# Single-Server Private Outsourcing of zk-SNARKs

Kasra Abbaszadeh*
University of Maryland
kasraz@umd.edu

Hossein Hafezi
New York University
h.hafezi@nyu.edu

Jonathan Katz
Google
jkatz2@gmail.com

Sarah Meiklejohn
Google & UCL
s.meiklejohn@ucl.ac.uk

**Abstract**

Succinct zero-knowledge arguments (zk-SNARKs) enable a prover to convince a verifier of the truth of a statement via a succinct and efficiently verifiable proof without revealing any additional information about the secret witness. A barrier to practical deployment of zk-SNARKs is their high proving cost. With this motivation, we study *server-aided* zk-SNARKs, where a client/prover outsources most of its work to a *single, untrusted* server while the server learns nothing about the witness or even the proof. We formalize this notion and show how to realize server-aided proving for widely deployed zk-SNARKs, including 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× and reduce the proving latency by up to 9×.

*Portions of this work were done while at Google. Work done at the University of Maryland was supported in part by NSF award CNS-2154705.

# Contents

# 1 Introduction

Succinct zero-knowledge non-interactive arguments of knowledge (zk-SNARKs) enable a prover to convince a verifier that it knows a private witness for an NP relation. They have two properties that enable broad applications. First, proofs are *succinct* and can be easily verified; i.e., the proof size and verification time are sublinear in the size of the NP relation. Second, proofs are *zero knowledge* and reveal no information about the witness. Recent years have seen many zk-SNARK constructions [37, 47, 31, 41], applications [7, 50, 12, 58, 1], and real-world deployments [66, 49, 36].

Adoption of zk-SNARKs, however, is hindered by a key limitation: while proof verification is fast, proof generation is expensive and thousands of times slower than directly checking the NP relation [27]. This overhead typically stems from the need for the prover to execute a number of costly cryptographic operations (e.g., elliptic-curve operations) that scale at least linearly with the relation size. Applications such as private smart contracts [12], auditable key transparency [61], and verifiable machine learning [1] remain limited to only small instances due to high proving costs.

One way to address this limitation is for provers to outsource proof generation to more-powerful cloud servers [63, 44, 57], an idea being actively pursued by several startups [51, 60, 54, 29]. For applications in which privacy is critical—such as private transactions [7] and anonymous credentials [58, 36]—the servers must learn no information about the witness, which may include sensitive client data (e.g., private keys or user credentials). This challenge has motivated a recent line of work on *private delegation* of zk-SNARKs [52, 21, 33, 35, 32], where clients outsource proving to servers that remain oblivious to the client's witness. As we discuss next, however, existing schemes fall short of the efficiency, usability, and security required for real-world applications.

## 1.1 Prior Work

Prior work on private delegation of zk-SNARKs can be categorized into two groups: *multi-server* protocols [21, 33] relying on secret sharing and *single-server* protocols [35] using fully homomorphic encryption (FHE). We discuss each in turn. (See Table 1 for a summary comparison.)

**Multi-server delegation.** One solution to privately delegate zk-SNARK proving is for a client to secret-share its witness with multiple servers, who then run a secure multi-party computation (MPC) protocol to compute the proof. Boneh et al. [52] called this framework *collaborative* zk-SNARKs and proposed concrete instantiations by applying standard MPC protocols to several zk-SNARKs, including Marlin [20], Groth16 [37], and Plonk [31]. Chiesa et al. [21] (for Marlin) and Garg et al. [33] (for Groth16, Plonk, and Marlin) advanced this approach and showed that when relying on sufficiently many servers, it is even possible to reduce the latency of proof generation. Subsequent works have improved the efficiency and security of this framework [64, 38, 46, 28, 34], though in some cases only by modifying the underlying zk-SNARK.

The main limitation of this approach is the need to deploy multiple servers, with the client having to trust that some threshold of them behave honestly. (If the number of corrupted servers exceeds the threshold, the client's witness is exposed.) Another limitation of these schemes is that, although they keep the witness private, they reveal information about the statement and proof to all the servers.[1] This is unacceptable for applications where the client must remain unlinkable to the proof as in, e.g., private transactions [7] or anonymous credentials [58].

[1]These protocols reveal hash-function inputs to the servers so they can compute Fiat–Shamir challenges; one could avoid this using generic MPC, but this would incur a significant cost, since the hashes are evaluated inside the MPC.

| Scheme | # of servers | Improved latency | Unlinkability | Malicious security | Instantiations |
|---|---|---|---|---|---|
| co-SNARKs [52] | $\geq 2$ | ✗ | ✗ | ✓ | Groth16; Marlin; Plonk; Fractal |
| EOS [21] | $\geq 2$ | ✓ | ✗ | ✗ | Marlin |
| ZKSaaS [33] | $\geq 8$ | ✓ | ✗ | ✗ | Groth16; Plonk; Marlin |
| DFS [38] | $\geq 2$ | ✓ | ✗ | ✓ | Marlin* |
| Siniel [64] | $\geq 2$ | ✓ | ✗ | ✓ | Marlin |
| Liu et al. [46] | $\geq 8$ | ✓ | ✗ | ✓ | HyperPlonk |
| Fang et al. [28] | $\geq 2$ | ✓ | ✗ | ✓ | Plonk* |
| Blind FRI [35, 32] | 1 | ✗ | ✗ | ✗ | Fractal* |
| **This paper** | 1 | ✓ | ✓ | ✓ | Nova; Groth16; Plonk |

* Delegation-specific adaptations.

Table 1: Private delegation schemes. Number of servers assumes the scheme tolerates at least one corrupted server. Although EOS claims malicious security, an attack is known [38].

**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 necessary trust assumptions. While single-server private delegation can be generically realized using fully homomorphic encryption (FHE), doing so directly will not yield a practical scheme due to the associated overhead [65]. Garg et al. [35] reduce this overhead for FRI-based zk-SNARKs [6] by minimizing non-black-box use of the underlying primitives. Follow-up work [2, 32] showed optimizations, but even in the best existing schemes (i) the server computation is orders of magnitude slower than the native proving time, thus *increasing* the latency of proof generation, and (ii) the client's cost—though smaller than native proving—is still fairly high.

Besides the overhead introduced by using FHE, existing single-server schemes have a number of other drawbacks. Most prominent is that they change proof verification; that is, the proofs they generate are not compatible with existing zk-SNARK verifiers, and in fact verification time increases due to the need to verify FHE decryption. Moreover, existing single-server schemes only support FRI-based zk-SNARKs (e.g., Fractal [22]) 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 it would be prohibitive to prevent this (cf. footnote 1). 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.

## 1.2 Our Contributions

We propose concretely efficient single-server private delegation schemes that match the best features of prior work while addressing their limitations (see Table 1). In particular, our constructions support group-based zk-SNARKs without any modifications; achieve unlinkability since they hide the witness, statement, and final proof from the server; and are secure even against a malicious server. Furthermore, our schemes are more efficient—for both client and server—than existing

single-server solutions, and we improve latency compared to local computation by the client.

We focus on group-based zk-SNARKs [37, 31, 41] as they are among the most widely deployed proof systems in practice (due to their minimal verification overhead), yet their proof generation time is typically slower than in alternate schemes. The prover overhead in these zk-SNARKs is dominated by elliptic-curve operations, which are expensive to realize in the circuits needed for FHE; thus, single-server delegation for theze zk-SNARKs using FHE is impractical.

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

- **Encrypted multi-scalar multiplication.** As a foundational building block, we introduce *encrypted MSM* (EMSM), a new primitive of independent interest that allows a client to privately delegate MSM computation to an untrusted server. We then 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 [17]. In our scheme, the server computation matches that of the 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. Finally, we observe that our scheme supports *public linear maps* applied to 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 Pedersen and KZG commitments [40]. As KZG commitments are a key building block in many zk-SNARKs and other applications [55, 20, 19, 30], we believe that private outsourcing of KZG commitments is of independent interest.

- **Server-aided zk-SNARKs.** As our main contribution, we introduce and formalize the notion of SAP-SNARKs—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 then show how to achieve server-aided proving for the widely-deployed Nova [41], Groth16 [37], and Plonk [31] zk-SNARKs. In each case, the group operations done by the prover are in the form of MSMs and can be privately delegated using our EMSM scheme. 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 zk-SNARKs (e.g., Marlin [20] or Hyperplonk [19]), as well as recursive arguments such as HyperNova [42] and KZH-fold [39].

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

### 1.3 Example Applications

SAP-SNARKs are useful in any scenario where a lightweight client wants to outsource proof generation to a more-powerful entity. 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 [67]. It can be even slower in private blockchains [12], where users prove correct execution of an arbitrary (private) smart contract. SAP-SNARKs reduce clients' work and the latency for generating transactions, enhancing overall performance and user experience.

**Verifiable and private machine learning.** zk-SNARKs can enable verification of various properties of machine-learning models, e.g., inference correctness [45] or model provenance and integrity [1]. Circuits that arithmetize these properties are typically large and their size scales with the size of the model and the dataset, which makes zk-SNARKs computationally intensive for realistic deployments. Our SAP-SNARKs 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 [59, 24], yet such hardware may not be readily accessible to average users. Prover markets [60] 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, limiting their applicability to real-world settings. SAP-SNARKs address this problem and enable efficient use of servers' resources.

**Private identity verification.** zk-SNARKs also have applications to private authentication and anonymous credentials [58], with a prominent example of zkLogin [4]. In such scenarios, users typically operate on lightweight devices such as smartphones, where proof generation is quite slow. SAP-SNARKs would enable users to offload this work 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 [25]. Despite efforts to optimize zk-SNARKs for such applications, proving time remains slow, often taking tens of minutes. SAP-SNARKs enable a client to offload most of its workload to a server while keeping images/edits hidden.

## 2 Preliminaries

We use $x := z$ for assignment and $k \leftarrow \mathcal{K}$ for uniform sampling from a set $\mathcal{K}$. We define $[n] = \{0, 1, \ldots, n-1\}$. We use bold lowercase letters to denote column vectors (e.g., $\mathbf{a}$) and bold uppercase letters for matrices (e.g., $\mathbf{A}$). We write $a_i$ for the $i$th element of $\mathbf{a}$, and $A_{i,j}$ for the element of $\mathbf{A}$ in the $i$th row and $j$th column. We let $\mathbf{A}_{i,*}$ and $\mathbf{A}_{*,j}$ denote the $i$th row and the $j$th column of $\mathbf{A}$, respectively. We use $\mathrm{wt}(\mathbf{a})$ to denote the Hamming weight of $\mathbf{a}$. We let $\lambda$ be the computational security parameter and let $\mathsf{negl}(\lambda)$ denote a negligible function in $\lambda$. An algorithm is *efficient* if its running time is polynomial in $\lambda$ and its input length.

**Algebraic notation.** We use $\mathbb{F}_q$ to denote the field of integers modulo a prime $q$, with $\mathbb{F}_q^* = \mathbb{F}_q \setminus \{0\}$. We let $\mathbb{G}$ be a group of prime order $q$ with generator $g \in \mathbb{G}$, written additively. *Multi-scalar multiplication* (MSM) of $\mathbf{a} \in \mathbb{F}_q^m$ and $\mathbf{g} \in \mathbb{G}^m$ is denoted by $\langle \mathbf{a}, \mathbf{g} \rangle = \sum_{i \in [m]} a_i g_i$.

**R1CS circuit format.** A *rank-1 constraint system* (R1CS) is a representation of arithmetic circuits used in many zk-SNARKs [37, 20]. 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. An R1CS is satisfied by an instance $\mathbf{x} \in \mathbb{F}_q^{\ell}$ and witness $\mathbf{w} \in \mathbb{F}_q^{m-\ell}$ if $(\mathbf{Az}) \odot (\mathbf{Bz}) = \mathbf{Cz}$, where $\mathbf{z} := \mathbf{x} \| \mathbf{w}$ and $\odot$ is the element-wise (Hadamard) product.

**Universal composability framework.** We use the *universal composability* (UC) framework [18] to define and prove security of our protocols. We say protocol $\Pi$ *UC-realizes* an ideal functionality $\mathcal{F}$ if for any adversary $\mathcal{A}$ there is an efficient *simulator* $\mathsf{Sim}$ such that for any environment $\mathcal{Z}$ with arbitrary auxiliary input $z$, the output of $\mathcal{Z}$ in the real-world execution (where the parties execute $\Pi$ and interact with $\mathcal{A}$) is indistinguishable from the output of $\mathcal{Z}$ in the ideal-world execution (where the parties interact with $\mathsf{Sim}$ and $\mathcal{F}$). We also consider $\mathcal{F}$-hybrid models, which means that the parties running a protocol also have access to an auxiliary ideal functionality $\mathcal{F}$.

