Samaritan: Linear-time Prover SNARK from New Multilinear Polynomial Commitments

Chaya Ganesh[1], Sikhar Patranabis[2], and Nitin Singh[3]

[1]Indian Institute of Science
[2,3]IBM Research India

Abstract

We study linear-time prover SNARKs and make the following contributions:

We provide a framework for transforming a univariate polynomial commitment scheme into a multilinear polynomial commitment scheme. Our transformation is *generic*, can be instantiated with any univariate scheme and improves on prior transformations like Gemini (EUROCRYPT 2022) and Virgo (S&P 2020) in all relevant parameters: proof size, verification complexity, and prover complexity. Instantiating the above framework with the KZG univariate polynomial commitment scheme, we get SamaritanPCS – the *first* multilinear polynomial commitment scheme with constant proof size and linear-time prover. The proof size is just 368 bytes, which is the smallest among all multilinear polynomial commitment schemes. Our scheme also has excellent batching properties, wherein proving k evaluations over the hypercube of size n incurs $O(n + k\sqrt{n})$ cryptographic work, resulting in substantially amortized prover work over several evaluations.

We construct LogSpartan – a new multilinear PIOP for R1CS based on recent techniques for lookup arguments. Compiling this PIOP using SamaritanPCS gives Samaritan – a SNARK in the universal and updatable SRS setting. Samaritan has linear-time prover, logarithmic verification and logarithmic proof size. Concretely, its proof size is one of the smallest among other known linear-time prover SNARKs without relying on concretely expensive proof recursion techniques. For an R1CS instance with 1 million constraints, Samaritan (over the BLS12-381 curve) has a proof size of 6.2KB.

We compare Samaritan with other linear-time prover SNARKs in the updatable setting. We asymptotically improve on the $\log^2 n$ proof size of Spartan. Unlike Libra (CRYPTO 2019), the argument size of Samaritan is independent of the circuit depth. Compared to Gemini (EUROCRYPT 2022), Samaritan achieves $3\times$ smaller argument size at 1 million constraints. We are competitive with the very recently proposed MicroSpartan (S&P 2025) and linear-time SNARKs for the Plonkish constraint system such as HyperPlonk (EUROCRYPT 2023).

Contents

1 Introduction

A Succinct Non-interactive ARgument of Knowledge (SNARK) allows a prover to convince the verifier about the integrity of a computation such that the proof size and the verifier's work to check the proof do not scale with the size of the computation. For a relation $\mathcal{R}$, a prover can produce a proof π to attest to $\mathcal{R}(\mathsf{x}, \mathsf{w}) = 1$ so that the verifier can check π non-interactively. Succinctness refers to the requirement that the size of π be $O_\lambda(\log t)$ and the verifier's complexity be $O_\lambda(n + \log t)$ where $|\mathsf{x}| = n$ and t is the size of the verification circuit for $\mathcal{R}$ (the number of gates in the circuit that checks $\mathcal{R}(\mathsf{x}, \mathsf{w}) = 1$).

Micali [Mic94] constructed argument systems with communication complexity smaller than the length of the witness based on Probabilistically Checkable Proofs (PCP). This succinct argument was transformed by Kilian [Kil92] into a SNARK in the Random Oracle Model (ROM). Several early SNARKs [Gro10, Lip12, BCCT12, BCI+13, GGPR13, PHGR13, BCG+13, Lip13, BCTV14] with constant proof size and verification complexity are based on linear PCP compiled using a cryptographic tool like linear-only encodings (for instance, exponentiation in a bilinear group).

Modular SNARK Constructions. A modular approach for designing efficient argument systems consists of two components: an information theoretic protocol in an idealized model (e.g., probabilistically checkable proof or PCP, linear PCP, interactive oracle proofs or IOPs), and a cryptographic compiler. The information-theoretic protocol is then compiled cryptographically to obtain an argument system.

Many popular SNARKs [CHM+20, GWC19] use Polynomial Interactive Oracle Proofs (PIOPs) as the information-theoretic component, and then compile the PIOP into a succinct argument system using a cryptographic tool called a Polynomial Commitment Scheme (PCS). In a PIOP, the prover and the verifier interact where the prover provides oracle access to a set of polynomials, and the verifier sends random challenges. The verifier can then query these polynomial oracles at challenge points to obtain evaluations, and in the end output accept or reject. A PCS allows a prover to commit to a polynomial f of bounded degree such that a verifier can query for evaluations $P(x)$ together with proofs that the provided evaluations are indeed consistent with the commitment. Compiling a PIOP with a PCS involves realizing the polynomial oracles using polynomial commitments and the queries using PCS evaluation proofs. This yields a public-coin succinct argument, which is then compiled using Fiat-Shamir [FS87] to obtain a SNARK in the Random Oracle Model (ROM).

Models for SNARKs. Typical SNARKs are either in the Structured Reference String (SRS) model or in idealized models (like ROM, Generic Group Model or GGM, Algebraic Group Model or AGM). Some SNARKs in the SRS model have a randomized preprocessing phase with secret random coins and therefore a *trusted setup*. SNARKs where the setup phase only uses public randomness and verifier randomness consists of only *public coins*

are called *transparent*. In SNARKS with *Universal* SRS, the one-time setup can be used to prove statements about any computation, as opposed to a circuit-dependent setup required in preprocessing-based SNARKs. The SRS is *updatable* if there is a mechanism for parties to update by contributing to the randomness of the SRS, and an SRS is trusted as long as at least one of the updates is honest.

Univariate and Multilinear PCS. SNARKs compiled using the PCS-based modular approach outlined above inherit the complexity and setup assumptions from the underlying PCS. Examples of SNARKs compiled using *univariate* PCS include Sonic [MBKM19], Marlin [CHM+20], Plonk [GWC19]. Using the KZG [KZG10] univariate PCS yields SNARKs in the updatable SRS model with constant proof size and constant verification complexity, and a prover that is $O(n \log n)$ where n is the size of the computation (often represented by the size of the circuit or the number of R1CS constraints). More recently, SNARKs have been obtained by compiling multilinear PIOPs using multilinear PCS. Prominent examples include Hyrax [WTs+18], Libra [XZZ+19], Spartan [Set20], Kopis [SL20], Xiphos [SL20], Gemini [BCHO22], Orion [XZS22], Brakedown [GLS+23] and Hyperplonk [CBBZ23].

SNARKs with Linear-time Prover. While early research on SNARKs focused on reducing argument size and verification time, many recent works have focused on optimizing prover time. This is especially important for applications requiring large computations [rol21, Lab17] where the prover effort can be a bottleneck. Several recent SNARKs based on multilinear PIOPs and multilinear PCS achieve linear-time prover [WTs+18, XZZ+19, Set20, SL20, CBBZ23] [BCHO22, XZS22, GLS+23]. In this paper, we focus on constructing a SNARK where the *prover time is linear* in the size of the circuit, while retaining the (concrete) verification efficiency and argument size of comparable SNARKs.

1.1 Our Contributions

In this paper, we propose Samaritan – a SNARK with updatable setup, linear-time prover, logarithmic proof size, and logarithmic verification. Samaritan improves over state-of-the-art linear-time prover SNARKs in (at least one of) the following two respects: (i) the verifier in Samaritan only performs *constant many* cryptographic operations (and logarithmic field work), and (ii) Samaritan achieves concretely small proof size. Table 1 presents a comparison of Samaritan with state-of-the-art linear-time prover SNARKs (throughout the paper, all asymptotics are stated for growing instance sizes, and have implicit dependence on a fixed security parameter λ). The proof size of Samaritan is competitive with the state-of-the-art SNARKs with linear prover time and comparable verification time [1].

Technically, we propose a new multilinear PCS called SamaritanPCS that is the first (to the best of our knowledge) multilinear PCS with *constant-sized evaluation proofs* and

[1] As per [CBBZ23] the proof size is 4.7KB; however, this number does not account for multilinear oracle queries. We report the corrected number, as confirmed by the authors of [CBBZ23].

Table 1: Comparison of pairing-based SNARKs with linear-time prover. Here, n denotes the number of R1CS constraints (or the number of gates in a circuit), and $(\mathbb{F}, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$ denote the field and groups underlying a bilinear pairing. $\mathbb{P}$ denotes a pairing evaluation. We report concrete proof sizes with respect to the BLS12-381 curve. The concrete proof size for Libra depends on the circuit depth d.

SNARK	Prover Time	Verifier Time	Proof Size	$n = 2^{20}$	Setup
Dory [Lee21]	$O(n)\,\mathbb{G}_1$	$O(\log n)\,\mathbb{G}_T$	$O(\log n)\,\mathbb{G}_T$	24KB	Transparent
Kopis [SL20]		$O(\sqrt{n})\,\mathbb{G}_T$	$O(\log n)\,\mathbb{G}_T$	39KB	
Xiphos [SL20]		$O(\log n)\,\mathbb{G}_T$	$O(\log n)\,\mathbb{G}_T$	61KB	
Spartan [Set20] + KZG [KZG10]		$O(\log^2 n)\,\mathbb{F}$, $O(\log n)\,\mathbb{G}_1$	$O(\log^2 n)\,\mathbb{F}$	50KB	Updatable
Gemini [BCHO22]		$O(\log n)\,\mathbb{G}_1$	$O(\log n)\,\mathbb{G}_1$	18KB	
HyperPlonk [CBBZ23] + PST [PST13]		$O(\log n \log(\log n))\,\mathbb{F}$, $O(\log n)\,\mathbb{P}$	$O(\log n)\,\mathbb{G}_1$	5.5KB	
HyperPlonk + Gemini + KZG		$O(\log n \log(\log n))\,\mathbb{F}$, $O(\log n)\,\mathbb{G}_1$	$O(\log n)\,\mathbb{G}_1$	6.9KB	
Libra [XZZ+19]		$O(d \log n)\,\mathbb{F}$, $O(\log n)\,\mathbb{G}_1$	$O(d \log n)\,\mathbb{G}_1$	–	
MicroSpartan [ZSCZ24]		$O(\log n)\,\mathbb{F}$, $O(\log n)\,\mathbb{G}_1$	$O(\log n)\,\mathbb{F}$, $O(\log n)\,\mathbb{G}_1$	6–7KB	
Samaritan (this work)		$O(\log n)\,\mathbb{F}$, $O(1)\,\mathbb{G}_1$	$O(\log n)\,\mathbb{F}$, $O(1)\,\mathbb{G}_1$	6.2KB	
HyperPlonk + SamaritanPCS (this work)		$O(\log n \log(\log n))\,\mathbb{F}$, $O(\log n)\,\mathbb{G}_1$	$O(\log n)\,\mathbb{G}_1$	5KB	

linear time evaluation. We also show how to leverage recent developments in *lookup arguments* based on logarithmic derivatives of polynomials to construct a new multilinear PIOP with highly compact proof size, which we call the LogSpartan PIOP. Compiling the LogSpartan PIOP with different multilinear PCS yields linear-prover SNARKs with varying proof sizes and verification overheads. In particular, compiling the LogSpartan PIOP using SamaritanPCS yields Samaritan in the updatable setting. We also obtain a new instantiation of HyperPlonk by substituting PST or Gemini+KZG with SamaritanPCS that has the smallest concrete proof size among all linear-time prover SNARKs in Table 1. We elaborate on our contributions below.

New Compiler for Multilinear PCS from Univariate PCS. As our main technical contribution, we present a new generic transformation for realizing a multilinear PCS from *any* univariate PCS. Our transformation improves upon prior generic compilers for obtaining multilinear PCS from univariate PCS such as Virgo [ZXZS20] and Gemini [BCHO22] in terms of the number of auxiliary oracles that need to be constructed and queried. We then discuss two concrete instances of our proposed compiler based on: (i) any *homomorphic* univariate PCS, and (ii) the KZG univariate PCS with updatable setup. Notably, the second instantiation based on the KZG univariate PCS, which we call SamaritanPCS, is the first (to the best of our knowledge), multilinear PCS with *constant-sized evaluation proofs* and linear time evaluation. In fact, SamaritanPCS is a drop-in replacement for the popular PST scheme, and improves upon PST in all relevant parameters (see Table 2 for a comparison). We discuss the instantiations below.

Transformation with Homomorphic Univariate PCS. When instantiated with a *homomorphic* univariate PCS, our transformation yields a multilinear PCS where generating an evaluation proof only incurs an additive overhead of $O(n)$ $\mathbb{F}$-operations over generating an evaluation proof in the underlying univariate PCS (where n is the size of

Table 2: Comparison of Multilinear PCS with linear-time prover. Here, n denotes the size of the multilinear polynomial and $\log n$ is the number of variables in the multilinear polynomial, $(\mathbb{F}, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$ again denote the field and groups underlying a bilinear pairing, and $\mathbb{P}$ denotes a pairing evaluation. SamaritanPCS improves substantially upon all of the other schemes, both in terms of proof size and verifier time.

