* AHE *is a threshold additively homomorphic encryption scheme, and Π_{PCD} is a leaf-decoupled PCD scheme built from a simulation extractable argument and a knowledge sound accumulation scheme. Then, the secure aggregation protocol of Figures 4.9, 4.10, and 4.11 realizes $\mathcal{F}_{\mathsf{Agg}}$ in Figure 4.1 in the (RO, Setup)-hybrid model against a static active adversary corrupting the server, any number of clients, and $\leq 2t - k - 1$ decryptors, with* AHE *threshold t and committee size k.*

Proof. We prove security of the scheme in the (RO,Setup)-hybrid model and assume AHE.KeyGen is realized via a secure distributed key generation (see [17] for a formal proof of our DKG instan-

tiation). Below, we only focus on the aggregation and give hybrids.

Proof Overview. There are a couple of differences between the real world and the ideal world. First, the random oracle is reprogrammed. However, since seeds are encrypted and only revealed after reprogramming, this remains indistinguishable from the real world. Second, Sim simulates honest clients with input 0^ℓ. This is also indistinguishable from the real world because: (i) KAHE is secure even under leakage of the sum of the keys; (ii) AHE is ciphertext correlation intractable; therefore the adversary's keys are uncorrelated with honest clients' keys and do not change their sum distribution which implies that it is calibrated correctly to invoke the leakage-resilient property of the KAHE scheme; and (iii) PCD leaves are zero-knowledge and keep clients' inputs hidden.

Note that zero-knowledge is required only for PCD leaves, since internal nodes are deterministic and carry no witness. The same holds for simulation extractability.

The hybrids are similar for malicious and semi-honest settings; they differ only in how Sim recovers the set of included honest clients S_H, to subsequently retrieve $\sum_{i \in S_H} \boldsymbol{x}_i$ from the ideal functionality, and reprogram the random oracle. For simplicity, in the sequel we assume that the server does not drop or replace any honest contribution, i.e., $H = S_H$ as the simulator can extract S_H from PCD leaves.

Hybrid Argument. $\mathcal{A}$ may corrupt the server, any number of clients, and $\leq 2t-k-1$ decryptors.

- $\mathsf{Hyb}^{(0)}$: The real execution, and $\mathcal{A}$'s view include honest contributions

$$(\mathsf{ct}_i^{\mathsf{key}}, \mathsf{ct}_i^{\mathsf{seed}}, \mathsf{ct}_i^{\mathsf{masked}}, \Pi_i^{\mathsf{PCD}})$$

and honest partial decryptions pd_j. It outputs $(\mathsf{rt}, g_i, \pi_i)$ to honest clients and

$$(\mathsf{encKey}, g_{\mathsf{pv}}, \Pi^{\mathsf{PCD}}), \Pi^{\mathsf{ID}}$$

to honest decryptors.

- $\mathsf{Hyb}^{(1)}$: The same as $\mathsf{Hyb}^{(0)}$, except that we set one of the honest client's input to $\sum_{i \in H} \boldsymbol{x}_i$ and other honest clients' inputs to 0^ℓ. $\mathsf{Hyb}^{(1)}$ is indistinguishable from $\mathsf{Hyb}^{(0)}$ since we know that KAHE is n-semantically secure under key sum leakage and hides the clients' inputs. Note that AHE is ciphertext correlation intractable due to Theorem 4.5.1, which implies that the keys $\{k_i\}_{i \in C}$ used by corrupted parties are independent of the keys $\{k_i\}_{i \in H}$ used by honest clients. Hence, the adversary learns nothing about $\{k_i\}_{i \in H}$ beyond $\sum_{i \in H} k_i$.

- $\mathsf{Hyb}^{(2)}$: The same as $\mathsf{Hyb}^{(1)}$, except that we set all honest clients' input to 0^ℓ and reprogram the RO so that $\mathcal{H}(\mathsf{seed}_h) \mapsto \sum_i \mathbf{x}_i$ for one of the honest clients $h \in H$. Since seeds are uniformly random and AHE is semantically secure, the probability that the adversary invokes PRG on input seed_h before reprogramming is negligible. Thus, $\mathsf{Hyb}^{(2)}$ is indistinguishable from $\mathsf{Hyb}^{(1)}$.

- $\mathsf{Hyb}^{(3)}$: The same as $\mathsf{Hyb}^{(2)}$, except that the partial decryptions are replaced with the output of the AHE simulator. By the security of threshold decryption in the AHE scheme, $\mathsf{Hyb}^{(3)}$ is indistinguishable from $\mathsf{Hyb}^{(1)}$.

$\mathsf{Hyb}^{(3)}$ almost matches the ideal world simulated by Sim up to a single left difference: Sim aborts if any duplicate IDs appear in the leaf proofs. By simulation extractability of Π^{PCD} and soundness of Π^{ID}, the real-world execution would also abort in this case. Thus, $\mathsf{Hyb}^{(3)}$ is indistinguishable from the ideal world. □

4.6 Instantiation with Spartan+KZH-Fold

WILLOWFOLD is generic over any leaf-decoupled PCD scheme that satisfies leaf-SE. In this work, we instantiate WILLOWFOLD using a variant of Spartan+KZH-Fold [93]. Spartan+KZH-Fold builds on Spartan+KZH as the underlying SNARK and is compiled into a PCD via the BCLMS compiler [42] using KZH-Fold. Unlike most state-of-the-art PCD schemes [40, 96, 97], Spartan+KZH-Fold has sublinear proof size and verification time, namely $O(k \cdot N^{1/k})$ for an arbitrary parameter k, which becomes $O(\log N)$ when $k = \log N$. To illustrate the concrete impact, consider the client circuit in WILLOWFOLD, which has roughly 1.4M constraints. For this circuit, Nova [96] yields a proof of about 120 MB and a verification time of about 7 seconds. In our benchmarks, Spartan+KZH-Fold produces proofs below 50 KB with verification time around 470 ms, while the prover is only about $3\times$ slower than Nova.

The main challenge is that Spartan+KZH, as described in [93], is not zero-knowledge and therefore cannot be simulation extractable. We describe how to obtain a zero-knowledge variant of Spartan+KZH, which we call zk-Spartan+zk-KZH. We then explain how to build a leaf-decoupled PCD from zk-Spartan+zk-KZH using KZH-Fold, and how to show that this scheme satisfies leaf-SE.

4.6.1 Zero-Knowledge Variant of Spartan+KZH

Spartan+KZH builds on Spartan polynomial IOP and KZH polynomial commitment. As shown in [48, 132], Spartan can be made zero-knowledge via a zero-knowledge sumcheck [146]. IronDict [88], also presents a zero-knowledge variant of the KZH commitment scheme. Composing zero-knowledge Spartan (zk-Spartan) with zero-knowledge KZH (zk-KZH) would then

yield a zero-knowledge SNARK via a standard compilation [50]. This transformation requires a logarithmic amount of randomness, since both zk-Spartan and zk-KZH only use logarithmic randomness.

The main caveat is that the zero-knowledge sumcheck protocol from prior work [146] requires committing to higher-degree multivariate polynomials, rather than multilinear ones. However, KZH is designed for multilinear commitments. Below, we elaborate on this issue.

Zero-Knowledge Sumcheck. Consider a sumcheck instance

$$H := \sum_{\boldsymbol{x} \in \{0,1\}^\ell} f(\boldsymbol{x}),$$

where $f(\boldsymbol{X})$ is an ℓ-variate polynomial of individual degree d. To achieve zero knowledge, the prover masks $f(\boldsymbol{X})$ with a random polynomial $r(\boldsymbol{X})$ of the same degree and then sends $G := \sum_{\boldsymbol{x} \in \{0,1\}^\ell} r(\boldsymbol{x})$ and a commitment to $r(\boldsymbol{X})$. Upon receiving a challenge ρ, the parties run a sumcheck protocol on

$$G + \rho \cdot H \;=\; \sum_{\boldsymbol{x} \in \{0,1\}^\ell} \big(r(\boldsymbol{x}) + \rho \cdot f(\boldsymbol{x})\big).$$

As shown by Xie et al. [146], it suffices for the masking polynomial $r(\boldsymbol{X})$ to be sampled from a highly structured family, where $r(\boldsymbol{X}) \;:=\; r_1(X_1) + r_2(X_2) + \cdots + r_\ell(X_\ell)$, such that each $r_i(X_i)$ is a univariate polynomial of degree d. This representation uses only $d\ell + 1$ random coefficients.

The barrier is that, even though the required randomness is only logarithmic, the polynomial commitment scheme must support degree-d polynomials in order to commit to $r(\boldsymbol{X})$. In Spartan, some sumcheck instances have $d = 2$ or $d = 3$, whereas KZH supports only multilinear

polynomials.

To resolve this issue, one may use a generic transformation from multivariate to multilinear polynomials [154], however, this would blow up the polynomial size to $O((d+1)^\ell)$. Here, we exploit the structure of the randomness, and propose an method to commit to $r(\boldsymbol{X})$ with only $O(d\ell)$ overhead. The scheme is given in Figure 4.15.

Theorem 4.6.1. Π_{commit} *in Figure 4.15 satisfies zero-knowledge and knowledge soundness.*

Proof. The scheme is zero-knowledge because PCS is zero-knowledge, and each coefficient of $r(\boldsymbol{X})$ has been masked with a random coefficient of $h(\boldsymbol{X})$. Soundness follows from the soundness of PCS, the soundness of sumcheck, and from the fact that, over the choice of ρ, for any $y^* \neq r(\boldsymbol{x})$, it holds $\rho \cdot y^* + y_h \neq \rho \cdot r(\boldsymbol{x}) + h(\boldsymbol{x})$ except for a negligible probability $1/|\mathbb{F}|$. Finally, the commitment is extractable since PCS is extractable, and $r(\boldsymbol{x})$ can be extracted from c. □

Remark 4.6.1. *The KZH-Fold accumulation scheme for Spartan+KZH commitments can be adapted to zk-Spartan+zk-KZH with only minor modifications: the Spartan PIOP messages are handled exactly as before. For zk-KZH, the accumulator performs only a constant number of additional group operations beyond those in KZH-Fold.*

4.6.2 Simulation Extractability of Spartan+KZH-Fold

Recall that zk-Spartan+zk-KZH is a NARK with an updatable SRS. Prior work [74] showed that such an argument is simulation extractable if it has three properties: (i) special soundness, (ii) trapdoor-less zero-knowledge, and (iii) unique response. To prove simulation extractability of zk-Spartan+zk-KZH, each of these must be shown.

The special soundness of zk-Spartan+zk-KZH follows from the knowledge soundness of KZH polynomial commitments and Spartan PIOP. Below, we revisit the proof of knowledge

Let PCS be a zero-knowledge multilinear polynomial commitment scheme with public parameters $\mathsf{pp}_{\mathsf{PCS}}$. Moreover, let $r(\boldsymbol{X}) = v_0 + \sum_{i\in[\ell],j\in[d]} v_{i,j} X_i^j$ denote the sumcheck masking polynomial.

Commitment:

1. Set $\widetilde{v}$ to be the multilinear extension of $(v_0, v_{1,1}, \ldots, v_{\ell,d})$.
2. Return $\mathsf{PCS.Commit}(\mathsf{pp}_{\mathsf{PCS}}, \widetilde{v}, r) \to c$.

Opening at point $\boldsymbol{x}$:

1. Sample random $h(\boldsymbol{X}) = u_0 + \sum_{i\in[\ell],j\in[d]} u_{i,j} X_i^j$.
2. Set $\widetilde{u}$ to be the multilinear extension of $(u_0, u_{1,1}, \ldots, u_{\ell,d})$.
3. Run $\mathsf{PCS.Commit}(\mathsf{pp}_{\mathsf{PCS}}, \widetilde{u}, r) \to c_h$.
4. Send c_h, $y_h = h(\boldsymbol{x})$, and $y = r(\boldsymbol{x})$, and receive a challenge ρ.
5. Set $\widetilde{x}^*$ to be the multilinear extension of $(x_i^j)_{i,j}$.
6. Run the sumcheck protocol on

$$\rho y + y_h = \sum_{\boldsymbol{b}\in\{0,1\}^{\log\lceil d\ell+1\rceil}} (\rho\widetilde{v}(\boldsymbol{b}) + \widetilde{u}(\boldsymbol{b})) \cdot \widetilde{x}^*(\boldsymbol{b}),$$

   which ends with evaluations $p := \widetilde{v}(\boldsymbol{s})$, $q := \widetilde{u}(\boldsymbol{s})$, and $t := \widetilde{x}^*(\boldsymbol{s})$ for a random challenge point $\boldsymbol{s}$.
7. Run $\mathsf{PCS.Open}(\mathsf{pp}_{\mathsf{PCS}}, \widetilde{v}(\boldsymbol{X}), \boldsymbol{s}) \to \pi_r$ and $\mathsf{PCS.Open}(\mathsf{pp}_{\mathsf{PCS}}, \widetilde{u}(\boldsymbol{X}), \boldsymbol{s}) \to \pi_h$.
8. Return the sumcheck messages, π_r and π_h.

Verification:

1. Verify the sumcheck messages and their consistency with y and y_h.
2. Run $\mathsf{PCS.Verify}(\mathsf{pp}_{\mathsf{PCS}}, c, \boldsymbol{s}, p)$ and $\mathsf{PCS.Verify}(\mathsf{pp}_{\mathsf{PCS}}, c_h, \boldsymbol{s}, q)$, and check if they accept.
3. Check if $t = \widetilde{x}^*(\boldsymbol{s})$.

Figure 4.15: Commitment to sumcheck masking randomness via a multilinear PCS.

soundness for the KZH polynomial commitment and adapt it to the updatable-SRS setting—Kadianakis et al. [93, Theorem 3] proved knowledge soundness only in the standalone setting where the adversary receives one SRS and does not call the SRS update oracle.

Lemma 4.6.2. *KZH satisfies knowledge soundness in the updatable-SRS setting.*

Proof. Kadianakis et al. [93, Theorem 3] showed that KZH-k is straight-line extractable in the AGM under the DLog and CDH assumptions. Here, we restate their proof in the updatable-SRS setting, where the adversary may also query the SRS update oracle $\mathcal{O}$ arbitrarily many times.

Let s denote the number of SRS updates, and $\mathsf{srs}^{(u)} = (G, V, \{\mathbf{H}_r^{(u)}\}_{r\in[k]}, \mathbf{V}^{(u)})$ be the SRS after the uth update, where $\mathsf{srs}^{(0)}$ denotes the initial SRS. For each $u \in [s]$ and $r \in [k]$, we have

$$\mathbf{V}_{*,r}^{(u)} := \alpha_r^{(u)} \cdot \mathbf{V}_{*,r}^{(u-1)} \quad \text{and} \quad \mathbf{H}_r^{(u)} = \alpha_r^{(u)} \cdot \mathbf{H}_r^{(u-1)},$$

where $\alpha_r^{(u)}$ is the update randomness sampled by the update oracle $\mathcal{O}$ on the uth update query.

Let $\mathcal{A}$ be a PPT adversary that, on input $\{\mathsf{srs}^{(u)}\}_{u\in[s]}$, outputs a commitment C and a tuple $((x_1, \ldots, x_k), y, \pi)$ that is accepted by the verifier. We construct an extractor $\mathcal{E}$ that outputs a polynomial $p(X_1, \ldots, X_k)$ such that $p(x_1, \ldots, x_k) = y$. Under the AGM, $\mathcal{A}$ outputs C together with coefficients c and $\{\boldsymbol{c}_r^{(u)}\}_{u\in[s],r\in[k]}$, where

$$C = c \times G + \sum_{u\in[s],r\in[k]} \langle \boldsymbol{c}_r^{(u)}, \mathbf{H}_r^{(u)} \rangle.$$

$\mathcal{E}$ observes $\{\boldsymbol{c}_r^{(u)}\}_{u\in[s],r\in[k]}$ and outputs $p(X_1, \ldots, X_k) := (\boldsymbol{c}_1^{(s)}, \ldots, \boldsymbol{c}_k^{(s)}) \otimes (X_1, \ldots, X_k)$. Then, we must show that $p(X_1, \ldots, X_k)$ is consistent with the tuple $((x_1, \ldots, x_k), y, \pi)$.

Let parse the opening proof π as $(\{\mathbf{D}_j\}_{j\in[k-1]}, \mathbf{T}_k)$. Under the AGM, there exist coef-

ficients $\{d_{j,i}\}_{j\in[k-1],i\in[d]}$ and $\{\boldsymbol{d}^{(u)}_{r,j,i}\}_{u\in[s],\, r\in[k],j\in[k-1],\, i\in[d]}$ such that, for each $j \in [k-1]$ and $i \in [d]$,

$$D_j[i] = d_{j,i} \times G + \sum_{u\in[s],\, r\in[k]} \langle \boldsymbol{d}^{(u)}_{r,j,i}, \mathbf{H}^{(u)}_r \rangle.$$

Since the verifier accepts π, it holds that $e(C, V) = \sum_{i\in[d]} e(D_1[i], V^{(s)}_{i,1})$. In other words,

$$e(c \times G + \sum_{u\in[s],r\in[k]} \langle \boldsymbol{c}^{(u)}_r, \mathbf{H}^{(u)}_r \rangle, V) = \sum_{i\in[d]} e(d_{1,i} \times G + \sum_{u\in[s],\, r\in[k]} \langle \boldsymbol{d}^{(u)}_{r,1,i}, \mathbf{H}^{(u)}_r \rangle, \mu^{(s)}_{i,1} \times V).$$

Observing that every term on the right-hand side has a factor of $\mu^{(s)}_{i,1}$, we conclude that $c = 0$, $\boldsymbol{c}^{(u)}_r = 0$ for all $u \neq s$, and $\boldsymbol{c}^{(s)}_r = 0$ for all $r \neq 1$; otherwise, $\mathcal{A}$ would break the DLog assumption. Then, we have $\boldsymbol{d}^{(u)}_{r,1,i} = 0$ for all $u \neq s$ and $r \notin \{1, 2\}$ and $d_{1,i} = 0$ for all $i \in [d]$ due to the DLog assumption. The term $(\mu^{(s)}_{i,1})^2$ appears only on the right-hand side through $\langle \boldsymbol{d}^{(s)}_{1,1,i}, \mathbf{H}^{(s)}_1 \rangle$. Therefore, we must have $\boldsymbol{d}^{(s)}_{1,1,i} = 0$; otherwise, $\mathcal{A}$ would break the CDH assumption. Finally, note that for all $i \in [d]$ and all $t \neq i$, the tth entry of $\boldsymbol{d}^{(s)}_{2,1,i}$ must be zero; otherwise, the product term $\mu^{(s)}_{i,1} \cdot \mu^{(s)}_{t,2}$ would appear on the right-hand side, yet it cannot be generated on the left-hand side under the CDH assumption. With the above implications, the verification equation would be simplified to:

$$e(\langle \boldsymbol{c}^{(s)}_1, \mathbf{H}^{(s)}_1 \rangle, V) = \sum_{i\in[d]} e(\langle \boldsymbol{d}^{(s)}_{2,1,i}, \mathbf{H}^{(s)}_2 \rangle, \mu^{(s)}_{i,1} \times V).$$

Then, by the DLog assumption, we must have $\boldsymbol{c}^{(s)}_1 = \sum_{i\in[d]} \boldsymbol{d}^{(s)}_{2,1,i}$. The verifier, similarly, checks

that $e(C_{j-1}, V) = \sum_{i\in[d]} e(D_j[i], V_{i,j}^{(s)})$ for all $j = 2, 3, \ldots, k-1$ such that $C_j = \langle \mathbf{x}_j, \mathbf{D}_j \rangle$. Thus,

$$e(\langle \mathbf{x}_{j-1}, \mathbf{D}_{j-1} \rangle, V) = \sum_{i\in[d]} e(d_{j,i} \times G + \sum_{u\in[s],\, r\in[k]} \langle \boldsymbol{d}_{r,j,i}^{(u)}, \mathbf{H}_r^{(u)} \rangle, \mu_{i,j}^{(s)} \times V).$$

We conclude that $\mathbf{D}_{j-1} = \langle \boldsymbol{d}_{j,j-1,*}^{(s)}, \mathbf{H}_j^{(s)} \rangle$, as above, due to the DLog or CDH assumptions. Then,

$$e(\langle \mathbf{x}_{j-1}, \langle \boldsymbol{d}_{j,j-1,*}^{(s)}, \mathbf{H}_j^{(s)} \rangle \rangle, V) = \sum_{i\in[d]} e(\langle \boldsymbol{d}_{j+1,j,i}^{(s)}, \mathbf{H}_{j+1}^{(s)} \rangle, \mu_{i,j}^{(s)} \times V).$$

Thus, $\langle \mathbf{x}_{j-1}, \langle \boldsymbol{d}_{j,\,j-1,\,*}^{(s)}, \mathbf{H}_j^{(s)} \rangle \rangle = \langle \sum_{i\in[d]} \boldsymbol{d}_{j+1,\,j,\,i}^{(s)}, \mathbf{H}_j^{(s)} \rangle$, and so, $\sum_{i\in[d]} d_{j+1,\,j,\,i}^{(s)} = \langle \mathbf{x}_{j-1}, \boldsymbol{d}_{j,\,j-1,\,*}^{(s)} \rangle$. Finally, since $C_{k-1} = \langle \mathbf{T}_k, \mathbf{H}_k \rangle = \left\langle \mathbf{x}_{k-1}, \left\langle \boldsymbol{d}_{k,\,k-1,\,*}^{(s)}, \mathbf{H}_k^{(s)} \right\rangle \right\rangle$, we obtain $\mathbf{T}_k = \left\langle \mathbf{x}_{k-1}, \boldsymbol{d}_{k,\,k-1,\,*}^{(s)} \right\rangle$. Hence, $\mathbf{T}_k = (\boldsymbol{c}_1^{(s)}, \ldots, \boldsymbol{c}_{k-1}^{(s)}) \otimes (x_1, \ldots, x_{k-1})$ and $\langle \mathbf{T}_k, \mathbf{x}_k \rangle = y$. This completes the proof. □

Below, we recall a lemma of Hafezi et al. [88] which shows that zk-KZH satisfies 2-special soundness if KZH is knowledge sound. Since the extractor for zk-KZH only rewinds the adversary and does not rely on the AGM, their result naturally extends to the updatable-SRS setting.

Lemma 4.6.3. *[88, Lemma 3]* zk-*KZH satisfies* 2*-special soundness in the updatable-SRS setting under the knowledge soundness of KZH in the updatable-SRS setting.*

Lemma 4.6.4. zk-*Spartan* + zk-*KZH satisfies special soundness in the updatable-SRS setting.*

Proof. We can extract a witness and construct a tree of accepting transcripts by invoking the tree-extractor for zk-KZH. Lemma 4.6.2 guarantees that this procedure succeeds with overwhelming probability. Moreover, since Spartan is a sound PIOP [132, Lemma A.1], the extracted witness is satisfiable, and the resulting transcript tree is accepting. Soundness of the zero-knowledge Spartan PIOP (built from zero-knowledge sumchecks) was shown in [48, Theorem A.3]. □

Next, we show that zk-Spartan+zk-KZH satisfies the TLZK property.

Lemma 4.6.5. *Spartan + zk-KZH with r rounds satisfies $(r-1)$-trapdoor-less zero-knowledge.*

Proof. Both zk-Spartan (built from zero-knowledge sumchecks) and zk-KZH are zero-knowledge, and hence, their composition is also zero-knowledge [50]. The simulator for zk-Spartan+zk-KZH invokes the zk-KZH simulator to generate commitments and openings, and invokes the Spartan simulator to generate the sumcheck messages. The simulator for Spartan is information-theoretic and neither uses any trapdoor nor reprograms the random oracle. The simulator for KZH does not employ any trapdoor, but it reprograms the random oracle once when simulating the opening proof. Since openings are invoked in round $r-1$, zk-Spartan+zk-KZH is $(r-1)$-TLZK. □

Finally, we need show that zk-Spartan+zk-KZH satisfies the unique response property. For this purpose, we first prove that zk-KZH satisfies 1-unique response.