## 2.1 zk-SNARKs

A non-interactive *argument system* for a relation $\mathcal{R}$ allows a prover $\mathcal{P}$ to convince a verifier $\mathcal{V}$ that $\exists w : (x, w) \in \mathcal{R}$, where $x$ is a public instance and $w$ is a private witness. It has the following syntax:

- $\mathsf{Setup}(1^\lambda) \to \mathsf{pp}$: a setup algorithm that, on security parameter $\lambda$, outputs public parameters $\mathsf{pp}$.
- $\mathsf{Prove}(\mathsf{pp}; x, w) \to \pi$: a proving algorithm that, given public parameters $\mathsf{pp}$, a public instance $x$, and a private witness $w$, outputs a proof $\pi$.
- $\mathsf{Verify}(\mathsf{pp}; x, \pi) \to \{0, 1\}$: a verification algorithm that, given public parameters $\mathsf{pp}$, an instance $x$, and a proof $\pi$, outputs either 1 (accept) or 0 (reject).

A non-interactive argument system is *complete* if $\mathcal{V}(\mathsf{pp}; x, \mathsf{Prove}(\mathsf{pp}; x, w)) = 1$ for all $\mathsf{pp}$ output by $\mathsf{Setup}$ and all $(x, w) \in \mathcal{R}$. Informally, it satisfies *soundness* if no efficient $\mathcal{P}^\star$ can output an accepting proof for $x$ if $\nexists w : (x, w) \in \mathcal{R}$. *Knowledge soundness* further ensures there is an efficient *extractor* such that if an efficient $\mathcal{P}^\star$ convinces $\mathcal{V}$ to accept, then the extractor can use $\mathcal{P}^\star$ to compute a witness $w$ such that $(x, w) \in \mathcal{R}$. *Zero knowledge* means there is an efficient *simulator* that, given only $x$, outputs $(\mathsf{pp}, \pi)$ indistinguishable from the honestly generated outputs of $\mathsf{Setup}$ and $\mathcal{P}$. *Succinctness* means the proof size and verification time are sublinear in the size of the relation $\mathcal{R}$. zk-SNARK stands for (z)ero-(k)nowledge (S)uccinct (N)on-interactive (Ar)gument of (K)nowledge.

## 2.2 Commitment Schemes

A *commitment scheme* for a message space $\mathbf{M}$ consists of algorithms $\langle \mathsf{Setup}, \mathsf{Commit} \rangle$ as follows:

- $\mathsf{Setup}(1^\lambda) \to \mathsf{pp}$: a setup algorithm that, on security parameter $\lambda$, outputs public parameters $\mathsf{pp}$.
- $\mathsf{Commit}(\mathsf{pp}; m; r) \to c$: a commitment algorithm that, given the public parameters $\mathsf{pp}$, a message $m \in \mathbf{M}$, and randomness $r$, outputs a commitment $c$.

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

**Pedersen commitments.** Pedersen commitments [53] for message space $\mathbb{F}_q^n$ use uniform group generators $(h, g_0, g_1, \cdots, g_{n-1}) \in \mathbb{G}^{n+1}$ as public parameters. The commitment to $\mathbf{a} \in \mathbb{F}_q^n$ using randomness $r \in \mathbb{F}_q$ is $c := rh + \langle \mathbf{a}, \mathbf{g} \rangle$, and so involves computing a single MSM.

**Polynomial commitments.** A polynomial commitment is a commitment for the message space $\mathbb{F}_q^{<n}[X]$ that enables proving evaluations of committed polynomials. Formally, it consists of:

- $\mathsf{Setup}(1^\lambda) \to \mathsf{pp}$: a setup algorithm that, on security parameter $\lambda$, outputs public parameters $\mathsf{pp}$.
- $\mathsf{Commit}(\mathsf{pp}; f(X); r) \to c$: a commitment algorithm that, given public parameters $\mathsf{pp}$, a polynomial $f(X) \in \mathbb{F}_q^{<n}[X]$, and randomness $r$, outputs a commitment $c$.
- $\mathsf{Open}(\mathsf{pp}; c, f(X), x; r) \to (y, \pi)$: an opening algorithm that, given public parameters $\mathsf{pp}$, the commitment $c$, polynomial $f(X)$, an evaluation point $x$, and randomness $r$, outputs the evaluation $y = f(x)$ and an opening proof $\pi$.
- $\mathsf{Verify}(\mathsf{pp}; c, x, y, \pi) \to \{0, 1\}$: a verification algorithm that, given public parameters $\mathsf{pp}$, some commitment $c$, an evaluation pair $(x, y)$, and a proof $\pi$, outputs 1 (accept) or 0 (reject).

A polynomial commitment is *hiding* if the commitment is hiding and the opening proof is a zero-knowledge argument of correct evaluation. It is *extractable* if the commitment is binding and the proof is a knowledge-sound argument of evaluation.

**KZG commitments.** Let $\mathbb{G}_1, \mathbb{G}_2$ be groups of prime order $q$ with generators $g_1 \in \mathbb{G}_1$ and $g_2 \in \mathbb{G}_2$, with an associated bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$. KZG commitments [40] work as follows:

- $\mathsf{Setup}$: output $\mathsf{pp} := (g_1, \tau \cdot g_1, \ldots, \tau^{n-1} \cdot g_1, \tau \cdot g_2)$, where $\tau \leftarrow \mathbb{F}_q$ is a secret trapdoor.
- $\mathsf{Commit}$: given a polynomial $f(X) = \sum_{i=0}^{n-1} f_i X^i$, output $c := f(\tau) \cdot g_1 = \sum_{i=0}^{n-1} f_i \cdot (\tau^i \cdot g_1)$.
- $\mathsf{Open}$: given $x$, output the evaluation $y = f(x)$ and $\pi := q(\tau) \cdot g_1 = \sum_{i=0}^{n-2} q_i \cdot (\tau^i \cdot g_1)$, where $q(X) = \frac{f(X)-y}{X-x}$.
- $\mathsf{Verify}$: given $c$, $x$, $y$, and $\pi$, output accept if and only if $e(\pi, \tau \cdot g_2 - x \cdot g_2) = e(c - y \cdot g_1, g_2)$.

This variant of KZG is binding, but not hiding; hiding is easy to add if needed. Note that $\mathsf{Commit}$ and $\mathsf{Open}$ each involve only a single MSM.

### 2.3 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*. The *rate* of $\mathcal{C}$ is $R = n/N$. The *distance* of $\mathcal{C}$ is the minimum Hamming distance between two codewords; for a linear code, it equals the minimum weight of a nonzero codeword. The *relative distance* of $\mathcal{C}$ is $\delta := \min_{\mathbf{v} \in \mathcal{C} \setminus \{0^N\}} \mathrm{wt}(v)/N$.

**RAA codes.** *Repeat-accumulate-accumulate* (RAA) codes are linear error-correcting codes with linear encoding time [17]. For block length $N$, dimension $n$, and constant $r = (N/n) \in \mathbb{N}$, we define the following:

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

- $\mathbf{A} \in \mathbb{F}^{N \times N}$ is the 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.

Permutations $\sigma_1, \sigma_2 : [N] \to [N]$ define a generator for the RAA code as $\mathbf{G}_{\sigma_1,\sigma_2} := \mathbf{F}_r \cdot \mathbf{M}_{\sigma_1} \cdot \mathbf{A} \cdot \mathbf{M}_{\sigma_2} \cdot \mathbf{A}$.

### 2.4 Learning Parity with Noise

At a high level, the *learning parity with noise* (LPN) problem [10] over a general 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, $(\mathbf{A},\ \mathbf{A}\mathbf{s} + \mathbf{e})$ is computationally indistinguishable from $(\mathbf{A},\ \mathbf{b})$ with $\mathbf{b} \leftarrow \mathbb{F}^N$ uniform. The *dual-LPN* assumption is an equivalent formulation [48, 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 $(\mathbf{G}, \mathbf{G}\mathbf{e})$ 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 [43] for a detailed security analysis. Recent work [13, 16, 14, 56, 8] has instantiated $\mathcal{G}$ as the uniform distribution over generator matrices of linear codes with fast encoding. It is widely believed that the dual-LPN assumption 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 $\mathbf{G}\mathbf{e} \in \mathbb{F}^n$ when $\mathbf{e}$ is sampled from $\mathcal{E}$. All known attacks against LPN and its variants [5, 11, 62, 68] involve finding a $\mathbf{v} \in \mathbb{F}^n$ 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) there does not exist such a $\mathbf{v}$. Prior work [14] shows that this can be done by ensuring that, for some $\delta$, (i) the code defined by $\mathbf{G}$ has relative distance at least $\delta$ and (ii) $\epsilon = \max_{\mathsf{wt}(\mathbf{v}) \geq \delta N} \mathsf{bias}_{\mathbf{v}}(\mathcal{E}_{\mathbf{G}})$ is small. Since the noise distribution $\mathcal{E}$ we use is uniform over vectors of Hamming weight $t$, we have $\epsilon < e^{-2\delta t}$ [14]; thus, to achieve $\lambda$-bit security of dual-LPN against known attacks we require

$$e^{-2\delta t} \leq 2^{-\lambda + \log_2 N} \quad \Longleftrightarrow \quad \delta \ \geq \ \frac{\ln 2 \cdot (\lambda - \log_2 N)}{2t}. \tag{1}$$

We ensure that the code defined by $\mathbf{G}$ has relative distance $\delta$ with overwhelming probability over choice of $\mathbf{G}$ sampled from $\mathcal{G}$.

There are three scenarios that do not directly fit into the linear-test framework [14]: 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 [3]; and when the field $\mathbb{F}$ has nontrivial subfields. We make sure to avoid these cases.

**Instantiation with RAA codes.** In this work, we instantiate $\mathcal{G}$ as a distribution over generators of an RAA code (cf. Section 2.3), which has been conjectured to be safe for dual-LPN [8]. To

provide further evidence, we analyze the minimum distance of RAA codes over arbitrary finite fields (see Appendix A) and then use the linear-test framework to set parameters; further details are given in Section 5.1.

# 3 Encrypted Multi-Scalar Multiplication

We first formalize the notion of *encrypted multi-scalar multiplication* (EMSM) and then propose a construction based on the dual-LPN assumption. We then explore applications to the delegation of commitment schemes.

## 3.1 Definition

Fix a public group basis $\mathbf{g} \in \mathbb{G}^n$. An EMSM scheme allows a client who holds a private input $\mathbf{z} \in \mathbb{F}_q^n$ to outsource the computation of $\langle \mathbf{z}, \mathbf{g} \rangle$ to a single, untrusted server. More formally, an EMSM scheme has the following syntax:

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

Let $\mathcal{F}_{\mathsf{MSM}}$ (cf. Figure 1) be the reactive two-party ideal functionality that is initialized with a public $\mathbf{g}$ and, when given input $\mathbf{z}$ from the client, returns $\langle \mathbf{z}, \mathbf{g} \rangle$; reactive means that $\mathcal{F}_{\mathsf{MSM}}$ can be queried multiple times after being initialized. A corrupted server can be either *semi-honest* or *malicious*, giving two flavors of security for EMSM. A malicious server can abort the functionality and thus 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 $\Pi$—in which the client runs $\mathsf{Encrypt}$ and sends the ciphertext to the server, the server runs $\mathsf{Evaluate}$ and sends back the result, and the client computes $\mathsf{Decrypt}$—UC-realizes $\mathcal{F}_{\mathsf{MSM}}$ in the $\mathsf{Setup}$-hybrid model. This implies the following properties:

- *Correctness*: in an honest execution of $\Pi$, the client always outputs the correct result $\langle \mathbf{z}, \mathbf{g} \rangle$.
- *Privacy*: the server learns nothing about the input $\mathbf{z}$.
- *Malicious security*: a malicious server cannot make the client output an incorrect value; the most it can do is make the client abort and output $\perp$. A malicious server learns no information about $\mathbf{z}$ based on the client's decision to abort (i.e., selective-failure attacks are ruled out).