Multilinear PCS	**Commit + Eval Time**	**Verifier Time**	**Proof Size**	$n = 2^{20}$	**Setup**
Hyrax [WTs+18]	$O(n)\,\mathbb{G}_1$	$O(\sqrt{n})\,\mathbb{G}_1$	$O(\sqrt{n})\,\mathbb{G}_1$	140KB	Transparent
Dory [Lee21]	$O(n)\,\mathbb{G}_1$, $O(\sqrt{n})\,\mathbb{P}$	$O(\log n)\,\mathbb{G}_T$	$O(\log n)\,\mathbb{G}_T$	≈24KB	
Bulletproofs [BBB+18]	$O(n)\mathbb{G}_1$	$O(n)\,\mathbb{G}_1$	$O(\log n)\,\mathbb{G}_1$	1.8KB	
PST [PST13]	$O(n)\,\mathbb{G}_1$	$O(\log n)\,\mathbb{P}$	$O(\log n)\,\mathbb{G}_1$	1KB	Updatable
Gemini+KZG [BCHO22]	$O(n)\,\mathbb{G}_1$	$O(\log n)\,\mathbb{G}_1$	$O(\log n)\,\mathbb{G}_1$	2.3KB	
Virgo+KZG [ZXZS20]	$O(n \log n)\mathbb{F}$, $O(n)\mathbb{G}_1$	$O(\log^2 n)\,\mathbb{F}, O(\log n)\mathbb{G}_1$	$O(\log^2 n)\,\mathbb{F}, O(\log n)\mathbb{G}_1$	$\geq$ 10KB	
Zeromorph [KT24]	$O(n)\,\mathbb{G}_1$	$O(\log n)\,\mathbb{G}_1$	$O(\log n)\,\mathbb{G}_1$	1.1KB	
Mercury [EG25]	$O(n)\mathbb{G}_1$	$O(\log n)\,\mathbb{F}$, $O(1)\,\mathbb{G}_1$	$O(1)\,\mathbb{G}_1$	0.57KB	
SamaritanPCS (this work)	$O(n)\,\mathbb{G}_1$	$O(\log n)\,\mathbb{F}$, $O(1)\,\mathbb{G}_1$	$O(1)\,\mathbb{G}_1$	0.36KB	

the multilinear polynomial as well as the degree-bound for the univariate PCS, and $\log n$ is the number of variables in the multilinear PCS). In contrast, when instantiated with a (homomorphic) univariate PCS, the compiler in Gemini [BCHO22] incurs $3\times$ *multiplicative* blowup over the underlying univariate PCS for evaluation proof generation, while the compiler in Virgo [ZXZS20] incurs a quasilinear $O(n \log n)$ additive overhead in addition to the $3\times$ multiplicative overhead. In terms of evaluation proof size, our transformation incurs $O(\log n)$ additive overhead over the evaluation proof size of the underlying univariate PCS. This matches the (asymptotic) proof size for the compiler in Gemini, but is better than $O(\log^2 n)$ proof size overhead incurred by Virgo. Finally, the compiler in [PH23] incurs $O(n \log n)$ evaluation proof generation as it employs univariate sumcheck over products of n-degree polynomials, while the compiler in [GNS24] incurs $O(n \log^2 n)$ prover.

Instantiation from KZG: SamaritanPCS. We showcase a further optimized instantiation of our generic transformation based on the KZG univariate PCS. We call the resulting multilinear PCS SamaritanPCS. Notably, the evaluation proof in SamaritanPCS incurs only a *constant* overhead over the evaluation proof size of the KZG univariate PCS, thus resulting in constant overall proof size while retaining linear-time evaluation. Verification requires $O(\log n)$ $\mathbb{F}$-operations, $O(1)$ scalar multiplications, and a single pairing operation. SamaritanPCS has updatable setup and is efficient over non FFT-friendly elliptic curve groups. We prove its security in the algebraic group model (AGM). Our contribution is summarized in Theorem 4.2.

Comparison with Existing Pairing-based Multilinear PCS. Table 2 compares SamaritanPCS with several pairing-based multilinear PCS with linear-time prover. SamaritanPCS improves upon all of these schemes in terms of proof size and verifier time. Notably, SamaritanPCS substantially improves over PST, which has the smallest proof size among existing multilinear PCS, and requires $O(\log n)$ pairings for verification. SamaritanPCS also improves

significantly over the multilinear PCS obtained by instantiating the compiler in Gemini with the KZG univariate PCS, which has logarithmic proof size and requires $O(\log n)$ scalar multiplications in $\mathbb{G}_1$ for verification. Finally, Mercury [EG25] is a multilinear PCS introduced in a concurrent work which features similar techniques to ours, and also features a constant proof size with similarly efficient prover and verifier. Concretely, the proofs are about 574 bytes, consisting of $8\,\mathbb{G}_1$ and $6\,\mathbb{F}$ elements.

Comparison with Multilinear PCS with Transparent Setup. Several recent works have proposed (plausibly post-quantum) multilinear PCS schemes with transparent setup and linear prover from hash-based assumptions (such as Brakedown [GLS+23] and Blaze [BCF+25]), and lattice-based assumptions (e.g., Greyhound [NS24] and [CMNW24]). Brakedown and Blaze incur $O(\log^2 n)$-sized opening proofs and $O(\log^2 n)$ verification complexity, while the lattice-based multilinear PCS schemes incur $O(\log^2 n)$-sized opening proofs and $O(\sqrt{n})$ verification complexity. In comparison, SamaritanPCS achieves $O(1)$ opening proof size and $O(\log n)$ verification complexity (where the cryptographic work for verification is $O(1)$), albeit in the updatable SRS model.

Batching Properties of SamaritanPCS. We highlight that SamaritanPCS has excellent batching properties wherein a prover can batch the generation of k evaluation proofs while performing only $O(n + k\sqrt{n})$ cryptographic work. The batched proof consists of $O(k)$ $\mathbb{G}$ and $\mathbb{F}$ elements, while batch verification requires $O(k \log n)$ $\mathbb{F}$-operations, $O(k)$ $\mathbb{G}$ operations, and two pairing operations. We refer to Section 4.4, Lemma 4.2 for more details.

To the best of our knowledge, existing multilinear PCS schemes in the updatable SRS setting such as PST [PST13], Virgo compiled with KZG [ZXZS20], and Gemini complied with KZG [BCHO22] do not natively support efficient batching across evaluations on distinct points (cryptographic work would grow linearly in the number of evaluations). Mercury [EG25] (concurrent work) also does not seem to support efficient batching (and there is no discussion on batching in their paper). Finally, the HyperPlonk paper ([CBBZ23], Section 3.8) points out a generic technique for batching multilinear PCS evaluation proofs (possibly over distinct points), which requires only one query to the underlying multilinear PCS (assuming homomorphism), but requires extra $O(\log n)$ communication due to sumcheck. For small constant batch evaluations, this would not preserve $O(1)$ proof size of SamaritanPCS.

LogSpartan PIOP. The starting point of the LogSpartan PIOP is the multilinear PIOP underlying Spartan [Set20] – a popular zkSNARK with an efficient prover and a succinct verifier. All known variants of Spartan have large concrete proof size due to the presence of $O(\log^2 n)$ $\mathbb{F}$ elements ($\mathbb{F}$ being the base finite field). In this work, we introduce LogSpartan PIOP that uses a lookup argument based on logarithmic derivatives of polynomials [Hab22a] and reduces the proof size of the Spartan PIOP from $O(\log^2 n)$ to

$O(\log n)$. Technically, we show how to replace memory-checking techniques used to realize a sparse polynomial commitment (to R1CS matrices) in the Spartan PIOP with a lookup argument-based protocol. We also point out that LogSpartan does not degrade the zero-knowledge properties of the original Spartan PIOP since our modifications to the Spartan PIOP only involve proving evaluations of public R1CS matrices. The use of logarithmic derivative-based lookups incurs increased cost for the prover. We present two variants with different tradeoffs between proof size and prover overhead. The variant with smaller proof size (6.2KB) incurs around 2.3× prover overhead over Spartan (Section 5.3). To mitigate this overhead, we describe a variant with 1.3× prover overhead and proof size of 8.3KB (Section 5).

We highlight certain existing attempts to reduce the argument size of Spartan in the discussion below. Quarks [SL20] attempt to reduce the proof size of Spartan by using *inner pairing product* (IPP)-based commitments. While asymptotically the enhancements ensure $O(\log n)$ proof-size, concretely the proof-sizes are still > 35 KB. Testudo [CGG$^+$23] also uses IPP based commitments, but overcomes the blow-up in proof size by recursively composing sum-check verification and IPP verification with a Groth16 [Gro16] SNARK, which results in constant proof size. However, such a recursive composition requires non-native operations (such as target group scalar multiplications) to be encoded inside arithmetic circuits which makes it practically challenging, limits the choice of elliptic curves to instantiate the scheme, and prevents the possibility of proving security in the ROM. LogSpartan imposes no such restrictions, and allows proving security in the ROM. Other recent efforts [GNS24] have used "dual commitments" to move certain parts of the verification such as the grand-product check to those over univariate polynomials. However, this comes at a considerable overhead for the prover as compared to LogSpartan. We finally note that certain very recent and concurrent works have also introduced variants of the Spartan PIOP using lookup arguments based on logarithmic derivatives of polynomials. These include MicroSpartan [ZSCZ24], Shout [ST25], and DFS [HMW$^+$25]. While these PIOPs share broad similarities with LogSpartan, we introduce several optimizations to reduce the prover overhead that are unique to LogSpartan.

2 Overview of Results and Techniques

In this section, we present an overview of our main results and techniques.

2.1 Multilinear PCS from Homomorphic Univariate PCS

In this subsection, we present an overview of our framework for obtaining a multilinear PCS from a homomorphic univariate PCS.

The Core Protocol. Let $n = 2^\mu$ for some $\mu \in \mathbb{N}$ and let B_μ denote the hypercube $\{0,1\}^\mu$. Let $\widetilde{eq}_i$, $i \in [n]$ denote the basis of $\mathbb{F}^{\leq 1}[X_1, \ldots, X_\mu]$ where $\widetilde{eq}_i(\mathbf{x}) = \widetilde{eq}_\mu(\langle\, i \,\rangle, \mathbf{x})$. We

use the following isomorphism between multilinear polynomials and univariate polynomials $\mathbb{F}[X]$ of degree at most n. More precisely, we define the isomorphism of the $\mathbb{F}$-vector spaces as:

$$\varphi : \mathbb{F}^{\leq 1}[X_1, \ldots, X_\mu] \longrightarrow \mathbb{F}^{<n}[X]$$
$$\sum_{i=1}^{n} f_i \widetilde{eq}_i(X_1, \ldots, X_\mu) \mapsto \sum_{i=1}^{n} f_i X^{i-1}$$

We use the above isomorphism to commit to multilinear polynomials using a univariate PCS. For $\widetilde{f} \in \mathbb{F}^{\leq 1}[X_1, \ldots, X_\mu]$, we use $\hat{f}$ to denote the univariate polynomial $\varphi(\widetilde{f})$. Let uPC be a univariate polynomial commitment scheme. We define the multilinear polynomial commitment scheme mPC using uPC as a black-box. For a μ-variate multilinear polynomial $\widetilde{f}$, we define $\mathsf{mPC.Com}(\widetilde{f}) \to (\mathsf{cm}_f, \tilde{\omega})$, where $(\mathsf{cm}_f, \tilde{\omega}) \leftarrow \mathsf{uPC.Com}(\mathsf{pp}, \hat{f})$.

The Evaluation Protocol. We now present an overview of the evaluation protocol for a multilinear polynomial committed as outlined above. Let $\mathbf{z} = (z_1, \ldots, z_\mu)$ be the evaluation point. The evaluation claim $\widetilde{f}(\mathbf{z}) = v$ is equivalent to proving inner product $\langle \mathbf{f}, \phi_z \rangle = v$ where $\mathbf{f}$ is the coefficient vector of $\widetilde{f}$ and $\phi_z = (\widetilde{eq}_1(\mathbf{z}), \ldots, \widetilde{eq}_n(\mathbf{z}))$. The vector ϕ_z can be described as the tensor product:

$$\phi_z = \begin{pmatrix} 1 - z_\mu \\ z_\mu \end{pmatrix} \otimes \cdots \otimes \begin{pmatrix} 1 - z_1 \\ z_1 \end{pmatrix}$$

Let $\hat{\Psi}(X; \mathbf{z})$ be the univariate polynomial with ϕ_z in reverse order as the coefficient vector. It can be seen that

$$\hat{\Psi}(X; \mathbf{z}) = (z_1 + (1 - z_1)X) \cdot (z_2 + (1 - z_2)X^2) \cdots (z_\mu + (1 - z_\mu)X^{2^{\mu-1}}).$$

The claimed dot product v is then the coefficient of X^{n-1} in the product $\hat{f}(X) \cdot \hat{\Psi}(X; \mathbf{z})$, which can be shown by the prover by exhibiting univariate oracles $\hat{h}(X)$ and $\hat{g}(X)$ of degree at most $(n - 2)$ such that:

$$\hat{f}(X) \cdot \hat{\Psi}(X; \mathbf{z}) = X^n \hat{h}(X) + vX^{n-1} + g(X)$$

The above identity can be checked at a random point by querying the univariate oracles $\hat{f}, \hat{g}$ and $\hat{h}$, whereas the verifier can evaluate $\hat{\Psi}(X; \mathbf{z})$ itself in $O(\log n)$ $\mathbb{F}$-operations. We note that the prover can compute the product $\hat{f}(X) \cdot \hat{\Psi}(X; \mathbf{z})$ using $O(n \log n)$ field multiplications *without* using FFT by successive multiplication with factors of $\hat{\Psi}(X; \mathbf{z})$ and subsequently compute $\hat{h}$ and $\hat{g}$.

Obtaining Linear Time Prover. We use divide and conquer to reduce prover complexity from $O(n \log n)$ in the above protocol to $O(n)$. Let $n = \ell m$, $\nu = \log \ell$ and $\kappa = \log m$,

and thus $\mu = \nu + \kappa$. We observe that the evaluation of multilinear polynomial over hypercube of size n can be expressed in terms of evaluations of ℓ multilinear polynomials defined over hypercube of size m. Specifically, we have:

$$\widetilde{f}(\mathbf{x}, \mathbf{y}) = \sum_{\tau \in B_\nu} \widetilde{eq}_\nu(\tau, \mathbf{y}) \widetilde{f}(\mathbf{x}, \tau) = \sum_{i=1}^{\ell} \widetilde{eq}(\langle\, i \,\rangle, \mathbf{y}) \widetilde{f}(\mathbf{x}, \langle\, i \,\rangle)$$

Next, we define $\widetilde{g}_i(\mathbf{x}) = \widetilde{f}(\mathbf{x}, \langle\, i \,\rangle)$ as κ-variate multilinear polynomials for $i \in [\ell]$ and write $\hat{f}(X)$ in base X^m as:

$$\hat{f}(X) = \hat{g}_1(X) + X^m \hat{g}_2(X) + \cdots + X^{m(\ell-1)} \hat{g}_\ell(X)$$

where the polynomials $\hat{g}_i(X)$ are uniquely determined polynomials of degree $< m$. We observe that if we write the coefficient vector $\mathbf{f} = (\mathbf{g}_1, \ldots, \mathbf{g}_\ell)$, with $\mathbf{g}_i \in \mathbb{F}^m$, then $\mathbf{g}_i$ is the common coefficient vector of the multilinear polynomial $\widetilde{g}_i(\mathbf{x})$ and $\hat{g}_i(X) \in \mathbb{F}^{<m}[\, X \,]$.