Lemma 4.6.6. zk-*KZH satisfies* 1-*unique response.*

Proof. The opening procedure of zk-KZH consists of two rounds. In the first round, the prover sends a commitment to a masked polynomial and its evaluation at the query point, and the verifier issues a challenge. In the second round, the prover provides a KZH opening proof for the masked polynomial. Therefore, to show that zk-KZH satisfies 1-unique response, it suffices to show that, for a fixed commitment and evaluation point, KZH admits a unique opening proof.

Suppose there exist two distinct openings (y, π) and (y^*, π^*) for a commitment C at the point $(\mathbf{x}_1, \ldots, \mathbf{x}_k)$. Since both openings are accepted, the following three conditions must hold:

- $\forall j \in [k-1]$, we have $\sum_{i\in[d]} e(D_j[i], \mathbf{V}_{i,j}^{(s)}) = e(C_{j-1}, V) = \sum_{i\in[d]} e(D_j^*[i], \mathbf{V}_{i,j}^{(s)})$. By the Setup-Find-Repo assumption, it follows that $D_j[i] = D_j^*[i]$ for all $i \in [d]$ and $j \in [k-1]$.

- $C_{k-1} = \langle \mathbf{T}_k, \mathbf{H}_k^{(s)} \rangle = \langle \mathbf{T}_k^*, \mathbf{H}_k^{(s)} \rangle$. By the DLog assumption, we must have $\mathbf{T}_k = \mathbf{T}_k^*$. Together with the previous item, this implies that $\pi = \pi^*$.

- $y = \langle \mathbf{T}_k, \mathbf{x}_k \rangle = \langle \mathbf{T}_k^*, \mathbf{x}_k \rangle = y^*$, contradicting that (y, π) and (y^*, π^*) are distinct.

□

Then, we can use Lemma 4.6.6 to show that zk-Spartan+zk-KZH satisfies $(r-1)$-unique response, where r denotes the total number of rounds.

Lemma 4.6.7. *zk-Spartan+zk-KZH with r rounds satisfies $(r-1)$-unique response.*

Proof. In the rth round, zk-Spartan+zk-KZH opens a KZH commitment at a random evaluation point. By Lemma 4.6.6, KZH satisfies 1-unique response, and thus the opening proof is unique. Therefore, zk-Spartan+zk-KZH satisfies $(r-1)$-unique response. □

Lemmas 4.3.1, 4.6.4, 4.6.5, and 4.6.7 imply that zk-Spartan+zk-KZH is SE.

Theorem 4.6.8. *zk-Spartan+zk-KZH satisfies simulation extractability.*

Spartan+KZH-Fold PCD is built from the zk-Spartan+zk-KZH argument and the KZH-Fold accumulator. SE of zk-Spartan+zk-KZH has been shown above. Knowledge soundness of KZH-Fold was shown in [93]. Since their extractor only rewinds the adversary and does not rely on the AGM, their proof extends to the updatable-SRS setting.

Lemma 4.6.9. *[93, Theorem 6] The KZH-Fold accumulation scheme satisfies knowledge soundness in the updatable-SRS setting.*

Lemmas 4.5.2 and 4.6.9 and Theorem 4.6.8 imply that Spartan+KZH-Fold is SE.

Theorem 4.6.10. *Leaf-decoupled Spartan+KZH-Fold satisfies leaf simulation extractability.*

4.7 Implementation and Evaluation

Our implementation of WILLOWFOLD builds on arkworks [10], KZH-Fold[1] and Willow[2] codebases. We instantiate KZH-Fold with KZH-3. Our codebase will be publicly available soon.

Experimental Setup. We use a high-performance Google Cloud machine (c4-highcpu-288) as the server, with 576 GB of RAM and 288 vCPUs. For the client and committee members, we use a MacBook Air M3 with an 8-core CPU and 8 GB of RAM. In our experiments, the server requires at most 20 GB of RAM to perform aggregation.

Lattice Parameter Selection. Since we employ the same lattice-based schemes, AHE and KAHE, as Willow, our choice of lattice parameters are identical to Willow. For further details, we refer the reader to Section 7 and Table 2 of Willow [17].

4.7.1 Optimizations

In this section, we discuss several optimization techniques to improve the performance.

Smallness Test. For smallness tests of the form $x \leq L$ with a bound $L \ll |\mathbb{F}|$, we proceed as follows. First, we embed the bit-decomposition of L in the circuit to be used across all tests. For each x, the prover gives its bit-decomposition $b_0, b_1, \ldots, b_{\lceil \log L \rceil - 1}$. The circuit checks $b_j \cdot (1 - b_j) = 0$ for all j and that $x = \sum_j b_j 2^j$, where the powers of two are hardcoded in the circuit. By a lexicographic comparison of the bit-decompositions of x and L, we can verify the test. This technique requires a $\lceil \log L \rceil$-sized witness, while the generic arkworks smallness test has a $\log |\mathbb{F}|$-sized witness.

Efficient Non-Native Arithmetic. We use a power-of-two cyclotomic ring $R = \mathbb{W}[X]/(X^N + 1)$

[1] https://github.com/h-hafezi/kzh_fold
[2] https://github.com/google/secure-aggregation

in our AHE scheme. Elements of R are polynomials with coefficients in a finite field $\mathbb{W}$, which differs from the field $\mathbb{F}$ used by our KZH-Fold circuit (e.g., the scalar field of BN254). The field $\mathbb{W}$ is much smaller, typically of size 2^{64}–2^{96}. Since elements of $\mathbb{W}$ are represented in $\mathbb{F}$, the circuit includes range checks that values are less than $|\mathbb{W}|$. To enforce modular correctness, each modular operation $a \star b = c$ is expressed as $a \star b - c = |\mathbb{W}| \cdot q$, with $a, b, c < |\mathbb{W}|$.

The main operation in the circuit is a ring multiplication, expressed as a polynomial product $u(r) \cdot v(r) = w(r)$, where u, v, w have coefficients in $\mathbb{W}$ and degree n. Thus, we design an efficient interpolation method for *non-native* polynomials. As the first step, we compute powers $\{r^i\}_{i\in[n]}$ recursively using $r^i = r^{i-1} \cdot r + |\mathbb{W}| \cdot m_i$, where m_i is the quotient from division. Since r^{i-1} is already known to be small from the previous step, only r^i must be checked to be less than $|\mathbb{W}|$. We then evaluate $u(r) = \sum u_i r^i$ as an inner product between $(1, r, \ldots, r^n)$ and $(u_0, u_1, \ldots, u_n)$. We make two observations that boost our efficiency. Let 2^ℓ be the smallest power of two greater than $|\mathbb{W}|$; it suffices to prove $r^i, u_i < 2^\ell$. Since $n \cdot |\mathbb{W}| < n \cdot 2^\ell < |\mathbb{F}|$, we can safely compute the inner product (or polynomial evaluation) over $\mathbb{F}$ since no overflow would happen, and then perform a single modular reduction by $\mathbb{W}$ at the end. Importantly, checking a value is less than 2^ℓ is much more efficient in the circuit than checking an arbitrary value, since it doesn't require a lexicographic comparison.

Probabilistic Verification of Ring Operations. The client circuit requires computing the product of two ring elements $\mathbf{u}, \mathbf{v} \in \mathbb{W}/(X^N + 1)$, which is performed using the *negacyclic matrix*. This is the method currently used in Willow to compute $\mathbf{v} \cdot \mathbf{u}$ within an arithmetic circuit. However, this approach is highly inefficient: with $N = 4096$, it yields a circuit with roughly 16 million constraints. We propose an alternative method that leverages the underlying polynomial

structure. Let $\mathbf{w} = \mathbf{v} \cdot \mathbf{u}$ be the product of the corresponding ring element. By evaluating the underlying polynomials of $\mathbf{v}$, $\mathbf{u}$ and $\mathbf{w}$ at a randomly chosen point r, we can check the equality probabilistically: if $\mathbf{w}(r) = \mathbf{v}(r) \cdot \mathbf{u}(r)$, then with high probability we have that $\mathbf{w} = \mathbf{v} \cdot \mathbf{u}$.[3]

This reduces the verification cost from quadratic to linear in the number of constraints. To securely obtain the randomness r, we need to hash the polynomials (ring elements) $\mathbf{v}, \mathbf{u}, \mathbf{w}$ and derive r from the output. However, hashing these polynomials directly is prohibitively expensive: even with a zk-friendly hash function such as Poseidon [85], which requires about 100 constraints per element, hashing three ring elements of length 4096 yields nearly 1.2 million constraints. To avoid this cost, we adopt the *commit-and-prove* paradigm, i.e., the circuit is provided with commitments to $\mathbf{v}$, $\mathbf{u}$ and $\mathbf{w}$ and derives the challenge r by hashing the commitments.

4.7.2 Setup and Registration Costs

Our setup (excluding DKG) comprises two main phases: (i) PCD setup, and (ii) generating client IDs and the Merkle tree over them. As shown in Table 4.2, a single server can complete the KZH-related setup in under 11 seconds for $n = 2^{23}$ clients. Note that due to the leaf decoupling mode, we require two separate KZH-Fold setups: one for the client circuit and one for the server circuit. An advantage of KZH-Fold is that a larger setup can be used to derive smaller ones—thus, we can run the client setup and then derive the server setup. Each role needs to store only a subset of the public parameters. For the Merkle tree, we use MiMC [6] as the hash function with exponent 3 and 91 rounds, which provides approximately 128-bit security.

4.7.3 Client, Server, and Decryptor Costs

Below, we discuss the costs incurred by each role in the aggregation session.

[3]To achieve 128-bit security, we check the relation at two independent random challenges, r_1 and r_2.

n	One-time Setup		Server					
	KZH-Fold	Merkle	Verify time (Per Client)	Contribution Proof Size	Aggregate (Per Client)	Proof of Uniqueness	IDs	Setup
2^{10}	2.55 s	32 ms	470 ms	38 KB	3.8 s	0.11 s	40 KB	16.3 MB
2^{17}	2.8 s	4.03 s	475 ms	38 KB	3.8 s	0.33 s	4.22 MB	16.3 MB
2^{23}	10.8 s	255 s	478 ms	38 KB	3.8 s	3.20 s	310 MB	16.3 MB

Table 4.2: Microbenchmarks for WILLOWFOLD setup and server for number of clients $n \in \{2^{10}, 2^{17}, 2^{23}\}$. We break up the setup cost into the KZH-fold PCD setup and the ID Merkle Tree generation. We measure the server's cost to verify a client contribution proof, as well as that proof size, the aggregation time per client, the time it takes to generate the proof of uniqueness, as well as the size of all ids and of the setup. Reported sizes assume compression of group elements.

(n, ℓ)	Client			Committee Member			
	Setup	Enc	Prove	Setup	PCD Verify / Size	Zerocheck Verify / Size	Decryption
$(2^{10}, 10^3)$	64.5 MB	2.3 ms	2.9 s	35 KB	910 ms 48 KB	47 ms 1.05 KB	74 ms
$(2^{10}, 10^5)$		8.2 ms					148 ms
$(2^{10}, 10^7)$		296 ms					297 ms
$(2^{17}, 10^3)$	64.5 MB	2.3 ms	2.9 s	39 KB	910 ms 48 KB	81 ms 3.0 KB	84 ms
$(2^{17}, 10^5)$		8.2 ms					148 ms
$(2^{17}, 10^7)$		296 ms					283 ms
$(2^{23}, 10^3)$	64.5 MB	2.3 ms	3.0 s	48 KB	910 ms 48 KB	104 ms 7.6 KB	84 ms
$(2^{23}, 10^5)$		8.2 ms					144 ms
$(2^{23}, 10^7)$		296 ms					324 ms

Table 4.3: Microbenchmarks of WILLOWFOLD's client and committee for input lengths $\ell \in \{10^3, 10^5, 10^7\}$ and number of clients $n \in \{2^{10}, 2^{17}, 2^{23}\}$. The values filling out multiple cells are independent of ℓ. For the client we measure the setup/proving key size, the encryption time, and the proving time. For the committee we report the setup size, the time to verify the PCD proof, the ZeroCheck proof.

Client Costs. The client's main cost is constructing a zero-knowledge proof of knowledge of the secret key used in the AHE scheme. As shown in Table 4.3, prover time increases only slightly as the number of clients grows. This is because Merkle inclusion—the only component that depends on the number of clients—accounts for only a small fraction of the overall circuit; for example, it contributes roughly 30k constraints even for one billion clients. We note a subtle limitation of our protocol: each client must store an SRS of size 64.5 MB.

Server Costs. The server performs three main tasks: (i) it aggregates the clients' contributions,

i.e., aggregates the ciphertexts and builds the PCD tree; (ii) once aggregation is complete, it proves that each client's contribution is used at most once, and, (iii) it performs the decryption.

- *Per-Client Cost.* Each client submits a PCD proof along with several ciphertexts to the server. Since ciphertext aggregation is computationally negligible, the main server workload per client involves (i) verifying the client's PCD proof and (ii) aggregating it into the ongoing PCD proof. The server circuit, i.e., internal nodes of the PCD tree, consists of 2^{16} R1CS constraints.

- *Proof of Uniqueness.* After aggregating all client contributions, the final PCD proof includes a commitment to the participation vector. The server runs a zerocheck to prove the underlying vector is binary. Concretely, this step is very efficient, taking ≈ 3 seconds for 2^{23} clients.

Committee Cost. Each committee member verifies the protocol execution to ensure correctness before acting as a decryptor. This verification cost is sublinear in the number of clients and independent of the input dimension; as shown in Table 4.3, it grows only logarithmically. Committee members are also responsible for decryption, which has the same cost as in Willow.

Comparison to Prior Work. Before comparison, we emphasize that our protocol is the first to enable single-server aggregation with dynamic clients, where the committee's work is sublinear in the number of clients, and no additional roles, such as a *verifier*, are required.

The closest work for comparison is Willow, since we build our scheme on Willow and only modify its underlying proof system. In Table 4.4, we provide a concrete comparison of Willow with WILLOWFOLD for $N = 2^{23}$ clients. Since both schemes employ similar encryption schemes and have similar decryptor overhead, we exclude the decryptor role from this comparison.

While Willow achieves lower client and server costs, its high verification overhead for committee members forces an unfavorable trade-off: either needs uses a large committee—weakening

Scheme	Client Proof Gen.	Server per Client	Total Verifier	Proof Size
Willow	0.8 s	0.2 s	460 h	15 GB
WILLOWFOLD	2.9 s	4.3 s	1.0 s	57 KB

Table 4.4: Comparison of Willow and WILLOWFOLD for aggregating $N = 2^{23}$ clients. The client's proof generation time, the server's per-client aggregation time, the total verifier time, and the total proof size are shown.

trust assumptions by requiring an honest majority among more parties—or one keeps the committee small and incurs high per-member computation overhead, which limits scalability.

Chapter 5

Single-Server Private Outsourcing of zk-SNARKs

Deployment of zk-SNARKs is hindered by a key limitation: although proof verification is fast, proof generation is expensive and often thousands of times slower than directly checking the NP relation [63]. This overhead typically stems from the need for the prover to perform a number of expensive cryptographic operations (e.g., group operations) that scale with the instance size. As a result, many applications of zk-SNARKs, such as private smart contracts [32], auditable key transparency [139], and verifiable machine learning [3], remain restricted to small instances.

One way to address this limitation is to allow provers/clients to outsource proof generation to more powerful cloud servers [106, 129, 145], an idea actively pursued by several startups in the industry [67, 81, 121, 134, 135]. For many applications, such as private transactions [21] and anonymous credentials [130], it is crucial that the servers learn nothing about the clients' secrets, which motivates the study of *private delegation* of zk-SNARKs [51, 75, 79]. As we discuss in Section 5.1, however, existing schemes fall short of the efficiency, security, and usability needed for real-world applications.

Our Contributions. We propose efficient single-server private delegation schemes that match the best features of prior work while addressing their limitations (see Table 5.1). In particular, our constructions support group-based zk-SNARKs without modification, achieve unlinkability by hiding the witness, the statement, and the final proof from the server, provide security even against a malicious server, and improve proof generation latency relative to local proving.

We focus on group-based zk-SNARKs [72, 86, 96] because they are among the most widely deployed proof systems in practice, due to their minimal verification cost; yet, their proof-generation time is often slower than that of alternative schemes. In these zk-SNARKs, the prover cost is dominated by elliptic-curve operations, which are expensive to realize in the circuits required for FHE; thus, single-server delegation of these zk-SNARKs using FHE is impractical.

Our approach to single-server delegation for group-based zk-SNARKs begins with the basic observation that the group operations in these proof systems are primarily *multi-scalar multiplications* (MSMs), which are linear operations with a simple algebraic structure. This motivates us to construct an efficient subroutine for outsourcing MSMs, allowing the expensive parts of proving (group operations) to be offloaded while the cheaper parts (field operations) are run locally. Specifically, we show the following results:

- *Encrypted Multi-Scalar Multiplication:* As a key building block, we introduce *encrypted MSM* (EMSM), a new primitive of independent interest that allows a client to privately outsource MSM computation to an untrusted server. We propose an EMSM scheme based on variants of the *learning parity with noise* (LPN) assumption. We optimize the efficiency of our scheme by instantiating the LPN assumption with *repeat-accumulate-accumulate* (RAA) codes [38]. In our scheme, the server computation matches that of a native (plaintext) MSM, while the client performs only a *constant* number of group operations. We also show how to achieve security against a malicious server with only $2\times$ overhead. We observe that our scheme supports *public linear maps* applied to the MSM inputs, which further reduces the cost in our applications.

- *Delegating Polynomial Commitments:* As an immediate application of EMSM, we show how it can be used to outsource KZG polynomial commitments [95]. Since KZG commitments are,

in particular, a key building block in zk-SNARKs and many other applications [48, 50, 71, 72, 95, 114, 126, 149], we believe that outsourcing KZG is of independent interest.

- *Server-Aided zk-SNARKs:* As our main contribution, we introduce and formalize the notion of zk-SNARKs with server-aided proving, whereby a client generates a proof with the help of an untrusted server who learns nothing about the client's witness or the final proof.

  We show how to realize server-aided proving for the widely-deployed Nova [96], Groth16 [86], and Plonk [72] zk-SNARKs. In each case, the group operations done by the prover are in the form of MSMs and can be delegated using our EMSM. Moreover, we show that some of the scalar operations performed by the client—which correspond to public linear maps applied to MSM inputs—can also be offloaded, thereby further reducing latency and the client's workload. Our technique is broadly applicable to other group-based proof systems such as Marlin [50], Hyperplonk [48], HyperNova [97], and KZH-fold [93].

- *Implementation and Evaluation:* We experimentally evaluate our EMSM scheme and its applications to delegating KZG commitments and server-aided proving for Nova and Groth16. When proving 4 million constraints using Nova and delegating from a small AWS instance (`r6i.large`) to a larger instance (`c7i.24xlarge`), our scheme lowers the client's computation by up to $\mathbf{20}\times$ and reduces the proof-generation latency by up to $\mathbf{9}\times$.

Example Applications. Server-aided proving is useful in any scenario where a lightweight client wants to outsource proof generation to powerful entities. We give several real-world examples.

- *Private transactions and smart contracts.* Generating even a simple private transaction in Zcash can take several minutes on a mobile device. It can be even slower in private blockchains,

where users prove the correct execution of an arbitrary smart contract. Server-aided proving can reduce clients' work and the latency for generating transactions, enhancing performance.

- *Verifiable and private machine learning.* zk-SNARKs can enable verification of various properties of machine-learning models, e.g., inference correctness [107] or model provenance and integrity [3]. Circuits that arithmetize these properties are typically large, and their size scales with the size of the model/dataset, which makes zk-SNARKs computationally intensive for realistic deployments. Our server-aided proving schemes can accelerate proof generation without revealing sensitive model parameters or datasets to the server.

- *Prover markets.* Hardware acceleration can substantially boost the performance of zk-SNARK provers [57, 131], yet such hardware may not be readily accessible to average users. Prover markets [134] are networks of servers with specialized hardware (e.g., GPUs or FPGAs) to which clients can delegate proof generation. Due to the prohibitive communication and/or computation costs of existing private delegation schemes, most currently deployed prover markets operate only in a non-private mode, which limits their deployability to real-world settings. Our schemes address this problem and enable efficient use of servers' resources.

- *Private identity verification.* zk-SNARKs have applications to private authentication and anonymous credentials [130], with a prominent example of zkLogin [14]. In such scenarios, users typically operate on lightweight devices such as smartphones, where proof generation is quite slow. Our schemes enable users to offload this computation to more powerful servers without revealing their credentials.

- *Image authentication.* zk-SNARKs can be used to certify the provenance of images, even after

(an approved set of) edits have been applied [59]. Despite efforts to optimize zk-SNARKs for such applications, proving time remains slow, often taking tens of minutes. Our server-aided proving schemes enable a client to offload most of its workload to a server while keeping images/edits hidden.

5.1 Related Work

Prior work on the private delegation of zk-SNARKs can be categorized into two groups: *multi-server* protocols [51, 75], which rely on secret sharing, and *single-server* protocols [79], which use fully homomorphic encryption (FHE). We discuss each in turn; see Table 5.1 for a summary comparison.

Multi-Server Delegation. One approach to privately delegating zk-SNARKs is for the client to secret-share its witness among multiple servers, which then run a secure multi-party computation (MPC) protocol to generate the proof. Boneh et al. [124] termed this framework *collaborative* zk-SNARKs and proposed concrete instantiations by applying standard MPC techniques to several zk-SNARKs, including Marlin [50], Groth16 [86], and Plonk [72]. Chiesa et al. [51] and Garg et al. [75] advanced this approach and showed that, when relying on sufficiently many servers, it can even reduce the latency of proof generation. Subsequent works further improved the efficiency and security of this framework [66, 76, 90, 108, 148]—e.g., by modifying the zk-SNARK itself.

The main limitation of this approach is the need to deploy multiple servers, with the client having to trust that a threshold of them behave honestly; if the number of corrupted servers exceeds the threshold, the witness is exposed. Another limitation is that, although these schemes keep the witness private, they reveal information about the proof to the servers. This is unac-

Scheme	# of servers	Improved latency	Unlinkability	Malicious security	Instantiations
co-SNARKs [124]	≥ 2	✗	✗	✓	Groth16; Marlin; Plonk; Fractal
EOS [51]	≥ 2	✓	✗	✗	Marlin
ZKSaaS [75]	≥ 8	✓	✗	✗	Groth16; Plonk; Marlin
DFS [90]	≥ 2	✓	✗	✓	Marlin*
Siniel [148]	≥ 2	✓	✗	✓	Marlin
Liu et al. [108]	≥ 8	✓	✗	✓	HyperPlonk
Fang et al. [66]	≥ 2	✓	✗	✓	Plonk*
Blind FRI [73, 79]	1	✗	✗	✗	Fractal*
This paper	1	✓	✓	✓	Nova; Groth16; Plonk

* Delegation-specific adaptations.

Table 5.1: Private delegation schemes. The number of servers assumes the scheme tolerates at least one corrupted server. Although EOS claims malicious security, an attack is known [90].

ceptable for applications in which the client must remain unlinkable to the proof, such as private transactions [21]. Although such linkability can be mitigated by assuming anonymous communication between the client and the servers [78], implementing such communication channels may be infeasible or expensive.

Single-Server Delegation. Alternatively, one can delegate proof generation to a single untrusted server; this avoids the need to deploy multiple servers and reduces the required trust assumptions. While single-server private delegation can be generically realized using FHE, doing so directly does not yield a practical scheme due to the associated overhead [69]. Garg et al. [79] reduce this cost for FRI-based zk-SNARKs [20] (e.g., Fractal [52]) by minimizing the non-black-box use of the underlying primitives. Follow-up work [9, 73] showed further optimizations, but even in the best existing schemes, server computation remains orders of magnitude slower than native proving runtime, thereby *increasing* the overall latency of proof generation.