**Efficiency.** The trivial EMSM scheme, where the client computes the MSM locally, is secure; we are interested in EMSM schemes where the client's work is less than this local computation.

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

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

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

Figure 1: The multi-scalar multiplication ideal functionality.

**Realizing the setup.** At a high level, there are two ways the setup can be realized for an EMSM scheme. First, each client could compute $\mathsf{Setup}$ by itself; this may be more expensive than computing a single MSM locally, but the cost would be *amortized* over multiple MSM evaluations. The other possibility is to have a semi-trusted entity who computes $\mathsf{Setup}$ 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, in our EMSM scheme $\mathsf{Setup}$ is *transparent*, meaning there is no secret trapdoor that needs to be deleted, and also *verifiable*, meaning that anyone can verify the correctness of the setup—though this verification requires $\Theta(n)$ 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 an incorrect setup, but would risk being caught by doing so.

## 3.2 Construction

We present the framework for our EMSM scheme based on the dual-LPN assumption. Then, we discuss a concretely efficient instantiation of our framework using RAA codes.

### 3.2.1 Protocol Design

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, which returns $\langle \mathbf{z} + \mathbf{r}, \mathbf{g} \rangle$; the client outputs $\langle \mathbf{z} + \mathbf{r}, \mathbf{g} \rangle - \langle \mathbf{r}, \mathbf{g} \rangle$. In this naive approach the client computes $\langle \mathbf{r}, \mathbf{g} \rangle$, which costs the same as computing the MSM. Our main insight is to mask $\mathbf{z}$ with some *structured, pseudorandom* $\mathbf{r}$ such that $\langle \mathbf{r}, \mathbf{g} \rangle$ can be computed using sublinear 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-$t$ noise vector chosen by the client, with $t = O(1)$. Note that $\mathbf{r}$ is pseudorandom if the dual-LPN assumption holds. Furthermore,

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

where the $\{\langle \mathbf{G}_{*,i},\ \mathbf{g} \rangle\}_{i \in [N]}$ are independent of the client's input. Thus, once $\mathbf{h} = \mathbf{G}^T \mathbf{g}$ (so that $h_i = \langle \mathbf{G}_{*,i}, \mathbf{g} \rangle$) is published, 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 only $O(\mathsf{wt}(\mathbf{e})) = O(t) = O(1)$ group operations.

Note that, by a standard hybrid argument, dual-LPN 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. Note that the setup algorithm is transparent (assuming $\mathbf{G}$ is generated using the output of 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 $c \leftarrow \mathbb{F}_q$ 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 $\mathsf{dm}$; 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, the client outputs $\perp$ except with negligible probability $1/q$.

Our EMSM scheme is specified in Figure 2 with components for malicious security in gray.

**Theorem 3.1.** *Assume the dual-LPN holds for some generator distribution $\mathcal{G}$ and an error distribution $\mathcal{E}$. Then the semi-honest (resp., malicious) EMSM of Figure 2 is secure against a semi-honest (resp., malicious) server.*

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

We show that the protocol obtained from the malicious version of $\Pi_{\mathsf{MSM}}$ (as described earlier) UC-realizes $\mathcal{F}_{\mathsf{MSM}}$ in the $\mathsf{Setup}$-hybrid model. Consider a simulator $\mathsf{Sim}$ that interacts with an adversary $\mathcal{A}$ corrupting the server. In each execution, the simulator $\mathsf{Sim}$ chooses uniform $\mathbf{v}, \mathbf{v}_{\mathsf{ck}}$ and sends these to the $\mathcal{A}$; it then receives $\mathsf{em}, \mathsf{em}_{\mathsf{ck}}$ from $\mathcal{A}$. If $\mathsf{em} = \langle \mathbf{v}, \mathbf{g}\rangle, \mathsf{em}_{\mathsf{ck}} = \langle \mathbf{v}_{\mathsf{ck}}, \mathbf{g}\rangle$, then $\mathsf{Sim}$ sends $\mathsf{continue}$ to $\mathcal{F}_{\mathsf{MSM}}$; otherwise, it sends $\mathsf{abort}$ to $\mathcal{F}_{\mathsf{MSM}}$. Computational indistinguishability of the adversary's view in the ideal world and its view in the real world follows from the dual-LPN assumption; correctness of the client's output follows from the argument given above. □

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

**One input, multiple MSMs.** A client may need to outsource $\langle \mathbf{z}, \mathbf{g}_1\rangle, \ldots, \langle \mathbf{z}, \mathbf{g}_\ell\rangle$ for multiple bases $\mathbf{g}_1, \ldots, \mathbf{g}_\ell$ but the same input $\mathbf{z}$. While this can be done using $\ell$ independent executions of our EMSM scheme, it is possible to do better when it is acceptable to reveal to the server that the same input is being used. First, for $\mathsf{Setup}$, it suffices to fix a single generator matrix $\mathbf{G}$ (though $\ell$ 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, the client only needs to send a *single* ciphertext $\mathbf{v}$. The server then returns $\{\mathsf{em}_i = \langle \mathbf{v}, \mathbf{g}_i\rangle\}$, from which the client recovers the outputs $\{\mathsf{dm}_i\}$ as before. The same idea applies to the malicious version of the scheme.

### 3.3 Efficient Instantiation

We instantiate the EMSM scheme of Figure 2 with an error distribution $\mathcal{E}$ that is uniform over weight-$t$ vectors in $\mathbb{F}^N$, where $t \in O(1)$ and $N = O(n)$ (cf. Section 2.4). The client then performs

$\Pi_{\mathsf{EMSM}}$

**Parameters:** $n$ the input length; $N = O(n)$ block length; $\mathcal{G}$ a code generator distribution over $\mathbb{F}_q^{n \times N}$; $\mathcal{E}$ an error distribution over weight-$t$ noise vectors in $\mathbb{F}_q^N$ with $t = O(1)$.

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

- 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 $\mathbf{v} := \mathbf{r} + \mathbf{z}$ , $\mathbf{v}_{\mathsf{ck}} := \mathbf{r}_{\mathsf{ck}} + c \cdot \mathbf{z}$ .
- Return $\mathsf{ct} := (\mathbf{v}, \mathbf{v}_{\mathsf{ck}})$ and $\mathsf{st} := (\mathbf{e}, \mathbf{e}_{\mathsf{ck}}, c)$.

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

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

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

- Parse $\mathsf{em}^*$ 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 $\mathsf{dm}$.

Figure 2: The encrypted multi-scalar multiplication protocol.

$O(t) = O(1)$ group operations. For a uniform matrix $\mathbf{G}$, however, the client might have to perform up to $n \cdot t$ field multiplications[2] and $n \cdot (t-1)$ field additions to compute $\mathbf{G}\mathbf{e}$; the same cost applies to computing $\mathbf{G}\mathbf{e}_{\mathsf{ck}}$. If we let $\mathbf{G}$ be a *structured* matrix for which dual-LPN still holds, we can reduce this encoding cost.

Several such "LPN-friendly" choices of $\mathbf{G}$ have been studied in prior work [15, 14]. Here, we let

[2]For instance, with $n = 2^{20}$, $N = 2^{22}$, and $\mathbf{G}$ sampled uniformly from $\mathbb{F}^{n \times N}$, we require $t \geq 40$ to achieve 100-bit security [13].

$\mathbf{G}$ be a generator for an RAA code (cf. Section 2.3) since such codes offer linear-time and concretely efficient encoding. Recall this means $\mathbf{G} = \mathbf{F}_r\mathbf{M}_{\sigma_1}\mathbf{A}\mathbf{M}_{\sigma_2}\mathbf{A}$, 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 $\sigma_1, \sigma_2$ over $[N]$; and $\mathbf{A}$ is the accumulation matrix. It is then possible to compute $\mathbf{Ge}$ using at most $3N$ field additions: each $\mathbf{A}$ and $\mathbf{F}_r$ requires $N-1$ additions, while applying $\mathbf{M}_{\sigma_1}$ and $\mathbf{M}_{\sigma_2}$ requires no field operations. When $N = 4n$ (see Section 5), this reduces the client's work to $3N = 12n$ field *additions*.

Using RAA codes also makes the setup procedure more efficient. When $\mathbf{G}$ is arbitrary, computing $\mathbf{h}$ requires $\Theta(nN)$ group multiplications, but when $\mathbf{G}$ is the generator matrix of an RAA code this drops to $2N$ group multiplications as discussed above. Note that this also holds when delegating a public linear map $\mathbf{H}$, as long as $\mathbf{H}$ is a structured matrix that enables fast multiplication; i.e., even if $\mathbf{H}^T\mathbf{G}^T$ is not structured, we can use the structure of $\mathbf{G}^T$ and $\mathbf{H}^T$ by applying them sequentially.

We discuss concrete parameter choices and the resulting concrete security in Section 5.1. We also note that several other families of error-correcting codes [26, 14] could also be used to instantiate our framework. We leave a full exploration of the trade-offs for future work.

### 3.4 Outsourced KZG Commitments

An immediate application of EMSM is outsourcing the Pedersen and KZG commitment schemes. Since KZG commitments are a key building block underlying many protocols, including zk-SNARKs, we focus on the latter. (The application to Pedersen commitments is similar.)

Recall from Section 2.2 that the commitment and opening algorithms of the KZG each require only a single 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 $\langle \mathbf{q}, \mathbf{g}\rangle$ where $\mathbf{q}$ is defined similarly. In both cases, the dominant computation—the MSMs—can be outsourced via an EMSM. We refer to the result as *encrypted KZG* (EKZG). For completeness, we present the ideal functionality for outsourced KZG commitments in Figure 3, and an EKZG protocol $\Pi_{\mathsf{KZG}}$ in Figure 4. The following is immediate:

**Theorem 3.2.** *If $\Pi_{\mathsf{EMSM}}$ is a secure semi-honest (resp., malicious) EMSM scheme, then $\Pi_{\mathsf{KZG}}$ UC-realizes $\mathcal{F}_{\mathsf{KZG}}$ in the $\mathsf{Setup}$-hybrid model for semi-honest (resp., malicious) adversaries.*

## 4 zk-SNARKs with Server-Aided Proving

We formalize the notion of *zk-SNARKs with server-aided proving* (SAP-SNARKs) that enable a client to (partially) outsource the work of proof generation for a zk-SNARK to a more powerful server. We then propose server-aided proving protocols for Nova [41], Groth16 [37], and Plonk [31].

### 4.1 Definition

Let $\Sigma = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ be a zk-SNARK. A server-aided proving scheme for $\Sigma$ consists of:

- $\mathsf{Setup}(\mathsf{pp}_\Sigma) \to \mathsf{pp}$: a setup algorithm that, given $\mathsf{pp}_\Sigma := \Sigma.\mathsf{Setup}$, outputs public parameters $\mathsf{pp}$.
- $\langle \mathsf{Prove}.S(\mathsf{pp}_\Sigma, \mathsf{pp}), \mathsf{Prove}.C(\mathsf{pp}_\Sigma, \mathsf{pp}, x, w)\rangle \to \pi$: an interactive protocol between a server $S$ who holds the public parameters $\mathsf{pp}_\Sigma, \mathsf{pp}$ and a client $C$ that holds the public parameters $\mathsf{pp}_\Sigma, \mathsf{pp}$, a statement $x$, and a witness $w$. Finally, $C$ outputs a proof $\pi := \Sigma.\mathsf{Prove}(\mathsf{pp}_\Sigma; x, w)$ (possibly $\perp$).

**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 $\mathsf{com}$ to the server and adversary. 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 $f(X)$. If the server sends $\mathsf{abort}$, send $\perp$ to the client.

**Opening:** Upon receiving $(\mathsf{open}, f(X), x)$ from the client, send $\mathsf{open}$ 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 the polynomial $q(X) = \frac{f(X)-f(x)}{X-x}$. Otherwise, if the server sends $\mathsf{abort}$, then send $\perp$ to the client.

Figure 3: Ideal functionality for KZG commitments.

**Protocol $\Pi_{\mathsf{KZG}}$**

**Parameters:** $\mathbf{g} = (g_1, \tau \cdot g_1, \ldots, \tau^{n-1} \cdot g_1, \tau \cdot g_2)$ KZG public parameters; $\Pi_{\mathsf{EMSM}}$ an EMSM.