Evaluation Proof. We now present a high-level overview of how to prove an evaluation of a multilinear polynomial committed using the above technique. To prove the claim $\widetilde{f}(\mathbf{z}) = v$, the prover proceeds as follows:

- It first sends to the verifier commitments $\mathsf{cm}_1, \ldots, \mathsf{cm}_\ell$ to the univariate polynomials $\hat{g}_1(X), \ldots, \hat{g}_\ell(X)$ respectively, which are also commitments to the corresponding multilinear polynomials $\widetilde{g}_i(\mathbf{x})$, $i \in [\ell]$.
- It also sends the multilinear polynomial evaluations $v_1 = \widetilde{g}_1(\mathbf{z}_x), \ldots, v_\ell = \widetilde{g}_\ell(\mathbf{z}_x)$, where $\mathbf{z} = (\mathbf{z}_y, \mathbf{z}_x)$ with $\mathbf{z}_y \in \mathbb{F}^\mu$, $\mathbf{z}_x \in \mathbb{F}^\kappa$.

The verifier in turn checks that *each* of following holds:

- The committed polynomials $\hat{g}_i(X), i \in [\ell]$ represent the correct decomposition of $\hat{f}(X)$.
- The multilinear evaluations $v_1, \ldots, v_\ell$ are correct.
- The evaluation v follows correctly from the evaluations of $\widetilde{g}_i$, $i \in [\ell]$.

The above blueprint yields a black-box realization of a multilinear PCS from any homomorphic univariate PCS, with $O(\log n)$ communication overhead. The details and the corresponding efficiency analyses appear in Section 4.2, with complete protocol in Figure 1. The contribution is summarized in Theorem 4.1.

SamaritanPCS: Instantiation based on KZG. In the preceding discussion we required proving ℓ evaluation claims $\hat{g}_i(X) = v_i$, $i \in [\ell]$. To employ straightforward batching we required committing to each claim $(\hat{g}_i(X), v_i)$ (where the polynomial is committed, while the evaluation is sent in the clear). Then, for a uniform $\gamma \leftarrow \mathbb{F}$, we could check the aggregate claim: $\big(\sum_{i=1}^{\ell} \gamma^{i-1}\hat{g}_i(X), \sum_{i=1}^{\ell} \gamma^{i-1} v_i\big)$. To succinctly commit to vector $(v_1, \ldots, v_\ell)$, we can simply commit to the polynomial $\hat{v}(X)$ with the corresponding coefficients. Next, we observe that commitment to $\hat{f}(X)$ also implicitly commits to polynomials $\hat{g}_1(X), \ldots, \hat{g}_\ell(X)$ given the uniqueness of the decomposition. The key challenge is to relate the polynomial $\hat{P}(X) = \sum_{i=1}^{\ell} \gamma^{i-1}\hat{g}_i(X)$ to $\hat{f}(X)$. It turns out that if we express the decomposition of $\hat{f}$ in terms of a bivariate polynomial Q by treating X^m and X as separate variables X and Y, we have $\hat{f}(X) = Q(X^m, X)$ and $\hat{P}(X) = Q(\gamma, X)$ where $Q(X, Y) = \sum_{i=1}^{\ell} X^{i-1}\hat{g}_i(Y)$. Moreover, such a polynomial Q is unique (Lemma 3.1). Thus, in principle, any bivariate PCS helps to relate $\hat{f}$ and $\hat{G}$. Our construction in Section 4.3 employs a bivariate PCS based on KZG, which has a constant proof size. Our eventual protocol SamaritanPCS uses several additional optimizations to reduce concrete proof size by exploiting the specific properties of the KZG and the bivariate PCS. Along the way, in Lemma 4.1, we also generalize the existing description of the bivariate PCS in [ZBK+22], by showing that a univariate PCS can be generically used to construct a bivariate PCS (Lemma 4.1). The properties of SamaritanPCS are summarised in Theorem 4.2. We also highlight the efficient batching properties of the PCS in Lemma 4.2.

2.2 LogSpartan: PIOP with Log-Up based Lookups

In this subsection, we provide an overview of our new Spartan PIOP based on Log-Up based lookups, which we call the LogSpartan PIOP. We begin by providing a brief recap of the Spartan PIOP for ease of exposition. Readers familiar with the details of the Spartan PIOP can skip this.

Revisiting the Spartan PIOP. For integers $m, n, k \in \mathbb{N}$ and matrices $A, B, C \in \mathbb{F}^{m \times n}$, Spartan presents an argument of knowledge for the following relation:

$$\mathcal{R}^{\mathsf{R1CS}}_{m,n,k} = \{(\mathbf{x}, \mathbf{w}) \in \mathbb{F}^k \times \mathbb{F}^{n-k-1} : A\mathbf{z} \circ B\mathbf{z} = C\mathbf{z} \text{ where } \mathbf{z} = (1, \mathbf{x}, \mathbf{w})\}$$

In Spartan, the authors make the simplification that $|\mathbf{x}| + 1 = n/2 = |\mathbf{w}|$, which allows them to conveniently express the multilinear extension (MLE) of the vector $\mathbf{z}$ as $\widetilde{z} = (1 - X_\mu)\widetilde{(1, \mathbf{x})} + X_\mu \widetilde{w}$. We will also make the same simplification and further consider R1CS instances with $m = n$, using n to denote both. Neither impacts generality, as the matrices or the witness can be zero-padded to match the dimensions. As seen later, we commit to sparse representation of matrices, so zero-padding does not impact concrete efficiency of the protocol. We will henceforth ignore public input, and simply use $\mathbf{z}$ to denote the entire witness. With these considerations, we define the R1CS relation as:

$$\mathcal{R}^{\mathsf{R1CS}}_{n} = \{\big((A, B, C), \mathbf{z}\big) \in (\mathbb{F}^{n \times n})^3 \times \mathbb{F}^n : A\mathbf{z} \circ B\mathbf{z} = C\mathbf{z}\} \tag{1}$$

Let $\mu = \log n$. We view matrices A, B and C as functions $B_{2\mu} \to \mathbb{F}$ (here $B_{2\mu}$ denotes the hypercube $\{0,1\}^{2\mu}$), and view the witness $\mathbf{z}$ as function $B_\mu \to \mathbb{F}$. Let $\widetilde{A}$, $\widetilde{B}$, $\widetilde{C}$ and $\widetilde{z}$ be MLEs of matrices A, B, C and vector $\mathbf{z}$ respectively. Then, the relation in Equation (1) implies the following multilinear PIOP:

$$\left(\sum_{\mathbf{y}\in B_\mu} \widetilde{A}(\mathbf{x},\mathbf{y})\widetilde{z}(\mathbf{y})\right) \cdot \left(\sum_{\mathbf{y}\in B_\mu} \widetilde{B}(\mathbf{x},\mathbf{y})\widetilde{z}(\mathbf{y})\right) - \left(\sum_{\mathbf{y}\in B_\mu} \widetilde{C}(\mathbf{x},\mathbf{y})\widetilde{z}(\mathbf{y})\right) = \mathbf{0}\, \forall \mathbf{x} \in B_\mu$$

With overwhelming probability over the choice of $\tau \leftarrow \mathbb{F}^\mu$, the above is equivalent to the following sum-check:

$$\sum_{\mathbf{x}\in B_\mu} \widetilde{eq}(\mathbf{x},\tau)\Bigg[\Big(\sum_{\mathbf{y}\in B_\mu} \widetilde{A}(\mathbf{x},\mathbf{y})\widetilde{z}(\mathbf{y})\Big) \cdot \Big(\sum_{\mathbf{y}\in B_\mu} \widetilde{B}(\mathbf{x},\mathbf{y})\widetilde{z}(\mathbf{y})\Big) - \Big(\sum_{\mathbf{y}\in B_\mu} \widetilde{C}(\mathbf{x},\mathbf{y})\widetilde{z}(\mathbf{y})\Big)\Bigg] = 0 \tag{2}$$

Naively, the matrices A, B and C are defined over the hypercube of size n^2, and thus the prover complexity in the resulting sum-check instance would be $\Omega(n^2)$. To avoid this quadratic overhead, Spartan [Set20] proposes *sparse commitment scheme* Spark for the R1CS matrices, which allows one to execute the above sum-check in cost $O(K)$ where K is the upper bound on the number of non-zero entries in A, B and C. Typically we assume $K = O(n)$. For matrix $M \in \{A, B, C\}$, we write $M = \big(M(x_k, y_k), x_k, y_k\big)_{k\in[K]}$ where $(x_k, y_k), k \in [K]$ is a canonical ordering of the non-zero positions of M. Then, for $\kappa = \log K$, we compute multilinear extensions of the functions $\mathsf{val}_M, \mathsf{R}_M$ and C_M from B_κ to $\mathbb{F}$ defined by $\mathsf{val}_M(\langle k \rangle_\kappa) = M(x_k, y_k)$, $\mathsf{R}_M(\langle k \rangle_\kappa) = x_k$ and $\mathsf{C}_M(\langle k \rangle_\kappa) = y_k$. We view M as the triple $(\widetilde{\mathsf{val}}_M, \widetilde{\mathsf{R}}_M, \widetilde{\mathsf{C}}_M)$. We can write the MLE $\widetilde{M}$ of the matrix $M \in \{A, B, C\}$ as:

$$\widetilde{M}(\mathbf{x},\mathbf{y}) = \sum_{\mathbf{k}\in B_\kappa} \widetilde{\mathsf{val}}_M(\mathbf{k}) \cdot \widetilde{eq}_\mu(\mathbf{x}, \langle\, \widetilde{\mathsf{R}}_M(\mathbf{k})\, \rangle_\mu) \cdot \widetilde{eq}_\mu(\mathbf{y}, \langle\, \widetilde{\mathsf{C}}_M(\mathbf{k})\, \rangle_\mu)$$

With the preceding representation of R1CS matrices, we can execute the sum-check in Equation (2). Using challenges $\mathbf{r}_x \in \mathbb{F}^\mu$ in a sum-check protocol, the verifier can reduce the claim in Equation 2 to the following claims for some v_A, v_B, v_C and $v \in \mathbb{F}$.

$$\sum_{\mathbf{y}\in B_\mu} \widetilde{M}(\mathbf{r}_x,\mathbf{y})\widetilde{z}(\mathbf{y}) = v_M \text{ for } M \in \{A, B, C\}$$
$$\widetilde{eq}(\mathbf{r}_x,\tau)(v_A \cdot v_B - v_C) = v$$

Using another instance of sum-check with verifier challenges $\mathbf{r}_y \in B_\mu$, the claim $\sum_{\mathbf{y}\in B_\mu} \widetilde{M}(\mathbf{r}_x,\mathbf{y})\widetilde{z}(\mathbf{y}) = v_M$ reduces to the claims

$$\widetilde{M}(\mathbf{r}_x,\mathbf{r}_y)\widetilde{z}(\mathbf{r}_y) = \bar{v}_M \text{ for } M \in \{A, B, C\}$$

Now, we can write $\widetilde{M}(\mathbf{r}_x, \mathbf{r}_y)$ in terms of the MLEs $\widetilde{\mathsf{val}}_M, \widetilde{\mathsf{R}}_M$ and $\widetilde{\mathsf{C}}_M$ as

$$\widetilde{M}(\mathbf{r}_x, \mathbf{r}_y) = \sum_{\mathbf{k} \in B_\kappa} \widetilde{\mathsf{val}}_M(\mathbf{k}) \cdot \widetilde{eq}_\mu(\mathbf{r}_x, \langle\, \widetilde{\mathsf{R}}_M(\mathbf{k}) \,\rangle_\mu) \cdot \widetilde{eq}_\mu(\mathbf{r}_y, \langle\, \widetilde{\mathsf{C}}_M(\mathbf{k}) \,\rangle_\mu) \tag{3}$$

To prove the correctness of the above computation of $\widetilde{M}(\mathbf{r}_x, \mathbf{r}_y)$ using Equation (3), we use logarithmic derivative-based lookups [EFG22, Hab22b] as a more "algebraic" alternative to the "combinatorial" memory-checking techniques originally used in [Set20], which allows us to obtain a multilinear PIOP with smaller proof size. Logarithmic derivative based-lookup arguments have also been used in recent work [ZSCZ24] to obtain variant of Spartan, called MicroSpartan with logarithmic proof size. A similar PIOP for lookups has been used for sparse polynomial evaluation in DFS [HMW+25] (concurrent work).

The usage of log-derivative approach for lookups incurs increased proving costs due to the need to commit to auxiliary oracles. We present lookups in the context of *oracle composition*. Our goal is to formulate PIOPs to minimize this overhead, while keeping proof sizes small. For reference, commitments to oracles in the original Spartan PIOP [Set20] requires multi-exponentiation involving $6n$ large exponents and n small exponents. In the LogSpartan PIOP presented in Section 5 we obtain commitment cost equivalent to $8n$ large exponents and $7n$ small exponents, with proof size $\approx 11 \log n$. In Appendix 5.3, we present an alternative version of the LogSpartan PIOP with smaller proof size $\approx 8 \log n$, but requiring commitment to $14n$ large exponents and n small exponents. Thus our variants incur $1.3\times$ - $2.3\times$ prover overhead over Spartan.

Our Technique: Oracle Composition Using Log-Derivatives. The key challenge in proving the correct evaluation of MLE of sparse matrix M using Equation (3) is to prove correctness of the oracle $\widetilde{f}_M : B_\kappa \to \mathbb{F}$, defined by $\mathbf{y} \mapsto \widetilde{eq}(\mathbf{r}_x \langle\, \widetilde{\mathsf{R}}_M(\mathbf{y}) \,\rangle)$. Our core technical observation is that the oracle $\widetilde{f}$ can be treated as a composition of the oracles $\widetilde{eq}(\mathbf{r}_x, \cdot) : B_\mu \to \mathbb{F}$ and $\widetilde{\mathsf{R}}_M(\cdot) : B_\kappa \to \mathbb{F}$, where the function $\langle \cdot \rangle$ maps the co-domain of $\widetilde{\mathsf{R}}_M$ to the domain of $\widetilde{eq}(\mathbf{r}_x, \cdot)$. We first present a PIOP for the oracle composition relation in isolation. We then apply this PIOP for oracle composition to obtain a more efficient variant of the Spartan PIOP. We expand further on our techniques below.