Besides the overhead introduced by FHE, existing single-server schemes have a number of other drawbacks. The most prominent is that they alter proof verification; that is, the proofs they

generate are not compatible with existing zk-SNARK verifiers, and verification time increases due to the need to verify FHE decryption. Furthermore, existing single-server schemes support only FRI-based zk-SNARKs and not popular group-based zk-SNARKs such as Groth16 or Plonk. As with multi-server schemes, existing single-server schemes reveal the statement and the final proof to the server, and preventing this would be prohibitive. Finally, current single-server schemes consider only a semi-honest server, and a malicious server could use a selective-failure attack to learn information about the witness based on whether the client aborts.

5.2 Additional Preliminaries

For $n > 0$, we let $\omega \in \mathbb{F}_q^*$ be a primitive nth root of unity if $\omega^n = 1$ and $\omega^m \neq 1$ for $1 \leq m < n$. Then, we define the multiplicative subgroup $D_\omega := \{\omega^0, \omega^1, \ldots, \omega^{n-1}\}$.

Fast Fourier Transform and Polynomial Operations. Fast Fourier Transform (FFT) enables the evaluation of a degree-n polynomial at $O(n)$ points—typically the points over the domain D_ω for a primitive nth root of unity ω—in time $O(n \log n)$. Its inverse operation, the inverse FFT (iFFT), reconstructs the polynomial from its value at those points in time $O(n \log n)$. Note FFT and iFFT are both linear operators. One application of these transforms is multiplication of two degree-n polynomials in time $O(n \log n)$. In particular, given the coefficients of two polynomials, the algorithm first applies an FFT to each polynomial to obtain their evaluations at $2n - 1$ points, multiplies the evaluations pointwise, and then applies an iFFT operation to recover the result. This also extends to polynomial division; any division is reduced to two multiplications [25].

The Product-Check Protocol. The *product-check* protocol enables a prover to convince a verifier that for a polynomial $f(X)$, $\prod_{i=0}^{n-1} f(\omega^i) = \gamma$ where ω is a primitive nth root of unity. Let

$t(X)$ be a polynomial with $t(\omega^s) = \prod_{i\in[s-1]} f(\omega^i)$ for all $1 \leq s \leq n$. In the product-check protocol, the prover sends $t(X)$ as an oracle, and the verifier checks whether $t(\omega^{n-1}) = \gamma$. The prover shows that $t(X)$ is well-formed, i.e., $t(\omega \cdot x) - t(x) \cdot f(\omega \cdot x) = 0$ for all $x \in D_\omega$. To do this, the prover computes a quotient $q(X)$ by dividing $t(\omega \cdot x) - t(x) \cdot f(\omega \cdot x)$ by $X^n - 1$ and sending it as an oracle. The verifier opens the polynomials at random points and checks for their consistency. Oracles can be instantiated with polynomial commitments.

5.2.1 Arithmetizations

We recall two standard arithmetizations used by zk-SNARK protocols.

R1CS Circuit Format. The *rank-1 constraint system* (R1CS) is a widely used representation of arithmetic circuits [50, 86]. An R1CS with n constraints is defined by matrices $\mathbf{A}, \mathbf{B}, \mathbf{C} \in \mathbb{F}_q^{n\times m}$, each with $\Theta(1)$ non-zero entries per row. Then, we say a constraint system is satisfied by a given public instance $\mathbf{x} \in \mathbb{F}_q^\ell$ and a private witness $\mathbf{w} \in \mathbb{F}_q^{m-\ell}$ if $(\mathbf{Az}) \odot (\mathbf{Bz}) = \mathbf{Cz}$, where $\mathbf{z} := \mathbf{x} \,\|\, \mathbf{w}$.

Plonk Arithmetization. Given integers $m, n \in \mathbb{N}$, the Plonk arithmetization for a fan-in-two circuit with unlimited fan-out, m wires, and n arithmetic gates is defined by two tuples: (i) $(\mathbf{a}, \mathbf{b}, \mathbf{c}) \in [m]^{3n}$, where $\mathbf{a}$, $\mathbf{b}$, and $\mathbf{c}$ denote the left, right, and output sequences, respectively, and (ii) $(\mathbf{q}_L, \mathbf{q}_R, \mathbf{q}_O, \mathbf{q}_M, \mathbf{q}_C) \in \mathbb{F}^{5n}$, where $\mathbf{q}_L$, $\mathbf{q}_R$, $\mathbf{q}_O$, $\mathbf{q}_M$, and $\mathbf{q}_C$ denote the left, right, output, multiplication, and constant selector vectors, respectively. Given a circuit assignment $\mathbf{z} := (\mathbf{x} \,\|\, \mathbf{w}) \in \mathbb{F}_q^m$, we say the circuit is satisfied if for all $i \in [n]$:

$$(q_L)_i \cdot z_{a_i} + (q_R)_i \cdot z_{b_i} + (q_O)_i \cdot z_{c_i} + (q_M)_i \cdot (z_{a_i} z_{b_i}) + (q_C)_i = 0.$$

The wire values must also be correctly ordered. Let $\mathbf{v} := (z_{a_i})_{i\in[n]} \,\|\, (z_{b_i})_{i\in[n]} \,\|\, (z_{c_i})_{i\in[n]}$. Plonk

enforces this by defining a permutation $\sigma : [3n] \to [3n]$ such that $v_i = v_{\sigma(i)}$ for all $i \in [3n]$.

5.2.2 Unrolled Protocols of zk-SNARKs

We recall the designs of Groth16 [86], Nova [96], and Plonk [72].

The Nova Protocol. The key building block in Nova is a folding scheme for the *committed, relaxed* R1CS—a generalization of R1CS where, in addition to the matrices $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$, the relation includes a private error vector $\mathbf{e} \in \mathbb{F}_q^n$ and a public scalar $u \in \mathbb{F}_q$. Given $\mathbf{z} = \mathbf{x} \| \mathbf{w}$, the relation is satisfied if and only if $(\mathbf{Az}) \odot (\mathbf{Bz}) = u \cdot (\mathbf{Cz}) + \mathbf{e}$. Here, the instance includes commitments to vectors $\mathbf{w}$ and $\mathbf{e}$. Given fixed matrices $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$, consider two instances $(\bar{\mathbf{e}}_0, u_0, \bar{\mathbf{w}}_0, \mathbf{x}_0)$ and $(\bar{\mathbf{e}}_1, u_1, \bar{\mathbf{w}}_1, \mathbf{x}_1)$ where, for each $b \in \{0, 1\}$, $\bar{\mathbf{e}}_b$, and $\bar{\mathbf{w}}_b$ are Pedersen commitments to $\mathbf{e}_b$ and $\mathbf{w}_b$. For folding, the prover computes $\mathbf{t} = (\mathbf{Az}_0) \odot (\mathbf{Bz}_1) + (\mathbf{Az}_1) \odot (\mathbf{Bz}_0) - u_0 \cdot (\mathbf{Cz}_1) - u_1 \cdot (\mathbf{Cz}_0)$, and sends a Pedersen commitment to $\mathbf{t}$—denoted by $\bar{\mathbf{t}}$—to the verifier. Then, given a random challenge $r \leftarrow \mathbb{F}_q$, we define the folded instance $(\bar{\mathbf{e}}, u, \bar{\mathbf{w}}, \mathbf{x})$ as

$$u = u_0 + r \cdot u_1,\ \bar{\mathbf{e}} = \bar{\mathbf{e}}_0 + r \cdot \bar{\mathbf{t}} + r^2 \cdot \bar{\mathbf{e}}_1, \mathbf{x} = \mathbf{x}_0 + r \cdot \mathbf{x}_1,\ \bar{\mathbf{w}} = \bar{\mathbf{w}}_0 + r \cdot \bar{\mathbf{w}}_1.$$

The prover computes the updated witness $(\mathbf{e}, \mathbf{w})$ where $\mathbf{e} = \mathbf{e}_0 + r \cdot \mathbf{t} + r^2 \cdot \mathbf{e}_1$ and $\mathbf{w} = \mathbf{w}_0 + r \cdot \mathbf{w}_1$. Folding is correct since the Pedersen commitment is additively homomorphic; it is sound since if any input instance is invalid, the folded instance is unsatisfied over the choice of r. Let F be the function of each iteration, and F' be an augmented function that, in addition to computing F, runs the folding verifier; i.e., given two instances and a cross-term commitment, it folds the instances. Then, the setup and proving algorithms in Nova are as follows:

- Setup: it outputs the Pedersen commitment scheme parameters.

- Prove: it proceeds in two steps: (i) first, for each invocation of F', extracts the instance–witness pair and incrementally folds them into a single instance–witness pair; and (ii) second, it applies a general-purpose zk-SNARK to this final instance.

The Groth16 Protocol. Let $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$ be matrices defining an R1CS relation to a given relation. Let $\omega \in \mathbb{F}_q$ be a $2n$th primitive root of unity and $\omega' := \omega^2$. We define $f_{A,j}(X), f_{B,j}(X), f_{C,j}(X)$:

$$f_{A,j}(\omega'^i) := A_{i,j},\ f_{B,j}(\omega'^i) := B_{i,j},\ f_{C,j}(\omega'^i) := C_{i,j},$$

$\forall i \in [n], j \in [m]$. It is well-known [80] that R1CS with $\mathbf{z} := \mathbf{x} \| \mathbf{w}$ is satisfied iff for all $i \in [n]$

$$\sum_{j\in[m]} z_j f_{A,j}(\omega'^i) \cdot \sum_{j\in[m]} z_j f_{B,j}(\omega'^i) = \sum_{j\in[m]} z_j f_{C,j}(\omega'^i).$$

Let $t(X) := \prod_{i\in[n]}(X - w'_i)$. The above condition holds iff there is a quotient $q(X)$ such that

$$\sum_{j\in[m]} z_j f_{A,j}(X) \cdot \sum_{j\in[m]} z_j f_{B,j}(X) - \sum_{j\in[m]} z_j f_{C,j}(X) = q(X) \cdot t(X).$$

The prover shows that $q(X)$ exists. The setup and proving algorithms of Groth16 are as follows:

- Setup: Let $\alpha, \beta, \gamma, \delta, v \leftarrow \mathbb{F}_q$ and $f_j := \alpha f_{A,j} + \beta f_{B,j} + f_{C,j}$. Public parameters pp include

$$\{\alpha, \beta, \delta, \{v^i\}_{i\in[n]}, \{\gamma^{-1} f_j(v)\}_{j\in[\ell]}, \{\delta^{-1} f_j(v)\}_{j=\ell}^{m-1}, \{\delta^{-1} v^i t(v)\}_{j\in[n-1]},$$

$$\{f_{A,j}(v)\}_{j\in[m]}, \{f_{B,j}(v)\}_{j\in[m]}\} \cdot g_1, \{\beta, \gamma, \delta, \{v^i\}_{i\in[n]}, \{f_{B,j}(v)\}_{j\in[m]}\} \cdot g_2, \alpha\beta \cdot g_T,$$

  where $g_1 \in \mathbb{G}_1$, $g_2 \in \mathbb{G}_2$, and $g_T \in \mathbb{G}_T$ are generators of the source and target groups.

- Prove: Consider the following polynomials:

$$f_A(X) := \sum_{j\in[m]} z_j f_{A,j}(X), f_B(X) := \sum_{j\in[m]} z_j f_{B,j}(X), f_C(X) := \sum_{j\in[m]} z_j f_{C,j}(X).$$

The prover can compute $q(X)$ within the following steps:

1. Over the domain $D_{\omega'}$, $f_A(X)$, $f_B(X)$, and $f_C(X)$ are evaluated to $\mathbf{Az}$, $\mathbf{Bz}$, and $\mathbf{Cz}$. The prover computes these evaluations in $O(n)$ time, as the matrices $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$ are sparse.
2. Let $D := D_\omega \setminus D_{\omega'}$. The prover applies iFFT to get the coefficients of $f_A(X)$, $f_B(X)$, and $f_C(X)$, and applies FFT to evaluate them over D. The prover also evaluates $t(X)$ over D.
3. The prover evaluates $q(X)$ over the domain D using the evaluations of $f_A(X)$, $f_B(X)$, $f_C(X)$, and $t(X)$, and then applies iFFT to obtain the coefficient representation of $q(X)$.

Let $q(X) = \sum_{i\in[n-1]} q_i X^i$. The prover samples $r, s \leftarrow \mathbb{F}_q$ and outputs $\pi := (\pi_a, \pi_b, \pi_c)$ where

$$\pi_a := \alpha \cdot g_1 + r \cdot (\delta \cdot g_1) + \sum_{j\in[m]} z_j \cdot (f_{A,j}(v) \cdot g_1),$$

$$\pi_b := \beta \cdot g_2 + s \cdot (\delta \cdot g_2) + \sum_{j\in[m]} z_j \cdot (f_{B,j}(v) \cdot g_2),$$

$$\pi_c := s \cdot \pi_a + r \cdot (\beta \cdot g_1 + \sum_{j\in[m]} z_j \cdot (f_{B,j}(v) \cdot g_1))$$

$$+ \sum_{j\in[\ell:m]} z_j \cdot (\delta^{-1} f_j(v) \cdot g_1) + \sum_{i\in[n-1]} q_i \cdot (\delta^{-1} v^i t(v) \cdot g_1).$$

The Plonk Protocol. Plonk proceeds in five rounds, which we outline below:

1. The prover computes three polynomials $a(X)$, $b(X)$, and $c(X)$, each of degree $n-1$, corresponding to the left, right, and output wire values, respectively. In particular, let ω be a

primitive n-th root of unity. Then, we have $a(\omega^i) = z_{a_i}$, $b(\omega^i) = z_{b_i}$, and $c(\omega^i) = z_{c_i}$ for all $i \in [n]$. These polynomials are obtained by applying an iFFT to the corresponding wire values. The prover sends KZG commitments to $a(X)$, $b(X)$, and $c(X)$.

2. Let $\kappa, \eta \in \mathbb{F}_q^*$ with distinct domains D_ω, $D_{\kappa\omega}$, and $D_{\eta\omega}$. For random $\beta, \tau \in \mathbb{F}_q$, let

$$h(X) := (a(X) + \beta X + \tau)(b(X) + \beta\kappa X + \tau)(c(X) + \beta\eta X + \tau),$$
$$g(X) := (a(X) + \beta\sigma_a(X) + \tau)(b(X) + \beta\kappa\sigma_b(X) + \tau)(c(X) + \beta\eta\sigma_c(X) + \tau),$$

where $\sigma_a(X)$, $\sigma_b(X)$, and $\sigma_c(X)$ are defined as $\sigma_a(w^i) := w^{\sigma(i)}$, $\sigma_b(w^i) := w^{\sigma(n+i)}$, and $\sigma_c(w^i) := w^{\sigma(2n+i)}$ for all $i \in [n]$. Then, the wiring is consistent with the permutation σ iff

$$\prod_{j \in [n]} \frac{h(\omega^j)}{g(\omega^j)} = \prod_{j \in [n]} f(\omega^j) = 1,$$

which can be validated via a product-check protocol, i.e., the prover sends a KZG commitment to a polynomial $t(X)$ such that for all $s \in [n]$, $t(w^s) = \prod_{i \in [s]} \frac{h(\omega^i)}{g(\omega^i)}$.

3. Define $q_L(X)$, $q_R(X)$, $q_O(X)$, $q_M(X)$, and $q_C(X)$ such that each evaluates to the corresponding selector over the domain D_ω. We let $p_0(X)$, $p_1(X)$, and $p_2(X)$ be as follows:

$$p_0(X) = q_L(X) \cdot a(X) + q_R(X) \cdot b(X) + q_O(X) \cdot c(X) + q_M(X) \cdot a(X) \cdot b(X) + q_C(x),$$
$$p_1(X) = t(\omega \cdot X) - t(X) \cdot f(X), \quad p_2(X) = \ell_n(X) \cdot (t(X) - 1),$$

where $\ell_n(X)$ denotes the nth Lagrange basis over D_ω. All three polynomials must evaluate to zero over the domain D_ω. For random ρ, let $p(X) := p_0(X) + \rho p_1(X) + \rho^2 p_2(X)$.

4. The prover computes a quotient $q(X)$ by dividing $p(X)$ by $X^n - 1$ and sends a commitment to $q(X)$; that is, the prover decomposes $q(X)$ into degree-n polynomials $q_{\mathsf{low}}(X)$, $q_{\mathsf{mid}}(X)$, and $q_{\mathsf{high}}(X)$ with $q(X) = q_{\mathsf{low}}(X) + X^n \cdot q_{\mathsf{mid}}(X) + X^{2n} \cdot q_{\mathsf{high}}(X)$, and commits to each.

5. Once all the commitments are received, the verifier queries the committed polynomials at random points, and the prover provides KZG openings for each of them.

5.2.3 Repeat-Accumulate-Accumulate Codes

Repeat-accumulate-accumulate (RAA) is a family of error-correcting codes with efficient encoding [38]. For a block length N, dimension n, and $r = (N/n) \in \mathbb{N}$, we define the following:

- $\mathbf{F}_r \in \mathbb{F}^{n \times N}$ is a *repeat* matrix that repeats each entry in the input vector r times; that is, $(F_r)_{i,j} = 1$ if $ir \leq j < (i+1)r$ and $(F_r)_{i,j} = 0$ otherwise.

- $\mathbf{A} \in \mathbb{F}^{N \times N}$ is an upper-triangular *accumulator* matrix with all nonzero entries equal to 1; that is, $(A_r)_{i,j} = 1$ if $i \leq j$, and $(A_r)_{i,j} = 0$ otherwise.

- For $\sigma : [N] \to [N]$, a permutation, $\mathbf{M}_\sigma \in \mathbb{F}^{N \times N}$ denotes the corresponding *permutation* matrix; that is, $(M_\sigma)_{i,j} = 1$ if $\sigma(j) = i$, and $(M_\sigma)_{i,j} = 0$ otherwise.

Then, a pair of permutations $\sigma_1, \sigma_2 : [N] \to [N]$ defines a generator as $\mathbf{G} := \mathbf{F}_r \cdot \mathbf{M}_{\sigma_1} \cdot \mathbf{A} \cdot \mathbf{M}_{\sigma_2} \cdot \mathbf{A}$.

5.2.4 The Learning Parity with Noise Assumption

At a high level, the *learning parity with noise* (LPN) problem [28] over a field $\mathbb{F}$ states that for a public matrix $\mathbf{A} \in \mathbb{F}^{N \times m}$, a uniform secret $\mathbf{s} \in \mathbb{F}^m$, and a *sparse* vector $\mathbf{e} \in \mathbb{F}^N$ from some error distribution, $(\mathbf{A},\ \mathbf{As} + \mathbf{e})$ is computationally indistinguishable from $(\mathbf{A},\ \mathbf{b})$ with $\mathbf{b} \leftarrow \mathbb{F}^N$ uniform. The *dual-LPN* assumption is an equivalent formulation [117, Lemma 4.9] stating that

for a public $\mathbf{G} \in \mathbb{F}^{n \times N}$ chosen from a distribution $\mathcal{G}$ and a sparse noise vector $\mathbf{e} \in \mathbb{F}^N$ chosen from a distribution $\mathcal{E}$, it holds that $(\mathbf{G}, \mathbf{Ge})$ is indistinguishable from $(\mathbf{G}, \mathbf{b})$ where $\mathbf{b} \leftarrow \mathbb{F}^n$ is uniform.

For dual-LPN, various choices for the distributions $\mathcal{G}$ and $\mathcal{E}$ have been considered. A standard choice for the noise distribution $\mathcal{E}$ is the uniform distribution over vectors in $\mathbb{F}^N$ of Hamming weight t, where $t = O(1)$; see [105] for a detailed security analysis. Recent work [24, 34, 35, 37, 127] has instantiated $\mathcal{G}$ with the uniform distribution over generator matrices of linear codes with fast encoding. It is believed that dual-LPN holds under such instantiations when certain conditions hold, as we discuss next.

Security against linear tests. For a distribution $\mathcal{D}$ over $\mathbb{F}^n$ and a nonzero vector $\mathbf{v} \in \mathbb{F}^n$, the *bias* of $\mathcal{D}$ with respect to $\mathbf{v}$ is

$$\mathsf{bias}_{\mathbf{v}}(\mathcal{D}) := \max_{c \in \mathbb{F}} \left| \Pr_{\mathbf{x} \leftarrow \mathcal{D}}[\mathbf{v}^T \cdot \mathbf{x} = c] - \tfrac{1}{|\mathbb{F}|} \right|.$$

For a generator matrix $\mathbf{G} \in \mathbb{F}^{n \times N}$ and a noise distribution $\mathcal{E}$ over $\mathbb{F}^N$, let $\mathcal{E}_{\mathbf{G}}$ denote the distribution of $\mathbf{Ge} \in \mathbb{F}^n$ when $\mathbf{e}$ is sampled from $\mathcal{E}$. All known attacks against LPN and its variants [18, 141, 151] involve finding a vector $\mathbf{v} \in \mathbb{F}^n$ for which the bias of $\mathcal{E}_{\mathbf{G}}$ with respect to $\mathbf{v}$ is large. Thus, to thwart known attacks, we must ensure that (with high probability over choice of $\mathbf{G}$) no such $\mathbf{v}$ exists. Prior work [35] shows that this can be done by ensuring that, for some δ, (i) the code defined by $\mathbf{G}$ has relative distance at least δ and (ii) $\epsilon = \max_{\mathsf{wt}(\mathbf{v}) \geq \delta N} \mathsf{bias}_{\mathbf{v}}(\mathcal{E}_{\mathbf{G}})$ is small. If $\mathcal{E}$ is the uniform distribution over vectors of Hamming weight t, we have $\epsilon < e^{-2\delta t}$ [35];

in that case, we achieve λ-bit security of dual-LPN against known attacks if

$$e^{-2\delta t} \leq 2^{-\lambda+\log_2 N} \iff \delta \geq \frac{\ln 2 \cdot (\lambda - \log_2 N)}{2t}. \tag{5.1}$$

We choose a distribution $\mathcal{G}$ such that with overwhelming probability over choice of $\mathbf{G}$ from $\mathcal{G}$, the code defined by $\mathbf{G}$ has relative distance δ.

There are three cases that do not directly fit into the linear-test framework [35]: when the code admits an efficient decoding algorithm, yielding a trivial attack on the dual-LPN problem; when the noise is highly structured and the adversary is given sufficiently many samples for a given noise pattern [11]; and when the field $\mathbb{F}$ has nontrivial subfields. We make sure to avoid these cases.

Instantiation with RAA codes. In this work, we take $\mathcal{G}$ as a distribution over generators of an RAA code, which is conjectured to be safe for dual-LPN [24]. To provide further evidence for this claim, we analyze the minimum distance of RAA codes over arbitrary finite fields (see Section 5.3.1) and then use the linear-test framework to set parameters; see Section 5.3.2.

5.3 Concrete Security of Dual-LPN

In this section, we analyze the security of dual-LPN instantiated with RAA codes. Specifically, we first analyze the distance of the RAA codes, and then use the linear-test framework to choose the dual-LPN parameters.

5.3.1 Distance Analysis of RAA Codes

Prior work [38] provides a provable analysis of the distance of RAA codes, but its approach applies only to binary fields. Here, we employ a technique for analyzing linear codes [27] and provide a numerical analysis over fields $\mathbb{F}_q$ for $q \geq 3$. Specifically, we aim to derive an upper bound on the probability that the relative distance of a random RAA code of a given dimension falls below 0.05; this is known as the *failure probability*.