**Setup:** The EMSM setup $\Pi_{\mathsf{EMSM}}.\mathsf{Setup}(1^\lambda, \mathbf{g})$ is executed, giving the output $\mathsf{pp}_{\Pi_{\mathsf{EMSM}}}$.

**Commitment:** On input coefficient vector $\mathbf{f}$, the client runs $(\mathsf{ct}, \mathsf{st}) \leftarrow \mathsf{Encrypt}(\mathsf{pp}_{\Pi_{\mathsf{EMSM}}}; \mathbf{f})$, and sends $\mathsf{ct}$ to the server. The server then executes $\mathsf{em} \leftarrow \mathsf{Evaluate}(\mathsf{pp}_{\Pi_{\mathsf{EMSM}}}; \mathsf{ct})$ and sends $\mathsf{em}$ to the client. Finally, the client recovers the KZG commitment $\mathsf{dm} \leftarrow \mathsf{Decrypt}(\mathsf{pp}_{\Pi_{\mathsf{EMSM}}}; \mathsf{em}; \mathsf{st})$.

**Opening:** Given $x$, the client evaluates $y = f(x)$ and computes the quotient $q(X) = \frac{f(X)-y}{X-x}$ with the coefficient vector $\mathbf{q}$. The client then runs $(\mathsf{ct}, \mathsf{st}) \leftarrow \mathsf{Encrypt}(\mathsf{pp}_{\Pi_{\mathsf{EMSM}}}; \mathbf{q})$ and sends $\mathsf{ct}$ to the server. The server executes $\mathsf{em} \leftarrow \mathsf{Evaluate}(\mathsf{pp}_{\Pi_{\mathsf{EMSM}}}; \mathsf{ct})$ and sends $\mathsf{em}$ to the client. Finally, the client recovers the KZG evaluation opening proof $\mathsf{dm} \leftarrow \mathsf{Decrypt}(\mathsf{pp}_{\Pi_{\mathsf{EMSM}}}; \mathsf{em}; \mathsf{st})$.

Figure 4: Outsourcing KZG commitments.

As in the case of EMSM, we define security via a reactive two-party functionality. Let $\mathcal{F}_{\mathsf{Prove}}$ (cf. Figure 5) be the ideal functionality corresponding to the proving algorithm of the zk-SNARK $\Sigma$. The functionality is parameterized by the public parameters $\mathsf{pp}_\Sigma$ 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 $\Pi_\Sigma$ for the zk-SNARK $\Sigma$ is *secure* if it UC-realizes the ideal functionality $\mathcal{F}_{\mathsf{Prove}}$ in the $(\Sigma.\mathsf{Setup}, \Pi.\mathsf{Setup})$-hybrid model. As before, we consider semi-honest or malicious servers. The 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 $\pi$ output by $C$ satisfies $\Sigma.\mathsf{Verify}(\mathsf{pp}; x, \pi) = 1$.
- *Witness privacy*: A corrupted $S$ learns nothing about the witness $w$, even if it later obtains the

**Functionality $\mathcal{F}_{\mathsf{Prove}}$**

**Initialization:** The client and the server initialize the functionality with an NP relation $\mathcal{R}$, a zk-SNARK $\Sigma = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ for $\mathcal{R}$, and its public parameters $\mathsf{pp}_\Sigma \leftarrow \Sigma.\mathsf{Setup}(1^\lambda)$.

**Proof generation:** Upon receiving $(\mathsf{prove}, x, w)$ from the client, send $\mathsf{prove}$ to the server and adversary. 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 $\mathsf{abort}$, then send $\perp$ to the client.

Figure 5: The server-aided proving ideal functionality.

statement $x$ and the proof $\pi$; this is due to the security of $\Pi_\Sigma$ and the zero-knowledge of $\Sigma$.

- *Unlinkability*: A corrupted $S$ learns no information about the statement, or the proof output by $C$. This ensures that proofs cannot be linked to a particular execution of the protocol, even if statements of proofs are later revealed to $S$. Note this property is independent of witness privacy; in particular, existing work on private delegation achieve witness privacy but do not achieve unlinkability since they reveals the statement/proof to the servers (see Table 1).
- *Malicious security*: A malicious $S$ cannot cause $C$ to output an incorrect proof $\pi$; 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 $\perp$.

**Efficiency.** Any zk-SNARK $\Sigma$ admits a trivial server-aided proving $\Pi_\Sigma$ in which $\Pi_\Sigma.\mathsf{Setup}$ does nothing and $\Pi_\Sigma.\mathsf{Prove}$ simply 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 less than its work to compute $\Sigma.\mathsf{Prove}$ locally and, for group-based schemes in particular, where the number of group operations required by the client is sublinear in $|R|$.

### 4.2 Constructions

We now describe how to obtain server-aided proving for three zk-SNARKs: Nova [41], Groth16 [37], and Plonk [31]. Here we provide only a high-level overview of how our EMSM scheme can be applied to each zk-SNARK; more details are given in Appendix C. In general, zk-SNARK proving consists of the following steps:

1. A witness $w$ for $(x, w) \in \mathcal{R}$ is transformed into a witness for a constraint system, e.g., R1CS.
2. Cryptographic operations are used to generate an argument showing that the constraint system is satisfiable.

The first step is relation-specific, non-cryptographic, and computationally inexpensive; hence, we assume it can be performed by the client locally and focus only on outsourcing the second step.

### 4.2.1 Server-Aided Proving for Nova

Nova is a recursive zk-SNARK designed for iterative computations. Its key component is a *folding scheme* that merges two R1CS instances into one of the same size; if the folded instance is satisfiable, then—except with negligible probability—both input instances are satisfiable. The Nova prover derives the instances and witnesses for each iteration and combines them into one instance–witness pair using the folding scheme. Finally, a generic zk-SNARK is applied to the last folded instance to prove its satisfiability, which implies the satisfiability of the initial instances. The folding scheme used by Nova commits to each instance/witness using a Pedersen commitment and merges the commitments via a random linear combination. This works since Pedersen commitments are *additively homomorphic*. Further details about the Nova protocol are provided in Appendix B.1.

The prover bottleneck in Nova is the Pedersen commitments, which can be computed using MSMs. Our server-aided proving protocol for Nova has the client outsource commitment generation using an EMSM scheme, while the other steps of the prover algorithm—consisting only of field operations—are run by the client locally. Note that the overhead of the zk-SNARK on the last folded instance (which is computed locally by the client) is proportional to the circuit size of a *single* iteration, and is independent of the number of iterations; thus, it represents only a minor portion of the overall proving cost. We give the full server-aided proving protocol for Nova in Appendix C.1.

**Per-iteration efficiency.** Let $n$ denote the constraint size of an iteration. The prover's computation in Nova is dominated by two $O(n)$-sized MSMs. In our server-aided protocol for Nova, the client's computation is reduced to $O(1)$ group operations, while the server's computation 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.

### 4.2.2 Server-Aided Proving for Groth16

The second zk-SNARK we consider is Groth16. Let $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$ be matrices defining an R1CS relation (cf. Section 2). Given a statement $\mathbf{x}$, a witness $\mathbf{w}$, and $\mathbf{z} := \mathbf{x}\|\mathbf{w}$, let $\mathbf{a} = \mathbf{A}\mathbf{z}$, $\mathbf{b} = \mathbf{B}\mathbf{z}$, and $\mathbf{c} = \mathbf{C}\mathbf{z}$. The R1CS relation is satisfied when $\mathbf{a} \odot \mathbf{b} = \mathbf{c}$. The Groth16 scheme also defines three polynomials $f_a(X)$, $f_b(X)$, and $f_c(X)$ that, over a public evaluation domain $D$, interpolate to $\mathbf{a}$, $\mathbf{b}$, and $\mathbf{c}$, respectively. To prove the satisfiability of the R1CS instance, the prover shows that there exists a quotient $q(X)$ such that $f_a(X) \cdot f_b(X) - f_c(X) = t(X) \cdot q(X)$, where $t(X)$ is the vanishing polynomial over $D$. More details about the Groth16 protocol are provided in Appendix B.2.

The prover's computation is dominated by five MSMs: three involving the statement–witness $\mathbf{z}$, one involving $r\mathbf{z}$ for a uniform $r$ sampled by the prover, and one involving $\mathbf{q}$, the coefficient vector of $q(X)$. The R1CS matrices $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$ are embedded in the MSM bases. We note that computing $q(X)$ additionally requires eight fast Fourier transforms (FFTs) over the field.

Our server-aided proving protocol for Groth16 outsources all five MSMs to the server using an EMSM scheme, which requires the client to encrypt three vectors, $\mathbf{z}$, $r\mathbf{z}$, and $\mathbf{q}$. We further observe that four of the eight FFTs required to compute $q(X)$ are applied directly to the MSM inputs (i.e., either $\mathbf{z}$ or $\mathbf{q}$). Since our EMSM supports public linear maps (cf. Section 3.2) and FFTs are linear operations, these can be reduced from the client's work at the cost of an additional MSM by the server; the extra MSM can be executed in parallel with the others. The remaining four FFTs are not compatible with EMSM and are run locally by the client. Further details are given in Appendix C.2.

**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 $O(1)$

group operations and four $O(n)$-sized FFTs. The server work is six $O(n)$-sized MSMs (twelve in the malicious setting). Since Groth16 is non-interactive by design and does not rely on the Fiat-Shamir heuristic, our scheme is also non-interactive (i.e., it has one communication round). The communication cost is $O(n)$, as the client runs the EMSM scheme on inputs $\mathbf{z}$, $r\mathbf{z}$, and $\mathbf{q}$.

#### 4.2.3 Server-Aided Proving for Plonk

Finally, we consider server-aided Plonk. Plonk is built on *polynomial interactive oracle proofs* [20] (poly-IOPs), which are designed in an idealized model where the prover can construct "polynomial oracles" that the verifier can query. A poly-IOP can be instantiated in the real world by replacing these oracles with polynomial commitments; that is, first the prover commits to the polynomial, and then when the verifier wants to query the oracle at a particular point $x$, the prover responds with the evaluation of the polynomial at $x$ along with a proof of correct evaluation. Plonk instantiates the polynomial commitments with KZG. The protocol is made non-interactive via the Fiat-Shamir heuristic, wherein verifier challenges are replaced with the output of a random oracle, which is instantiated with a cryptographic hash function. More details about Plonk are given in Appendix B.3.

In our server-aided proving scheme for Plonk, we outsource the KZG polynomial commitments (cf. Section 3.4), while the information-theoretic component (i.e., generating poly-IOP messages) is executed locally by the client. We further show that the client can defer part of the field operations required in the poly-IOP to the final steps of the protocol execution, thereby improving the proof generation latency. We present more details as well as the formal protocol in Appendix C.3.

**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 reduces these costs to $O(1)$ group operations for the client, while the server's work is equivalent to the MSMs in Plonk (double in the malicious scheme). Our server-aided proving scheme requires five rounds of interaction with $O(n)$ communication cost.

## 5 Implementation and Evaluation

We built a Rust library implementing our EMSM scheme, EKZG scheme, and SAP-SNARKs for Nova and Groth16. (We leave the implementation for Plonk as future work.) Our library builds on arkworks [23], a collection of Rust libraries for zkSNARKs that provides interfaces for finite fields, elliptic curves, and commitments; it also includes an implementation of Groth16. Our implementation is publicly available at https://github.com/h-hafezi/server-aided-snarks.

**Experimental setup.** In 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 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 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. Reported times are averaged over 10 runs.

### 5.1 Dual-LPN Parameters

As in prior work [13, 14], we employ the *linear test* framework to choose LPN parameters. (See Section 2.4.) Specifically, we use Equation (1) to set parameters for $\lambda$-bit security.