Oracle Composition. For integers μ, κ, we define the oracle relation $\mathcal{R}^{\mathsf{comp}}_{\mu,\kappa}$ as the set of pairs $(\mathbf{x}, \mathbf{w})$ where $\mathbf{x} = ([\![\widetilde{f}]\!], [\![\widetilde{g}]\!], [\![\widetilde{h}]\!])$ is the statement, and $\mathbf{w} = (\widetilde{f}, \widetilde{g}, \widetilde{h})$ is the witness, such that $\widetilde{g}$, is a μ-variate multilinear polynomial and $\widetilde{f}, \widetilde{h}$ are κ-variate multilinear polynomials satisfying $\widetilde{f}(\mathbf{y}) = \widetilde{g}(\langle\, \widetilde{h}(\mathbf{y}) \,\rangle_\mu)$. Here $[\![\cdot]\!]$ denotes the oracle access to the polynomial. We define oracle composition formally in Definition 5.1 (Section 5.1).

Viewing Oracle Composition as Indexed-Lookup. It turns out that the oracle composition above can be viewed as an *indexed-lookup relation* over the evaluation vectors of the multilinear polynomials over their respective domains. Concretely, for integers m, n, we say that a triple of vectors $(\mathbf{t}, \mathbf{v}, \mathbf{a})$, where $\mathbf{t} = (t_1, \ldots, t_n) \in \mathbb{F}^n$, $\mathbf{v} = (v_1, \ldots, v_m) \in \mathbb{F}^m$ and $\mathbf{a} = (a_1, \ldots, a_m) \in \mathbb{F}^m$, satisfy the *indexed lookup* relation if $v_i = t_{a_i}$ for all $i \in [m]$ (here,

$\mathbf{t}$ is the table/vector being looked up from, $\mathbf{v}$ is the vector consisting of looked up entries from $\mathbf{t}$, and $\mathbf{a}$ is the vector of lookup indices). See Definition 5.2 for a formal exposition.

We now recall a result on logarithmic derivatives of polynomials introduced in [Hab22a] for proving indexed lookup relation. The result, which appears later in Lemma 5.1, states that for integers m, n and a field $\mathbb{F}$ of characteristic $p > n$, a triple of vectors $(\mathbf{t}, \mathbf{v}, \mathbf{a}) \in \mathbb{F}^n \times \mathbb{F}^m \times \mathbb{F}^m$ satisfy the indexed lookup relation if and only if there exists vector $\mathbf{m} = (m_1, \ldots, m_n) \in \mathbb{F}^n$ such that

$$\sum_{i=1}^{n} \frac{m_i}{X + iY + t_i} = \sum_{i=1}^{m} \frac{1}{X + a_i Y + v_i}$$

Now, identifying the vectors in the above result [Hab22a] with the evaluations of multilinear polynomials, we observe that the relation $\widetilde{f}(\mathbf{y}) = \widetilde{g}(\langle \widetilde{h}(\mathbf{y}) \rangle)$ is implied by the existence of multilinear polynomial $\widetilde{\chi}$ (which interpolates the vector $(m_{\mathbf{y}})_{\mathbf{y} \in B_\mu}$ in the above result from [Hab22a]) satisfying

$$\sum_{\mathbf{y} \in B_\mu} \frac{\widetilde{\chi}(\mathbf{y})}{X + \widetilde{\mathsf{id}}_\mu(\mathbf{y}) Y + \widetilde{g}(\mathbf{y})} = \sum_{\mathbf{y} \in B_\kappa} \frac{1}{X + \widetilde{h}(\mathbf{y}) Y + \widetilde{f}(\mathbf{y})} \tag{4}$$

Based on above observations, we show a PIOP for the oracle composition relation $\mathcal{R}^{\mathsf{comp}}_{\mu,\kappa}$ in Section 5.1 and then use it to obtain a PIOP for the R1CS relation (Equation 1). The resulting PIOP is called LogSpartan. It is compiled using the multilinear PCS SamaritanPCS to obtain the SNARK Samaritan.

3 Preliminaries

We present preliminary background material in this section.

Notation. We denote the set of integers $\{1, \ldots, n\}$ by $[n]$ for $n \in \mathbb{N}$, and $\mathbb{F}$ to denote a prime field of order p. We denote by λ a security parameter. We use negl to denote a negligible function: for any integer $c > 0$, there exists $n \in \mathbb{N}$, such that $\forall\ x > n$, $\mathsf{negl}(x) \leq 1/x^c$. We assume a bilinear group generator BG which on input λ outputs parameters for the protocols. Specifically, $\mathsf{BG}(1^\lambda)$ outputs $(\mathbb{F}, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2, g_t)$ where: $\mathbb{F} = \mathbb{F}_p$ is a prime field of super-polynomial size in λ, with $p = \lambda^{\omega(1)}$; $\mathbb{G}_1, \mathbb{G}_2$ and $\mathbb{G}_T$ are groups of order p, and e is an efficiently computable non-degenerate bilinear pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$; Generators g_1, g_2 are uniformly chosen from $\mathbb{G}_1$ and $\mathbb{G}_2$ respectively and $g_t = e(g_1, g_2)$. We write groups $\mathbb{G}_1$ and $\mathbb{G}_2$ additively, and use the shorthand notation $[x]_1$ and $[x]_2$ to denote group elements $x \cdot g_1$ and $x \cdot g_2$ respectively for $x \in \mathbb{F}$. We implicitly assume that all the setup algorithms for the protocols invoke BG to generate descriptions of groups and fields over which the protocol is instantiated. We will use sets $\mathbb{F}, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ to specify the type of operations, where additionally, we have $\mathbb{P}$ to denote pairings and $\mathbb{M}$ to denote multiexponentiation.

Sets. For $\mu \in \mathbb{N}$, we use B_μ to denote the set $\{0,1\}^\mu$. For $\mathbf{y} = (y_1, \ldots, y_\mu) \in B_\mu$, we use the notation $\mathsf{id}_\mu(\mathbf{y})$ to denote the integer $1 + \sum_{i=1}^{\mu} y_i 2^{i-1}$, and similarly for an integer i, we use $\langle\, i \,\rangle_\mu$ to denote the μ-bit binary decomposition of $i-1$. We note that the surjective maps $\mathsf{id}_\mu : B_\mu \to [2^\mu]$ and $\mathsf{bin}_\mu : [\,2^\mu\,] \to B_\mu$ are inverses of each other. We drop the subscript μ when it is clear from the context.

3.1 Succinct Argument of Knowledge

Let $\mathcal{R}$ be a NP-relation and $\mathcal{L}$ be the corresponding NP-language, where $\mathcal{L} = \{x : \exists\ w$ such that $(x,w) \in \mathcal{R}\}$. Here, a prover $\mathcal{P}$ aims to convince a verifier $\mathcal{V}$ that $x \in \mathcal{L}$ by proving that it knows a witness w for a public statement x such that $(x,w) \in \mathcal{R}$. An interactive argument of knowledge for a relation $\mathcal{R}$ consists of a PPT algorithm Setup that takes as input the security parameter λ, and outputs the public parameters pp, and a pair of interactive PPT algorithms $\langle \mathcal{P}, \mathcal{V} \rangle$, where $\mathcal{P}$ takes as input (pp, x, w) and $\mathcal{V}$ takes as input (pp, x). An interactive argument of knowledge $\langle \mathcal{P}, \mathcal{V} \rangle$ must satisfy completeness and knowledge soundness.

Definition 3.1 (Completeness)**.** *For all security parameter $\lambda \in \mathbb{N}$ and statement x and witness w such that $(x,w) \in \mathcal{R}$, we have*

$$\Pr\left(b = 1 \ : \ \begin{array}{c} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda) \\ b \leftarrow \langle \mathcal{P}(w), \mathcal{V} \rangle (\mathsf{pp}, x) \end{array}\right) = 1.$$

Definition 3.2 (Knowledge Soundness)**.** *For any PPT malicious prover $\mathcal{P}^* = (\mathcal{P}_1^*, \mathcal{P}_2^*)$, there exists a PPT algorithm $\mathcal{E}$ such that the following probability is negligible:*

$$\Pr\left(\begin{array}{c} b = 1 \wedge \\ (x,w) \notin \mathcal{R} \end{array} \ : \ \begin{array}{c} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda) \\ (x, \mathsf{st}) \leftarrow \mathcal{P}_1^*(1^\lambda, \mathsf{pp}) \\ b \leftarrow \langle \mathcal{P}_2^*(\mathsf{st}), \mathcal{V} \rangle (\mathsf{pp}, x) \\ w \leftarrow \mathcal{E}^{\mathcal{P}_2^*}(\mathsf{pp}, x) \end{array}\right).$$

A *succinct* argument of knowledge $\langle \mathcal{P}, \mathcal{V} \rangle$ for a relation $\mathcal{R}$, must satisfy completeness and knowledge soundness and additionally be *succinct*, that is, the communication complexity between prover and verifier, as well as the verification complexity is bounded by $\mathsf{poly}(\lambda, \log |w|)$.

3.2 Polynomials and Multilinear Extensions

We use $\mathbb{F}^{\leq 1}[X_1, \ldots, X_\mu]$ to denote the set of μ-variate multilinear polynomials over the field $\mathbb{F}$. We define the following 2μ-variate polynomial

$$\widetilde{eq}_\mu(\mathbf{x}, \mathbf{y}) = \prod_{i=1}^{\mu} (x_i y_i + (1 - x_i)(1 - y_i))$$

for $x \in \mathbb{F}^\mu$ and $y \in \mathbb{F}^\mu$. The polynomials $\{\widetilde{eq}_\mu(x, \langle i-1 \rangle) : i \in [2^\mu]\}$ are linearly independent over $\mathbb{F}$ form the Lagrange basis polynomials for the set B_μ. For $x, y \in B_\mu$, $\widetilde{eq}_\mu(x, y) = 1$ if $x = y$, and is 0 otherwise. For a function $f : B_\mu \to \mathbb{F}$, the (unique) polynomial $\widetilde{f}(x) = \sum_{i=1}^{2^\mu} f(\langle i-1 \rangle)\widetilde{eq}(x, \langle i-1 \rangle)$ is called the *multilinear extension* (MLE) of the function f. We also naturally view vectors $\mathbf{f} \in \mathbb{F}^{2^\mu}$ as functions $f : B_\mu \to \mathbb{F}$, and define MLE of the vector as that of the implied function. For clarity of notation, we will denote multilinear polynomials as $\widetilde{f}$ (with a tilde), its associated coefficient vector of evaluations at B_μ as $\mathbf{f}$, and the univariate polynomial with $\mathbf{f}$ as the coefficient vector (in power basis $1, X, \ldots, X^{n-1}$ for $n = 2^\mu$) as $\hat{f}(X)$. Thus, the univariate and multilinear polynomials sharing the coefficient vector $\mathbf{f}$ in the respective bases are denoted as $\hat{f}$ and $\widetilde{f}$ respectively.

μ-variate Sumcheck. Let $f(X_1, \ldots, X_\mu) \in \mathbb{F}[X_1, \ldots, X_\mu]$. Consider,

$$\sum_{x_1 \in \{0,1\}} \sum_{x_2 \in \{0,1\}} \cdots \sum_{x_\mu \in \{0,1\}} f(x_1, \ldots, x_\mu) = y$$

This sum over the hypercube B_μ, $\sum_{\mathbf{x} \in B_\mu} f(\mathbf{x}) = y$ takes time $O(|B_\mu|)$ to compute. The sumcheck protocol [LFKN90, CBBZ23] allows the verifier to outsource this computation to a prover, where the prover sends number of field elements that is logarithmic in the size of the hypercube, and the verifier needs to evaluate f at a single point.

Lemma 3.1 ([BSS08])**.** *Given any pair of polynomials $G(X)$, $q(X)$ there exists a unique bivariate polynomial $Q(X, Y)$ with $\deg_X(Q) < \lfloor \deg(G)/\deg(q) \rfloor$ and $\deg_Y(Q) < \deg(q)$ such that $G(X) = Q(q(X), X)$.*

3.3 Algebraic Preliminaries

Algebraic Group Model. We analyze the security of our protocols in the Algebraic Group Model (AGM) introduced in [FKL18]. An adversary $\mathcal{A}$ is called *algebraic* if every group element output by $\mathcal{A}$ is accompanied by a representation of that group element in terms of all the group elements that $\mathcal{A}$ has seen so far (input and output). In the AGM, an adversary $\mathcal{A}$ is restricted to be *algebraic*, which in our SRS-based protocol means a PPT algorithm satisfying the following: Given $\mathsf{srs} = (\mathsf{srs}_1, \mathsf{srs}_2)$, whenever $\mathcal{A}$ outputs an element $A \in \mathbb{G}_i, i \in 1, 2$, it is accompanied by its representation, i.e, $\mathcal{A}$ also outputs a vector $\mathbf{v}$ over $\mathbb{F}$ such that $A = \langle \mathbf{v}, \mathsf{srs}_i \rangle$.

The q-DLOG Assumption. We define the q-DLOG assumption below.

Definition 3.3 (q-DLOG Assumption)**.** *The q-DLOG assumption with respect to $\mathcal{G}$ holds if for all λ and for all PPT $\mathcal{A}$, we have:*

$$\Pr\left[\begin{matrix}\tau = \tau' \\ \tau' \leftarrow \mathcal{A}(1^\lambda, \mathsf{pp})\end{matrix} : \begin{matrix}(\mathbb{F}, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2, g_t) \leftarrow \mathsf{BG}(1^\lambda),\ \tau \leftarrow \mathbb{F} \\ \mathsf{pp} := (g_1^\tau, g_1^{\tau^2}, \ldots, g_1^{\tau^q}, g_2^\tau, g_2^{\tau^2}, \ldots, g_2^{\tau^q})\end{matrix}\right] \leq \mathsf{negl}(\lambda)$$

3.4 Fiat-Shamir

An interactive protocol is *public-coin* if the verifier's messages are uniformly random strings. Public-coin protocols can be transformed into non-interactive arguments in the Random Oracle Model (ROM) by using the Fiat-Shamir (FS) [FS87] heuristic to derive the verifier's messages as the output of a Random Oracle. All protocols in this work are public-coin interactive protocols in the structured reference string (SRS) model where both the parties have access to a SRS, that are then compiled into non-interactive arguments using FS.