Let $\mathbf{G} \in \mathbb{F}^{n \times N}$ be a generator with $\mathbf{G} = \mathbf{F}_r \mathbf{M}_{\sigma_1} \mathbf{A} \mathbf{M}_{\sigma_2} \mathbf{A}$, where $\mathbf{F}_r$ is an r-repetition matrix for $r = N/n$; $\mathbf{M}_{\sigma_1}$ and $\mathbf{M}_{\sigma_2}$ are permutation matrices for permutations σ_1, σ_2 over $[N]$; and $\mathbf{A}$ is the accumulation matrix. Analyzing the distance of the code for a distance δ amounts to bounding $\Pr[\exists x \in \mathbb{F}^n \setminus \{0\} : \mathsf{wt}(\mathbf{G}^T\mathbf{x}) \leq \delta N]$. We recall the following two lemmas [27].

Lemma 5.3.1. *For any* $\mathbf{x} \in \mathbb{F}^N$*, it holds that* $\mathsf{wt}(\mathbf{A}^T\mathbf{x}) \geq \lceil \mathsf{wt}(\mathbf{x})/2 \rceil$.

Lemma 5.3.2. *For a uniform* $\mathbf{x} \in \mathbb{F}^N$ *of weight* $w \neq 0$ *and any* $h \geq \lceil w/2 \rceil$*,*

$$\Pr[\mathsf{wt}(\mathbf{A}^T\mathbf{x}) = h] \leq \frac{\sum_{i=0}^{w-1} \binom{N-h}{\lfloor \frac{w-i}{2} \rfloor} \binom{h-1}{\lceil \frac{w-i}{2} \rceil - 1} \binom{h - \lceil \frac{w-i}{2} \rceil}{i}}{\binom{N}{w}}.$$

We then apply a union bound on the failure probability:

$$\begin{aligned}\Pr[\exists x \in \mathbb{F}^n \setminus \{0\} : \mathsf{wt}(\mathbf{G}^T\mathbf{x}) \leq \delta N] \leq \sum_{w_1=1}^{n} \binom{n}{w_1} \\ \cdot \sum_{w_2=1}^{N} \Pr[\mathsf{wt}((\mathbf{F}_r \mathbf{M}_{\sigma_1} \mathbf{A})^T\mathbf{x}) = w_2 | \mathsf{wt}(\mathbf{x}) = w_1] \\ \cdot \sum_{w_3=1}^{\delta N} \Pr[\mathsf{wt}((\mathbf{M}_{\sigma_2} \mathbf{A})^T\mathbf{x}) = w_3 | \mathsf{wt}(\mathbf{x}) = w_2].\end{aligned}$$

Note that $\mathbf{M}_{\sigma_1}$ and $\mathbf{M}_{\sigma_2}$ keep the input/output weights unchanged. For the repetition matrix, it

holds that $\mathsf{wt}(\mathbf{F}_r^T\mathbf{x}) = r \cdot \mathsf{wt}(\mathbf{x})$. Moreover, following Lemma 5.3.1, we know that $w_2 \geq \lceil rw_1/2 \rceil$ and $w_3 \geq \lceil w_2/2 \rceil$. With these observations, we can simplify the above upper bound as follows:

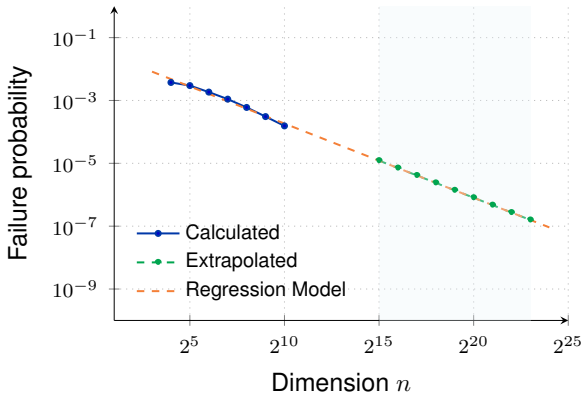
$$\Pr[\exists x \in \mathbb{F}^n \setminus \{0\} : \mathsf{wt}(\mathbf{G}^T\mathbf{x}) \leq \delta N] \leq \sum_{w_1=1}^{n} \binom{n}{w_1}$$
$$\cdot \sum_{w_2=\lceil rw_1/2 \rceil}^{2\delta N} \Pr[\mathsf{wt}(\mathbf{A}^T\mathbf{x}) = w_2 | \mathsf{wt}(\mathbf{x}) = rw_1]$$
$$\cdot \sum_{w_3=\lceil w_2/2 \rceil}^{\delta N} \Pr[\mathsf{wt}(\mathbf{A}^T\mathbf{x}) = w_3 | \mathsf{wt}(\mathbf{x}) = w_2].$$

Then, we apply Lemma 5.3.2 to obtain the following upper bound:

$$\Pr[\exists x \in \mathbb{F}^n \setminus \{0\} : \mathsf{wt}(\mathbf{G}^T\mathbf{x}) \leq \delta N] \leq \sum_{w_1=1}^{n} \binom{n}{w_1}$$
$$\cdot \sum_{w_2=\lceil rw_1/2 \rceil}^{2\delta N} \frac{\sum_{i=0}^{rw_1-1} \binom{N-w_2}{\lfloor \frac{rw_1-i}{2} \rfloor} \binom{w_2-1}{\lceil \frac{rw_1-i}{2} \rceil - 1} \binom{w_2 - \lceil \frac{rw_1-i}{2} \rceil}{i}}{\binom{N}{rw_1}}$$
$$\cdot \sum_{w_3=\lceil w_2/2 \rceil}^{\delta N} \frac{\sum_{i=0}^{w_2-1} \binom{N-w_3}{\lfloor \frac{w_2-i}{2} \rfloor} \binom{w_3-1}{\lceil \frac{w_2-i}{2} \rceil - 1} \binom{w_3 - \lceil \frac{w_2-i}{2} \rceil}{i}}{\binom{N}{w_2}}.$$

To minimize t, we fix the code rate to $R = 1/4$ (thus $N = 4n$) and set the relative distance to $\delta = 0.05$. Calculating the bound for dimensions n of interest is infeasible, thus we evaluate it for $n = 2^4, 2^5, \ldots, 2^{10}$ and extrapolate to larger dimensions using exponential regression. (See Figure 5.1.) Our heuristic analysis shows that, for our parameter choices, the failure probability is at most 2^{-21} for $n \geq 2^{15}$.

Prior work [38] uses a procedure for sampling a generator of an RAA code that lowers the failure probability exponentially in a parameter w at the cost of $O(N^w \log(N))$. We do not employ this procedure in our analysis; however, the same technique can be used in our case.



Dimension n	2^{15}	2^{16}	2^{17}	2^{18}	2^{19}
Noise weight t	589	582	575	568	561

Dimension n	2^{20}	2^{21}	2^{22}	2^{23}	2^{24}
Noise weight t	554	547	540	533	526

Figure 5.1: Failure probability of RAA codes and noise weight t as a function of the dimension n to achieve 100-bit security when dual-LPN is instantiated with RAA codes for rate $R = 0.25$ and relative distance $\delta = 0.05$. The shaded region marks the extrapolation domain.

5.3.2 Parameter Selection for Dual-LPN

As in prior work [34,35,56,105], we employ the *linear test* framework (specifically, Equation 5.1) to choose LPN parameters. We set $\lambda = 100$ to match the security provided by the BN254 curve used in our zk-SNARK implementations, but as can be observed from Equation (5.1) the value of t would not change much if instead we used $\lambda = 128$. We tabulate the noise weight as a function of the input dimension in the Table of Figure 5.1.

5.4 Encrypted Multi-Scalar Multiplication

In this section, we first define the notion of *encrypted multi-scalar multiplication* (EMSM). We then propose an efficient construction based on the dual-LPN assumption. Finally, we explore the applications of the EMSM primitive to the delegation of KZG commitments.

5.4.1 Definition

Fix a public $\mathbf{g} \in \mathbb{G}^n$. Then, an EMSM scheme with respect to $\mathbf{g}$ follows the syntax below:

- $\mathsf{Setup}(1^\lambda, \mathbf{g}) \to \mathsf{pp}$: a setup algorithm that, given λ and $\mathbf{g} \in \mathbb{G}^n$, outputs public parameters pp.
- $\mathsf{Encrypt}(\mathsf{pp}, \mathbf{z}) \to (\mathsf{ct}, \mathsf{st})$: an encryption algorithm that, given the public parameters pp and the input vector $\mathbf{z} \in \mathbb{F}_q^m$, outputs a ciphertext ct and client secret state st.
- $\mathsf{Evaluate}(\mathsf{pp}, \mathsf{ct}) \to \mathsf{em}$: an evaluation algorithm that, given the public parameters pp and ciphertext ct, outputs an encrypted result em.
- $\mathsf{Decrypt}(\mathsf{pp}, \mathsf{em}, \mathsf{st}) \to \mathsf{dm}$: a decryption algorithm that, given the public parameters pp, an encrypted result em, and secret state st, outputs a result dm (possibly $\perp$).

An EMSM scheme is secure if it realizes a (reactive) secure two-party computation of $\langle \mathbf{z}, \mathbf{g} \rangle$ in the Setup-hybrid model. In particular, let $\mathcal{F}_{\mathsf{MSM}}$ (shown in Figure 5.2) be the reactive two-party ideal functionality that is initialized with a public basis $\mathbf{g}$ and, when given input $\mathbf{z}$ from the client, returns $\langle \mathbf{z}, \mathbf{g} \rangle$. (reactive means the setup can be reused.) A corrupted server can be *semi-honest* or *malicious*, giving two flavors of security for EMSM. A malicious server can abort the functionality and hence prevent the client from receiving output. We say an EMSM scheme $\Pi = (\mathsf{Setup}, \mathsf{Encrypt}, \mathsf{Evaluate}, \mathsf{Decrypt})$ is *secure* if the natural two-round protocol based on Π—in which the client runs $\mathsf{Encrypt}$ and sends the result to the server, the server runs $\mathsf{Evaluate}$ and sends the result to the client, and the client computes $\mathsf{Decrypt}$ and outputs the result—realizes the ideal functionality $\mathcal{F}_{\mathsf{MSM}}$ in the Setup-hybrid model. This implies the following properties:

- *Correctness*: in an honest execution, the client always outputs a correct evaluation $\langle \mathbf{z}, \mathbf{g} \rangle$.
- *Privacy*: the server learns nothing about the client's input $\mathbf{z}$.
- *Malicious Security*: a malicious server cannot make the client output an incorrect result. The malicious server also learns nothing about $\mathbf{z}$ based on the client's decision to abort.

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

Initialization: The client and the server initialize the functionality with a public $\mathbf{g} \in \mathbb{G}^n$.

Evaluation: Upon receiving $(\mathsf{eval}, \mathbf{z})$ from the client, send eval to the server. If the server sends $\mathsf{continue}$, send $\langle \mathbf{z}, \mathbf{g} \rangle$ to the client. If the server sends abort, send $\perp$ to the client.

Figure 5.2: Multi-scalar multiplication ideal functionality.

Efficiency. The trivial EMSM in which the client computes the MSM locally is secure. We are interested in EMSMs where the work done by the client is less than that native computation.

Realizing the setup. At a high level, there are two ways the setup can be realized in an EMSM. First, each client could compute its own setup parameters; this may be more expensive than computing a single MSM locally, but the cost could be *amortized* over multiple MSM evaluations. The other possibility is to have a semi-trusted entity who computes Setup once and for all, with the result being used by multiple clients. While in general it may be undesirable to introduce an additional trusted entity, for our EMSM scheme the Setup algorithm is *transparent*, meaning there is no secret trapdoor that needs to be deleted, and *verifiable*, meaning that anyone can verify the correctness of setup—though this verification costs $\Theta(n)$ group operations. In particular, this means that for our EMSM scheme the server itself could generate the requisite setup; a malicious server could potentially generate incorrect setup, but would risk being caught by doing so.

5.4.2 Construction

We present the framework for our EMSM scheme based on the dual-LPN assumption.

5.4.2.1 Protocol Design

We observe that MSM is a linear operation, i.e., $\langle \mathbf{z} + \mathbf{r}, \mathbf{g} \rangle - \langle \mathbf{r}, \mathbf{g} \rangle = \langle \mathbf{z}, \mathbf{g} \rangle$. Thus, a natural idea is to have the client choose a uniform $\mathbf{r}$ and send $\mathbf{z} + \mathbf{r}$ to the server, who sends

back $\langle \mathbf{z} + \mathbf{r}, \mathbf{g} \rangle$; the client then outputs $\langle \mathbf{z} + \mathbf{r}, \mathbf{g} \rangle - \langle \mathbf{r}, \mathbf{g} \rangle$. Naively, however, this requires the client to locally compute $\langle \mathbf{r}, \mathbf{g} \rangle$, and hence, does not reduce the work. Our insight is to mask $\mathbf{z}$ with a *structured, pseudorandom* $\mathbf{r}$ such that $\langle \mathbf{r}, \mathbf{g} \rangle$ can be computed using $o(n)$ group operations. Specifically, let $\mathbf{r} = \mathbf{G}\mathbf{e}$ where $\mathbf{G} \in \mathbb{F}^{n \times N}$ is a generator matrix with $N = O(n)$ and $\mathbf{e} \in \mathbb{F}^N$ is a secret *weight-t* vector where $t = O(1)$; Note $\mathbf{r}$ is pseudorandom under dual-LPN. Furthermore,

$$\langle \mathbf{r}, \mathbf{g} \rangle = \langle \mathbf{G}\mathbf{e}, \mathbf{g} \rangle = \langle \sum_{i \in [N]} e_i \cdot \mathbf{G}_{*,i},\ \mathbf{g} \rangle = \sum_{i \in [N], e_i \neq 0} e_i \cdot \langle \mathbf{G}_{*,i},\ \mathbf{g} \rangle,$$

where the $\{\langle \mathbf{G}_{*,i},\ \mathbf{g} \rangle\}_{i \in [N]}$ are independent of the client's input. Thus, if the vector $\mathbf{h} = \mathbf{G}^T\mathbf{g}$ (with $h_i = \langle \mathbf{G}_{*,i}, \mathbf{g} \rangle$) is published as part of the setup outputs, the client can then compute

$$\langle \mathbf{r}, \mathbf{g} \rangle = \sum_{i \in [N], e_i \neq 0} e_i \cdot h_i = \langle \mathbf{e}, \mathbf{h} \rangle$$

using $O(t)$ group operations. (We discuss the efficiency of computing $\mathbf{G}\mathbf{e}$ in Section 5.4.3.)

Note that, by a standard hybrid argument, the dual-LPN assumption holds even if the client uses the same generator $\mathbf{G}$ to mask polynomially many inputs, as long as the client samples a fresh noise vector $\mathbf{e}$ each time. Thus, if $\mathbf{G}$ is public and fixed, the corresponding $\{h_i\}_{i \in [N]}$ can be published once and reused forever. Note further that the setup algorithm is transparent (assuming $\mathbf{G}$ is generated using a random oracle), and the values $\{h_i\}$ can be verified by anyone.

Achieving malicious security. As described, the scheme is secure against a *semi-honest* server, but a malicious server can violate correctness by causing the client to output an incorrect result. (Note, however, that the protocol is not vulnerable to a selective-failure attack violating privacy.) We show that our protocol can be adapted to achieve malicious security with only $2\times$ overhead.

To do so, we have the client choose a uniform $c \xleftarrow{\$} \mathbb{F}_q$ and then run the semi-honest protocol *twice* on inputs $\mathbf{z}$ and $\mathbf{z}_{\mathsf{ck}} = c \cdot \mathbf{z}$ to get outputs $\mathsf{dm}, \mathsf{dm}_{\mathsf{ck}}$. Then, if $\mathsf{dm}_{\mathsf{ck}} = c \cdot \mathsf{dm}$, the client outputs dm; otherwise, it outputs $\perp$. To see that this ensures security against a malicious server, let $\mathbf{v}, \mathbf{v}_{\mathsf{ck}}$ be the vectors sent by the client to the server and assume the server responds with $\mathsf{em} = \langle \mathbf{v}, \mathbf{g} \rangle + \epsilon$ and $\mathsf{em}_{\mathsf{ck}} = \langle \mathbf{v}_{\mathsf{ck}}, \mathbf{g} \rangle + \epsilon_{\mathsf{ck}}$ with $\epsilon \neq 0$. This causes the client to reconstruct $\mathsf{dm} = \langle \mathbf{z}, \mathbf{g} \rangle + \epsilon$ and $\mathsf{dm}_{\mathsf{ck}} = \langle c \cdot \mathbf{z}, \mathbf{g} \rangle + \epsilon_{\mathsf{ck}}$; thus, the client outputs $\perp$ unless $\epsilon_{\mathsf{ck}} = c \cdot \epsilon$. Since c is uniform and hidden from the server, we see that the client outputs $\perp$ except with negligible probability $1/q$.

Our EMSM is specified in Figure 5.3; components for malicious security are in blue.

Theorem 5.4.1. *Assuming the dual-LPN assumption for distributions $\mathcal{G}, \mathcal{E}$. Then the semi-honest (resp. malicious) scheme in Figure 5.3 is secure for semi-honest (resp. malicious) adversaries.*

Proof. We focus on the malicious setting; semi-honest follows by an analogous argument.

We show that the protocol (as described earlier) obtained from the malicious version of Π_{MSM} UC-realizes $\mathcal{F}_{\mathsf{MSM}}^{\mathbf{g}}$ in the Setup-hybrid model. Security against a malicious client is trivial. For a malicious server, consider a simulator interacting with an adversary $\mathcal{A}$ corrupting the server. In each execution, the simulator chooses uniform $\mathbf{v}, \mathbf{v}_{\mathsf{ck}}$ and sends these to the $\mathcal{A}$; in response, it receives $\mathsf{em}, \mathsf{em}_{\mathsf{ck}}$ from $\mathcal{A}$. If $\mathsf{em} = \langle \mathbf{v}, \mathbf{g} \rangle$ and $\mathsf{em}_{\mathsf{ck}} = \langle \mathbf{v}_{\mathsf{ck}}, \mathbf{g} \rangle$, the simulator sends $\mathsf{continue}$ to $\mathcal{F}_{\mathsf{MSM}}$; otherwise, it sends abort to $\mathcal{F}_{\mathsf{MSM}}$. The security follows from dual-LPN, as above. □

Supporting public linear maps. We observe that any EMSM naturally supports the application of a fixed (public) linear map to the client's input before computing the MSM. That is, we can delegate the computation of $\langle \mathbf{Hz}, \mathbf{g} \rangle$, where $\mathbf{H}$ is public and $\mathbf{z}$ is private. To do so, we can use the fact that $\langle \mathbf{Hz}, \mathbf{g} \rangle = \langle \mathbf{z}, \mathbf{H}^T \mathbf{g} \rangle$; thus, we can run the setup of the EMSM scheme on input $\mathbf{H}^T \mathbf{g}$.

Π_{EMSM}

Let n be the input length, $\mathcal{G}$ be a generator distribution over matrices in $\mathbb{F}_q^{n\times N}$, and $\mathcal{E}$ an noise distribution over weight-t vectors in $\mathbb{F}_q^N$ with $t = O(1)$ and block length $N = O(n)$.

$\mathsf{Setup}(1^\lambda, \mathbf{g})$:

- Sample $\mathbf{G} \leftarrow \mathcal{G}$, and compute $\mathbf{h} := \mathbf{G}^T\mathbf{g}$.
- Return the public parameters $\mathsf{pp} := (\mathbf{G}, \mathbf{h})$.

$\mathsf{Encrypt}(\mathsf{pp}; \mathbf{z})$:

- Sample $\mathbf{e} \leftarrow \mathcal{E}$ and evaluate $\mathbf{r} := \mathbf{G}\mathbf{e}$.
- Sample $\mathbf{e}_{\mathsf{ck}} \leftarrow \mathcal{E}$ and evaluate $\mathbf{r}_{\mathsf{ck}} := \mathbf{G}\mathbf{e}_{\mathsf{ck}}$.
- Sample a uniform random challenge $c \leftarrow \mathbb{F}_q$.
- Compute $\mathbf{v} := \mathbf{r} + \mathbf{z}$ as well as $\mathbf{v}_{\mathsf{ck}} := \mathbf{r}_{\mathsf{ck}} + c \cdot \mathbf{z}$.
- Return the ciphertext $\mathsf{ct} := (\mathbf{v}, \mathbf{v}_{\mathsf{ck}})$ and the secret state $\mathsf{st} := (\mathbf{e}, \mathbf{e}_{\mathsf{ck}}, c)$.

$\mathsf{Evaluate}(\mathsf{pp}; \mathsf{ct})$:

- Return the encrypted result $\mathsf{em}^* := (\langle \mathbf{v}, \mathbf{g}\rangle, \langle \mathbf{v}_{\mathsf{ck}}, \mathbf{g}\rangle)$.

$\mathsf{Decrypt}(\mathsf{pp}; \mathsf{em}^*, \mathsf{st})$:

- Parse em^* as $\mathsf{em}, \mathsf{em}_{\mathsf{ck}}$
- Compute $\mathsf{dm} := \mathsf{em} - \langle \mathbf{e}, \mathbf{h}\rangle$.
- Compute $\mathsf{dm}_{\mathsf{ck}} := \mathsf{em}_{\mathsf{ck}} - \langle \mathbf{e}_{\mathsf{ck}}, \mathbf{h}\rangle$.
- If $\mathsf{dm}_{\mathsf{ck}} \neq c \cdot \mathsf{dm}$, return $\perp$; else return the result dm.

Figure 5.3: Encrypted multi-scalar multiplication scheme.

One input, multiple MSMs. We note that a client may need to outsource $\langle \mathbf{z}, \mathbf{g}_1\rangle, \ldots, \langle \mathbf{z}, \mathbf{g}_\ell\rangle$ for multiple group bases $\mathbf{g}_1, \ldots, \mathbf{g}_\ell$ but the same input $\mathbf{z}$. While this can be done using ℓ independent executions of our EMSM protocol, it is possible to do better when it is acceptable to reveal to the server that the same input is being used. First, for Setup it suffices to fix a single $\mathbf{G}$ (though ℓ vectors $\mathbf{h}_1 = \mathbf{G}^T\mathbf{g}_1, \ldots, \mathbf{h}_\ell = \mathbf{G}^T\mathbf{g}_\ell$ are still needed). More significantly, in the protocol itself,

the client only needs to compute and send a *single* value $\mathbf{v}$. (For simplicity, we focus only on the semi-honest setting; everything extends naturally to the malicious case.) The server then computes the ℓ responses $\{\mathsf{em}_i = \langle \mathbf{v}, \mathbf{g}_i \rangle\}$ and the client computes the ℓ outputs $\{\mathsf{dm}_i\}$ as before.

5.4.3 Efficient Instantiation

In the EMSM scheme of Figure 5.3, the client performs $O(t) = O(1)$ group operations. We note that for an arbitrary matrix $\mathbf{G}$, however, the client might have to perform up to $n \cdot t$ field multiplications to compute $\mathbf{Ge}$ [1]; the same cost applies to $\mathbf{Ge}_{\mathsf{ck}}$. However, letting $\mathbf{G}$ be a *structured* matrix for which dual-LPN is still conjectured to hold, we can further reduce this cost.

There are several choices for $\mathbf{G}$ [7, 35, 36, 111, 127]. Here, we let $\mathbf{G}$ be a generator for an RAA code due to the fast encoding runtime this achieves. Recall this would mean $\mathbf{G} = \mathbf{F}_r\mathbf{M}_{\sigma_1}\mathbf{A}\mathbf{M}_{\sigma_2}\mathbf{A}$, such that $\mathbf{F}_r$ is an r-repetition matrix for $r = N/n$; $\mathbf{M}_{\sigma_1}$ and $\mathbf{M}_{\sigma_2}$ are permutation matrices corresponding to permutations σ_1, σ_2 over $[N]$; and $\mathbf{A}$ is the accumulation matrix. It is then possible to compute $\mathbf{Ge}$ using at most than $3N$ field additions: each application of $\mathbf{A}$ and $\mathbf{F}_r$ requires at most $N - 1$ additions, while applying $\mathbf{M}_{\sigma_1}$ and $\mathbf{M}_{\sigma_2}$ requires no field operations. The net result is to reduce the client's computation to $3N = 12n$ field *additions*.