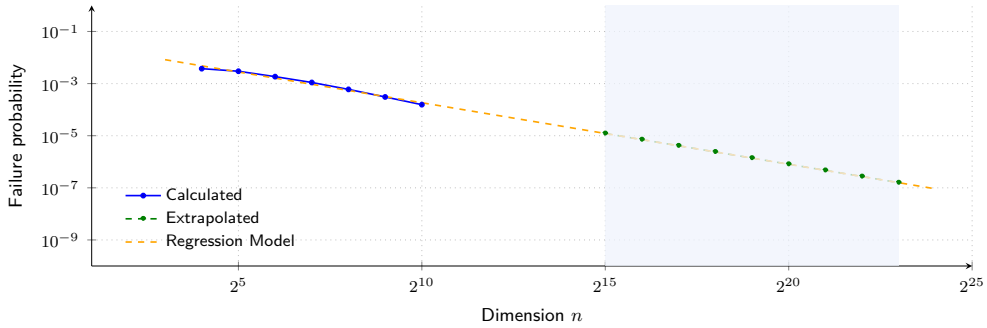



Table 2: Failure probability of RAA codes with rate $R = 0.25$ and relative distance $\delta = 0.05$ as a function of the dimension. The shaded region marks the extrapolation domain.

| Dimension $n$ | $2^{15}$ | $2^{16}$ | $2^{17}$ | $2^{18}$ | $2^{19}$ | $2^{20}$ | $2^{21}$ | $2^{22}$ | $2^{23}$ | $2^{24}$ |
|---|---|---|---|---|---|---|---|---|---|---|
| Noise weight $t$ | 589 | 582 | 575 | 568 | 561 | 554 | 547 | 540 | 533 | 526 |

Table 3: Noise weight $t$ as a function of the dimension $n$ to achieve 100-bit security when dual-LPN is instantiated using RAA codes with rate $R = 0.25$ and relative distance $\delta = 0.05$.

While prior work [17] provides a provable analysis of the distance of RAA codes, their approach applies only to binary fields. Following recent work [14], we rely instead on a numerical analysis of RAA codes over $\mathbb{F}_q$ for $q \geq 3$. To minimize $t$ and reduce the client's work, we fix the code rate to $R = 1/4$ (so $N = 4n$) and set the relative distance to $\delta = 0.05$. We then employ a technique for analyzing linear codes [9] to derive a combinatorial upper bound on the probability that the relative distance of a random RAA code of a given dimension falls below 0.05; we refer to this as the *failure probability*.[3] (Further details about the upper bound derivation are given in Appendix A.) Since calculating this bound for dimensions $n$ of interest is infeasible, we evaluate it for $n = 2^4, 2^5, \ldots, 2^{10}$ and extrapolate to larger dimensions using exponential regression. (See Figure 3.) Our numerical analysis shows that, for our parameter choices, the failure probability is at most $2^{-21}$ for $n \geq 2^{15}$. Prior work [17] shows a procedure for sampling a generator of an RAA code that lowers the failure probability exponentially in a parameter $w$ in time $O(N^w \log(N))$. We do not employ this procedure in our analysis, but note that it could be used in our setting.

Given $R = 1/4$ and $\delta = 0.05$ as above, we use Equation (1) to set $t$ as a function of the dimension $n$ (determined by the length of the public basis $\mathbf{g}$ for the MSM we want to outsource) and the desired security parameter $\lambda$. Here, 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 (1) the value of $t$ would not change much if instead we used $\lambda = 128$. We tabulate the results in Table 2.

## 5.2 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 4, we tabulate microbenchmarks for each algorithm of the

[3] Even if a code with generator $G$ fails to achieve distance $\delta$, it may still have a pseudo-distance $\delta$ [14]; that is, no efficient adversary can find an input message $x$ such that $\mathsf{wt}(G^T x) \leq \delta N$. Thus, security may hold even when the relative distance exceeds $\delta$.

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 computation of $\mathbf{Ge}$), but this reverses as the instance size grows. We also see that, as expected, the running times of each algorithm increase 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 **33×** and improves the latency by up to **10×**. Since the EMSM time in the malicious setting doubles, the improvements change by roughly a factor of two in that setting.

| Size | Enc. | Dec. | 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×** |

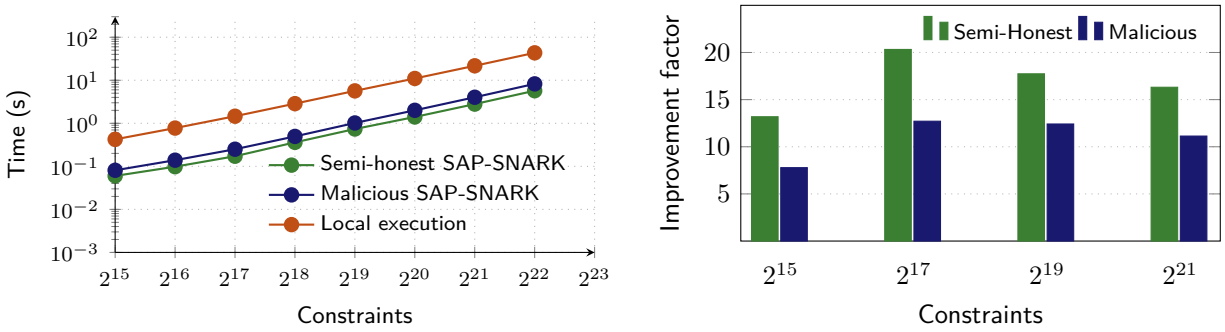
Table 4: EMSM encryption, decryption, and evaluation times compared to local computation of the MSM, in milliseconds. Numbers for malicious EMSM are in gray.

| 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: EKZG opening compared to local computation of KZG opening, in milliseconds. (Witness computation is run by the client.) Numbers for malicious EKZG are in gray.

**Setup.** We measure the cost of EMSM setup when run on a machine as powerful as the server. The setup of our EMSM scheme 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) and two permutations over $N$ elements. For instance sizes $2^{15}$ and $2^{23}$, setup requires 630 ms and 163 s, respectively.

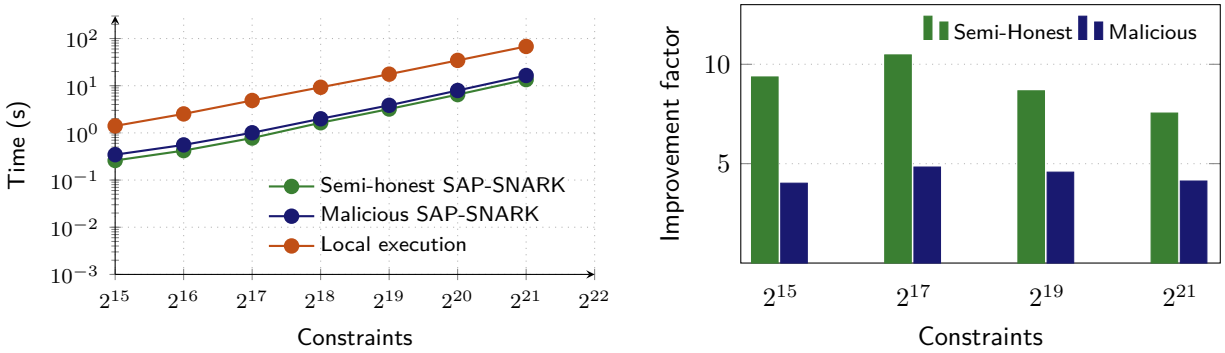
**EKZG scheme.** For completeness, we also implement and evaluate the application of our EMSM scheme to the outsourcing of KZG commitments, i.e., our EKZG scheme. Generating a KZG commitment is exactly computing one MSM; thus, the performance of the commitment phase of the EKZG scheme is identical to the performance of our EMSM scheme in Table 4. Opening a KZG commitment, however, also involves *witness computation* (evaluating the committed polynomial and computing the quotient) using field operations executed locally by the client. Our evaluation (see Table 5) 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 **22×** and improves the latency by up to **9×**; in the malicious setting, these are **13×** and **6×**, respectively.




(a) Latency of proof generation (log-log scale).

(b) Improvement in the client's active runtime.

Figure 6: Evaluation results for our server-aided Nova.




(a) Latency of proof generation (log-log scale).

(b) Improvement in the client's active runtime.

Figure 7: Evaluation results for our server-aided Groth16.

### 5.3 Server-Aided Proving for zk-SNARKs

We evaluate server-aided schemes for Nova and Groth16. Our implementation for Groth16 does not include the FFT optimization; we estimate that doing so would further improve the runtime and latency by 1.2×. We report the proof-generation latency in Figures 6(a) and 7(a), and plot the improvement in the client's running time in Figures 6(b) and 7(b). For Nova, our semi-honest scheme reduces the client's work by up to **20**× and the latency by up to **9**×; even in the malicious setting, these are **14**× and **6**×, respectively. For Groth16, our scheme reduces the client's work by up to **10**× and the latency by up to **6**× in the semi-honest case; these are **8**× and **5**×, respectively, in the malicious case.

**Comparison to prior work.** Prior work in the single-server setting [35] only applies to FRI-based zk-SNARKs (e.g., Fractal [22]) 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 [21, 38, 64], Plonk [28], and HyperPlonk [46]. However, since we do not implement SAP-SNARKs for those zk-SNARKs, a direct comparison is against not possible. Collaborative zk-SNARKs [52] is not a delegation scheme; it neither implements a client role nor improves proof-generation latency.

| | 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 6: Latency and cost for generating 100 Groth16 proofs for a circuit of size $|C| = 2^{19}$ .

The only directly comparable work is ZKSaaS [35], which implements and evaluates private 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 6, 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 separately; however, in their scheme the client only secret-shares the witnesses across the servers, so we expect lower client costs than in our scheme.

**Server-aided Plonk.** Our library does not currently support server-aided Plonk. Recall (cf. Appendix B.3) that 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 including partial products and polynomial multiplication and division. Prior work [35, 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), we estimate that our scheme for Plonk would reduce the client's work by up to 5× and latency by up to 4×; deferring FFTs to the last round (cf. Appendix C.3) could reduce the latency by up to 5×.

## Acknowledgement

We thank Alexander Block, Benedikt Bünz and Mahdi Sedaghat for their helpful comments.

## References


[1] K. Abbaszadeh, C. Pappas, J. Katz, and D. Papadopoulos. "Zero-Knowledge Proofs of Training for Deep Neural Networks". In: *31st Conference on Computer and Communications Security (CCS)*. Ed. by B. Luo, X. Liao, J. Xu, E. Kirda, and D. Lie. ACM Press, 2024, pp. 4316–4330. DOI: 10.1145/3658644.3670316.

[2] D. F. Aranha, A. Costache, A. Guimarães, and E. Soria-Vazquez. "HELIOPOLIS: Verifiable Computation over Homomorphically Encrypted Data from Interactive Oracle Proofs is Practical". In: *Advances in Cryptology—Asiacrypt 2024, Part V*. Ed. by K.-M. Chung and Y. Sasaki. Vol. 15488. LCNS. Springer, 2024, pp. 302–334. DOI: 10.1007/978-981-96-0935-2_10.

[3] S. Arora and R. Ge. "New Algorithms for Learning in Presence of Errors". In: *38th Intl. Colloquium on Automata, Languages and Programming (ICALP), Part I*. Ed. by L. Aceto, M. Henzinger, and J. Sgall. Vol. 6755. LCNS. Zurich, Switzerland: Springer, 2011, pp. 403–415. DOI: 10.1007/978-3-642-22006-7_34.

[4] F. Baldimtsi, K. K. Chalkias, Y. Ji, J. Lindstrøm, D. Maram, B. Riva, A. Roy, M. Sedaghat, and J. Wang. "zkLogin: Privacy-Preserving Blockchain Authentication with Existing Credentials". In: *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*. CCS '24. Salt Lake City, UT, USA: Association for Computing Machinery, 2024, 3182–3196. ISBN: 9798400706363. DOI: 10.1145/3658644.3690356. URL: https://doi.org/10.1145/3658644.3690356.

[5] M. Bellare and D. Micciancio. "A New Paradigm for Collision-Free Hashing: Incrementality at Reduced Cost". In: *Advances in Cryptology—Eurocrypt '97*. Ed. by W. Fumy. Vol. 1233. LCNS. Springer, 1997, pp. 163–192. DOI: 10.1007/3-540-69053-0_13.

[6] E. Ben-Sasson, I. Bentov, Y. Horesh, and M. Riabzev. "Fast Reed-Solomon Interactive Oracle Proofs of Proximity". In: *45th Intl. Colloquium on Automata, Languages and Programming (ICALP)*. Ed. by I. Chatzigiannakis, C. Kaklamanis, D. Marx, and D. Sannella. Vol. 107. Leibniz International Proceedings in Informatics (LIPIcs). Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018, 14:1–14:17. DOI: 10.4230/LIPIcs.ICALP.2018.14.