Given a public-coin interactive proof system $\Pi = (\mathsf{Setup}, \mathcal{P}, \mathcal{V})$, we denote the corresponding FS-compiled non-interactive proof system by $\Pi_{\mathsf{FS}} = (\mathsf{Setup}_{\mathsf{FS}}, \mathcal{P}_{\mathsf{FS}}, \mathcal{V}_{\mathsf{FS}})$. We write $\mathcal{P}_{\mathsf{FS}}^{\mathsf{H}}$ and $\mathcal{V}_{\mathsf{FS}}^{\mathsf{H}}$ to denote that the prover and verifier have oracle access to H. We denote by $\mathsf{Prove}, \mathsf{Verify}$, the non-interactive prover and verifier algorithms obtained by applying FS to the Eval public-coin interactive protocol, giving a non-interactive PCS scheme $(\mathsf{pp}, \mathsf{H}) \leftarrow \mathsf{Setup}(1^\lambda, n, d), C \leftarrow \mathsf{Com}(\mathsf{pp}, f(\mathbf{X})), (v, \pi) \leftarrow \mathsf{Prove}^{\mathsf{H}}(\mathsf{pp}, f(\mathbf{X}), x)$, $b \leftarrow \mathsf{Verify}^{\mathsf{H}}(\mathsf{pp}, C, v, x, \pi)$.

3.5 Polynomial Commitment Scheme

A polynomial commitment scheme (PCS) introduced in [KZG10] allows a prover to open evaluations of the committed polynomial succinctly. A PCS over $\mathbb{F}$ is a tuple $\mathsf{PC} = (\mathsf{Setup}, \mathsf{Com}, \mathsf{Open}, \mathsf{Eval})$ where:

- $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, n, \{D_i\}_{i\in[n]})$. On input security parameter λ, number of variables n and upper bounds $D_i \in \mathbb{N}$ on the degree of each variable X_i for a n-variate polynomial, Setup generates public parameters pp.

- $(C, \tilde{\mathbf{c}}) \leftarrow \mathsf{Com}(\mathsf{pp}, f(\mathbf{X}), \mathbf{d})$. On input the public parameters pp, and a n-variate polynomial $f(X_1, \cdots, X_n) \in \mathbb{F}[X_1, \ldots, X_n]$ with degree at most $\deg(X_i) = d_i \leq D_i$ for all i, Com outputs a commitment to the polynomial C, and additionally an opening hint $\tilde{\mathbf{c}}$.

- $b \leftarrow \mathsf{Open}(\mathsf{pp}, f(\mathbf{X}), \mathbf{d}, C, \tilde{\mathbf{c}})$. On input the public parameters pp, the commitment C and the opening hint $\tilde{\mathbf{c}}$, a polynomial $f(X_1, \cdots, X_n)$ with $d_i \leq D_i$, Open outputs a bit indicating accept or reject.

- $b \leftarrow \mathsf{Eval}(\mathsf{pp}, C, \mathbf{d}, \mathbf{x}, v; f(\mathbf{X}))$. A public coin interactive protocol $\langle P_{\mathsf{eval}}(f(\mathbf{X})), V_{\mathsf{eval}}\rangle(\mathsf{pp}, C, \mathbf{d}, \mathbf{x}, v)$ between a PPT prover and a PPT verifier. The parties have as common input public parameters pp, commitment C, degree d, evaluation point x, and claimed evaluation v. The prover has, in addition, the opening $f(X_1, \cdots, X_n)$ of C, with $\deg(X_i) \leq d_i$. At the end of the protocol, the verifier outputs 1 indicating accepting the proof that $f(x_1, \ldots, x_n) = v$, or outputs 0 indicating rejecting the proof.

A polynomial commitment scheme must satisfy completeness, binding and extractability.

Definition 3.4 (Completeness)**.** *For all polynomials* $f(X_1, \cdots, X_n) \in \mathbb{F}[X_1, \ldots, X_n]$ *with degree* $\deg(X_i) = d_i \leq D_i$*, for all* $(x_1, \ldots, x_n) \in \mathbb{F}^n$*,*

$$\Pr\left[\begin{array}{c|c} b = 1 & \begin{array}{c} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, n, \{D_i\}_{i\in[n]}), \\ v \leftarrow f(\mathbf{x}), \\ b \leftarrow \mathsf{Eval}(\mathsf{pp}, C, \mathbf{d}, \mathbf{x}, v; f(\mathbf{X})) \end{array} \end{array}\right] = 1$$

Definition 3.5 (Binding)**.** *A polynomial commitment scheme* PC *is binding if for all PPT* $\mathcal{A}$*, the following probability is negligible in* λ*:*

$$\Pr\left[\begin{array}{l|l} \mathsf{Open}(\mathsf{pp}, f_0, \mathbf{d_0}, C, \tilde{\mathbf{c}}_\mathbf{0}) = 1 \wedge & \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, n, \mathbf{D}) \\ \mathsf{Open}(\mathsf{pp}, f_1, \mathbf{d_1}, C, \tilde{\mathbf{c}}_\mathbf{1}) = 1 \wedge & (C, f_0, f_1, \tilde{\mathbf{c}}_0, \\ f_0 \neq f_1 & \quad \tilde{\mathbf{c}}_1, \mathbf{d_0}, \mathbf{d_1}) \leftarrow \mathcal{A}(\mathsf{pp}) \end{array}\right]$$

Definition 3.6 (Knowledge Soundness)**.** *For any PPT adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$*, there exists a PPT algorithm* $\mathcal{E}$ *such that the following probability is negligible in* λ*:*

$$\Pr\left(\begin{array}{c} b = 1 \wedge \\ \mathcal{R}_{\mathsf{Eval}}(\mathsf{pp}, C, \mathbf{x}, v; \tilde{f}, \tilde{\mathbf{c}}) = 0 \end{array} : \begin{array}{c} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, n, \{D_i\}_{i\in[n]}) \\ (C, \mathbf{d}, \mathbf{x}, v, \mathsf{st}) \leftarrow \mathcal{A}_1(\mathsf{pp}) \\ (\tilde{f}, \tilde{\mathbf{c}}) \leftarrow \mathcal{E}^{\mathcal{A}_2}(\mathsf{pp}, C, d) \\ b \leftarrow \langle \mathcal{A}_2(\mathsf{st}), V_{\mathsf{eval}} \rangle (\mathsf{pp}, C, \mathbf{d}, \mathbf{x}, v) \end{array}\right).$$

where the relation $\mathcal{R}_{\mathsf{Eval}}$ *is defined as follows:*

$$\begin{aligned} \mathcal{R}_{\mathsf{Eval}} = \{((\mathsf{pp}, C \in \mathbb{G},\ \mathbf{x} \in \mathbb{F}^n,\ v \in \mathbb{F});\ (f(X_1, \cdots, X_n), \tilde{\mathbf{c}})) : \\ (\mathsf{Open}(\mathsf{pp}, f, \mathbf{d}, C, \tilde{\mathbf{c}}_\mathbf{0}) = 1) \wedge v = f(\mathbf{x})\} \end{aligned}$$

Definition 3.7 (Knowledge Soundness for Non-Interactive PCS)**.** *For any PPT adversary* $\mathcal{A}$*, there exists a PPT algorithm* $\mathcal{E}$ *such that the following probability is negligible in* λ*:*

$$\Pr\left(\begin{array}{c} b = 1 \wedge \\ \mathcal{R}_{\mathsf{Eval}}(\mathsf{pp}, C, \mathbf{x}, v; \tilde{f}, \tilde{\mathbf{c}}) = 0 \end{array} : \begin{array}{c} (\mathsf{pp}, \mathsf{H}) \leftarrow \mathsf{Setup}(1^\lambda, n, \{D_i\}_{i\in[n]}) \\ (C, \mathbf{d}, \mathbf{x}, v, \pi) \leftarrow \mathcal{A}^{\mathsf{H}}(\mathsf{pp}) \\ (\tilde{f}, \tilde{\mathbf{c}}) \leftarrow \mathcal{E}^{\mathcal{A}}(\mathsf{pp}, C, d) \\ b \leftarrow \mathsf{Verify}^{\mathsf{H}}(\mathsf{pp}, C, \mathbf{d}, \mathbf{x}, v, \pi) \end{array}\right).$$

where the relation $\mathcal{R}_{\mathsf{Eval}}$ *is defined as follows:*

$$\begin{aligned} \mathcal{R}_{\mathsf{Eval}} = \{((\mathsf{pp}, C \in \mathbb{G},\ \mathbf{x} \in \mathbb{F}^n,\ v \in \mathbb{F});\ (f(X_1, \cdots, X_n), \tilde{\mathbf{c}})) : \\ (\mathsf{Open}(\mathsf{pp}, f, \mathbf{d}, C, \tilde{\mathbf{c}}_\mathbf{0}) = 1) \wedge v = f(\mathbf{x})\} \end{aligned}$$

Definition 3.8 (Succinctness)**.** *We require the commitments and the evaluation proofs to be of size independent of the degree of the polynomial, that is the scheme is* proof succinct *if* $|C|$ *is* $\mathsf{poly}(\lambda)$*,* $|\pi|$ *is* $\mathsf{poly}(\lambda)$ *where* π *is the transcript obtained by applying FS to* Eval*. Additionally, the scheme is* verifier succinct *if* Eval *runs in time* $\mathsf{poly}(\lambda) \cdot \mathsf{log}(d)$ *for the verifier.*

We refer to Appendix 3.6 for background material on the KZG univariate PCS.

3.6 The KZG PCS

The KZG univariate PCS was introduced in [KZG10]. We denote the KZG scheme by the tuple of PPT algorithms (KZG.Setup,KZG.Commit, KZG.Prove, KZG.Verify) as defined below.

Definition 3.9 (KZG PCS)**.** *Let* $(\mathbb{F}, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2, g_t)$ *be output of bilinear group generator* $\mathsf{BG}(1^\lambda)$*.*

- KZG.Setup *on input* $(1^\lambda, d)$*, where* d *is the degree bound, outputs*

$$\mathsf{srs} = (\{[\tau]_1, \ldots, [\tau^d]_1\}, \{[\tau]_2, \ldots, [\tau^d]_2\}$$

- KZG.Commit *on input* $(\mathsf{srs}, p(X))$*, where* $p(X) \in \mathbb{F}_{\leq d}[X]$*, outputs* $C = [p(\tau)]_1$
- KZG.Prove *on input* $(\mathsf{srs}, p(X), \alpha)$*, where* $p(X) \in \mathbb{F}_{\leq d}[X]$ *and* $\alpha \in \mathbb{F}$*, outputs* (v, π) *such that* $v = p(\alpha)$ *and* $\pi = [q(\tau)]_1$*, for* $q(X) = \frac{p(X) - p(\alpha)}{X - \alpha}$
- KZG.Verify *on input* $(\mathsf{srs}, C, v, \alpha, \pi)$*, outputs* 1 *if the following equation holds, and* 0 *otherwise:* $e(C - v[1]_1 + \alpha\pi, [1]_2) \stackrel{?}{=} e(\pi, [\tau]_2)$

KZG is shown to be evaluation binding assuming q-DLOG (Definition 3.3) and knowledge sound in the Algebraic Group Model (AGM). At a high level, AGM [FKL18] considers *algebraic* adversaries that are algorithms $\mathcal{A}$ such that every group element output by $\mathcal{A}$ is accompanied by a representation of that group element in terms of all the group elements that $\mathcal{A}$ has seen so far (input and output).

3.7 Polynomial IOP

Definition 3.10 (Polynomial IOP)**.** *A polynomial IOP is a public-coin interactive proof for a relation* $\mathcal{R} = (\mathsf{x}, \mathsf{w})$*.* $\mathcal{R}$ *is an oracle relation such that* x *consists of* oracles *to* μ*-variate polynomials over* $\mathbb{F}$*. These oracles can be queried at arbitrary points in* $\mathbb{F}^\mu$ *to evaluate the polynomial at these points. In every round in the protocol, the prover sends multi-variate polynomial oracles. The verifier in every round sends a random challenge. At the end of the protocol, the verifier (with oracle access to all the polynomial oracles sent so far) and given its own randomness outputs accept/reject. A PIOP satisfies completeness and knowledge-soundness.*

We use the following results about PIOPs and their compilation.

Lemma 3.2 ([CHM+20, BFS20, CBBZ23])**.** *If a PIOP is sound for an oracle relation* $\mathcal{R}$ *with soundness error* δ*, then it is knowledge sound for* $\mathcal{R}$ *with knowledge error* δ *and the extractor running in time polynomial in the witness size.*

In a *holographic* proof system, the verifier's direct access to the circuit is replaced with query access to encodings of the circuit. Consider proving knowledge of w such that $C(\mathsf{x}, \mathsf{w}) = 1$ where circuit C and input x are public. Here, the statement is divided into an index i that corresponds to the circuit description of C and an instance x that corresponds to the public input of the circuit. The ability to preprocess a circuit in an offline phase is captured by holographic proofs where the verifier does not receive the circuit description as an input but, makes a small number of queries to an encoding of it, instead. In a holographic PIOP, the circuit is encoded into polynomials called the index polynomials i in a preprocessing phase and the verifier has query access to i, in addition to queries that the verifier makes to the oracles sent by the prover.

Lemma 3.3 ([CHM+20, BFS20, CBBZ23])**.** *Let $\mathcal{R}$ be a relation over $\mathbb{F}$. Let $PIOP = (I, P, V)$ be a PIOP over $\mathbb{F}$ for $\mathcal{R}$ with negligible soundness error, and $\mathsf{PC} = (\mathsf{Setup}, \mathsf{Com}, \mathsf{Open}, \mathsf{Eval})$ be a polynomial commitment scheme over $\mathbb{F}$ that satisfies completeness, binding and extractability with negligible extraction error. Then there exists a compiler that compiles the PIOP using PC to obtain a public-coin argument of knowledge $\Pi = (\mathsf{Setup}, \mathcal{P}, \mathcal{V})$ for $\mathcal{R}$ with negligible knowledge error. If the PIOP is holographic, the argument system is a preprocessing argument system.*

This succinct argument system with a public-coin verifier is finally transformed into a SNARK via Fiat-Shamir.