Using RAA codes also makes the setup procedure more efficient. When $\mathbf{G}$ is arbitrary, computing $\mathbf{h}$ requires $O(nN)$ group multiplications, but when $\mathbf{G}$ is the generator matrix of an RAA code this cost drops to $O(N)$ group multiplications. We note that this also holds when delegating a public linear map $\mathbf{H}$, as long as $\mathbf{H}$ is structured that enables fast multiplication. (Even if $\mathbf{H}^T\mathbf{G}^T$ is not structured, we can apply $\mathbf{G}^T$ and $\mathbf{H}^T$ sequentially to enable fast multiplication.)

[1] For instance, with $n = 2^{20}$, $N = 2^{22}$, and $\mathbf{G} \in \mathbb{F}^{n \times N}$ uniform, we require $t \geq 40$ for 100-bit security [34].

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

Initialization: The client and the server initialize the functionality with $\mathbf{g} = (\tau^i \cdot g_1)_{i\in[n]}$.

Commitment: Upon receiving $(\mathsf{com}, f(X))$ from the client, where $f \in \mathbb{F}^{<n}[X]$, send com to the server. If the server sends $\mathsf{continue}$, send $c = \langle \mathbf{f}, \mathbf{g} \rangle$ to the client, where $\mathbf{f}$ is the coefficient vector of $f(X)$ and store $\mathbf{f}$. If the server sends abort, send $\perp$ to the client.

Opening: Upon receiving $(\mathsf{open}, f(X), x)$ from the client, send open to the server and adversary. If the server sends $\mathsf{continue}$, send $\langle \mathbf{q}, \mathbf{g} \rangle$ to the client, where $\mathbf{q}$ is the coefficient vector of $q(X) = \frac{f(X)-f(x)}{X-x}$. Otherwise, if the server sends abort, then send $\perp$ to the client.

Figure 5.4: KZG commitment ideal functionality.

5.4.4 Outsourced KZG Commitments

An immediate application of EMSM schemes is outsourcing KZG polynomial commitments. The commitment and opening algorithms of the KZG scheme each require only one MSM. In particular, for $\mathbf{g} = (\tau^i \cdot g_1)_{i\in[n]}$, the commitment algorithm outputs $\langle \mathbf{f}, \mathbf{g} \rangle$, where $\mathbf{f}$ is the coefficient vector of a polynomial $f(X) \in \mathbb{F}_q^{<n}[X]$; to prove that $f(x) = y$, the opening algorithm computes $q(X) = \frac{f(X)-y}{X-x}$ and outputs $\langle \mathbf{q}, \mathbf{g} \rangle$. In both cases, the dominant computation—the MSMs—can be outsourced via EMSM. We refer to the result as an *encrypted KZG* (EKZG).

Theorem 5.4.2. *If Π is a secure semi-honest (resp., malicious) EMSM, then Π_{KZG} UC-realizes $\mathcal{F}_{\mathsf{KZG}}$ in the* Setup*-hybrid model for semi-honest (resp., malicious) adversaries.*

5.5 zk-SNARKs with Server-Aided Proving

We first formalize the notion of *zk-SNARKs with server-aided proving* that enables a client to (partially) outsource the work of proof generation for a zk-SNARK to a more powerful server. We then show server-aided proving protocols for widely deployed zk-SNARKs.

Protocol Π_{KZG}

Let $(\mathbf{g} = (g_1, \tau \cdot g_1, \ldots, \tau^{n-1} \cdot g_1),\ \tau \cdot g_2)$ be public parameters for KZG polynomial commitment scheme, and let $\Pi = (\mathsf{Setup}, \mathsf{Encrypt}, \mathsf{Evaluate}, \mathsf{Decrypt})$ be an EMSM.

Setup: The EMSM setup algorithm $\Pi.\mathsf{Setup}(1^\lambda, \mathbf{g})$ is executed, giving the output pp_Π.

Commitment: On input coefficient vector $\mathbf{f}$, the client runs $(\mathsf{ct}, \mathsf{st}) \leftarrow \mathsf{Encrypt}(\mathsf{pp}_\Pi; \mathbf{f})$, and sends ct to the server. The server runs $\mathsf{em} \leftarrow \mathsf{Evaluate}(\mathsf{pp}_\Pi; \mathsf{ct})$ and sends em to the client. Finally, the client outputs the commitment $\mathsf{dm} \leftarrow \mathsf{Decrypt}(\mathsf{pp}_\Pi; \mathsf{em}; \mathsf{st})$.

Opening: Given x, the client evaluates $y = f(x)$ and computes $q(X) = \frac{f(X)-y}{X-x}$ with the coefficient vector $\mathbf{q}$. The client then runs $(\mathsf{ct}, \mathsf{st}) \leftarrow \mathsf{Encrypt}(\mathsf{pp}_\Pi; \mathbf{q})$ and sends ct to the server. The server executes $\mathsf{em} \leftarrow \mathsf{Evaluate}(\mathsf{pp}_\Pi; \mathsf{ct})$ and sends em to the client. Finally, the client outputs the KZG opening $\mathsf{dm} \leftarrow \mathsf{Decrypt}(\mathsf{pp}_\Pi; \mathsf{em}; \mathsf{st})$.

Figure 5.5: Encrypted KZG commitment scheme.

Remark 5.5.1. *zk-SNARK proving consists of two steps: (i) A witness w for a relation $(x, w) \in \mathcal{R}$ is transformed into a witness for a certain constraint system, such as R1CS. (ii) A cryptographic argument system is applied to generate an argument of the satisfiability of the constraint system. Since the first step is relation-specific, non-cryptographic, and computationally cheap, we assume it is performed by the client and only focus on the outsourcing of the second step.*

5.5.1 Definition

Let $\Sigma = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ be a zk-SNARK. Given $\mathsf{pp}_\Sigma \leftarrow \Sigma.\mathsf{Setup}$, a server-aided proving scheme for Σ consists of the following efficient algorithms:

- $\mathsf{Setup}(\mathsf{pp}_\Sigma) \to \mathsf{pp}$: a setup algorithm that, given pp_Σ, outputs public parameters pp.
- $\langle \mathsf{Prove.S}(\mathsf{pp}_\Sigma, \mathsf{pp}), \mathsf{Prove.C}(\mathsf{pp}_\Sigma, \mathsf{pp}, x, w) \rangle \to \pi$: this is an interactive protocol between a server S who holds $\mathsf{pp}_\Sigma, \mathsf{pp}$ and a client C who holds $\mathsf{pp}_\Sigma, \mathsf{pp}$, a statement x, and a witness w. Finally, C outputs a proof π (possibly $\perp$).

As in the case of EMSM schemes, we define security via a reactive two-party functionality.

Let $\mathcal{F}^{\Sigma}_{\mathsf{Prove}}$ (cf. Figure 5.6) be the ideal functionality corresponding to the proving algorithm of the zk-SNARK Σ. The functionality is parameterized by public parameters pp_{Σ} generated by $\Sigma.\mathsf{Setup}$, and repeatedly evaluates $\Sigma.\mathsf{Prove}(\mathsf{pp}; x, w)$ for inputs (x, w) provided by the client. The server does not provide input or receive output; however, a malicious server may abort and prevent the client from receiving the output. A server-aided proving scheme Π_{Σ} for the zk-SNARK Σ is *secure* if it UC-realizes $\mathcal{F}^{\Sigma}_{\mathsf{Prove}}$ in the $\Pi_{\Sigma}.\mathsf{Setup}$-hybrid model. As before, we may consider a semi-honest or malicious corrupted server, giving rise to two notions of security. (Security for a corrupted client is trivial.) This definition implies the following properties:

- *Correctness*: In an honest execution between C and S, where C provides input (x, w) such that $(x, w) \in \mathcal{R}$, the proof π output by C satisfies $\Sigma.\mathsf{Verify}(\mathsf{pp}; x, \pi) = 1$.

- *Witness privacy*: A corrupted S running the protocol with C learns nothing about the witness w, even if S later obtains the statement x and the proof π; this is due to the security of the protocol and the zero-knowledge property of Σ.

- *Unlinkability*: A corrupted S learns no information about the proof output by C. This ensures that proofs (and thereby statements) cannot be linked to the transcript of an execution of the server-aided proving protocol, even if proofs or statements are later revealed to S. Note that this property is independent of witness privacy; in particular, existing work on private delegation of zk-SNARKs [51, 79, 79, 124] achieves witness privacy but reveals the statement and proof.

- *Malicious security*: A malicious S cannot cause C to output an incorrect proof π; the most it can do is make the client output $\perp$. The definition also rules out selective-failure attacks in which S learns (information) about C's inputs based on whether or not C outputs $\perp$.

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

Initialization: The client and the server initialize the functionality with a relation $\mathcal{R}$ with a zk-SNARK $\Sigma = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$, and its public parameters $\mathsf{pp}_\Sigma \leftarrow \Sigma.\mathsf{Setup}(1^\lambda)$.

Proof generation: Upon receiving (prove, x, w) from the client, send prove to the server. If the server responds with $\mathsf{continue}$, then send the proof $\pi \leftarrow \Sigma.\mathsf{Prove}(\mathsf{pp}_\Sigma, x, w)$ to the client. Otherwise, if the server responds with abort, then send $\perp$ to the client.

Figure 5.6: Server-aided proving ideal functionality.

Efficiency. Any zk-SNARK Σ admits a trivial Π_Σ for Σ in which $\Pi_\Sigma.\mathsf{Setup}$ does nothing, and $\Pi_\Sigma.\mathsf{Prove}$ has C run $\Sigma.\mathsf{Prove}$ locally while S remains idle. We are thus interested in schemes in which C's work in $\Pi_\Sigma.\mathsf{Prove}$ is strictly less than its computation to execute $\Sigma.\mathsf{Prove}$ locally.

5.5.2 Server-Aided Proving for Nova

We describe a server-aided proving scheme for Nova. Recall the Nova protocol from Section 5.2.2. We observe that the prover cost is dominated by Pedersen commitments to $\mathbf{t}$ and $\mathbf{w}$. Note that when we extract an instance-witness for an iteration, $\mathbf{e}$ is always set to a zero vector, and thus its commitment is fixed. During the folding steps, $\mathbf{e}$ changes based on the cross term. Since a Pedersen commitment can be written as a single MSM, we use EMSM to delegate commitment generation. The client then requires constant group operations and linear field operations to run folding and update the instances and witnesses. In particular, in each folding invocation, the client first computes the cross term $\mathbf{t}$ and delegates its commitment. Then, after decoding the commitment, the client derives the Fiat-Shamir challenge, updates the current folded instance and witness, and delegates the commitment to $\mathbf{w}$. We present the formal protocol in the $\mathcal{F}_{\mathsf{MSM}}$-hybrid model in Figure 5.7, where $\mathcal{F}_{\mathsf{MSM}}$ can be instantiated by EMSM. The following is immediate:

Theorem 5.5.1. *In the $\mathcal{F}_{\mathsf{MSM}}$-hybrid model, Π_{Nova} UC-realizes the functionality $\mathcal{F}_{\mathsf{Prove}}$ for Nova.*

$\underline{\Pi_{\mathsf{Nova}}}$

Let $\mathsf{pp}_{\mathsf{Nova}}$ be the public parameters of Nova—including an MSM basis $\mathbf{g} \in \mathbb{G}^n$—and let F be an iteration function with the corresponding augmented function F'—defined below. Let $\mathcal{R}'$ be a committed, relaxed R1CS relation corresponding to F', defined over instances $(\bar{\mathbf{e}}, u, \bar{\mathbf{w}}, \mathbf{x})$ and witnesses $(\mathbf{e}, \mathbf{w})$. Let $(\bar{\mathbf{e}}_f, u_f, \bar{\mathbf{w}}_f, \mathbf{x}_f)$ and $(\bar{\mathbf{e}}_0, u_0, \bar{\mathbf{w}}_0, \mathbf{x}_0)$ be initialized by trivially satisfying instances with corresponding witnesses $(\mathbf{e}_f, \mathbf{w}_f)$ and $(\mathbf{e}_0, \mathbf{w}_0)$, respectively. Then, given a zk-SNARK for R', $\Pi_{\mathsf{SNARK}} = \langle \mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify} \rangle$, the MSM functionality $\mathcal{F}_{\mathsf{MSM}}$, and initial input $\mathbf{z}_0$, server-aided Nova works as follows:

- The client and the server initialize $\mathcal{F}_{\mathsf{MSM}}$ with $\mathbf{g} \in \mathbb{G}^n$.
- For each i-th iteration of the computation, proceed as follows:
  1. C computes $\mathbf{t} := (\mathbf{A}\mathbf{z}_f) \odot (\mathbf{B}\mathbf{z}_{i-1}) + (\mathbf{A}\mathbf{z}_{i-1}) \odot (\mathbf{B}\mathbf{z}_f) - u_f \cdot (\mathbf{C}\mathbf{z}_{i-1}) - u_{i-1} \cdot (\mathbf{C}\mathbf{z}_f)$.
  2. C and S send $(\mathsf{eval}, \mathbf{t})$ to $\mathcal{F}^{\mathbf{g}}_{\mathsf{MSM}}$; C receives $\bar{\mathbf{t}} := \langle \mathbf{t}, \mathbf{g} \rangle$ from $\mathcal{F}^{\mathbf{g}}_{\mathsf{MSM}}$.
  3. C runs $F'(i-1, z_0, z_{i-1}, (\bar{\mathbf{e}}_f, u_f, \bar{\mathbf{w}}_f, \mathbf{x}_f), (\bar{\mathbf{e}}_{i-1}, u_{i-1}, \bar{\mathbf{w}}_{i-1}, \mathbf{x}_{i-1}), \bar{t}, \mathbf{w}_{i-1})$ as below:
     (a) Evaluate $\mathbf{z}_i := F(\mathbf{z}_{i-1}, \mathbf{w}_{i-1})$.
     (b) For Fiat-Shamir Challenge r, update $\bar{\mathbf{e}}_f, u_f, \bar{\mathbf{w}}_f, \mathbf{x}_f$ such that:
     $$u_f := u_f + r \cdot u_{i-1}, \quad \bar{\mathbf{e}}_f := \bar{\mathbf{e}}_f + r \cdot \bar{\mathbf{t}} + r^2 \cdot \bar{e}_{i-1},$$
     $$\mathbf{x}_f := \mathbf{x}_f + r \cdot \mathbf{x}_{i-1}, \quad \bar{\mathbf{w}}_f := \bar{\mathbf{w}}_f + r \cdot \bar{\mathbf{w}}_{i-1}.$$
     (c) Output $(i, \mathbf{z}_0, \mathbf{z}_i, (\bar{\mathbf{e}}_f, u_f, \bar{\mathbf{w}}_f, \mathbf{x}_f))$.
  4. C sets $\mathbf{e}_f := \mathbf{e}_f + r \cdot \mathbf{t} + r^2 \cdot \mathbf{e}_{i-1}, \mathbf{w}_f := \mathbf{w}_f + r \cdot \mathbf{w}_{i-1}$ and extracts the instance-witness $(\mathbf{x}_i, \mathbf{w}_i)$ for the execution of F' in Step 3. C sets $\mathbf{e}_i := 0^n$, $\bar{\mathbf{e}}_i := \mathsf{Commit}(0^n)$, $u_i := 1$.
  5. C and S send $(\mathsf{eval}, \mathbf{w}_i)$ to $\mathcal{F}^{\mathbf{g}}_{\mathsf{MSM}}$; C receives $\bar{\mathbf{w}}_i := \langle \mathbf{w}_i, \mathbf{g} \rangle$.
  6. If it is the final iteration, repeat Steps 1 to 5 except for Step 3.a.
- C outputs $(i, \mathbf{z}_0, \mathbf{z}_i)$, $(\bar{\mathbf{e}}_i, u_i, \bar{\mathbf{w}}_i, \mathbf{x}_i)$, $(\bar{\mathbf{e}}_f, u_f, \bar{\mathbf{w}}_f, \mathbf{x}_f)$, π_f, and π_i such that:
$$\pi_f := \Pi_{\mathsf{SNARK}}.\mathsf{Prove}(\mathsf{pp}_{\mathsf{SNARK}}; (\bar{\mathbf{e}}_f, u_f, \bar{\mathbf{w}}_f, \mathbf{x}_f), (\mathbf{e}_f, \mathbf{w}_f)),$$
$$\pi_i := \Pi_{\mathsf{SNARK}}.\mathsf{Prove}(\mathsf{pp}_{\mathsf{SNARK}}; (\bar{\mathbf{e}}_i, u_i, \bar{\mathbf{w}}_i, \mathbf{x}_i), (\mathbf{e}_i, \mathbf{w}_i)).$$

Figure 5.7: Server-aided proving for Nova.

We remark that while our presentation of Nova is simplified and minor steps are abstracted (e.g., hashing inputs/outputs), our technique immediately extends to the full Nova protocol.

Efficiency. Let n denote the constraint size of an iteration. The prover's (per-iteration) cost in Nova is dominated by two $O(n)$-sized MSMs. In our server-aided protocol for Nova, the client's

work is reduced to $O(1)$ group operations, while the server's work is two $O(n)$-sized MSMs in the semi-honest scheme (resp., four $O(n)$-sized MSMs in the malicious scheme). Moreover, the scheme requires two rounds of interaction per iteration, with $O(n)$ communication complexity.

5.5.3 Server-Aided Proving for Groth16

The second zk-SNARK we consider is Groth16. Recall the design of prover from Section 5.2.2. The client holds a satisfiable assignment $\mathbf{z} = \mathbf{x} \| \mathbf{w}$ for the R1CS relation specified by the matrices $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$, and wishes to privately outsource the computation of $\pi = (\pi_a, \pi_b, \pi_c)$. Let us define the following group bases, which are included in the public parameters pp:

$$\mathbf{g}_a := (f_{A,j})_{j\in[m]} \cdot g_1, \mathbf{g}_b := (f_{B,j})_{j\in[m]} \cdot g_2, \mathbf{g}_{c,rz} := (f_{B,j})_{j\in[m]} \cdot g_1,$$

$$\mathbf{g}_{c,z} := (\delta^{-1} f_j)_{j\in[\ell:m]} \cdot g_1, \mathbf{g}_{c,q} := (\delta^{-1} v^i t(v))_{i\in[n-1]} \cdot g_1.$$

Given $\mathbf{q}$, which is the coefficient vector of $q(X)$ and $\mathbf{z}$, we have

$$\pi_a = \alpha \cdot g_1 + r \cdot (\delta \cdot g_1) + \langle \mathbf{z}, \mathbf{g}_a \rangle, \pi_b = \beta \cdot g_2 + s \cdot (\delta \cdot g_1) + \langle \mathbf{z}, \mathbf{g}_b \rangle,$$

$$\pi_c = s \cdot \pi_a + r \cdot \beta \cdot g_1 + \langle r \cdot \mathbf{z}, \mathbf{g}_{c,rz} \rangle + \langle \mathbf{z}, \mathbf{g}_{c,z} \rangle + \langle \mathbf{q}, \mathbf{g}_{c,q} \rangle.$$

The client encrypts $\mathbf{z}$ and $r \cdot \mathbf{z}$ with EMSM and delegates $\langle \mathbf{z}, \mathbf{g}_a \rangle$, $\langle \mathbf{z}, \mathbf{g}_b \rangle$, $\langle r \cdot \mathbf{z}, \mathbf{g}_{c,rz} \rangle$, $\langle \mathbf{z}, \mathbf{g}_{c,z} \rangle$. The main remaining term is $\langle \mathbf{q}, \mathbf{g}_{c,q} \rangle$. Computing the coefficients of $q(X)$ is costly for the client, as it involves multiple (i)FFT operations. We use $\mathbf{F}_D$ to denote the matrix representation of the FFT over the domain D. Recall that the client computes $\mathbf{q} = \mathbf{F}_D^{-1} \cdot (\mathbf{h} \oslash (\mathbf{F}_D \cdot \mathbf{t}))$ such that

$$\mathbf{h} := \big((\mathbf{F}_D \mathbf{F}_{D_{\omega'}}^{-1} \cdot \mathbf{A}\mathbf{z}) \odot (\mathbf{F}_D \mathbf{F}_{D_{\omega'}}^{-1} \cdot \mathbf{B}\mathbf{z}) - (\mathbf{F}_D \mathbf{F}_{D_{\omega'}}^{-1} \cdot \mathbf{C}\mathbf{z}) \big),$$

where $\mathbf{t}$ denotes the coefficients of $t(X)$, and $\odot$ and $\oslash$ denote the element-wise multiplication and division, respectively. This requires eight (i)FFTs, which can be reduced as follows:

1. Computing the final iFFT, i.e., $\mathbf{F}_D^{-1}$, is unnecessary as our EMSM supports public linear maps; that is, the client can instead outsource $\langle \mathbf{y}, (\mathbf{F}_D^{-1})^T \mathbf{g}_{c,q} \rangle$ given $\mathbf{y} = \mathbf{h} \oslash (\mathbf{F}_D \cdot \mathbf{t})$.

2. The polynomial $t(X)$ is public and witness-independent; thus, $\mathbf{F}_D \cdot \mathbf{t}$ can be precomputed, and the client can instead outsource $\langle \mathbf{h}, \mathbf{g}_{c,h} \rangle$ where $\mathbf{g}_{c,h} := ((\mathbf{F}_D^{-1})^T \mathbf{g}_{c,q}) \oslash (\mathbf{F}_D \cdot \mathbf{t})$.

3. The client can delegate $\mathbf{F}_D \mathbf{F}_{D_{\omega'}}^{-1} \cdot \mathbf{C}$ to the server. In particular, we can decompose $\langle \mathbf{h}, \mathbf{g}_{c,h} \rangle$ as

$$\langle (\mathbf{F}_D \mathbf{F}_{D_{\omega'}}^{-1} \cdot \mathbf{Az}) \odot (\mathbf{F}_D \mathbf{F}_{D_{\omega'}}^{-1} \cdot \mathbf{Bz}), \mathbf{g}_{c,h} \rangle - \langle \mathbf{F}_D \mathbf{F}_{D_{\omega'}}^{-1} \cdot \mathbf{Cz}, \mathbf{g}_{c,h} \rangle,$$

   where the client can outsource $\langle \mathbf{z}, (\mathbf{F}_D \mathbf{F}_{D_{\omega'}}^{-1} \cdot \mathbf{C})^T \mathbf{g}_{c,h} \rangle$ to the server.

These optimizations reduce the number of (i)FFTs performed by the client to four. Note that $(\mathbf{F}_D \mathbf{F}_{D_{\omega'}}^{-1} \cdot \mathbf{Az}) \odot (\mathbf{F}_D \mathbf{F}_{D_{\omega'}}^{-1} \cdot \mathbf{Bz})$ is a quadratic term; hence, we cannot delegate it via EMSM. We provide the formal description of the scheme in the $\mathcal{F}_{\mathsf{MSM}}$-hybrid model in Figure 5.8.