[7] E. Ben-Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza. "Zerocash: Decentralized Anonymous Payments from Bitcoin". In: *2014 IEEE Symposium on Security and Privacy*. Berkeley, CA, USA: IEEE Computer Society Press, 2014, pp. 459–474. DOI: 10.1109/SP.2014.36.

[8] F. Benhamouda, C. Chen, S. Halevi, Y. Ishai, H. Krawczyk, T. Mour, T. Rabin, and A. Rosen. *Encrypted Matrix-Vector Products from Secret Dual Codes*. Cryptology ePrint Archive, Report 2025/858. 2025. URL: https://eprint.iacr.org/2025/858.

[9] A. R. Block, Z. Fang, J. Katz, J. Thaler, H. Waldner, and Y. Zhang. "Field-Agnostic SNARKs from Expand-Accumulate Codes". In: *Advances in Cryptology—Crypto 2024, Part X*. Ed. by L. Reyzin and D. Stebila. Vol. 14929. LCNS. Springer, 2024, pp. 276–307. DOI: 10.1007/978-3-031-68403-6_9.

[10] A. Blum, M. L. Furst, M. J. Kearns, and R. J. Lipton. "Cryptographic Primitives Based on Hard Learning Problems". In: *Advances in Cryptology—Crypto '93*. Ed. by D. R. Stinson. Vol. 773. LCNS. Springer, 1994, pp. 278–291. DOI: 10.1007/3-540-48329-2_24.

[11] A. Blum, A. Kalai, and H. Wasserman. "Noise-Tolerant Learning, the Parity Problem, and the Statistical Query Model". In: *32nd Annual ACM Symposium on Theory of Computing (STOC)*. ACM Press, 2000, pp. 435–440. DOI: 10.1145/335305.335355.

[12] S. Bowe, A. Chiesa, M. Green, I. Miers, P. Mishra, and H. Wu. "ZEXE: Enabling Decentralized Private Computation". In: *2020 IEEE Symposium on Security and Privacy*. San Francisco, CA, USA: IEEE Computer Society Press, 2020, pp. 947–964. DOI: 10.1109/SP40000.2020.00050.

[13] E. Boyle, G. Couteau, N. Gilboa, and Y. Ishai. "Compressing Vector OLE". In: *25th Conference on Computer and Communications Security (CCS)*. Ed. by D. Lie, M. Mannan, M. Backes, and X. Wang. ACM Press, 2018, pp. 896–912. DOI: 10.1145/3243734.3243868.

[14] E. Boyle, G. Couteau, N. Gilboa, Y. Ishai, L. Kohl, N. Resch, and P. Scholl. "Correlated Pseudorandomness from Expand-Accumulate Codes". In: *Advances in Cryptology—Crypto 2022, Part II*. Ed. by Y. Dodis and T. Shrimpton. Vol. 13508. LCNS. Springer, 2022, pp. 603–633. DOI: 10.1007/978-3-031-15979-4_21.

[15] E. Boyle, G. Couteau, N. Gilboa, Y. Ishai, L. Kohl, P. Rindal, and P. Scholl. "Efficient Two-Round OT Extension and Silent Non-Interactive Secure Computation". In: *26th Conference on Computer and Communications Security (CCS)*. Ed. by L. Cavallaro, J. Kinder, X. Wang, and J. Katz. ACM Press, 2019, pp. 291–308. DOI: 10.1145/3319535.3354255.

[16] E. Boyle, G. Couteau, N. Gilboa, Y. Ishai, L. Kohl, and P. Scholl. "Correlated Pseudorandom Functions from Variable-Density LPN". In: *61st Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE Computer Society Press, 2020, pp. 1069–1080. DOI: 10.1109/FOCS46700.2020.00103.

[17] M. Brehm, B. Chen, B. Fisch, N. Resch, R. D. Rothblum, and H. Zeilberger. "Blaze: Fast SNARKs from Interleaved RAA Codes". In: *Advances in Cryptology—Eurocrypt 2025, Part IV*. Ed. by S. Fehr and P.-A. Fouque. Springer, 2025, pp. 123–152. ISBN: 978-3-031-91134-7.

[18] R. Canetti. "Universally Composable Security". In: *J. ACM* 67.5 (2020), 28:1–28:94.

[19] B. Chen, B. Bünz, D. Boneh, and Z. Zhang. "HyperPlonk: Plonk with Linear-Time Prover and High-Degree Custom Gates". In: *Advances in Cryptology—Eurocrypt 2023, Part II*. Ed. by C. Hazay and M. Stam. Vol. 14005. LCNS. Lyon, France: Springer, 2023, pp. 499–530. DOI: 10.1007/978-3-031-30617-4_17.

[20] A. Chiesa, Y. Hu, M. Maller, P. Mishra, P. Vesely, and N. P. Ward. "Marlin: Preprocessing zkSNARKs with Universal and Updatable SRS". In: *Advances in Cryptology—Eurocrypt 2020, Part I*. Ed. by A. Canteaut and Y. Ishai. Vol. 12105. LCNS. Zagreb, Croatia: Springer, 2020, pp. 738–768. DOI: 10.1007/978-3-030-45721-1_26.

[21] A. Chiesa, R. Lehmkuhl, P. Mishra, and Y. Zhang. "Eos: Efficient Private Delegation of zkSNARK Provers". In: *32nd USENIX Security Symposium*. Ed. by J. A. Calandrino and C. Troncoso. USENIX Association, 2023, pp. 6453–6469. URL: https://www.usenix.org/conference/usenixsecurity23/presentation/chiesa.

[22] A. Chiesa, D. Ojha, and N. Spooner. "Fractal: Post-quantum and Transparent Recursive Proofs from Holography". In: *Advances in Cryptology—Eurocrypt 2020, Part I*. Ed. by A. Canteaut and Y. Ishai. Vol. 12105. LCNS. Zagreb, Croatia: Springer, 2020, pp. 769–793. DOI: 10.1007/978-3-030-45721-1_27.

[23] arkworks contributors. *arkworks zkSNARK ecosystem*. 2022. URL: https://arkworks.rs.

[24] A. Daftardar, J. Mo, J. Ah-kiow, B. Bünz, R. Karri, S. Garg, and B. Reagen. "Need for zkSpeed: Accelerating HyperPlonk for Zero-Knowledge Proofs". In: *ISCA 2025*. ACM, 2025, pp. 1986–2001. ISBN: 9798400712616. DOI: 10.1145/3695053.3731021. URL: https://doi.org/10.1145/3695053.3731021.

[25] T. Datta, B. Chen, and D. Boneh. "VerITAS: Verifying Image Transformations at Scale". In: *2025 IEEE Symposium on Security and Privacy*. 2025, pp. 4606–4623. DOI: 10.1109/SP61157.2025.00097.

[26] E. Druk and Y. Ishai. "Linear-Time Encodable Codes Meeting The Gilbert-Varshamov Bound and Their Cryptographic Applications". In: *5th Conference on Innovations in Theoretical Computer Science (ITCS)*. Ed. by M. Naor. Princeton, NJ, USA: Association for Computing Machinery, 2014, pp. 169–182. DOI: 10.1145/2554797.2554815.

[27] J. Ernstberger, S. Chaliasos, G. Kadianakis, S. Steinhorst, P. Jovanovic, A. Gervais, B. Livshits, and M. Orrù. "zk-Bench: A Toolset for Comparative Evaluation and Performance Benchmarking of SNARKs". In: *Security and Cryptography for Networks (SCN), Part I*. Ed. by C. Galdi and D. H. Phan. Vol. 14973. LNCS. Springer, 2024, pp. 46–72. DOI: 10.1007/978-3-031-71070-4_3. URL: https://doi.org/10.1007/978-3-031-71070-4_3.

[28] Z. Fang, S. Garg, B. Roberts, W. Wu, and Y. Zhang. *Collaborative zkSNARKs with Sublinear Prover Time and Constant Proof Size*. Cryptology ePrint Archive, Report 2025/1388. 2025. URL: https://eprint.iacr.org/2025/1388.

[29] Fermah. https://www.fermah.xyz.

[30] A. Gabizon and Z. J. Williamson. *plookup: A Simplified Polynomial Protocol for Lookup Tables*. Cryptology ePrint Archive, Report 2020/315. 2020. URL: https://eprint.iacr.org/2020/315.

[31] A. Gabizon, Z. J. Williamson, and O. Ciobotaru. *PLONK: Permutations over Lagrange-bases for Oecumenical Noninteractive arguments of Knowledge*. Cryptology ePrint Archive, Report 2019/953. 2019. URL: https://eprint.iacr.org/2019/953.

[32] M. Gama, E. H. Beni, J. Kang, J. Spiessens, and F. Vercauteren. *Blind zkSNARKs for Private Proof Delegation and Verifiable Computation over Encrypted Data*. Cryptology ePrint Archive, Paper 2024/1684. 2024. URL: https://eprint.iacr.org/2024/1684.

[33] S. Garg, A. Goel, A. Jain, G.-V. Policharla, and S. Sekar. "zkSaaS: Zero-Knowledge SNARKs as a Service". In: *32nd USENIX Security Symposium*. Ed. by J. A. Calandrino and C. Troncoso. USENIX Association, 2023, pp. 4427–4444. URL: https://www.usenix.org/conference/usenixsecurity23/presentation/garg.

[34] S. Garg, A. Goel, A. Jain, B. Roberts, and S. Sekar. "Malicious Security in Collaborative zk-SNARKs: More than Meets the Eye". In: *Adv. in Cryptology—Crypto 2025, Part VII*. Vol. 16006. LCNS. Available at https://ia.cr/2025/1026. 2025, pp. 396–428.

[35] S. Garg, A. Goel, and M. Wang. "How to Prove Statements Obliviously?" In: *Advances in Cryptology—Crypto 2024, Part X*. Ed. by L. Reyzin and D. Stebila. Vol. 14929. LCNS. Springer, 2024, pp. 449–487. DOI: 10.1007/978-3-031-68403-6_14.

[36] Google. *Private Identification in Google Wallet*. https://blog.google/products/google-pay/google-wallet-age-identity-verifications.

[37] J. Groth. "On the Size of Pairing-Based Non-interactive Arguments". In: *Advances in Cryptology—Eurocrypt 2016, Part II*. Ed. by M. Fischlin and J.-S. Coron. Vol. 9666. LCNS. Vienna, Austria: Springer, 2016, pp. 305–326. DOI: 10.1007/978-3-662-49896-5_11.

[38] Y. Hu, P. Mishra, X. Wang, J. Xie, K. Yang, Y. Yu, and Y. Zhang. "DFS: Delegation-Friendly zkSNARK and Private Delegation of Provers". In: *Proceedings of the 34th USENIX Conference on Security Symposium*. USA: USENIX Association, 2025. ISBN: 978-1-939133-52-6.

[39] G. Kadianakis, A. Zapico, H. Hafezi, and B. Bünz. *KZH-Fold: Accountable Voting from Sublinear Accumulation*. Cryptology ePrint Archive, Paper 2025/144. 2025. URL: https://eprint.iacr.org/2025/144.

[40] A. Kate, G. M. Zaverucha, and I. Goldberg. "Constant-Size Commitments to Polynomials and Their Applications". In: *Advances in Cryptology—Asiacrypt 2010*. Ed. by M. Abe. Vol. 6477. LCNS. Singapore: Springer, 2010, pp. 177–194. DOI: 10.1007/978-3-642-17373-8_11.

[41] A. Kothapalli, S. Setty, and I. Tzialla. "Nova: Recursive Zero-Knowledge Arguments from Folding Schemes". In: *Advances in Cryptology—Crypto 2022, Part IV*. Ed. by Y. Dodis and T. Shrimpton. Vol. 13510. LCNS. Springer, 2022, pp. 359–388. DOI: 10.1007/978-3-031-15985-5_13.

[42] A. Kothapalli and S. T. V. Setty. "HyperNova: Recursive Arguments for Customizable Constraint Systems". In: *Advances in Cryptology—Crypto 2024, Part X*. Ed. by L. Reyzin and D. Stebila. Vol. 14929. LCNS. Springer, 2024, pp. 345–379. DOI: 10.1007/978-3-031-68403-6_11.

[43] H. Liu, X. Wang, K. Yang, and Y. Yu. "The Hardness of LPN over Any Integer Ring and Field for PCG Applications". In: *Advances in Cryptology—Eurocrypt 2024, Part VI*. Ed. by M. Joye and G. Leander. Vol. 14656. LCNS. Zurich, Switzerland: Springer, 2024, pp. 149–179. DOI: 10.1007/978-3-031-58751-1_6.