4 Multilinear PCS from Univariate PCS

4.1 Core Protocol

Let $n = 2^\mu$ for some $\mu \in \mathbb{N}$ and let B_μ denote the hypercube $\{0,1\}^\mu$. Let $\widetilde{eq}_i$, $i \in [n]$ denote the basis of $\mathbb{F}^{\leq 1}[X_1, \ldots, X_\mu]$ where $\widetilde{eq}_i(\mathbf{x}) = \widetilde{eq}_\mu(\langle\, i \,\rangle, \mathbf{x})$. We use the following isomorphism between multilinear polynomials and univariate polynomials $\mathbb{F}[X]$ of degree at most n. More precisely, we define the isomorphism $\varphi : \mathbb{F}^{\leq 1}[X_1, \ldots, X_\mu] \longrightarrow \mathbb{F}^{<n}[X]$ of the $\mathbb{F}$-vector spaces as:

$$\sum_{i=1}^{n} f_i \widetilde{eq}_i(X_1, \ldots, X_\mu) \mapsto \sum_{i=1}^{n} f_i X^{i-1}$$

Multilinear Commitments from Univariate Commitments. We use the above isomorphism to commit to multilinear polynomials. For $\widetilde{f} \in \mathbb{F}^{\leq 1}[X_1, \ldots, X_\mu]$, we use $\hat{f}$ to denote the univariate polynomial $\varphi(\widetilde{f})$. Let uPC be a univariate polynomial commitment scheme. We define the multilinear polynomial commitment scheme mPC using uPC as a black-box. For a μ-variate multilinear polynomial $\widetilde{f}$, we define $\mathsf{mPC.Com}(\widetilde{f}) \to (\mathsf{cm}_f, \tilde{\omega})$, where $(\mathsf{cm}_f, \tilde{\omega}) \leftarrow \mathsf{uPC.Com}(\mathsf{pp}, \hat{f})$. Next, we describe evaluation protocol for committed multilinear polynomials.

Evaluation Protocol. Let $\mathbf{z} = (z_1, \ldots, z_\mu)$ be the evaluation point. The evaluation claim $\widetilde{f}(\mathbf{z}) = v$ is equivalent to proving inner product $\langle \mathbf{f}, \phi_z \rangle = v$ where $\mathbf{f}$ is the coefficient vector of $\widetilde{f}$ and $\phi_z = (\widetilde{eq}_1(\mathbf{z}), \ldots, \widetilde{eq}_n(\mathbf{z}))$. The vector ϕ_z can be described as the tensor product:

$$\phi_z = \begin{pmatrix} 1 - z_\mu \\ z_\mu \end{pmatrix} \otimes \cdots \otimes \begin{pmatrix} 1 - z_1 \\ z_1 \end{pmatrix}$$

Let $\hat{\Psi}(X; \mathbf{z})$ be the univariate polynomial with ϕ_z in reverse order as the coefficient vector. It can be seen that

$$\hat{\Psi}(X; \mathbf{z}) = (z_1 + (1 - z_1)X) \cdot (z_2 + (1 - z_2)X^2) \cdots (z_\mu + (1 - z_\mu)X^{2^{\mu-1}}).$$

The claimed dot product v is then the coefficient of X^{n-1} in the product $\hat{f}(X) \cdot \hat{\Psi}(X; \mathbf{z})$. This can be shown by the prover exhibiting univariate oracles $\hat{h}(X)$ and $\hat{g}(X)$ of degree at most $n - 2$ such that:

$$\hat{f}(X) \cdot \hat{\Psi}(X; \mathbf{z}) = X^n \hat{h}(X) + vX^{n-1} + \hat{g}(X) \tag{5}$$

We can check the above identity at a random point, by querying the univariate oracles $\hat{f}, \hat{g}$ and $\hat{h}$, whereas the verifier can evaluate $\hat{\Psi}(X; \mathbf{z})$ itself in $O(\log n)$ $\mathbb{F}$-operations. The prover can compute the product $\hat{f}(X) \cdot \hat{\Psi}(X; \mathbf{z})$ using $O(n \log n)$ field multiplications *without* using FFT, and subsequently compute $\hat{h}$ and $\hat{g}$.

4.2 Obtaining Linear Time Prover

We now use amortization to reduce prover complexity from $O(n \log n)$ in the core protocol to $O(n)$. Let $n = \ell m$, $\nu = \log \ell$ and $\kappa = \log m$, and thus $\mu = \nu + \kappa$. Now, we view the coefficient vector $\mathbf{f}$ of the multilinear polynomial $\widetilde{f}$ as $\ell \times m$ matrix. We write $\widetilde{f}$ as:

$$\widetilde{f}(\mathbf{x}, \mathbf{y}) = \sum_{\tau \in B_\nu} \widetilde{eq}_\nu(\tau, \mathbf{y}) \widetilde{f}(\mathbf{x}, \tau) = \sum_{i=1}^{\ell} \widetilde{eq}(\langle\, i \,\rangle, \mathbf{y}) \widetilde{f}(\mathbf{x}, \langle\, i \,\rangle) \tag{6}$$

In the above, we define $\widetilde{g}_i(\mathbf{x}) = \widetilde{f}(\mathbf{x}, \langle\, i \,\rangle)$ as κ-variate multilinear polynomials for $i \in [\ell]$. Next, we write $\hat{f}(X)$ in base X^m as:

$$\hat{f}(X) = \hat{g}_1(X) + X^m \hat{g}_2(X) + \cdots + X^{m(\ell-1)} \hat{g}_\ell(X) \tag{7}$$

where the polynomials $\hat{g}_i(X)$ are uniquely determined polynomials of degree $< m$. If we write the coefficient vector $\mathbf{f} = (\mathbf{g}_1, \ldots, \mathbf{g}_\ell)$, with $\mathbf{g}_i \in \mathbb{F}^m$, it is easily seen that $\mathbf{g}_i$ is the common coefficient vector of the multilinear polynomial $\widetilde{g}_i(\mathbf{x})$ and $\hat{g}_i(X) \in \mathbb{F}^{<m}[\,X\,]$. To prove the claim $\widetilde{f}(\mathbf{z}) = v$, the prover first sends commitments $\mathsf{cm}_1, \ldots, \mathsf{cm}_\ell$ to the univariate polynomials $\hat{g}_1(X), \ldots, \hat{g}_\ell(X)$ respectively, which are also commitments to the corresponding multilinear polynomials $\widetilde{g}_i(\mathbf{x})$, $i \in [\ell]$. It also sends the multilinear polynomial evaluations $v_1 = \widetilde{g}_1(\mathbf{z}_x), \ldots, v_\ell = \widetilde{g}_\ell(\mathbf{z}_x)$, where $\mathbf{z} = (\mathbf{z}_x, \mathbf{z}_y)$ with $\mathbf{z}_y \in \mathbb{F}^\nu$, $\mathbf{z}_x \in \mathbb{F}^\kappa$. The verifier checks the following:

Polynomials $\hat{g}_1(X), \ldots, \hat{g}_\ell(X)$ are correct: To check whether the committed polynomials $\hat{g}_i(X), i \in [\ell]$ represent the correct decomposition of $\hat{f}(X)$ according to Equation (6), the verifier needs to ensure a degree bound of $m-1$ on each of the committed polynomials in addition to checking the polynomial identity in Equation (7). To this end, the verifier sends a challenge $\gamma \leftarrow \mathbb{F}$, with the prover responding with the uPC commitment cm_T to the polynomial $\hat{t}(X) = X^{m-1} \sum_{i=1}^{\ell} \gamma^{i-1} \hat{g}_i(X^{-1})$. We note that with overwhelming probability over the choice of γ, $\hat{t}(X)$ is a polynomial if and only if $\deg(\hat{g}_i) < m$ for all $i \in [\ell]$. Next, the verifier sends and evaluation challenge $\beta \leftarrow \mathbb{F}$ and checks $\hat{t}(\beta) = \beta^{m-1} \sum_{i=1}^{\ell} \gamma^{i-1} \hat{g}_i(\beta^{-1})$. Using homomorphism of uPC, the verifier can compute commitment to the polynomial $\hat{G}(X) = \sum_{i=1}^{\ell} \gamma^{i-1} \hat{g}_i(X)$, and check evaluation proofs for $\hat{t}$ and $\hat{G}$ at β and β^{-1} respectively.

Multilinear evaluations $v_1, \ldots, v_\ell$ are correct: We can use the random challenge γ to batch verify the evalauations $\widetilde{g}_i(\mathbf{z}_x) = v_i$ as well. Note that the commitment to the univariate polynomial $\hat{G}(X)$ is also the commitment to the corresponding multilinear polynomial $\widetilde{G}(\mathbf{x}) = \sum_{i=1}^{\ell} \gamma^{i-1} \widetilde{g}_i(\mathbf{x})$. Thus, with overwhelming probability the correctness of all evaluations is implied by the evaluation $\widetilde{G}(\mathbf{z}_x) = \sum_{i=1}^{\ell} \gamma^{i-1} v_i$. To check this, the prover and the verifier execute the core protocol in Section 4.1 over the hypercube B_ν of size m.

Check correctness of evaluation v: Finally, we need to ensure that evaluation v follows from the evaluations of polynomials $\widetilde{g}_i$, $i \in [\ell]$. From Equation (6), we must have $v = \sum_{i=1}^{\ell} \widetilde{eq}(\langle\, i \,\rangle, \mathbf{z}_y) \cdot v_i$. The evaluations $\widetilde{eq}(\langle\, i \,\rangle, \mathbf{z}_y)$ for $i \in [\ell]$ can be computed in $O(\ell)$ $\mathbb{F}$-operations using standard techniques, which allows the verifier to check the correctness of v in $O(\ell)$ $\mathbb{F}$-operations.

Efficiency: We summarize the efficiency of the above construction. We first consider the prover complexity. Computing commitments to polynomials $\hat{g}_i$, $i \in [\ell]$ incurs a cost of $\ell \times \mathrm{t}^{\mathsf{com}}_{\mathsf{uPC}}(m)$ to the prover. Assuming a linear commitment complexity for uPC, we simplify this to $\mathrm{t}^{\mathsf{com}}_{\mathsf{uPC}}(n)$. Next, the first check involves computing polynomial $\hat{t}$, computing the commitment cm_T to $\hat{t}$ followed by evaluation proofs for polynomials $\hat{t}$ and $\hat{G}$ at β and β^{-1}. Thus, the prover effort for this step is $\mathrm{t}^{\mathsf{com}}_{\mathsf{uPC}}(m) + \mathrm{t}^{\mathsf{eval}}_{\mathsf{uPC}}(m) + O(n)\,\mathbb{F}$. In the second step, the prover executes the core protocol over the hypercube of size m, incurring $O(m \log m)\,\mathbb{F} + 2\mathrm{t}^{\mathsf{com}}_{\mathsf{uPC}}(m) + \mathrm{t}^{\mathsf{eval}}_{\mathsf{uPC}}(m)$ cost. Thus, the overall cost to the prover is:

$$\mathrm{t}^{\mathsf{eval}}_{\mathsf{mPC}}(n) = \mathrm{t}^{\mathsf{com}}_{\mathsf{uPC}}(n) + 3 \cdot \mathrm{t}^{\mathsf{com}}_{\mathsf{uPC}}(m) + 2 \cdot \mathrm{t}^{\mathsf{eval}}_{\mathsf{uPC}}(m) + O(n + m \log m)\,\mathbb{F}$$

Setting $\ell = \log n$ and $m = n/\log n$, the above simplifies to $O(n)\,\mathbb{F} + \mathrm{t}^{\mathsf{com}}_{\mathsf{uPC}}(n) + o(n)$. Similarly, the verification involves homomorphically combining ℓ commitments to obtain a commitment for the polynomial $\hat{G}$, and checking the correctness of v in the third step using $O(\ell)$ $\mathbb{F}$-operations, in addition to constant invocations of the uPC verifier. The proof size $|\pi|$ is dominated by ℓ commitments and ℓ evaluations in addition to constant number of uPC evaluation proofs. The complete scheme is detailed in Figure 1.