Theorem 5.5.2. *In the $\mathcal{F}_{\mathsf{MSM}}$-hybrid model, Π_{Groth} UC-realizes $\mathcal{F}_{\mathsf{Prove}}$ for Groth16.*

Remark 5.5.2. *Ciphertexts in our EMSM scheme are independent of the basis. A single ciphertext suffices to outsource multiple MSMs with the same input vector. In practice, one ciphertext suffices for outsourcing all four MSMs with the input vector* $\mathbf{z}$*.*

Efficiency. For an R1CS instance of size n, the proving overhead of Groth16 is dominated by five $O(n)$-sized MSMs and eight $O(n)$-sized FFTs. Our scheme reduces the client's work to

$\underline{\Pi_{\mathsf{Groth}}}$

Let $\mathcal{R}$ be an R1CS relation over instance-witness $(\mathbf{x}, \mathbf{w})$, specified by $\mathbf{A}, \mathbf{B}, \mathbf{C} \in \mathbb{F}_q^{n \times m}$. Let ω denote a primitive $2n$-th root of unity, and $\omega' := \omega^2$. Let $D := D_\omega \setminus D_{\omega'}$. Let $f_{A,j}(X)$, $f_{B,j}(X)$, and $f_{C,j}(X)$ be such that for all $i \in [n], j \in [m]$, $f_{A,j}(\omega'^i) := A_{i,j}$, $f_{B,j}(\omega'^i) := B_{i,j}$, and $f_{C,j}(\omega'^i) := C_{i,j}$. Let $\mathsf{pp}_{\mathsf{Groth}}$ be the public parameters of Groth16 for $\mathcal{R}$, including

$$\begin{aligned}
&\mathbf{g}_a := (f_{A,j})_{j\in[m]} \cdot g_1, \quad \mathbf{g}_b := (f_{B,j})_{j\in[m]} \cdot g_2,\\
&\mathbf{g}_{c,z} := (\delta^{-1} f_j)_{j\in[\ell:m]} \cdot g_1, \quad \mathbf{g}_{c,rz} := (f_{B,j})_{j\in[m]} \cdot g_1, \quad \mathbf{g}_{c,q} := (\delta^{-1} v^i t(v))_{i\in[n-1]} \cdot g_1,\\
&\alpha\beta \cdot g_T, \quad \{\alpha, \beta, \delta, \{v^i\}_{i\in[n]}, \{\gamma^{-1} f_j(x)\}_{j\in[\ell]}\} \cdot g_1, \quad \{\beta, \gamma, \delta, \{v^i\}_{i\in[n]}\} \cdot g_2.
\end{aligned}$$

Let $\mathbf{g}_{c,h}^L := ((\mathbf{F}_D^{-1})^T \mathbf{g}_{c,q}) \oslash (\mathbf{F}_D \cdot \mathbf{t})$ and $\mathbf{g}_{c,h}^R := (\mathbf{F}_D \mathbf{F}_{D_{\omega'}}^{-1} \cdot \mathbf{C})^T \mathbf{g}_{c,h}^L$ such that $\mathbf{t}$ denotes the coefficient vector of the vanishing polynomial $t(X) = \prod_{w \in D_{\omega'}} (X - w)$.

- The client and the server initialize $\mathcal{F}_{\mathsf{MSM}}$ with $\mathbf{g}_a$, $\mathbf{g}_b$, $\mathbf{g}_{c,z}$, $\mathbf{g}_{c,rz}$, $\mathbf{g}_{c,h}^L$, and $\mathbf{g}_{c,h}^R$.
- C samples a random $r \in \mathbb{F}$.
- Let $\mathbf{z} = \mathbf{x} \| \mathbf{w}$. C computes $\mathbf{z}_{ab} := (\mathbf{F}_D \mathbf{F}_{D_{\omega'}}^{-1} \cdot \mathbf{A}\mathbf{z}) \odot (\mathbf{F}_D \mathbf{F}_{D_{\omega'}}^{-1} \cdot \mathbf{B}\mathbf{z})$.
- C and S send $(\mathsf{eval}, \mathbf{z})$ to $\mathcal{F}_{\mathsf{MSM}}^{\mathbf{g}_a}$, $\mathcal{F}_{\mathsf{MSM}}^{\mathbf{g}_b}$, $\mathcal{F}_{\mathsf{MSM}}^{\mathbf{g}_{c,z}}$, $\mathcal{F}_{\mathsf{MSM}}^{\mathbf{g}_{c,h}^R}$, $(\mathsf{eval}, r \cdot \mathbf{z})$ to $\mathcal{F}_{\mathsf{MSM}}^{\mathbf{g}_{c,rz}}$, and $(\mathsf{eval}, \mathbf{z}_{ab})$ to $\mathcal{F}_{\mathsf{MSM}}^{\mathbf{g}_{c,h}^L}$; C receives $\langle \mathbf{z}, \mathbf{g}_a \rangle$, $\langle \mathbf{z}, \mathbf{g}_b \rangle$, $\langle \mathbf{z}, \mathbf{g}_{c,z} \rangle$, $\langle r \cdot \mathbf{z}, \mathbf{g}_{c,rz} \rangle$, $\langle \mathbf{z}, \mathbf{g}_{c,h}^R \rangle$, and $\langle \mathbf{z}_{ab}, \mathbf{g}_{c,h}^L \rangle$.
- C outputs the proof $\pi := (\pi_a, \pi_b, \pi_c)$ such that

$$\begin{aligned}
&\pi_a = \alpha \cdot g_1 + r \cdot (\delta \cdot g_1) + \langle \mathbf{z}, \mathbf{g}_a \rangle, \pi_b = \beta \cdot g_2 + s \cdot (\delta \cdot g_1) + \langle \mathbf{z}, \mathbf{g}_b \rangle,\\
&\pi_c = s \cdot \pi_a + r \cdot \beta \cdot g_1 + \langle r \cdot \mathbf{z}, \mathbf{g}_{c,rz} \rangle + \langle \mathbf{z}, \mathbf{g}_{c,z} \rangle + \langle \mathbf{q}, \mathbf{g}_{c,q} \rangle.
\end{aligned}$$

Figure 5.8: Server-aided proving for Groth16.

$O(1)$ group operations and four $O(n)$-sized FFTs. The server work is six $O(n)$-sized MSMs; it is twelve in the malicious setting. As Groth16 is non-interactive by design and does not use the Fiat-Shamir heuristic [68], our scheme is also non-interactive (i.e., one communication round). The communication cost is $O(n)$, as the client runs the EMSM scheme on inputs $\mathbf{z}$, $r\mathbf{z}$, and $\mathbf{q}$.

5.5.4 Server-Aided Proving for Plonk

Finally, we consider server-aided Plonk. Recall from Section 5.2.2 that the core subroutine of Plonk is the KZG commitments, which can be outsourced using EKZG.

Unlike Groth16, Plonk does not allow for outsourcing FFTs. However, we observe that the client can defer all FFTs to the final rounds of the protocol, which reduces latency. Recall that in rounds 1-3 of the Plonk proving algorithms, the prover computes the evaluations of polynomials $a(X)$, $b(X)$, $c(X)$, $t(X)$, and $p(X)$, and then applies (i)FFTs to transform these evaluations into coefficients, which can be committed to by the KZG scheme. Since each KZG commitment is simply an MSM, we can equivalently apply (i)FFTs to the MSM basis, and the client can instead send encryptions of the evaluations. To open each commitment, the client needs coefficients and must run (i)FFTs; however, since all these occur in rounds 4 and 5, (i)FFTs can be deferred until then. We give a formal description of server-aided Plonk in Figure 5.9.

Theorem 5.5.3. *In the $\mathcal{F}_{\mathsf{MSM}}$-hybrid model, Π_{Plonk} UC-realizes the functionality $\mathcal{F}_{\mathsf{Prove}}$ for Plonk.*

Our presentation of Plonk is simplified, and certain parameters to achieve zero knowledge are abstracted. Nevertheless, our technique immediately extends to the full Plonk, since we only delegate KZG commitments, and the poly-IOP component remains unchanged.

Efficiency. For a constraint system of size n, the prover in the Plonk protocol commits to nine polynomials of degree $O(n)$. Thus, each commitment and opening uses $O(n)$-sized MSMs, which is the dominant cost of proving. Our scheme lowers these costs to $O(1)$ group operations for the client, while server's work is equivalent to MSMs in Plonk (doubled in malicious scheme). Our server-aided proving scheme requires five interaction rounds with $O(n)$ communication cost.

5.6 Implementation and Evaluation

We built a Rust library implementing our EMSM, EKZG, and server-aided proving for Nova [96] and Groth16 [86]. Our library builds on arkworks [10], a collection of Rust libraries

$\underline{\Pi_{\mathsf{Plonk}}}$

For a circuit C, let $(\mathbf{a}, \mathbf{b}, \mathbf{c}) \in [m]^{3n}$ be the left, right, and output sequences, respectively, and $(\mathbf{q}_L, \mathbf{q}_R, \mathbf{q}_O, \mathbf{q}_M, \mathbf{q}_C) \in \mathbb{F}^{5n}$ be the left, right, output, multiplication, and constant selector vectors, respectively. Let $\sigma : [3n] \to [3n]$ be the wiring permutation. Consider the circuit satisfiability relation as $C(\mathbf{w}) = \mathbf{x}$ for a private input $\mathbf{w}$ and a public output $\mathbf{x}$. Let ω denote a primitive n-th root of unity. Let $\mathsf{pp}_{\mathsf{Plonk}}$ denote the public parameters of Plonk, which include an MSM basis $\mathbf{g} \in \mathbb{G}^n$ used for KZG. Let $\mathbf{g}_s := \mathbf{g}$ and $\mathbf{g}_i := (\mathbf{F}_{D_\omega}^{-1})^T \mathbf{g}$.

- The client and the server initialize $\mathcal{F}_{\mathsf{MSM}}$ with $\mathbf{g}_s$ and $\mathbf{g}_i$.
- Let $\mathbf{z} = \mathbf{x} \| \mathbf{w}$, $\mathbf{z}_a := (z_{a_i})_{i \in [n]}$, $\mathbf{z}_b := (z_{b_i})_{i \in [n]}$, and $\mathbf{z}_c := (z_{c_i})_{i \in [n]}$. Then, C and S send $(\mathsf{eval}, \mathbf{z}_a)$, $(\mathsf{eval}, \mathbf{z}_b)$, and $(\mathsf{eval}, \mathbf{z}_c)$ to $\mathcal{F}_{\mathsf{MSM}}^{\mathbf{g}_i}$; C receives $\langle \mathbf{z}_a, \mathbf{g}_i \rangle$, $\langle \mathbf{z}_b, \mathbf{g}_i \rangle$, and $\langle \mathbf{z}_c, \mathbf{g}_i \rangle$.
- Given the challenge β, η, C computes $\mathbf{z}_h$ and $\mathbf{z}_g$ as follows. Let $\mathbf{z}_t := \mathbf{z}_h \oslash \mathbf{z}_g$.

$$\mathbf{z}_h := ((z_{a_i} + \beta\omega^i + \tau)(z_{b_i} + \beta\kappa\omega^i + \tau)(z_{c_i} + \beta\eta\omega^i + \tau))_{i \in [n]}$$
$$\mathbf{z}_g := ((z_{a_i} + \beta\omega^{\sigma(i)} + \tau)(z_{b_i} + \beta\kappa\omega^{\sigma(n+i)} + \tau)(z_{c_i} + \beta\eta\omega^{\sigma(2n+i)} + \tau))_{i \in [n]}.$$

- C and S send $(\mathsf{eval}, \mathbf{z}_t)$ to $\mathcal{F}_{\mathsf{MSM}}^{\mathbf{g}_i}$; C receives $\langle \mathbf{z}_t, \mathbf{g}_i \rangle$.
- For thechallenge ρ, C computes $\mathbf{z}_p$ as follows.

$$\begin{aligned} \mathbf{z}_p \quad := \quad & ((q_L)_i z_{a_i} + (q_R)_i z_{b_i} + (q_O)_i z_{c_i} + (q_M)_i z_{a_i} z_{b_i} + (q_C)_i \\ & + \rho((\mathbf{z}_t)_{i+1} - ((\mathbf{z}_t)_i (\mathbf{z}_h)_i)/(\mathbf{z}_g)_i) + \rho^2(\ell_n(\omega^i)((\mathbf{z}_t)_i - 1)))_{i \in [n]}. \end{aligned}$$

- C and S send $(\mathsf{eval}, \mathbf{z}_p)$ to $\mathcal{F}_{\mathsf{MSM}}^{\mathbf{g}_i}$; C receives $\langle \mathbf{z}_p, \mathbf{g}_i \rangle$.
- C computes $\mathbf{p} = \mathbf{F}_{D_\omega}^{-1} \mathbf{z}_p$. Let $p(X) = \sum_i p_i X^i$. C computes a quotient $p(X)/(X^n - 1)$ and decomposes $q(X)$ into $q_l(X), q_m(X), q_h(X)$, where $q(X) = q_l(X) + X^n \cdot q_m(X) + X^{2n} \cdot q_h(X)$. Let $\mathbf{q}_l$, $\mathbf{q}_m$, and $\mathbf{q}_h$ be the coefficient vectors of $q_l(X)$, $q_m(X)$, and $q_h(X)$, respectively.
- C and S send $(\mathsf{eval}, \mathbf{q}_l)$, $(\mathsf{eval}, \mathbf{q}_m)$, $(\mathsf{eval}, \mathbf{q}_h)$ to $\mathcal{F}_{\mathsf{MSM}}^{\mathbf{g}_s}$; C receives $\langle \mathbf{q}_l, \mathbf{g}_s \rangle$, $\langle \mathbf{q}_m, \mathbf{g}_s \rangle$, $\langle \mathbf{q}_h, \mathbf{g}_s \rangle$.
- C computes $\mathbf{a} = \mathbf{F}_{D_\omega}^{-1} \mathbf{z}_a$, $\mathbf{b} = \mathbf{F}_{D_\omega}^{-1} \mathbf{z}_b$, $\mathbf{c} = \mathbf{F}_{D_\omega}^{-1} \mathbf{z}_c$, $\mathbf{t} = \mathbf{F}_{D_\omega}^{-1} \mathbf{z}_t$. Let $a(X) = \sum_i a_i X^i$, $b(X) = \sum_i b_i X^i$, $c(X) = \sum_i c_i X^i$, $t(X) = \sum_i t_i X^i$. Then, for each $r(X) \in \{a(X), b(X), c(X), t(X), p(X), q_l(X), q_m(X), q_h(X)\}$ and a point (x, y), C computes $u(X) = \frac{r(X) - y}{X - x}$.
- C and S send $(\mathsf{eval}, \mathbf{u})$ to $\mathcal{F}_{\mathsf{MSM}}^{\mathbf{g}_s}$ for all $\mathbf{u}$; C receives $\langle \mathbf{u}, \mathbf{g}_s \rangle$, which is a KZG opening.
- C outputs all the KZG commitments and openings (i.e., all the decrypted MSMs).

Figure 5.9: Server-aided proving for Plonk.

for zkSNARKs that provides interfaces for fields, groups, and commitments. Our implementation is publicly available at: `https://github.com/h-hafezi/server-aided-snarks`.

Experimental Setup. In all our experiments, the client is an AWS `r6i.large` instance with 16 GB of RAM and 2 Intel Xeon Platinum 8375C vCPUs running at 3.5 GHz, located in the `us-west-1` region; it costs $0.14/hr. The server is an AWS `c7i.24xlarge` instance with 192 GB of RAM and 96 Intel Xeon Platinum 8488C vCPUs running at 3.2 GHz, located in the `us-east-1` region; it costs $4.28/hr. The network throughput between the instances is 3.85 Gbps. Our implementation is multi-threaded, with field and group operations parallelized.

5.6.1 EMSM and EKZG Schemes

We evaluate the performance of our EMSM scheme in both the semi-honest and malicious settings for various instance sizes n. In Table 5.2, we tabulate microbenchmarks for each algorithm of the scheme; recall that encryption and decryption are run by the client, and evaluation is run by the server. Interestingly, for small instances, decryption (dominated by the computation of a low-weight MSM) is slower than encryption (dominated by the map $\mathbf{Ge}$), but this reverses as the instance size grows. We see that the runtime of each algorithm increases by roughly a factor of two when moving from the semi-honest setting to the malicious setting; note that the evaluation time increases less since the server has more cores and can exploit parallelization.

To quantify the benefits of our EMSM, we compare its execution to a local evaluation of an MSM of the same size by the client. We then look at two factors: (i) the improvement in the local running time of the client, used as a proxy for the computational work done by the client and measured as the ratio of the local running time and the sum of the encryption/decryption times, and (ii) the latency, measured by the ratio of the local running time at the client and the overall EMSM time (which includes encryption, evaluation, decryption, and the round-trip communication time). Our evaluations show that, in the semi-honest setting, our EMSM reduces

the client's work by as much as $\mathbf{33}\times$ and improves the latency by up to $\mathbf{10}\times$. Since the EMSM time in the malicious setting doubles, the improvements change by a factor of two in that setting.

In our experiments, MSM inputs are uniform vectors. In some applications, however, the inputs may be drawn from structured distributions, e.g., binary or sparse vectors. In such cases, our EMSM scheme may provide smaller reductions in latency and client work since the server evaluates MSMs on inputs masked by uniform field elements, and the client performs an MSM on a sparse error vector with uniform entries.

Setup. We measure the cost of EMSM setup when run on a machine as powerful as the server. The setup requires computing $\mathbf{h} = \mathbf{G}^T\mathbf{g}$, which can be done using $2N$ group additions (each repetition and accumulation requires N group additions). For instance sizes 2^{15} and 2^{23}, setup requires 630 ms and 163 s, respectively.

EKZG scheme. For completeness, we also implement and evaluate the application of our EMSM scheme to delegating KZG polynomial commitments, i.e., our EKZG scheme. Generating a KZG commitment is exactly computing one MSM; thus, the performance of the commitment phase is identical to the performance of our EMSM scheme in Table 5.2. Opening a KZG commitment, however, involves *witness computation* (evaluating the polynomial and computing the quotient) using field operations executed locally by the client. Our evaluation (see Table 5.2) shows that these field operations account for only a minor fraction of the local computation (less than 2%). Our results indicate that in the semi-honest setting, EKZG reduces the client's cost by up to $\mathbf{22}\times$ and improves the latency by up to $\mathbf{9}\times$; in the malicious setting, these are $\mathbf{13}\times$ and $\mathbf{6}\times$.

Size	Client Enc.	Client Dec.	Server Eval.	Local	Improvement Runtime	Improvement Latency
2^{15}	3.08	8.69	21.3	272	**23×**	**8×**
2^{15}	6.54	17.4	24.3	272	**11×**	**5×**
2^{17}	20.7	8.46	58.8	949	**33×**	**10×**
2^{17}	43.3	16.9	81.3	949	**16×**	**6×**
2^{19}	167	8.39	244	3740	**21×**	**8×**
2^{19}	277	16.6	315	3740	**13×**	**6×**
2^{21}	732	8.16	870	15115	**20×**	**9×**
2^{21}	1280	16.3	1060	15115	**12×**	**6×**
2^{23}	3160	8.00	3630	59496	**19×**	**8×**
2^{23}	5580	16.0	4410	59496	**11×**	**5×**

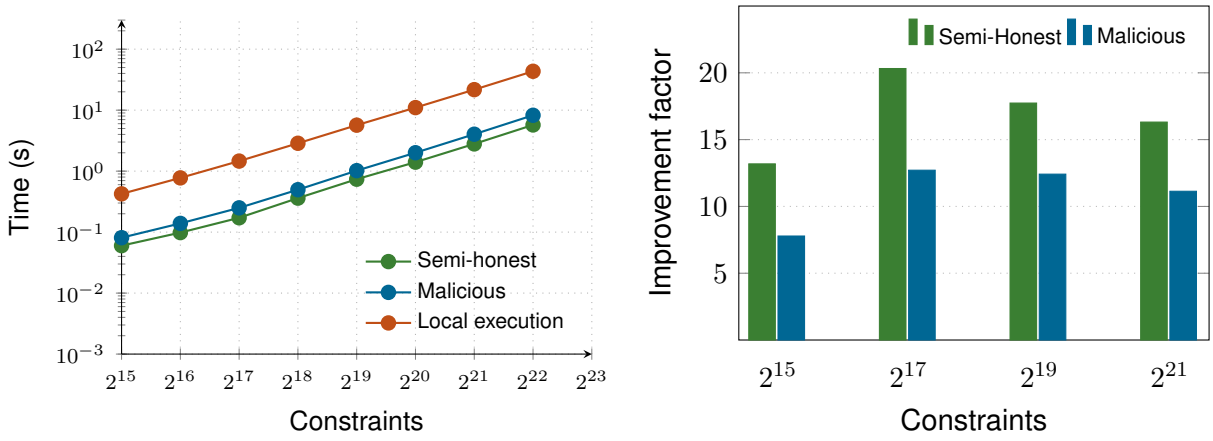
Size	Wit.	Client	Server	Local	Improvement Runtime	Improvement Latency
2^{15}	3.50	15.3	21.3	275	**18×**	**7×**
2^{15}	3.50	27.3	24.3	275	**10×**	**5×**
2^{17}	14.7	43.8	58.8	966	**22×**	**9×**
2^{17}	14.7	74.9	81.3	966	**13×**	**6×**
2^{19}	59.6	235	244	3830	**16×**	**7×**
2^{19}	59.6	353	315	3830	**11×**	**5×**
2^{21}	277	1017	870	15400	**15×**	**8×**
2^{21}	277	1570	1060	15400	**10×**	**5×**
2^{23}	1120	4280	3630	60500	**14×**	**7×**
2^{23}	1120	6720	4410	60500	**9×**	**5×**

Table 5.2: EMSM encryption, decryption, and evaluation times compared to local computation of the MSM (left), and EKZG opening compared to local computation of KZG opening (right). (Witness computation is run by the client.) Runtimes are in in milliseconds, and number for the malicious schemes are highlighted in blue.

5.6.2 Server-Aided Proving for zk-SNARKs

We evaluate our server-aided proving schemes for Nova and Groth16. Our Groth16 implementation does not include the FFT optimization; we estimate that doing so further improves the runtime and latency by $1.2\times$. We report the proof-generation latency in Figures 5.10(a) and 5.11(a), and plot the improvement in the client's running time in Figures 5.10(b) and 5.11(b). For Nova, our semi-honest scheme reduces the client's work by up to $\mathbf{20}\times$ and the latency by up to $\mathbf{9}\times$; in the malicious setting, these are $\mathbf{14}\times$ and $\mathbf{6}\times$, respectively. For Groth16, our scheme reduces the client's work by up to $\mathbf{10}\times$ and the latency by up to $\mathbf{6}\times$ in the semi-honest case and $\mathbf{8}\times$ and $\mathbf{5}\times$, respectively, in the malicious case.

Comparison. Prior work in the single-server setting [79] only applies to FRI-based zk-SNARKs (e.g., Fractal [52]) and cannot be applied to the group-based schemes we consider; hence, a direct comparison is not possible. Note, however, that this prior work actually *increases* the latency of proof generation. In the multi-server setting, there exist schemes that support (adaptations) of Marlin [51, 90, 148], Plonk [66], and HyperPlonk [108]. However, since we do not implement



(a) Latency of proof generation (log-log scale). (b) Improvement in the client's active runtime.

Figure 5.10: Evaluation results for our server-aided Nova.



(a) Latency of proof generation (log-log scale). (b) Improvement in the client's active runtime.

Figure 5.11: Evaluation results for our server-aided Groth16.

	ZKSaaS					
	$S = 8$	$S = 16$	$S = 32$	$S = 64$	$S = 128$	**Our work**
Latency (s)	934	520	271	143	81	320
Cost (USD)	132	92	64	51	50	38

Table 5.3: Latency and cost for 100 Groth16 proofs for a circuit of size $|C| = 2^{19}$.

server-aided proving for those zk-SNARKs, a direct comparison is not possible. Collaborative zk-SNARKs [124] is not a delegation scheme; it neither implements a client role nor improves proof-generation latency.

The only directly comparable work is ZKSaaS [79], which implements and evaluates pri-

vate outsourcing of Groth16 in a multi-server setting. Their scheme provides only semi-honest security, and so we compare against our semi-honest variant. ZKSaaS uses two types of servers: a single powerful server that performs most of the work, and many smaller servers that assist. We assume the powerful server runs on an AWS `r6i.large` instance equivalent to the server in our scheme, and the smaller servers run on AWS `c7i.24xlarge` instances like our client.