[44] T. Liu, T. Xie, J. Zhang, D. Song, and Y. Zhang. "Pianist: Scalable zkRollups via Fully Distributed Zero-Knowledge Proofs". In: *2024 IEEE Symposium on Security and Privacy*. San Francisco, CA, USA: IEEE Computer Society Press, 2024, pp. 1777–1793. DOI: 10.1109/SP54263.2024.00035.

[45] T. Liu, X. Xie, and Y. Zhang. "zkCNN: Zero Knowledge Proofs for Convolutional Neural Network Predictions and Accuracy". In: *28th Conference on Computer and Communications Security (CCS)*. Ed. by G. Vigna and E. Shi. ACM Press, 2021, pp. 2968–2985. DOI: 10.1145/3460120.3485379.

[46] X. Liu, Z. Zhou, Y. Wang, Y. Pang, J. He, B. Zhang, X. Yang, and J. Zhang. "Scalable Collaborative zk-SNARK and Its Application to Fully Distributed Proof Delegation". In: *Proceedings of the 34th USENIX Conference on Security Symposium*. USA: USENIX Association, 2025. ISBN: 978-1-939133-52-6.

[47] M. Maller, S. Bowe, M. Kohlweiss, and S. Meiklejohn. "Sonic: Zero-Knowledge SNARKs from Linear-Size Universal and Updatable Structured Reference Strings". In: *26th Conference on Computer and Communications Security (CCS)*. Ed. by L. Cavallaro, J. Kinder, X. Wang, and J. Katz. ACM Press, 2019, pp. 2111–2128. DOI: 10.1145/3319535.3339817.

[48] D. Micciancio and P. Mol. "Pseudorandom Knapsacks and the Sample Complexity of LWE Search-to-Decision Reductions". In: *Advances in Cryptology—Crypto 2011*. Ed. by P. Rogaway. Vol. 6841. LCNS. Springer, 2011, pp. 465–484. DOI: 10.1007/978-3-642-22792-9_26.

[49] Mina. https://minaprotocol.com.

[50] A. Naveh and E. Tromer. "PhotoProof: Cryptographic Image Authentication for Any Set of Permissible Transformations". In: *2016 IEEE Symposium on Security and Privacy*. San Jose, CA, USA: IEEE Computer Society Press, 2016, pp. 255–271. DOI: 10.1109/SP.2016.23.

[51] Nexus. https://nexus.xyz.

[52] A. Ozdemir and D. Boneh. "Experimenting with Collaborative zk-SNARKs: Zero-Knowledge Proofs for Distributed Secrets". In: *31st USENIX Security Symposium*. Ed. by K. R. B. Butler and K. Thomas. USENIX Association, 2022, pp. 4291–4308. URL: https://www.usenix.org/conference/usenixsecurity22/presentation/ozdemir.

[53] T. P. Pedersen. "Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing". In: *Advances in Cryptology—Crypto '91*. Ed. by J. Feigenbaum. Vol. 576. LCNS. Springer, 1992, pp. 129–140. DOI: 10.1007/3-540-46766-1_9.

[54] =nil; Proof Market. https://nil.foundation/blog/post/proof-market.

[55] Proto-Danksharding. https://www.eip4844.com.

[56] S. Raghuraman, P. Rindal, and T. Tanguy. "Expand-Convolute Codes for Pseudorandom Correlation Generators from LPN". In: *Advances in Cryptology—Crypto 2023, Part IV*. Ed. by H. Handschuh and A. Lysyanskaya. Vol. 14084. LCNS. Springer, 2023, pp. 602–632. DOI: 10.1007/978-3-031-38551-3_19.

[57] M. Rosenberg, T. Mopuri, H. Hafezi, I. Miers, and P. Mishra. "Hekaton: Horizontally-Scalable zkSNARKs Via Proof Aggregation". In: *31st Conference on Computer and Communications Security (CCS)*. Ed. by B. Luo, X. Liao, J. Xu, E. Kirda, and D. Lie. ACM Press, 2024, pp. 929–940. DOI: 10.1145/3658644.3690282.

[58] M. Rosenberg, J. D. White, C. Garman, and I. Miers. "zk-creds: Flexible Anonymous Credentials from zkSNARKs and Existing Identity Infrastructure". In: *2023 IEEE Symposium on Security and Privacy*. San Francisco, CA, USA: IEEE Computer Society Press, 2023, pp. 790–808. DOI: 10.1109/SP46215.2023.10179430.

[59] N. Samardzic, S. Langowski, S. Devadas, and D. Sanchez. "Accelerating Zero-Knowledge Proofs Through Hardware-Algorithm Co-Design". In: *2024 57th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 2024, pp. 366–379. DOI: 10.1109/MICRO61859.2024.00035.

[60] Succinct. https://testnet.succinct.xyz/prove.

[61] N. Tyagi, B. Fisch, A. Zitek, J. Bonneau, and S. Tessaro. "VeRSA: Verifiable Registries with Efficient Client Audits from RSA Authenticated Dictionaries". In: *29th Conference on Computer and Communications Security (CCS)*. Ed. by H. Yin, A. Stavrou, C. Cremers, and E. Shi. ACM Press, 2022, pp. 2793–2807. DOI: 10.1145/3548606.3560605.

[62] D. Wagner. "A Generalized Birthday Problem". In: *Advances in Cryptology—Crypto 2002*. Ed. by M. Yung. Vol. 2442. LCNS. Springer, 2002, pp. 288–303. DOI: 10.1007/3-540-45708-9_19.

[63] H. Wu, W. Zheng, A. Chiesa, R. A. Popa, and I. Stoica. "DIZK: A Distributed Zero Knowledge Proof System". In: *27th USENIX Security Symposium*. Ed. by W. Enck and A. P. Felt. USENIX Association, 2018, pp. 675–692. URL: https://www.usenix.org/conference/usenixsecurity18/presentation/wu.

[64] Y. Yang, Y. Cheng, K. Wang, X. Li, J. Sun, J. Shen, X. Dong, Z. Cao, G. Yang, and R. H. Deng. "Siniel: Distributed Privacy-Preserving zkSNARK". In: *ISOC Network and Distributed System Security Symposium – NDSS 2025*. San Diego, CA, USA: The Internet Society, 2025.

[65] Zama. *Benchmarks for TFHE-rs*. https://docs.zama.ai/tfhe-rs/get-started/benchmarks.

[66] Zcash. https://z.cash.

[67] Zcash. https://github.com/zcash/librustzcash.

[68] B. Zhang, L. Jiao, and M. Wang. "Faster Algorithms for Solving LPN". In: *Advances in Cryptology—Eurocrypt 2016, Part I*. Ed. by M. Fischlin and J.-S. Coron. Vol. 9665. LCNS. Vienna, Austria: Springer, 2016, pp. 168–195. DOI: 10.1007/978-3-662-49890-3_7.

# A Distance Analysis of RAA Codes

We bound the distance of an RAA code over arbitrary fields. Let $\mathbf{G} \in \mathbb{F}^{n \times N}$ be an RAA 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 $\sigma_1, \sigma_2$ over $[N]$; and $\mathbf{A}$ is the accumulation matrix. Analyzing the distance of the code for a fixed relative distance $\delta$ amounts to bounding the failure probability: $\Pr[\exists \mathbf{x} \in \mathbb{F}^n \setminus \{0\} : \mathsf{wt}(\mathbf{G}^T \mathbf{x}) \leq \delta N]$. We recall the following two lemmas from a prior work [9].

**Lemma A.1.** *For any* $\mathbf{x} \in \mathbb{F}^N$, $\mathsf{wt}(\mathbf{A}^T \mathbf{x}) \geq \lceil \mathsf{wt}(\mathbf{x})/2 \rceil$ .

**Lemma A.2.** *For a uniform* $\mathbf{x} \in \mathbb{F}^N$ *with* $w = \mathsf{wt}(\mathbf{x}) \neq 0$ *and any* $h \geq \lceil w/2 \rceil$, *it holds that*

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

We then apply a union bound on the failure probability:

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

Note that $\mathbf{M}_{\sigma_1}$ and $\mathbf{M}_{\sigma_2}$ are permutation matrices; thus, they keep the input and output weights unchanged. For the repetition matrix, it holds that $\mathsf{wt}(\mathbf{F}_r^T \mathbf{x}) = r \cdot \mathsf{wt}(\mathbf{x})$. Following Lemma A.1, we know that $w_2 \geq \lceil r w_1 / 2 \rceil$ and $w_3 \geq \lceil w_2 / 2 \rceil$. With these, we can simplify the bound as

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

Then, we apply Lemma A.2 to obtain the following upper bound on the failure probability:

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

# B Details of zk-SNARKs

We review the prover algorithms in Groth16 [37], Nova [41], and Plonk [31]. For $n > 0$, we say $\omega \in \mathbb{F}_q^*$ is a primitive $n$th root of unity if $\omega^n = 1, \omega^m \neq 1$ for $1 \leq m < n$. We let $D_\omega := \{\omega^0, \omega^1, \ldots, \omega^{n-1}\}$.

**Fast Fourier transform and polynomial operations.** The Fast Fourier Transform (FFT) enables the evaluation of a degree-$n$ polynomial at $O(n)$ points—typically the points over $D_\omega$ for a primitive $n$th root of unity $\omega$—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 to recover the result. This extends to polynomial division, since any polynomial division is reduced to two polynomial multiplications.

## B.1 The Nova Protocol

The key building block in Nova is a folding scheme for the *committed, relaxed* R1CS. Relaxed R1CS is 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 $u \in \mathbb{F}_q$. Then, given $\mathbf{z} = \mathbf{x} \| \mathbf{w}$, the relation is satisfied if and only if $(\mathbf{Az}) \odot (\mathbf{Bz}) = u \cdot (\mathbf{Cz}) + \mathbf{e}$. In committed, relaxed R1CS, 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 $\mathbf{w}_b$, respectively. Then, for folding the instances, the prover computes a cross term

$$\mathbf{t} = (\mathbf{Az}_0) \odot (\mathbf{Bz}_1) + (\mathbf{Az}_1) \odot (\mathbf{Bz}_0) - u_0 \cdot (\mathbf{Cz}_1) - u_1 \cdot (\mathbf{Cz}_0),$$

and sends a Pedersen commitment to $\mathbf{t}$—denoted by $\bar{\mathbf{t}}$—to the verifier. Then, given a random challenge $r \leftarrow \mathbb{F}_q$ from the verifier, 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 also 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. Moreover, it is sound since if either input instance is invalid, the folded instance will be unsatisfied over the choice of $r$.

Let $F$ be the function that computes 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 commitment scheme parameters.
- Prove: it proceeds in two steps: (i) first, for each invocation of $F'$, it 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.

Note that the main prover cost is the first step, and the final zk-SNARK is invoked only once.

## B.2 The Groth16 Protocol

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

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

$\forall i \in [n], j \in [m]$. R1CS with $\mathbf{z} := \mathbf{x}\|\mathbf{w}$ is satisfied iff:

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

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

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

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 $\mathbf{Cz}$, respectively. 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 $\pi := (\pi_a, \pi_b, \pi_c)$ such that

  $$\pi_a := \alpha \cdot g_1 + r \cdot (\delta \cdot g_1) + \sum_{j\in[m]} z_j \cdot (f_{A,j}(v) \cdot g_1),$$
  $$\pi_b := \beta \cdot g_2 + s \cdot (\delta \cdot g_2) + \sum_{j\in[m]} z_j \cdot (f_{B,j}(v) \cdot g_2),$$
  $$\pi_c := s \cdot \pi_a + r \cdot (\beta \cdot g_1 + \sum_{j\in[m]} z_j \cdot (f_{B,j}(v) \cdot g_1))$$
  $$+ \sum_{j\in[\ell:m]} z_j \cdot (\delta^{-1} f_j(v) \cdot g_1) + \sum_{i\in[n-1]} q_i \cdot (\delta^{-1} v^i t(v) \cdot g_1).$$

### B.3 The Plonk Protocol

We recall the building blocks and the Plonk constraint system; then, we describe the Plonk prover.