Setup: On input security parameter 1^λ and $\mu \in \mathbb{N}$, the setup outputs public parameters pp, where $\mathsf{pp} \leftarrow \mathsf{uPC.Setup}(1^\lambda, D)$ and $D \geq 2 \cdot 2^\mu$.
Commit: On input a polynomial $\widetilde{f} \in \mathbb{F}^{\leq 1}[X_1, \ldots, X_\mu]$, output $(C, \widetilde{\mathbf{c}})$ where $(C, \widetilde{\mathbf{c}}) \leftarrow \mathsf{uPC.Com}(\mathsf{pp}, \hat{f})$.
Eval: The interactive protocol between evaluation prover $\mathcal{P}$ and evaluation verifier $\mathcal{V}$ on common input $(C, \mathbf{z}, v)$, $\mathbf{z} \in \mathbb{F}^\mu$, $v \in \mathbb{F}$ and prover's input $\widetilde{f}$ such that $\widetilde{f}(\mathbf{z}) = v$ proceeds as:

1. $\mathcal{P}$ decomposes coefficient vector $\mathbf{f} \in \mathbb{F}^n$, $n = 2^\mu$ of the $\widetilde{f}$ as $(\mathbf{g}_1, \ldots, \mathbf{g}_\ell)$ with $\mathbf{g}_i \in \mathbb{F}^m$ where $\ell = \log n$ and $m = n/\log n$.
2. $\mathcal{P}$ computes commitment $\mathsf{cm}_i = \mathsf{uPC.Com}(\mathsf{pp}, \hat{g}_i(X))$, $i \in [\ell]$ where $\hat{g}_i(X)$ is the univariate polynomial with coefficient vector $\mathbf{g}_i$.
3. $\mathcal{P}$ computes multilinear evaluations $v_i = \widetilde{g}_i(\mathbf{z}_x)$, $i \in [\ell]$, where $\mathbf{z} = (\mathbf{z}_x, \mathbf{z}_y)$ with $\mathbf{z}_x \in \mathbb{F}^\kappa$ and $\mathbf{z}_y \in \mathbb{F}^\nu$, where $\kappa = \log m$ and $\nu = \log \ell$.
4. $\mathcal{P}$ sends (cm_i, v_i) for $i \in [\ell]$.
5. $\mathcal{V}$ sends $\gamma \leftarrow \mathbb{F}$.
6. $\mathcal{P}$ computes $\bar{v} = \sum_{i=1}^{\ell} \gamma^{i-1} v_i$. It then computes polynomials:
   - $\hat{G}(X) = \sum_{i=1}^{\ell} \gamma^{i-1} \hat{g}_i(X)$,
   - $\hat{h}(X)$ and $\hat{u}(X)$ satisfying
   $$\hat{G}(X) \cdot \hat{\Psi}(X; \mathbf{z}_y) = X^m \hat{h}(X) + \bar{v} X^{m-1} + \hat{u}(X) \tag{8}$$
   - $\hat{t}(X) = X^{m-1} \cdot \hat{G}(X^{-1}) + \gamma^m X^{m-2} \hat{u}(X^{-1})$.
7. $\mathcal{P}$ sends commitments $\mathsf{cm}_t, \mathsf{cm}_h, \mathsf{cm}_u$ to polynomials $\hat{t}(X), \hat{h}(X)$ and $\hat{u}(X)$.
8. $\mathcal{V}$ sends $\beta \leftarrow \mathbb{F}$.
9. $\mathcal{P}$ sends $V_t = \hat{t}(\beta)$, $V_G = \hat{G}(\beta^{-1})$, $V_h = \hat{h}(\beta^{-1})$ and $V_u = \hat{u}(\beta^{-1})$.
10. $\mathcal{V}$ computes $\mathsf{cm}_G = \sum_{i=1}^{\ell} \gamma^{i-1} \mathsf{cm}_i$.
11. $\mathcal{P}$ and $\mathcal{V}$ execute evaluation protocols to check evaluations of polynomials $\hat{t}, \hat{G}, \hat{h}$ and $\hat{u}$.
12. $\mathcal{V}$ checks:
    - $V_t = \beta^{m-1} \cdot V_G + \gamma^m \beta^{m-2} V_u$ (check degree bounds on $\hat{g}_i$, $\hat{u}$).
    - $V_G \cdot \hat{\Psi}(\beta^{-1}; \mathbf{z}_y) = \beta^{-m} V_h + \beta^{-(m-1)} \bar{v} + V_u$ (check Equation (8)).
    - $\sum_{i=1}^{\ell} \widetilde{eq}(\langle i \rangle, \mathbf{z}_y) \cdot v_i = v$ (check Equation (6)).
13. $\mathcal{V}$ outputs accepts if all the checks succeed. Otherwise it rejects.

Figure 1: Linear-prover multilinear PCS from homomorphic univariate PCS

Theorem 4.1. *Assuming that* uPC *is a homomorphic polynomial commitment scheme for univariate polynomials in* $\mathbb{F}[X]$*, the scheme* mPC *in Figure 1 is a homomorphic multilinear*

Setup: On input security parameter λ and degree bounds d_x, d_y, $\mathsf{Setup}(1^\lambda, d_x, d_y)$ outputs pp where $\mathsf{pp} \leftarrow \mathsf{uPC.Setup}(1^\lambda, d)$ where $d = d_x \cdot d_y$.
Commit: On input pp, $Q \in \mathbb{F}[X, Y]$ with $\deg_Y(Q) < n$ and $n \cdot \deg_X(Q) + \deg_Y(Q) < d$, Com outputs $(C, \mathbf{c})$ where $(C, \mathbf{c}) \leftarrow \mathsf{uPC.Com}(\mathsf{pp}, Q(X^n, X))$.
Eval: The protocol $\langle \mathcal{P}(Q(X,Y)), \mathcal{V}\rangle(\mathsf{pp}, C, (m,n), (\alpha, \beta), v)$ proceeds as:

1. $\mathcal{P}$ computes polynomials:
$$R(X,Y) = \frac{Q(X,Y) - Q(\alpha,Y)}{X - \alpha}, \quad u(Y) = \frac{Q(\alpha,Y) - v}{Y - \beta}$$
2. $\mathcal{P}$ computes commitments:
$$\mathsf{cm}_r = \mathsf{uPC.Com}(\mathsf{pp}, R(Y^n, Y)), \quad \mathsf{cm}_u = \mathsf{uPC.Com}(\mathsf{pp}, u(Y))$$
3. $\mathcal{P}$ sends cm_r and cm_u.
4. $\mathcal{V}$ sends $\delta \leftarrow \mathbb{F}$.
5. $\mathcal{P}$ sends evaluations $v_Q = Q(\delta^n, \delta)$, $v_r = R(\delta^n, \delta)$, $v_u = u(\delta)$.
6. $\mathcal{V}$ checks $v_Q = (\delta^n - \alpha) \cdot v_r + (\delta - \beta) \cdot v_u + v$.
7. $\mathcal{P}$ and $\mathcal{V}$ execute the following uPC evaluations:
$$\langle \mathcal{P}(Q(X^n, X)), \mathcal{V}\rangle(\mathsf{pp}, C, mn, \delta, v_Q)$$
$$\langle \mathcal{P}(R(X^n, X)), \mathcal{V}\rangle(\mathsf{pp}, \mathsf{cm}_r, mn - n, \delta, v_r)$$
$$\langle \mathcal{P}(u(X)), \mathcal{V}\rangle(\mathsf{pp}, \mathsf{cm}_u, n, \delta, v_u)$$
8. $\mathcal{V}$ accepts if all the evaluations accept and the checks accept.

Figure 2: Bivariate PCS from a Univariate PCS

PCS which achieves following efficiency parameters (n denotes the size of the polynomial):

$$\begin{aligned}
\textit{Commitment Cost}: &\quad \mathrm{t}^{\mathsf{com}}_{\mathsf{mPC}}(n) &&= \mathrm{t}^{\mathsf{com}}_{\mathsf{uPC}}(n) \\
\textit{Evaluation Cost}: &\quad \mathrm{t}^{\mathsf{eval}}_{\mathsf{mPC}}(n) &&= O(n)\,\mathbb{F} + \mathrm{t}^{\mathsf{com}}_{\mathsf{uPC}}(n) + o(n) \\
\textit{Verification Cost}: &\quad \mathrm{t}^{\mathsf{ver}}_{\mathsf{mPC}}(n) &&= O(\log n) + 2 \cdot \mathrm{t}^{\mathsf{ver}}_{\mathsf{uPC}}(n) \\
\textit{Proof Size}: &\quad |\pi_{\mathsf{mPC}}(n)| &&= O(\log n) + 2 \cdot |\pi_{\mathsf{uPC}}(n)|
\end{aligned}$$

4.3 SamaritanPCS: Multilinear PCS from KZG

We now exhibit SamaritanPCS: a multilinear PCS with constant-sized proofs in the AGM, obtained by transforming the KZG univariate PCS. The key ingredient we require is a bivariate PCS with constant proof-size. We use the bivariate PCS based on KZG scheme sketched in [ZBK+22], where a commitment to bivariate polynomial $u(X, Y)$ with $\deg_X(u) < m$ and $\deg_Y(u) < n$ is obtained as the commitment to univariate polynomial $u(Y^n, Y)$, which is committed using the usual KZG scheme. We present the detailed construction in Figure 2 where we slightly generalize the construction from [ZBK+22] using the above em-

bedding of bivariate polynomials into univariate polynomials from any univariate PCS. The key observation (see proof of Lemma 4.1) is that $u(\alpha, \beta) = v$ for such a polynomial u if and only if there exists a polynomial $p(Y)$ with $\deg(p) < n$ and $p(\beta) = v$ such that $u(Y^n, Y) = (Y^n - \alpha)r(Y) + p(Y)$ for some polynomial $r(Y)$. It also turns out that $p(Y) = u(\alpha, Y)$ which is the partial evaluation of the bivariate polynomial u at $X = \alpha$. We have the following:

Lemma 4.1. *Assuming that* uPC *is a polynomial commitment scheme for polynomials in* $\mathbb{F}[X]$*, the scheme in Figure 2 is a polynomial commitment scheme for polynomials in* $\mathbb{F}[X, Y]$ *which achieves following efficiency parameters where* m *and* n *are the* X *and* Y *degrees of the polynomial respectively.*

$$\begin{aligned}
\textit{Commit Cost:} \quad & \mathrm{t}^{\mathsf{com}}(m, n) && = \mathrm{t}^{\mathsf{com}}_{\mathsf{uPC}}(mn) \\
\textit{Evaluation Cost:} \quad & \mathrm{t}^{\mathsf{eval}}(m, n) && = O(mn)\,\mathbb{F} + \mathrm{t}^{\mathsf{eval}}_{\mathsf{uPC}}(mn) + \mathrm{t}^{\mathsf{eval}}_{\mathsf{uPC}}(n) \\
\textit{Verification Cost:} \quad & \mathrm{t}^{\mathsf{ver}} && = O(\mathrm{t}^{\mathsf{ver}}_{\mathsf{uPC}}) \\
\textit{Proof Size:} \quad & |\pi| && = 2 \cdot |\pi_{\mathsf{uPC}}| + O(1)
\end{aligned}$$

Proof. The completeness is trivial, so we skip it. Binding follows from the binding of the univariate PCS uPC and the fact that the map $u(X, Y) \mapsto u(Y^n, Y)$ is one-one on bivariate polynomials with degree in variable Y less than n. We now argue the knowledge soundness property. Suppose uPC satisfies Definition 3.6 for univariate polynomials. Let $\mathcal{E}_{\mathsf{uPC}}$ be the extractor for uPC and let $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ be the adversary in the knowledge soundness definition for the bivariate PCS in Figure 2. We construct the extractor $\mathcal{E}$ for bivariate PCS as follows: when $\mathcal{A}_1$ outputs $(C, (m, n), (\alpha, \beta), v, \mathsf{st})$, $\mathcal{E}$ invokes the extractor for uPC to extract $(q(Y), \tilde{\mathbf{c}}_q) \leftarrow \mathcal{E}^{\mathcal{A}_2}_{\mathsf{uPC}}(\mathsf{pp}, C, mn)$. Similarly, when adversary outputs commitments cm_r and cm_u in Step 2 $\mathcal{E}$ extracts as follows:

$$(r(Y), \tilde{\mathbf{c}}_r) \leftarrow \mathcal{E}^{\mathcal{A}_2}_{\mathsf{uPC}}(\mathsf{pp}, \mathsf{cm}_r, (m-1)n), \quad (u(Y), \tilde{\mathbf{c}}_u) \leftarrow \mathcal{E}^{\mathcal{A}_2}_{\mathsf{uPC}}(\mathsf{pp}, \mathsf{cm}_u, n)$$

If the verifier accepts, i.e, all the univariate evaluations accept, by knowledge-soundness of uPC we have

$$q(\delta) = v_Q, \quad r(\delta) = v_r, \quad u(\delta) = v_u$$

where $\deg(q) < mn$, $\deg(r) < (m-1)n$, and $\deg(u) < n-1$. with overwhelming probability. Note that the evaluation point (δ) for these commitments was determined by verifier's challenge; which is strictly weaker than the adversary in the knowledge-soundness game (Definition 3.6). Moreover, the polynomials satisfy $q(\delta) = (\delta^n - \alpha)r(\delta) + (\delta - \beta)u(\delta) + v$. Since δ was uniform in $\mathbb{F}$, with overwhelming probability, the polynomials satisfy the following identity:

$$q(Y) = (Y^n - \alpha)r(Y) + (Y - \beta)u(Y) + v \tag{9}$$

Now, let $Q(X, Y)$ be the unique bivariate polynomial with $\deg_X(Q) < m$ and $\deg_Y(Q) < n$ such that $q(Y) = Q(Y^n, Y)$. Note that such a Q exists from Lemma 3.1. Similarly, let $R(X, Y)$ be the bivariate polynomial with $\deg_X(R) < m - 1$ and $\deg_Y(R) < n$ such that

$r(Y) = R(Y^n, Y)$. Now, from Equation (9), we have $(Y-\beta)u(Y) + v = Q(Y^n, Y) - (Y^n - \alpha)R(Y^n, Y)$. Let $P(X, Y) = Q(X, Y) - (X - \alpha)R(X, Y)$. We note that $P(Y^n, Y) = (Y - \beta)u(Y) + v$. Writing $P(X, Y) = \sum_{i=1}^{m} X^{i-1} p_i(Y)$, we have $\sum_{i=1}^{m} Y^{n(i-1)} p_i(Y) = (Y - \beta)u(Y) + v$. Since, the right hand side has degree $< n$, we note that $p_i(Y) = 0$ for all $i > 1$. Thus, $P(X, Y) = (Y - \beta)u(Y) + v$. Thus, we hve $Q(X, Y) - (X - \alpha)R(X, Y) = (Y - \beta)u(Y) + v$, which implies $Q(\alpha, \beta) = v$. The extractor $\mathcal{E}$ outputs the polynomial $Q(X, Y)$. This proves knowledge soundness of the bivariate PCS. To see the efficiency claims, note that the prover can compute the polynomial $R(Y^n, Y)$ in Step 2 by performing long division of polynomial $Q(Y^n, Y)$ by $(Y^n - \alpha)$, which can be done in $O(mn)$ $\mathbb{F}$-operations. □

Remark: In Figure 2, we can also drop the degree bound on the variable X, in which case, the degree bounds need not be enforced for commitments C and cm_r in Step 7 (i.e, we implicitly set the bound to the maximum degree supported by setup parameters pp).