ZKSaaS is secure for at most $\frac{S}{4} - 1$ corrupted servers; hence, they require $S \geq 8$ servers to tolerate a single corruption. The latency improvement in ZKSaaS scales with the number of servers; for 2^{19} constraints their improvement factor is lower than ours unless $S > 25$. In Table 5.3, we compare the latency and financial cost of generating 100 Groth16 proofs for a relation with 2^{19} constraints using ZKSaaS and our scheme. As shown, their cost decreases as n increases because the cost of the largest server dominates and the reduction in latency lowers the overall deployment cost. For a huge number of servers ($S \geq 2^{22}$), their cost converges to $\approx \$40$. We do not report client work for ZKSaaS as it was not benchmarked; however, in their scheme, the client only secret-shares the witnesses, so we expect lower client costs than in our scheme.

Server-aided Plonk. Our library does not currently support server-aided Plonk. Plonk is a five-round protocol in which the prover commits to nine polynomials—with a degree bound equal to the constraint size—and performs field operations such as partial products, polynomial multiplication, and division. Prior work [79, Figure 7] shows that for constraint sizes $n \in \{2^{15}, 2^{16}, \ldots, 2^{21}\}$, field operations account for 15–30% of the overall proving computation. Based on this and our EKZG benchmarks (cf. Table 5.2), we estimate that our scheme for Plonk would reduce the client's work by up to $5\times$ and latency by up to $4\times$; deferring FFTs to the last round could reduce the latency by up to $5\times$.

Chapter 6

Conclusion and Future Work

In this dissertation, we first investigated two applications of recursive zk-SNARKs. We proposed KAIZEN, a zk-PoT for neural networks, and showed how recursive zk-SNARKs enabled succinct and efficient zk-PoTs despite the high computational complexity of training such models. We then proposed WILLOWFOLD, a secure aggregation scheme with a lightweight committee cost, and showed how recursive zk-SNARKs can address the linear committee-cost barrier in single-server secure aggregation protocols. Finally, we studied server-aided zk-SNARKs and showed how to realize single-server private outsourcing for widely deployed zk-SNARKs with minimal cryptographic overhead and strong security and privacy guarantees. The ideas in this dissertation may also inspire future work. Below, we suggest several potential research directions.

Proofs of Training. A zk-PoT is a powerful primitive with various applications. However, realizing this notion in practice for real-world models, such as LLMs at deployment scale [123], remains expensive. A promising direction for future work is to investigate weaker notions of zk-PoT that may enable more efficient constructions. For instance, recent work [136] introduced a relaxed notion based on *optimality vicinity*: instead of proving the correctness of the training procedure, it proves that the trained model lies sufficiently close to the mathematically optimal solution of the associated convex optimization problem. This approach, however, applies only to relatively simple models such as regression models. Extending similar relaxations to complex models, such as deep neural networks, remains an interesting open question.

Secure Aggregation. Compared to prior single-server secure aggregation protocols, our scheme WILLOWFOLD has two main drawbacks: (i) it requires a trusted setup, although this setup is updatable, and (ii) it imposes a relatively high cost on the client. A promising direction for future work is to study instantiations of WILLOWFOLD using linear-time, hash-based PCD schemes [41], e.g., the PCD scheme introduced in Chapter 3. Such schemes offer transparent setup and faster proving than group-based constructions. In addition, because they rely only on symmetric-key assumptions, and thus, they offer post-quantum security. The main obstacle is that no sublinear hash-based PCD scheme is currently known. Furthermore, in existing hash-based schemes, the cost of zero knowledge grows linearly with the constraint size.

Server-Aided zk-SNARKs. A limitation of our schemes is that the client's memory usage remains unchanged relative to local proving; server-aided proving improves only the client's runtime and proof-generation latency. In practical deployments, however, memory is often a critical bottleneck. This raises the open question of whether server-aided proving schemes can be designed to reduce client memory usage as well. In group-based zk-SNARKs, memory is typically dominated by two tasks: (i) storing MSM bases, and (ii) computing FFTs. Developing efficient schemes for outsourcing these tasks to a single server remains an interesting open problem.

Another limitation of our work is the relatively high failure probability for RAA codes in our analysis. A natural direction for future work is to obtain a tighter analysis of RAA codes that improves this bound. It would also be interesting to study the concrete security and performance of augmenting the sampling procedure for RAA generators with the tests proposed in [38].

Bibliography

[1] Circom circuits library for machine learning, 2023. `https://github.com/socathie/circomlib-ml`.

[2] Kasra Abbaszadeh, Hossein Hafezi, Jonathan Katz, and Sarah Meiklejohn. Single-server private outsourcing of zk-SNARKs. Cryptology ePrint Archive, Paper 2025/2113, 2025.

[3] Kasra Abbaszadeh, Christodoulos Pappas, Jonathan Katz, and Dimitrios Papadopoulos. Zero-knowledge proofs of training for deep neural networks. In Bo Luo, Xiaojing Liao, Jun Xu, Engin Kirda, and David Lie, editors, *ACM CCS 2024: 31st Conference on Computer and Communications Security*, pages 4316–4330, Salt Lake City, UT, USA, October 14–18, 2024. ACM Press.

[4] Behzad Abdolmaleki, Matteo Campanelli, Quang Dao, and Hamidreza Khoshakhlagh. On the simulation-extractability of proof-carrying data. Cryptology ePrint Archive, Report 2025/2037, 2025.

[5] Surya Addanki, Kevin Garbe, Eli Jaffe, Rafail Ostrovsky, and Antigoni Polychroniadou. Prio+: Privacy preserving aggregate statistics via boolean shares. In Clemente Galdi and Stanislaw Jarecki, editors, *SCN 22: 13th International Conference on Security in Communication Networks*, volume 13409 of *Lecture Notes in Computer Science*, pages 516–539, Amalfi, Italy, September 12–14, 2022. Springer, Cham, Switzerland.

[6] Martin R. Albrecht, Lorenzo Grassi, Christian Rechberger, Arnab Roy, and Tyge Tiessen. MiMC: Efficient encryption and cryptographic hashing with minimal multiplicative complexity. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *Advances in Cryptology – ASIACRYPT 2016, Part I*, volume 10031 of *Lecture Notes in Computer Science*, pages 191–219, Hanoi, Vietnam, December 4–8, 2016. Springer Berlin Heidelberg, Germany.

[7] Michael Alekhnovich. More on average case vs approximation complexity. In *44th Annual Symposium on Foundations of Computer Science*, pages 298–307, Cambridge, MA, USA, October 11–14, 2003. IEEE Computer Society Press.

[8] Aleo. `https://www.aleo.org`.

[9] Diego F. Aranha, Anamaria Costache, Antonio Guimarães, and Eduardo Soria-Vazquez. HELIOPOLIS: Verifiable computation over homomorphically encrypted data from interactive oracle proofs is practical. In Kai-Min Chung and Yu Sasaki, editors, *Advances in Cryptology – ASIACRYPT 2024, Part V*, volume 15488 of *Lecture Notes in Computer Science*, pages 302–334, Kolkata, India, December 9–13, 2024. Springer, Singapore, Singapore.

[10] arkworks contributors. `arkworks` zksnark ecosystem, 2022.

[11] Sanjeev Arora and Rong Ge. New algorithms for learning in presence of errors. In Luca Aceto, Monika Henzinger, and Jiri Sgall, editors, *ICALP 2011: 38th International Colloquium on Automata, Languages and Programming, Part I*, volume 6755 of *Lecture Notes in Computer Science*, pages 403–415, Zurich, Switzerland, July 4–8, 2011. Springer Berlin Heidelberg, Germany.

[12] Aztec. `https://aztec.network`.

[13] David Balbás, Dario Fiore, Maria Isabel González Vasco, Damien Robissout, and Claudio Soriente. Modular sumcheck proofs with applications to machine learning and image processing. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, CCS '23, page 1437–1451, New York, NY, USA, 2023. Association for Computing Machinery.

[14] Foteini Baldimtsi, Konstantinos Kryptos Chalkias, Yan Ji, Jonas Lindstrøm, Deepak Maram, Ben Riva, Arnab Roy, Mahdi Sedaghat, and Joy Wang. zklogin: Privacy-preserving blockchain authentication with existing credentials. In *ACM CCS 2024: 31st Conference on Computer and Communications Security*, page 3182–3196. ACM Press, 2024.

[15] James Bell, Adrià Gascón, Tancrède Lepoint, Baiyu Li, Sarah Meiklejohn, Mariana Raykova, and Cathie Yun. ACORN: Input validation for secure aggregation. In Joseph A. Calandrino and Carmela Troncoso, editors, *USENIX Security 2023: 32nd USENIX Security Symposium*, pages 4805–4822, Anaheim, CA, USA, August 9–11, 2023. USENIX Association.

[16] James Henry Bell, Kallista A. Bonawitz, Adrià Gascón, Tancrède Lepoint, and Mariana Raykova. Secure single-server aggregation with (poly)logarithmic overhead. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *ACM CCS 2020: 27th Conference on Computer and Communications Security*, pages 1253–1269, Virtual Event, USA, November 9–13, 2020. ACM Press.

[17] James Bell-Clark, Adrià Gascón, Baiyu Li, Mariana Raykova, and Phillipp Schoppmann. Willow: Secure aggregation with one-shot clients. In Yael Tauman Kalai and Seny F. Kamara, editors, *Advances in Cryptology – CRYPTO 2025, Part VIII*, volume 16007 of *Lecture Notes in Computer Science*, pages 285–318, Santa Barbara, CA, USA, August 17–21, 2025. Springer, Cham, Switzerland.

[18] Mihir Bellare and Daniele Micciancio. A new paradigm for collision-free hashing: Incrementality at reduced cost. In Walter Fumy, editor, *Advances in Cryptology – EUROCRYPT'97*, volume 1233 of *Lecture Notes in Computer Science*, pages 163–192, Konstanz, Germany, May 11–15, 1997. Springer Berlin Heidelberg, Germany.

[19] Alexandre Belling, Azam Soleimanian, and Olivier Bégassat. Recursion over public-coin interactive proof systems; faster hash verification. In *ACM Conf. Computer and Communications Security*. ACM, 2023.

[20] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Fast reed-solomon interactive oracle proofs of proximity. In *Intl. Colloquium on Automata, Languages, and Programming (ICALP)*, volume 107 of *LIPIcs*, pages 14:1–14:17. Schloss Dagstuhl—Leibniz-Zentrum für Informatik, 2018.

[21] Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from Bitcoin. In *2014 IEEE Symposium on Security and Privacy*, pages 459–474, Berkeley, CA, USA, May 18–21, 2014. IEEE Computer Society Press.

[22] Eli Ben-Sasson, Alessandro Chiesa, and Nicholas Spooner. Interactive oracle proofs. In *Theory of Cryptography (TCC)*, pages 31–60. Springer, 2016.

[23] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. Scalable zero knowledge via cycles of elliptic curves. In *Advances in Cryptology—Crypto 2014*, pages 276–294. Springer, 2014.

[24] Fabrice Benhamouda, Caicai Chen, Shai Halevi, Yuval Ishai, Hugo Krawczyk, Tamer Mour, Tal Rabin, and Alon Rosen. Encrypted matrix-vector products from secret dual codes. Cryptology ePrint Archive, Report 2025/858, 2025.

[25] Dario Bini and Victor Pan. Polynomial division and its computational complexity. *Journal of Complexity*, 2(3):179–203, 1986.

[26] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. Recursive composition and bootstrapping for SNARKS and proof-carrying data. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *45th Annual ACM Symposium on Theory of Computing*, pages 111–120, Palo Alto, CA, USA, June 1–4, 2013. ACM Press.

[27] Alexander R. Block, Zhiyong Fang, Jonathan Katz, Justin Thaler, Hendrik Waldner, and Yupeng Zhang. Field-agnostic SNARKs from expand-accumulate codes. In Leonid Reyzin and Douglas Stebila, editors, *Advances in Cryptology – CRYPTO 2024, Part X*, volume 14929 of *Lecture Notes in Computer Science*, pages 276–307, Santa Barbara, CA, USA, August 18–22, 2024. Springer, Cham, Switzerland.

[28] Avrim Blum, Merrick L. Furst, Michael J. Kearns, and Richard J. Lipton. Cryptographic primitives based on hard learning problems. In Douglas R. Stinson, editor, *Advances in Cryptology – CRYPTO'93*, volume 773 of *Lecture Notes in Computer Science*, pages 278–291, Santa Barbara, CA, USA, August 22–26, 1994. Springer Berlin Heidelberg, Germany.

[29] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H. Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. Practical secure aggregation for privacy-preserving machine learning. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017: 24th Conference on Computer and Communications Security*, pages 1175–1191, Dallas, TX, USA, October 31 – November 2, 2017. ACM Press.

[30] Dan Boneh, Justin Drake, Ben Fisch, and Ariel Gabizon. Halo infinite: Proof-carrying data from additive polynomial commitments. In *Advances in Cryptology—Crypto 2021, Part I*, pages 649–680. Springer, 2021.

[31] Jonathan Bootle, Alessandro Chiesa, and Katerina Sotiraki. Sumcheck arguments and their applications. In *Advances in Cryptology—Crypto 2021, Part I*, page 742–773. Springer, 2021.

[32] Sean Bowe, Alessandro Chiesa, Matthew Green, Ian Miers, Pratyush Mishra, and Howard Wu. ZEXE: Enabling decentralized private computation. In *2020 IEEE Symposium on Security and Privacy*, pages 947–964, San Francisco, CA, USA, May 18–21, 2020. IEEE Computer Society Press.

[33] Sean Bowe, Jack Grigg, and Daira Hopwood. Recursive proof composition without a trusted setup. Cryptology ePrint Archive, Paper 2019/1021, 2019.

[34] Elette Boyle, Geoffroy Couteau, Niv Gilboa, and Yuval Ishai. Compressing vector OLE. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018: 25th Conference on Computer and Communications Security*, pages 896–912, Toronto, ON, Canada, October 15–19, 2018. ACM Press.

[35] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, Nicolas Resch, and Peter Scholl. Correlated pseudorandomness from expand-accumulate codes. In Yevgeniy Dodis and Thomas Shrimpton, editors, *Advances in Cryptology – CRYPTO 2022, Part II*, volume 13508 of *Lecture Notes in Computer Science*, pages 603–633, Santa Barbara, CA, USA, August 15–18, 2022. Springer, Cham, Switzerland.

[36] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, Peter Rindal, and Peter Scholl. Efficient two-round OT extension and silent non-interactive secure computation. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019: 26th Conference on Computer and Communications Security*, pages 291–308, London, UK, November 11–15, 2019. ACM Press.

[37] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Correlated pseudorandom functions from variable-density LPN. In *61st Annual Symposium on Foundations of Computer Science*, pages 1069–1080, Durham, NC, USA, November 16–19, 2020. IEEE Computer Society Press.

[38] Martijn Brehm, Binyi Chen, Ben Fisch, Nicolas Resch, Ron D. Rothblum, and Hadas Zeilberger. Blaze: Fast SNARKs from interleaved RAA codes. In Serge Fehr and Pierre-Alain Fouque, editors, *Advances in Cryptology – EUROCRYPT 2025, Part IV*, volume 15604 of *Lecture Notes in Computer Science*, pages 123–152, Madrid, Spain, May 4–8, 2025. Springer, Cham, Switzerland.

[39] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *2018 IEEE Symposium on Security and Privacy*, pages 315–334, San Francisco, CA, USA, May 21–23, 2018. IEEE Computer Society Press.

[40] Benedikt Bünz and Binyi Chen. Protostar: Generic efficient accumulation/folding for special-sound protocols. In Jian Guo and Ron Steinfeld, editors, *Advances in Cryptology – ASIACRYPT 2023, Part II*, volume 14439 of *Lecture Notes in Computer Science*, pages 77–110, Guangzhou, China, December 4–8, 2023. Springer, Singapore, Singapore.

[41] Benedikt Bünz, Alessandro Chiesa, Giacomo Fenzi, and William Wang. Linear-time accumulation schemes. In Benny Applebaum and Huijia (Rachel) Lin, editors, *TCC 2025: 23rd Theory of Cryptography Conference, Part I*, volume 16268 of *Lecture Notes in Computer Science*, pages 369–399, Aarhus, Denmark, December 1–5, 2025. Springer, Cham, Switzerland.

[42] Benedikt Bünz, Alessandro Chiesa, William Lin, Pratyush Mishra, and Nicholas Spooner. Proof-carrying data without succinct arguments. In Tal Malkin and Chris Peikert, editors, *Advances in Cryptology – CRYPTO 2021, Part I*, volume 12825 of *Lecture Notes in Computer Science*, pages 681–710, Virtual Event, August 16–20, 2021. Springer, Cham, Switzerland.

[43] Benedikt Bünz and Jessica Chen. Proofs for deep thought: Accumulation for large memories and deterministic computations. Cryptology ePrint Archive, Paper 2024/325, 2024. `https://eprint.iacr.org/2024/325`.

[44] Benedikt Bünz, Pratyush Mishra, Wilson Nguyen, and William Wang. Accumulation without homomorphism. Cryptology ePrint Archive, Paper 2024/474, 2024. `https://eprint.iacr.org/2024/474`.

[45] Matteo Campanelli, Dario Fiore, and Anaïs Querol. Legosnark: Modular design and composition of succinct zero-knowledge proofs. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, CCS '19, page 2075–2092, New York, NY, USA, 2019. Association for Computing Machinery.

[46] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd Annual Symposium on Foundations of Computer Science*, pages 136–145, Las Vegas, NV, USA, October 14–17, 2001. IEEE Computer Society Press.

[47] Gian Carlo Cardarilli, Marco Re, and Luca Di Nunzio. A pseudo-softmax function for hardware-based high speed image classification. *Sci. Reports*, 2021.

[48] Binyi Chen, Benedikt Bünz, Dan Boneh, and Zhenfei Zhang. HyperPlonk: Plonk with linear-time prover and high-degree custom gates. In Carmit Hazay and Martijn Stam, editors, *Advances in Cryptology – EUROCRYPT 2023, Part II*, volume 14005 of *Lecture Notes in Computer Science*, pages 499–530, Lyon, France, April 23–27, 2023. Springer, Cham, Switzerland.

[49] Alessandro Chiesa, Michael A. Forbes, and Nicholas Spooner. A zero knowledge sumcheck and its applications. Cryptology ePrint Archive, Report 2017/305, 2017.

[50] Alessandro Chiesa, Yuncong Hu, Mary Maller, Pratyush Mishra, Psi Vesely, and Nicholas P. Ward. Marlin: Preprocessing zkSNARKs with universal and updatable SRS.

In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology – EUROCRYPT 2020, Part I*, volume 12105 of *Lecture Notes in Computer Science*, pages 738–768, Zagreb, Croatia, May 10–14, 2020. Springer, Cham, Switzerland.

[51] Alessandro Chiesa, Ryan Lehmkuhl, Pratyush Mishra, and Yinuo Zhang. Eos: Efficient private delegation of zkSNARK provers. In Joseph A. Calandrino and Carmela Troncoso, editors, *USENIX Security 2023: 32nd USENIX Security Symposium*, pages 6453–6469, Anaheim, CA, USA, August 9–11, 2023. USENIX Association.

[52] Alessandro Chiesa, Dev Ojha, and Nicholas Spooner. Fractal: Post-quantum and transparent recursive proofs from holography. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology – EUROCRYPT 2020, Part I*, volume 12105 of *Lecture Notes in Computer Science*, pages 769–793, Zagreb, Croatia, May 10–14, 2020. Springer, Cham, Switzerland.

[53] Alessandro Chiesa and Eran Tromer. Proof-carrying data and hearsay arguments from signature cards. In Andrew Chi-Chih Yao, editor, *ICS 2010: 1st Innovations in Computer Science*, pages 310–331, Tsinghua University, Beijing, China, January 5–7, 2010. Tsinghua University Press.

[54] Amrita Roy Chowdhury, Chuan Guo, Somesh Jha, and Laurens van der Maaten. EIFFeL: Ensuring integrity for federated learning. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *ACM CCS 2022: 29th Conference on Computer and Communications Security*, pages 2535–2549, Los Angeles, CA, USA, November 7–11, 2022. ACM Press.

[55] Henry Corrigan-Gibbs and Dan Boneh. Prio: private, robust, and scalable computation of aggregate statistics. page 259–282.

[56] Geoffroy Couteau, Peter Rindal, and Srinivasan Raghuraman. Silver: Silent VOLE and oblivious transfer from hardness of decoding structured LDPC codes. In Tal Malkin and Chris Peikert, editors, *Advances in Cryptology – CRYPTO 2021, Part III*, volume 12827 of *Lecture Notes in Computer Science*, pages 502–534, Virtual Event, August 16–20, 2021. Springer, Cham, Switzerland.

[57] Alhad Daftardar, Jianqiao Mo, Joey Ah-kiow, Benedikt Bünz, Ramesh Karri, Siddharth Garg, and Brandon Reagen. Need for zkSpeed: Accelerating HyperPlonk for zero-knowledge proofs. In *ISCA 2025*, pages 1986–2001. ACM, 2025.

[58] Quang Dao and Paul Grubbs. Spartan and bulletproofs are simulation-extractable (for free!). In Carmit Hazay and Martijn Stam, editors, *Advances in Cryptology – EUROCRYPT 2023, Part II*, volume 14005 of *Lecture Notes in Computer Science*, pages 531–562, Lyon, France, April 23–27, 2023. Springer, Cham, Switzerland.

[59] Trisha Datta, Binyi Chen, and Dan Boneh. Veritas: Verifying image transformations at scale. In *2025 IEEE Symposium on Security and Privacy (SP)*, pages 4606–4623, 2025.

[60] Li Deng. The MNIST database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

[61] Thorsten Eisenhofer, Doreen Riepel, Varun Chandrasekaran, Esha Ghosh, Olga Ohrimenko, and Nicolas Papernot. Verifiable and provably secure machine unlearning, 2023.

[62] Alessandro Epasto, Mohammad Mahdian, Vahab Mirrokni, and Manolis Zampetakis. Optimal approximation-smoothness tradeoffs for soft-max functions. In *Neural Information Proc. Systems (NeurIPS)*. Curran Associates Inc., 2020.

[63] Jens Ernstberger, Stefanos Chaliasos, George Kadianakis, Sebastian Steinhorst, Philipp Jovanovic, Arthur Gervais, Benjamin Livshits, and Michele Orrù. zk-Bench: A toolset for comparative evaluation and performance benchmarking of SNARKs. In Clemente Galdi and Duong Hieu Phan, editors, *Security and Cryptography for Networks (SCN), Part I*, volume 14973 of *LNCS*, pages 46–72. Springer, 2024.

[64] Congyu Fang, Hengrui Jia, Anvith Thudi, Mohammad Yaghini, Christopher A. Choquette-Choo, Natalie Dullerud, Varun Chandrasekaran, and Nicolas Papernot. On the fundamental limits of formally (dis)proving robustness in proof-of-learning, 2022.

[65] Congyu Fang, Hengrui Jia, Anvith Thudi, Mohammad Yaghini, Christopher A. Choquette-Choo, Natalie Dullerud, Varun Chandrasekaran, and Nicolas Papernot. Proof-of-learning is currently more broken than you think. In *European Symposium on Security and Privacy*, pages 797–816. IEEE, 2023.

[66] Zhiyong Fang, Sanjam Garg, Bhaskar Roberts, Wenxuan Wu, and Yupeng Zhang. Collaborative zkSNARKs with sublinear prover time and constant proof size. Cryptology ePrint Archive, Report 2025/1388, 2025.

[67] Fermah. `https://www.fermah.xyz`.

[68] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *Advances in Cryptology – CRYPTO'86*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194, Santa Barbara, CA, USA, August 1987. Springer Berlin Heidelberg, Germany.

[69] Benchmarks for TFHE-rs. `https://docs.zama.ai/tfhe-rs/get-started/benchmarks`.