**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 $\omega$ is a primitive $n$th 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 received polynomials at random points and checks their consistency. In practice, oracles are instantiated with polynomial commitments. In summary, the prover's work is to compute partial products $\prod_{i\in[s-1]} f(\omega^i)$, interpolate $t(X)$, and then invoke polynomial divisions and commitments.

**Plonk arithmetization.** Unlike Groth16 and Nova, Plonk employs an arithmetization that differs from R1CS. Given integers $m, n \in \mathbb{N}$, the constraint system for some fan-in-two circuit with unlimited fan-out, $m$ wires, and $n$ arithmetic gates is defined by 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 $\mathbf{q}_L$, $\mathbf{q}_R$, $\mathbf{q}_O$, $\mathbf{q}_C$, $\mathbf{q}_M$, and $\mathbf{q}_C$ denote the left, right, output, multiplication, and constant selector vectors, respectively. Given an assignment $\mathbf{z} := (\mathbf{x} \| \mathbf{w}) \in \mathbb{F}_q^m$, 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 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 correct ordering via a permutation $\sigma : [3n] \to [3n]$; the prover shows that $v_i = v_{\sigma(i)}$ for all $i \in [3n]$.

**Proving algorithm.** We outline the setup and proving algorithms of the Plonk protocol.

- Setup: outputs the commitment scheme parameters.
- Prove: consists of 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 $\omega$ 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 (cf. Section 2.2) to $a(X)$, $b(X)$, and $c(X)$.
2. Let $\kappa, \eta \in \mathbb{F}_q^*$ with distinct domains $D_\omega$, $D_{\kappa\omega}$, and $D_{\eta\omega}$. For random $\beta, \tau \in \mathbb{F}_q$, let

$$\begin{aligned} h(X) :=& (a(X) + \beta X + \tau)(b(X) + \beta\kappa X + \tau) \\ & (c(X) + \beta\eta X + \tau), \\ g(X) :=& (a(X) + \beta\sigma_a(X) + \tau)(b(X) + \beta\kappa\sigma_b(X) + \tau) \\ & (c(X) + \beta\eta\sigma_c(X) + \tau), \end{aligned}$$

   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 $\sigma$ 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 $D_\omega$. We let $p_0(X)$, $p_1(X)$, and $p_2(X)$ be as follows:

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

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

4. The prover computes a quotient $q(X)$ by dividing $p(X)$ by $X^n - 1$ and sends a commitment to $q(X)$. Concretely, 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.

# C Our Server-Aided Proving Protocols

## C.1 The Server-Aided Nova Protocol

We describe a server-aided proving scheme for Nova. Recall the Nova protocol from Section B.1. 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 8, where $\mathcal{F}_{\mathsf{MSM}}$ can be instantiated by our $\mathsf{EMSM}$ scheme. The following is immediate:

**Theorem C.1.** *In the $\mathcal{F}_{\mathsf{MSM}}$-hybrid model, $\Pi_{\mathsf{Nova}}$ UC-realizes the functionality $\mathcal{F}_{\mathsf{Prove}}$ for Nova.*

We remark that while our presentation of Nova is simplified and a few minor steps are abstracted (e.g., hashing inputs/outputs), our technique immediately extends to the full Nova protocol.

## C.2 The Server-Aided Groth16 Protocol

We build an SAP-SNARK from Groth16. Recall the design of the prover algorithm from Section B.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 the proof triple $\pi = (\pi_a, \pi_b, \pi_c)$. Given $\mathbf{q}$, which is the coefficient vector of $q(X)$ and $\mathbf{z}$, we have $\pi_a = \alpha \cdot g_1 + r \cdot (\delta \cdot g_1) + \langle \mathbf{z}, \mathbf{g}_a \rangle, \pi_b = \beta \cdot g_2 + s \cdot (\delta \cdot g_1) + \langle \mathbf{z}, \mathbf{g}_b \rangle, \pi_c = s \cdot \pi_a + r \cdot \beta \cdot g_1 + \langle r \cdot \mathbf{z}, \mathbf{g}_{c,rz} \rangle + \langle \mathbf{z}, \mathbf{g}_{c,z} \rangle + \langle \mathbf{q}, \mathbf{g}_{c,q} \rangle$. where $\mathbf{g}_a$, $\mathbf{g}_b$,

$\mathbf{g}_{c,rz}$, $\mathbf{g}_{c,z}$, and $\mathbf{g}_{c,q}$ are included in $\mathsf{pp}$. The client encrypts $\mathbf{z}$ and $r \cdot \mathbf{z}$ with EMSM and delegates $\langle \mathbf{z}, \mathbf{g}_a \rangle$, $\langle \mathbf{z}, \mathbf{g}_b \rangle$, $\langle r \cdot \mathbf{z}, \mathbf{g}_{c,rz} \rangle$, and $\langle \mathbf{z}, \mathbf{g}_{c,z} \rangle$.

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

$$\mathbf{h} := ((\mathbf{F}_D \mathbf{F}_{D_{\omega'}}^{-1} \cdot \mathbf{Az}) \odot (\mathbf{F}_D \mathbf{F}_{D_{\omega'}}^{-1} \cdot \mathbf{Bz}) - (\mathbf{F}_D \mathbf{F}_{D_{\omega'}}^{-1} \cdot \mathbf{Cz})),$$

where $\mathbf{t}$ denotes the coefficients of $t(X)$, and $\odot$ and $\oslash$ denote the element-wise multiplication and division, respectively. This computation 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 $\langle \mathbf{z}, (\mathbf{F}_D \mathbf{F}_{D_{\omega'}}^{-1} \cdot \mathbf{C})^T \mathbf{g}_{c,h} \rangle$.

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 9.

**Theorem C.2.** *In the $\mathcal{F}_{\mathsf{MSM}}$-hybrid model, $\Pi_{\mathsf{Groth}}$ UC-realizes the functionality $\mathcal{F}_{\mathsf{Prove}}$ for Groth16.*

**Remark C.1.** *Ciphertexts in our EMSM scheme are independent of the basis. As discussed in Section 3.2, a single ciphertext suffices to outsource multiple MSMs with the same input vector. Thus, in practice, one ciphertext suffices for outsourcing all four MSMs with input the vector $\mathbf{z}$.*

### C.3 The Server-Aided Plonk Protocol

We present server-aided Plonk. Recall from Section B.3 that the core subroutine of the Plonk prover is the KZG commitment scheme, which can be outsourced using EKZG, as discussed in Section 3.4.

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 that point.

We provide a more formal description of server-aided Plonk in the $\mathcal{F}_{\mathsf{MSM}}$-hybrid model in Figure 10.

**Theorem C.3.** *In the $\mathcal{F}_{\mathsf{MSM}}$-hybrid model, $\Pi_{\mathsf{Plonk}}$ 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.

$\underline{\Pi_{\mathsf{Nova}}}$

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 $(i, \mathbf{z}_0, \mathbf{z}_i)$, $(\bar{\mathbf{e}}_i, u_i, \bar{\mathbf{w}}_i, \mathbf{x}_i)$, $(\bar{\mathbf{e}}_f, u_f, \bar{\mathbf{w}}_f, \mathbf{x}_f)$, $\pi_f$, and $\pi_i$ 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 8: The server-aided proving scheme for Nova.

$\underline{\Pi_{\mathsf{Groth}}}$

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 $\omega$ 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

$$\mathbf{g}_a := (f_{A,j})_{j\in[m]} \cdot g_1, \quad \mathbf{g}_b := (f_{B,j})_{j\in[m]} \cdot g_2,$$
$$\mathbf{g}_{c,z} := (\delta^{-1} f_j)_{j\in[\ell:m]} \cdot g_1, \quad \mathbf{g}_{c,rz} := (f_{B,j})_{j\in[m]} \cdot g_1, \quad \mathbf{g}_{c,q} := (\delta^{-1} v^i t(v))_{i\in[n-1]} \cdot g_1,$$
$$\alpha\beta \cdot g_T, \quad \{\alpha, \beta, \delta, \{v^i\}_{i\in[n]}, \{\gamma^{-1} f_j(x)\}_{j\in[\ell]}\} \cdot g_1, \quad \{\beta, \gamma, \delta, \{v^i\}_{i\in[n]}\} \cdot g_2.$$

Let $\mathbf{g}^L_{c,h} := ((\mathbf{F}_D^{-1})^T \mathbf{g}_{c,q}) \oslash (\mathbf{F}_D \cdot \mathbf{t})$ and $\mathbf{g}^R_{c,h} := (\mathbf{F}_D \mathbf{F}^{-1}_{D_{\omega'}} \cdot \mathbf{C})^T \mathbf{g}^L_{c,h}$ 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 $\mathbf{g}_a$, $\mathbf{g}_b$, $\mathbf{g}_{c,z}$, $\mathbf{g}_{c,rz}$, $\mathbf{g}^L_{c,h}$, and $\mathbf{g}^R_{c,h}$.
- $C$ samples a random $r \in \mathbb{F}$.
- Let $\mathbf{z} = \mathbf{x} \| \mathbf{w}$. $C$ computes $\mathbf{z}_{ab} := (\mathbf{F}_D \mathbf{F}^{-1}_{D_{\omega'}} \cdot \mathbf{A}\mathbf{z}) \odot (\mathbf{F}_D \mathbf{F}^{-1}_{D_{\omega'}} \cdot \mathbf{B}\mathbf{z})$.
- $C$ and $S$ send $(\mathsf{eval}, \mathbf{z})$ to $\mathcal{F}^{\mathbf{g}_a}_{\mathsf{MSM}}$, $\mathcal{F}^{\mathbf{g}_b}_{\mathsf{MSM}}$, $\mathcal{F}^{\mathbf{g}_{c,z}}_{\mathsf{MSM}}$, $\mathcal{F}^{\mathbf{g}^R_{c,h}}_{\mathsf{MSM}}$, $(\mathsf{eval}, r \cdot \mathbf{z})$ to $\mathcal{F}^{\mathbf{g}_{c,rz}}_{\mathsf{MSM}}$, and $(\mathsf{eval}, \mathbf{z}_{ab})$ to $\mathcal{F}^{\mathbf{g}^L_{c,h}}_{\mathsf{MSM}}$; $C$ receives $\langle \mathbf{z}, \mathbf{g}_a \rangle$, $\langle \mathbf{z}, \mathbf{g}_b \rangle$, $\langle \mathbf{z}, \mathbf{g}_{c,z} \rangle$, $\langle r \cdot \mathbf{z}, \mathbf{g}_{c,rz} \rangle$, $\langle \mathbf{z}, \mathbf{g}^R_{c,h} \rangle$, and $\langle \mathbf{z}_{ab}, \mathbf{g}^L_{c,h} \rangle$.
- $C$ outputs the proof $\pi := (\pi_a, \pi_b, \pi_c)$ such that

$$\pi_a = \alpha \cdot g_1 + r \cdot (\delta \cdot g_1) + \langle \mathbf{z}, \mathbf{g}_a \rangle, \pi_b = \beta \cdot g_2 + s \cdot (\delta \cdot g_1) + \langle \mathbf{z}, \mathbf{g}_b \rangle,$$
$$\pi_c = s \cdot \pi_a + r \cdot \beta \cdot g_1 + \langle r \cdot \mathbf{z}, \mathbf{g}_{c,rz} \rangle + \langle \mathbf{z}, \mathbf{g}_{c,z} \rangle + \langle \mathbf{q}, \mathbf{g}_{c,q} \rangle.$$

Figure 9: The server-aided proving scheme for Groth16.

$\underline{\Pi_{\mathsf{Plonk}}}$

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 $\omega$ 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 commitments. 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 Fiat-Shamir challenge $\beta, \eta$, $C$ 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 the Fiat-Shamir challenge $\rho$, $C$ computes $\mathbf{z}_p$ as follows.

$$\begin{aligned} \mathbf{z}_p \;:=\; & ((q_L)_i z_{a_i} + (q_R)_i z_{b_i} + (q_O)_i z_{c_i} + (q_M)_i z_{a_i} z_{b_i} + (q_C)_i \\ & + \rho((\mathbf{z}_t)_{i+1} - ((\mathbf{z}_t)_i (\mathbf{z}_h)_i)/(\mathbf{z}_g)_i) + \rho^2(\ell_n(\omega^i)((\mathbf{z}_t)_i - 1)))_{i \in [n]}. \end{aligned}$$

- $C$ 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 10: The server-aided proving scheme for Plonk.