[70] Georg Fuchsbauer, Eike Kiltz, and Julian Loss. The algebraic group model and its applications. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018, Part II*, volume 10992 of *Lecture Notes in Computer Science*, pages 33–62, Santa Barbara, CA, USA, August 19–23, 2018. Springer, Cham, Switzerland.

[71] Ariel Gabizon and Zachary J. Williamson. plookup: A simplified polynomial protocol for lookup tables. Cryptology ePrint Archive, Report 2020/315, 2020.

[72] Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru. PLONK: Permutations over Lagrange-bases for oecumenical noninteractive arguments of knowledge. Cryptology ePrint Archive, Report 2019/953, 2019.

[73] Mariana Gama, Emad Heydari Beni, Jiayi Kang, Jannik Spiessens, and Frederik Vercauteren. Blind zkSNARKs for private proof delegation and verifiable computation over encrypted data. Cryptology ePrint Archive, Paper 2024/1684, 2024.

[74] Chaya Ganesh, Hamidreza Khoshakhlagh, Markulf Kohlweiss, Anca Nitulescu, and Michal Zajac. What makes fiat-shamir zkSNARKs (updatable SRS) simulation extractable? In Clemente Galdi and Stanislaw Jarecki, editors, *SCN 22: 13th International Conference on Security in Communication Networks*, volume 13409 of *Lecture Notes in Computer Science*, pages 735–760, Amalfi, Italy, September 12–14, 2022. Springer, Cham, Switzerland.

[75] Sanjam Garg, Aarushi Goel, Abhishek Jain, Guru-Vamsi Policharla, and Sruthi Sekar. zkSaaS: Zero-knowledge SNARKs as a service. In Joseph A. Calandrino and Carmela Troncoso, editors, *USENIX Security 2023: 32nd USENIX Security Symposium*, pages 4427–4444, Anaheim, CA, USA, August 9–11, 2023. USENIX Association.

[76] Sanjam Garg, Aarushi Goel, Abhishek Jain, Bhaskar Roberts, and Sruthi Sekar. Malicious security in collaborative zk-SNARKs: More than meets the eye. In *Adv. in Cryptology—Crypto 2025, Part VII*, volume 16006 of *Lecture Notes in Computer Science*, pages 396–428, 2025. Available at https://ia.cr/2025/1026.

[77] Sanjam Garg, Aarushi Goel, Somesh Jha, Saeed Mahloujifar, Mohammad Mahmoody, Guru-Vamsi Policharla, and Mingyuan Wang. Experimenting with zero-knowledge proofs of training. In Weizhi Meng, Christian Damsgaard Jensen, Cas Cremers, and Engin Kirda, editors, *ACM CCS 2023: 30th Conference on Computer and Communications Security*, pages 1880–1894, Copenhagen, Denmark, November 26–30, 2023. ACM Press.

[78] Sanjam Garg, Aarushi Goel, Dimitris Kolonelos, and Rohit Sinha. Jigsaw: Doubly private smart contracts.

[79] Sanjam Garg, Aarushi Goel, and Mingyuan Wang. How to prove statements obliviously? In Leonid Reyzin and Douglas Stebila, editors, *Advances in Cryptology – CRYPTO 2024, Part X*, volume 14929 of *Lecture Notes in Computer Science*, pages 449–487, Santa Barbara, CA, USA, August 18–22, 2024. Springer, Cham, Switzerland.

[80] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct NIZKs without PCPs. In Thomas Johansson and Phong Q. Nguyen, editors, *Advances in Cryptology – EUROCRYPT 2013*, volume 7881 of *Lecture Notes in Computer Science*, pages 626–645, Athens, Greece, May 26–30, 2013. Springer Berlin Heidelberg, Germany.

[81] Gevulot. `https://www.gevulot.com`.

[82] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. Delegating computation: Interactive proofs for muggles. In *ACM Symposium on Theory of Computing*, pages 113–122. ACM, 2008.

[83] Alexander Golovnev, Jonathan Lee, Srinath T. V. Setty, Justin Thaler, and Riad S. Wahby. Brakedown: Linear-time and field-agnostic SNARKs for R1CS. In Helena Handschuh and Anna Lysyanskaya, editors, *Advances in Cryptology – CRYPTO 2023, Part II*, volume 14082 of *Lecture Notes in Computer Science*, pages 193–226, Santa Barbara, CA, USA, August 20–24, 2023. Springer, Cham, Switzerland.

[84] Google. Increasing transparency in ai security, 2023. `https://security.googleblog.com/2023/10/increasing-transparency-in-ai-security.html`.

[85] Lorenzo Grassi, Dmitry Khovratovich, Christian Rechberger, Arnab Roy, and Markus Schofnegger. Poseidon: A new hash function for zero-knowledge proof systems. In Michael Bailey and Rachel Greenstadt, editors, *USENIX Security 2021: 30th USENIX Security Symposium*, pages 519–535. USENIX Association, August 11–13, 2021.

[86] Jens Groth. On the size of pairing-based non-interactive arguments. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology – EUROCRYPT 2016, Part II*, volume 9666 of *Lecture Notes in Computer Science*, pages 305–326, Vienna, Austria, May 8–12, 2016. Springer Berlin Heidelberg, Germany.

[87] Hossein Hafezi, Kasra Abbaszadeh, Adrià Gascón, Phillipp Schoppmann, Mariana Raykova, and Benedikt Bünz. WillowFold: Secure aggregation with a lightweight committee. Cryptology ePrint Archive, Paper 2026/264, 2026.

[88] Hossein Hafezi, Alireza Shirzad, Benedikt Bünz, and Joseph Bonneau. IronDict: Transparent dictionaries from polynomial commitments. Cryptology ePrint Archive, Report 2025/1580, 2025.

[89] Melissa Heikkilä. This artist is dominating ai-generated art. and he's not happy about it. MIT Technology Review, Sept 2022.

[90] Yuncong Hu, Pratyush Mishra, Xiao Wang, Jie Xie, Kang Yang, Yu Yu, and Yuwen Zhang. DFS: Delegation-friendly zkSNARK and private delegation of provers. In *USENIX Security 2025: 34th USENIX Security Symposium*, USA, 2025. USENIX Association.

[91] How Google Messages improves suggestions with federated technology. `https://support.google.com/messages/answer/9327902`.

[92] Hengrui Jia, Mohammad Yaghini, Christopher A. Choquette-Choo, Natalie Dullerud, Anvith Thudi, Varun Chandrasekaran, and Nicolas Papernot. Proof-of-learning: Definitions and practice. In *IEEE Symposium on Security and Privacy*, pages 1039–1056. IEEE, 2021.

[93] George Kadianakis, Arantxa Zapico, Hossein Hafezi, and Benedikt Bünz. KZH-fold: Accountable voting from sublinear accumulation. In Chun-Ying Huang, Jyh-Cheng Chen, Shiuh-Pyng Shieh, David Lie, and Véronique Cortier, editors, *ACM CCS 2025: 32nd Conference on Computer and Communications Security*, pages 409–422, Taipei, Taiwan, October 13–17, 2025. ACM Press.

[94] Harish Karthikeyan and Antigoni Polychroniadou. OPA: One-shot private aggregation with single client interaction and its applications to federated learning. In Yael Tauman Kalai and Seny F. Kamara, editors, *Advances in Cryptology – CRYPTO 2025, Part VIII*, volume 16007 of *Lecture Notes in Computer Science*, pages 319–353, Santa Barbara, CA, USA, August 17–21, 2025. Springer, Cham, Switzerland.

[95] Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. Constant-size commitments to polynomials and their applications. In Masayuki Abe, editor, *Advances in Cryptology – ASIACRYPT 2010*, volume 6477 of *Lecture Notes in Computer Science*, pages 177–194, Singapore, December 5–9, 2010. Springer Berlin Heidelberg, Germany.

[96] Abhiram Kothapalli, Srinath Setty, and Ioanna Tzialla. Nova: Recursive zero-knowledge arguments from folding schemes. In Yevgeniy Dodis and Thomas Shrimpton, editors, *Advances in Cryptology – CRYPTO 2022, Part IV*, volume 13510 of *Lecture Notes in Computer Science*, pages 359–388, Santa Barbara, CA, USA, August 15–18, 2022. Springer, Cham, Switzerland.

[97] Abhiram Kothapalli and Srinath T. V. Setty. HyperNova: Recursive arguments for customizable constraint systems. In Leonid Reyzin and Douglas Stebila, editors, *Advances in Cryptology – CRYPTO 2024, Part X*, volume 14929 of *Lecture Notes in Computer Science*, pages 345–379, Santa Barbara, CA, USA, August 18–22, 2024. Springer, Cham, Switzerland.

[98] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. Cifar-10 (canadian institute for advanced research). Available at `http://www.cs.toronto.edu/ kriz/cifar.html`.

[99] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Adv. in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012.

[100] Yan LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[101] Seunghwan Lee, Hankyung Ko, Jihye Kim, and Hyunok Oh. vCNN: Verifiable convolutional neural network based on zk-SNARKs, 2020.

[102] Ryan Lehmkuhl, Henry Corrigan-Gibbs, Emma Dauterman, and David J. Wu. Heli: Heavy-light private aggregation. Cryptology ePrint Archive, Paper 2026/059, 2026.

[103] Hanjun Li, Huijia Lin, Antigoni Polychroniadou, and Stefano Tessaro. LERNA: Secure single-server aggregation via key-homomorphic masking. In Jian Guo and Ron Steinfeld, editors, *Advances in Cryptology – ASIACRYPT 2023, Part I*, volume 14438 of *Lecture Notes in Computer Science*, pages 302–334, Guangzhou, China, December 4–8, 2023. Springer, Singapore, Singapore.

[104] Mu Li, David G. Andersen, Jun Woo Park, Alexander J. Smola, Amr Ahmed, Vanja Josifovski, James Long, Eugene J. Shekita, and Bor-Yiing Su. Scaling distributed machine

learning with the parameter server. In *Conference on Operating Systems Design and Implementation*, pages 583–598. USENIX Association, 2014.

[105] Hanlin Liu, Xiao Wang, Kang Yang, and Yu Yu. The hardness of LPN over any integer ring and field for PCG applications. In Marc Joye and Gregor Leander, editors, *Advances in Cryptology – EUROCRYPT 2024, Part VI*, volume 14656 of *Lecture Notes in Computer Science*, pages 149–179, Zurich, Switzerland, May 26–30, 2024. Springer, Cham, Switzerland.

[106] Tianyi Liu, Tiancheng Xie, Jiaheng Zhang, Dawn Song, and Yupeng Zhang. Pianist: Scalable zkRollups via fully distributed zero-knowledge proofs. In *2024 IEEE Symposium on Security and Privacy*, pages 1777–1793, San Francisco, CA, USA, May 19–23, 2024. IEEE Computer Society Press.

[107] Tianyi Liu, Xiang Xie, and Yupeng Zhang. zkCNN: Zero knowledge proofs for convolutional neural network predictions and accuracy. In Giovanni Vigna and Elaine Shi, editors, *ACM CCS 2021: 28th Conference on Computer and Communications Security*, pages 2968–2985, Virtual Event, Republic of Korea, November 15–19, 2021. ACM Press.

[108] Xuanming Liu, Zhelei Zhou, Yinghao Wang, Yanxin Pang, Jinye He, Bingsheng Zhang, Xiaohu Yang, and Jiaheng Zhang. *Scalable Collaborative zk-SNARK and Its Application to Fully Distributed Proof Delegation*. USENIX Association, USA, 2025.

[109] Carsten Lund, Lance Fortnow, Howard Karloff, and Noam Nisan. Algebraic methods for interactive proof systems. *J. ACM*, 39(4):859–868, oct 1992.

[110] Hidde Lycklama, Lukas Burkhalter, Alexander Viand, Nicolas Küchler, and Anwar Hithnawi. RoFL: Robustness of secure federated learning. In *2023 IEEE Symposium on Security and Privacy*, pages 453–476, San Francisco, CA, USA, May 21–25, 2023. IEEE Computer Society Press.

[111] Vadim Lyubashevsky and Daniel Masny. Man-in-the-middle secure authentication schemes from LPN and weak PRFs. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology – CRYPTO 2013, Part II*, volume 8043 of *Lecture Notes in Computer Science*, pages 308–325, Santa Barbara, CA, USA, August 18–22, 2013. Springer Berlin Heidelberg, Germany.

[112] Yiping Ma, Jess Woods, Sebastian Angel, Antigoni Polychroniadou, and Tal Rabin. Flamingo: Multi-round single-server secure aggregation with applications to private federated learning. In *2023 IEEE Symposium on Security and Privacy*, pages 477–496, San Francisco, CA, USA, May 21–25, 2023. IEEE Computer Society Press.

[113] Varun Madathil, Arthur Lazzaretti, Zeyu Liu, and Charalampos Papamanthou. TACITA: Threshold aggregation without client interaction. Cryptology ePrint Archive, Paper 2025/1579, 2025.

[114] Mary Maller, Sean Bowe, Markulf Kohlweiss, and Sarah Meiklejohn. Sonic: Zero-knowledge SNARKs from linear-size universal and updatable structured reference strings.

In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019: 26th Conference on Computer and Communications Security*, pages 2111–2128, London, UK, November 11–15, 2019. ACM Press.

[115] Ralph C. Merkle. A digital signature based on a conventional encryption function. In *Advances in Cryptology—Crypto '87*, pages 369–378. Springer, 1988.

[116] Silvio Micali. Computationally sound proofs. *SIAM J. Computing*, 30(4):1253–1298, 2000.

[117] Daniele Micciancio and Petros Mol. Pseudorandom knapsacks and the sample complexity of LWE search-to-decision reductions. In Phillip Rogaway, editor, *Advances in Cryptology – CRYPTO 2011*, volume 6841 of *Lecture Notes in Computer Science*, pages 465–484, Santa Barbara, CA, USA, August 14–18, 2011. Springer Berlin Heidelberg, Germany.

[118] Mina. `https://minaprotocol.com`.

[119] Payman Mohassel and Peter Rindal. Aby3: A mixed protocol framework for machine learning. In *Conf. on Computer and Comm. Security*, page 35–52. ACM, 2018.

[120] Assa Naveh and Eran Tromer. PhotoProof: Cryptographic image authentication for any set of permissible transformations. In *2016 IEEE Symposium on Security and Privacy*, pages 255–271, San Jose, CA, USA, May 22–26, 2016. IEEE Computer Society Press.

[121] Nexus. `https://nexus.xyz`.

[122] Wilson D. Nguyen, Trisha Datta, Binyi Chen, Nirvan Tyagi, and Dan Boneh. Mangrove: A scalable framework for folding-based SNARKs. In Leonid Reyzin and Douglas Stebila, editors, *Advances in Cryptology – CRYPTO 2024, Part X*, volume 14929 of *Lecture Notes in Computer Science*, pages 308–344, Santa Barbara, CA, USA, August 18–22, 2024. Springer, Cham, Switzerland.

[123] OpenAI. Introducing gpt-5. `https://openai.com/index/introducing-gpt-5/`, August 2025. Accessed: 2026-03-19.

[124] Alex Ozdemir and Dan Boneh. Experimenting with collaborative zk-SNARKs: Zero-knowledge proofs for distributed secrets. In Kevin R. B. Butler and Kurt Thomas, editors, *USENIX Security 2022: 31st USENIX Security Symposium*, pages 4291–4308, Boston, MA, USA, August 10–12, 2022. USENIX Association.

[125] Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In Joan Feigenbaum, editor, *Advances in Cryptology – CRYPTO'91*, volume 576 of *Lecture Notes in Computer Science*, pages 129–140, Santa Barbara, CA, USA, August 11–15, 1992. Springer Berlin Heidelberg, Germany.

[126] Proto-Danksharding. `https://www.eip4844.com`.

[127] Srinivasan Raghuraman, Peter Rindal, and Titouan Tanguy. Expand-convolute codes for pseudorandom correlation generators from LPN. In Helena Handschuh and Anna Lysyanskaya, editors, *Advances in Cryptology – CRYPTO 2023, Part IV*, volume 14084 of *Lecture Notes in Computer Science*, pages 602–632, Santa Barbara, CA, USA, August 20–24, 2023. Springer, Cham, Switzerland.

[128] Herbert Robbins and Sutton Monro. A stochastic approximation method. *The Annals of Mathematical Statistics*, 22(3):400 – 407, 1951.

[129] Michael Rosenberg, Tushar Mopuri, Hossein Hafezi, Ian Miers, and Pratyush Mishra. Hekaton: Horizontally-scalable zkSNARKs via proof aggregation. In Bo Luo, Xiaojing Liao, Jun Xu, Engin Kirda, and David Lie, editors, *ACM CCS 2024: 31st Conference on Computer and Communications Security*, pages 929–940, Salt Lake City, UT, USA, October 14–18, 2024. ACM Press.

[130] Michael Rosenberg, Jacob D. White, Christina Garman, and Ian Miers. zk-creds: Flexible anonymous credentials from zkSNARKs and existing identity infrastructure. In *2023 IEEE Symposium on Security and Privacy*, pages 790–808, San Francisco, CA, USA, May 21–25, 2023. IEEE Computer Society Press.

[131] Nikola Samardzic, Simon Langowski, Srinivas Devadas, and Daniel Sanchez. Accelerating zero-knowledge proofs through hardware-algorithm co-design. In *2024 57th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 366–379, 2024.

[132] Srinath Setty. Spartan: Efficient and general-purpose zkSNARKs without trusted setup. In Daniele Micciancio and Thomas Ristenpart, editors, *Advances in Cryptology – CRYPTO 2020, Part III*, volume 12172 of *Lecture Notes in Computer Science*, pages 704–737, Santa Barbara, CA, USA, August 17–21, 2020. Springer, Cham, Switzerland.

[133] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *Intl. Conf. on Learning Representations, ICLR*, 2015.

[134] Succinct. `https://testnet.succinct.xyz/prove`.

[135] TACEO. `https://taceo.io`.

[136] Gefei Tan, Adrià Gascón, Sarah Meiklejohn, Mariana Raykova, Xiao Wang, and Ning Luo. Founding zero-knowledge proof of training on optimum vicinity. In Chun-Ying Huang, Jyh-Cheng Chen, Shiuh-Pyng Shieh, David Lie, and Véronique Cortier, editors, *ACM CCS 2025: 32nd Conference on Computer and Communications Security*, pages 1173–1187, Taipei, Taiwan, October 13–17, 2025. ACM Press.

[137] Justin Thaler. Time-optimal interactive proofs for circuit evaluation. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology – CRYPTO 2013, Part II*, volume 8043 of *Lecture Notes in Computer Science*, pages 71–89, Santa Barbara, CA, USA, August 18–22, 2013. Springer Berlin Heidelberg, Germany.

[138] Florian Tramèr, Fan Zhang, Ari Juels, Michael K. Reiter, and Thomas Ristenpart. Stealing machine learning models via prediction apis. In *USENIX Security Symposium*, page 601–618, USA, 2016. USENIX Association.

[139] Nirvan Tyagi, Ben Fisch, Andrew Zitek, Joseph Bonneau, and Stefano Tessaro. VeRSA: Verifiable registries with efficient client audits from RSA authenticated dictionaries. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *ACM CCS 2022: 29th Conference on Computer and Communications Security*, pages 2793–2807, Los Angeles, CA, USA, November 7–11, 2022. ACM Press.

[140] Paul Valiant. Incrementally verifiable computation or proofs of knowledge imply time/space efficiency. In Ran Canetti, editor, *TCC 2008: 5th Theory of Cryptography Conference*, volume 4948 of *Lecture Notes in Computer Science*, pages 1–18, San Francisco, CA, USA, March 19–21, 2008. Springer Berlin Heidelberg, Germany.

[141] David Wagner. A generalized birthday problem. In Moti Yung, editor, *Advances in Cryptology – CRYPTO 2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 288–303, Santa Barbara, CA, USA, August 18–22, 2002. Springer Berlin Heidelberg, Germany.

[142] Riad S. Wahby, Ioanna Tzialla, abhi shelat, Justin Thaler, and Michael Walfish. Doubly-efficient zkSNARKs without trusted setup. In *2018 IEEE Symposium on Security and Privacy*, pages 926–943, San Francisco, CA, USA, May 21–23, 2018. IEEE Computer Society Press.

[143] Chenkai Weng, Kang Yang, Xiang Xie, Jonathan Katz, and Xiao Wang. Mystique: Efficient conversions for zero-knowledge proofs with applications to machine learning. In *USENIX Security Symposium*, pages 501–518. USENIX Association, 2021.

[144] Predicting Text Selections with Federated Learning. `https://research.google/blog/predicting-text-selections-with-federated-learning/`.

[145] Howard Wu, Wenting Zheng, Alessandro Chiesa, Raluca Ada Popa, and Ion Stoica. DIZK: A distributed zero knowledge proof system. In William Enck and Adrienne Porter Felt, editors, *USENIX Security 2018: 27th USENIX Security Symposium*, pages 675–692, Baltimore, MD, USA, August 15–17, 2018. USENIX Association.

[146] Tiancheng Xie, Jiaheng Zhang, Yupeng Zhang, Charalampos Papamanthou, and Dawn Song. Libra: Succinct zero-knowledge proofs with optimal prover computation. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology – CRYPTO 2019, Part III*, volume 11694 of *Lecture Notes in Computer Science*, pages 733–764, Santa Barbara, CA, USA, August 18–22, 2019. Springer, Cham, Switzerland.

[147] Tiancheng Xie, Yupeng Zhang, and Dawn Song. Orion: Zero knowledge proof with linear prover time. In Yevgeniy Dodis and Thomas Shrimpton, editors, *Advances in Cryptology – CRYPTO 2022, Part IV*, volume 13510 of *Lecture Notes in Computer Science*, pages 299–328, Santa Barbara, CA, USA, August 15–18, 2022. Springer, Cham, Switzerland.

[148] Yunbo Yang, Yuejia Cheng, Kailun Wang, Xiaoguo Li, Jianfei Sun, Jiachen Shen, Xiaolei Dong, Zhenfu Cao, Guomin Yang, and Robert H. Deng. Siniel: Distributed privacy-preserving zkSNARK. In *ISOC Network and Distributed System Security Symposium – NDSS 2025*, San Diego, CA, USA, February 24–28, 2025. The Internet Society.

[149] Arantxa Zapico, Vitalik Buterin, Dmitry Khovratovich, Mary Maller, Anca Nitulescu, and Mark Simkin. Caulk: Lookup arguments in sublinear time. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *ACM CCS 2022: 29th Conference on Computer and Communications Security*, pages 3121–3134, Los Angeles, CA, USA, November 7–11, 2022. ACM Press.

[150] Zcash. `https://z.cash`.

[151] Bin Zhang, Lin Jiao, and Mingsheng Wang. Faster algorithms for solving LPN. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology – EUROCRYPT 2016, Part I*, volume 9665 of *Lecture Notes in Computer Science*, pages 168–195, Vienna, Austria, May 8–12, 2016. Springer Berlin Heidelberg, Germany.

[152] Jiaheng Zhang, Tianyi Liu, Weijie Wang, Yinuo Zhang, Dawn Song, Xiang Xie, and Yupeng Zhang. Doubly efficient interactive proofs for general arithmetic circuits with linear prover time. In *Conf. on Computer and Communications Security*, page 159–177. ACM, 2021.

[153] Jiaheng Zhang, Tiancheng Xie, Yupeng Zhang, and Dawn Song. Transparent polynomial delegation and its applications to zero knowledge proof. In *IEEE Symposium on Security and Privacy*, pages 859–876, 2020.

[154] Zongyang Zhang, Weihan Li, Yanpei Guo, Kexin Shi, Sherman S. M. Chow, Ximeng Liu, and Jin Dong. Fast RS-IOP multivariate polynomial commitments and verifiable secret sharing. In Davide Balzarotti and Wenyuan Xu, editors, *USENIX Security 2024: 33rd USENIX Security Symposium*, Philadelphia, PA, USA, August 14–16, 2024. USENIX Association.