SamaritanPCS. We now return to the construction of constant-sized multilinear PCS. Once again, let $n = \ell m$ for some $\ell, m \in \mathbb{N}$ and further $\nu = \log \ell$, $\kappa = \log m$. We again consider the decomposition of polynomial $\hat{f}(X)$ as in Equation (7). Instead of sending commitments to polynomials $\hat{g}_i(X)$ and the claimed evaluations $v_i = \widetilde{g}_i(\mathbf{z}_x)$, the prover simply sends a commitment cm_Q to the bivariate polynomial $Q(X, Y) = \hat{g}_1(Y) + X \cdot \hat{g}_2(Y) + \cdots + X^{\ell-1}\hat{g}_l(Y)$. Note that Q is the unique polynomial with $\deg_X(Q) < \ell$ and $\deg_Y(Q) < m$ such that $\hat{f}(X) = Q(X^m, X)$. Similarly, instead of sending the full vector $\mathbf{v}$, the prover simply sends a commitment cm_v to the polynomial $\hat{v}(X) = \sum_{i=1}^{\ell} v_i X^{i-1}$. The verifier then sends a challenge $\gamma \leftarrow \mathbb{F}$, to which the prover responds by sending commitments cm_p to the polynomial $\hat{p}(X) = Q(\gamma, X)$ and evaluation $v_\gamma = \hat{v}(\gamma)$. We observe that the multilinear polynomial $\widetilde{p} = \sum_{i=1}^{\ell} \gamma^{i-1} \widetilde{g}_i$ and the univariate polynomial $\hat{p}$ have identical coefficients. Moreover, we have $\widetilde{p}(\mathbf{z}_x) = v_\gamma$. Invoking the core protocol on the polynomial $\widetilde{p}$ of size m, the prover can check this evaluation in time $O(m \log m)$ with $O(1)$ communication. The verifier also needs to check that the evaluations v_i encoded in $\hat{v}(X)$ satisfy Equation (6), i.e $\sum_{i=1}^{\ell} \widetilde{eq}(\langle\, i-1 \,\rangle, \mathbf{z}_y) v_i = v$. This is equivalent to checking that the corresponding multilinear polynomial $\tilde{v}$ (with coefficients v_i over the hypercube B_ν) evaluates to v at $\mathbf{z}_y$. This is again accomplished in cost $O(\ell \log \ell)$ using the core protocol on hypercube of size ℓ. Finally, it needs to be shown that the univariate polynomials $\hat{f}$ and $\hat{p}$ are correctly derived from bivariate polynomial Q as $Q(X^m, X)$ and $Q(\gamma, X)$ respectively. This can be shown by checking the identity a random point using the respective PCS. In summary, we require: (i) two instances of the core protocol on hypercubes of size m and ℓ respectively to show $\widetilde{p}(\mathbf{z}_x) = v(\gamma)$ and $\widetilde{v}(\mathbf{z}_y) = v$, and (ii) evaluations at random points to check $\hat{f}(X) = Q(X^m, X)$ and $\hat{p}(X) = Q(\gamma, X)$.

Optimized Protocol. We describe detailed and optimized version of the above blueprint to minimize the concrete proof size. Specifically, we use techniques from [CHM+20] to

enforce degree bounds on polynomials. Moreover, for our bivariate PCS, we use the fact that showing $\hat{f}(X) = Q(X^m, X)$ and $\hat{p}(X) = Q(\gamma, X)$ for the unique bivariate polynomial Q determined by $\hat{f}$ reduces to showing that $\hat{f}(X) = (X^m - \gamma)\hat{r}(X) + \hat{p}(X)$ for some polynomial $\hat{r}$, where $\deg(\hat{p}) < m$.

Let pp be the setup parameters. Both prover and verifier have access to the setup parameters pp, commitment C to the multilinear polynomial $\widetilde{f} \in \mathbb{F}^{\leq 1}[X_1, \ldots, X_\mu]$, $\mathbf{z} \in \mathbb{F}^\mu$ and claimed evaluation $v \in \mathbb{F}$. Additionally, the prover knows the polynomial $\widetilde{f}$ such that $\widetilde{f}(\mathbf{z}) = v$. The protocol between prover ($\mathcal{P}$) and verifier ($\mathcal{V}$) proceeds as follows. For clarity of presentation, we will describe the protocol in three phases:

- **Amortization Phase:**

  1. $\mathcal{P}$ computes: The prover decomposes coefficient vector $\mathbf{f} \in \mathbb{F}^n$, $n = 2^\mu$ of the $\widetilde{f}$ as $(\mathbf{g}_1, \ldots, \mathbf{g}_\ell)$. It computes $v_i = \widetilde{g}_i(\mathbf{z}_x)$ where $\mathbf{z} = (\mathbf{z}_x, \mathbf{z}_y)$ as before. The prover computes $\hat{v}(X) = \sum_{i=1}^{\ell} v_i X^{i-1}$.
  2. $\mathcal{P}$ sends commitment cm_v to polynomial $\hat{v}(X)$.
  3. $\mathcal{V}$ sends $\gamma \leftarrow \mathbb{F}$.
  4. $\mathcal{P}$ defines $Q(X, Y) = \sum_{i=1}^{\ell} X^{i-1}\hat{g}_i(Y)$, computes $\hat{p}(X) = Q(\gamma, X)$, $v_\gamma = \hat{v}(\gamma)$.
  5. $\mathcal{P}$ sends evaluation v_γ and commitment cm_p to the polynomial $\hat{p}$.

  Let ϕ_x denote the vector $\big(\widetilde{eq}_\kappa(\langle 1 \rangle, \mathbf{z}_x), \ldots, \widetilde{eq}_\kappa(\langle m \rangle, \mathbf{z}_x)\big)$, ϕ_y denote the vector

$$\big(\widetilde{eq}_\nu(\langle 1 \rangle, \mathbf{z}_y), \ldots, \widetilde{eq}_\nu(\langle \ell \rangle, \mathbf{z}_y)\big)$$

  and $\boldsymbol{\gamma}$ denote the vector $(1, \gamma, \ldots, \gamma^{\ell-1})$. Also let $\mathbf{v}$ and $\mathbf{p}$ denote coefficient vectors of polynomials $\hat{v}$ and $\hat{p}$ respectively. At the end of amortization phase, the verifier needs to check:

$$\langle \mathbf{v}, \phi_y \rangle = v, \quad \langle \mathbf{v}, \boldsymbol{\gamma} \rangle = v_\gamma,$$
$$\langle \mathbf{p}, \phi_x \rangle = v_\gamma, \quad \hat{f}(X) = (X^m - \gamma)\hat{r}(X) + \hat{p}(X)$$

  for some polynomial $\hat{r}$ and degree bounds $n-1$ and $m-1$ on $\hat{f}$ and $\hat{p}$ respectively. Next, the inner product checks involving $\mathbf{v}$ can be batched using a challenge $\alpha \leftarrow \mathbb{F}$ to a single check $\langle \mathbf{v}, \phi_y + \alpha\boldsymbol{\gamma} \rangle = v + \alpha v_\gamma$. As before, we turn the inner product checks into checking polynomial identities by constructing polynomials with "reversed" vectors as coefficients. We note that the polynomial $\hat{\Phi}(X; \gamma)$ defined by $(\gamma + X)(\gamma^2 + X^2) \cdots (\gamma^{2^{\nu-1}} + X^{2^{\nu-1}})$ has the reversed vector $\boldsymbol{\gamma}$ as its coefficients. In terms of polynomials, the prover needs to show there exist polynomials $\hat{h}(X)$, $\hat{u}(X)$, $\hat{a}(X)$, $\hat{b}(X)$ and $\hat{r}(X)$such that:

$$\begin{aligned} \hat{v}(X) \cdot (\hat{\Psi}(X; \mathbf{z}_y) + \alpha \cdot \hat{\Phi}(X; \gamma)) &= X^\ell \hat{a}(X) + (v + \alpha v_\gamma) \cdot X^{\ell-1} + \hat{b}(X) \\ \hat{p}(X) \cdot \hat{\Psi}(X; \mathbf{z}_x) &= X^m \hat{h}(X) + v_\gamma \cdot X^{m-1} + \hat{u}(X) \\ \hat{f}(X) &= (X^m - \gamma)\hat{r}(X) + \hat{p}(X) \end{aligned} \tag{10}$$

where we require $\deg(\hat{f}) < n$, $\deg(\hat{p}) < m$, $\deg(\hat{u}) < m-1$ and $\deg(\hat{b}) < \ell - 1$. As an optimization, the prover only sends commitments cm_u, cm_b to polynomials $\hat{u}(X)$ and $\hat{b}(X)$. To prove the existence of polynomials $\hat{h}(X), \hat{a}(X)$ and $\hat{r}(X)$ and enforce degree bounds on $\hat{f}, \hat{p}, \hat{u}$ and $\hat{b}$, the verifier provides a batching challenge $\beta \leftarrow \mathbb{F}$. As a response to the challenge β, the prover computes polynomials $\hat{t}(X)$ and $\hat{s}(X)$ as below (where D is the degree of srs in pp).

$$
\begin{aligned}
t(\hat{X}) = & \left(\frac{\hat{v}(X)(\hat{\Psi}(X; \mathbf{z}_y) + \alpha \cdot \hat{\Phi}(X; \gamma)) - (v + \alpha \cdot v_\gamma) X^{\ell - 1} - \hat{b}(X)}{X^\ell} \right) \\
& + \beta \cdot \left(\frac{\hat{p}(X)\hat{\Psi}(X; \mathbf{z}_x) - v_\gamma \cdot X^{m-1} - \hat{u}(X)}{X^m} \right) \\
& + \beta^2 \cdot \left(\frac{\hat{f}(X) - \hat{p}(X)}{X^m - \gamma} \right) \\
& + \beta^3 \cdot \hat{f}(X) + \beta^4 \cdot X^{n-m} \hat{p}(X) + \beta^5 \cdot X^{n-m+1} \hat{u}(X) + \beta^6 \cdot X^{n-\ell+1} \hat{b}(X), \\
\hat{s}(X) = & X^{D-n+1} \hat{t}(X)
\end{aligned} \tag{11}
$$

The prover sends commitments cm_t and cm_s to $\hat{t}(X)$ and $\hat{s}(X)$ respectively. We note that with high probability, $\hat{t}$ is a polynomial if and only if each component is a polynomial. Moreover, the pairing check $e(\mathsf{cm}_t, \left[X^{D-n+1}\right]_2) = e(\mathsf{cm}_s, [1]_2)$ ensures that $\deg(\hat{t}) < n$ which in turn ensures the degree bounds on polynomials $\hat{f}, \hat{p}, \hat{u}$ and $\hat{b}$. Finally, we note that cm_t can be computed in $o(n)$ multi-exponentiations as commitment C to $\hat{f}$ is known, while other polynomials are of asymptotically smaller size. Computing the commitment cm_s incurs n multi-exponentiations. The above interaction is summarized as the aggregation phase as below:

- **Aggregation Phase:**
  1. $\mathcal{V}$ sends $\alpha \leftarrow \mathbb{F}$.
  2. $\mathcal{P}$ computes polynomials $\hat{u}(X)$, $\hat{b}(X)$ as in Equation 10.
  3. $\mathcal{P}$ sends commitments cm_u, cm_b to polynomials $\hat{u}(X)$ and $\hat{b}(X)$.
  4. $\mathcal{V}$ sends $\beta \leftarrow \mathbb{F}$.
  5. $\mathcal{P}$ computes polynomials $\hat{t}(X)$ and $\hat{s}(X)$as in Equation 11.
  6. $\mathcal{P}$ sends commitments cm_t and cm_s to $\hat{t}(X)$ and $\hat{s}(X)$ respectively.

In the final phase, the verifier checks the identity describing polynomial $\hat{t}(X)$ at $\delta \leftarrow \mathbb{F}$. To do so, both the prover and verifier define the "linearized" polynomial $\hat{q}(X)$ obtained by

substituting $X = \delta$ in the public polynomials in the expression for $\hat{t}(X)$, i.e, they define:

$$\begin{aligned}\hat{q}(X) = t(\hat{X}) &- \left(\frac{\hat{v}(X)(\hat{\Psi}(\delta; \mathbf{z}_y) + \alpha \cdot \hat{\Phi}(\delta; \gamma)) - (v + \alpha \cdot v_\gamma)\delta^{\ell-1} - \hat{b}(X)}{\delta^\ell}\right) \\ &- \beta \cdot \left(\frac{\hat{p}(X)\hat{\Psi}(\delta; \mathbf{z}_x) - v_\gamma \cdot \delta^{m-1} - \hat{u}(X)}{\delta^m}\right) \\ &- \beta^2 \cdot \left(\frac{\hat{f}(X) - \hat{p}(X)}{\delta^m - \gamma}\right) \\ &- \beta^3 \cdot \hat{f}(X) + \beta^4 \cdot \delta^{n-m}\hat{p}(X) - \beta^5 \cdot \delta^{n-m+1}\hat{u}(X) - \beta^6 \cdot \delta^{n-\ell+1}\hat{b}(X) \qquad (12)\end{aligned}$$

The prover computes KZG proof Π to show $\hat{q}(\delta) = 0$. This incurs n multi-exponentiations for the prover. The verifier checks the proof Π using the commitment cm_q for polynomial $\hat{q}(X)$ which it can itself compute in $O(\log n)$ $\mathbb{F}$-operations and $O(1)$ $\mathbb{G}_1$ operations as $\hat{q}$ is a linear combination of committed polynomials. We summarize this final phase as the evaluation phase below:

- **Evaluation Phase:**
  1. $\mathcal{V}$ sends $\delta \leftarrow \mathbb{F}$.
  2. $\mathcal{P}$ and $\mathcal{V}$ define polynomial $\hat{q}(X)$ as in Equation 12.
  3. $\mathcal{P}$ sends KZG proof Π showing $\hat{q}(\delta) = 0$.
  4. $\mathcal{V}$ computes commitment cm_q to the polynomial $\hat{q}$.
  5. $\mathcal{V}$ verifies Π against the commitment cm_q and checks $e(\mathsf{cm}_t, \left[X^{D-n+1}\right]_2) = e(\mathsf{cm}_s, [1]_2)$.
  6. $\mathcal{V}$ accepts if all the checks pass, else it rejects.

Theorem 4.2. *The construction in this section is a multilinear PCS when instantiated over* $(\mathbb{F}, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2, g_t) \leftarrow \mathsf{BG}(1^\lambda)$ *with* KZG *as the univariate PCS, under the q-DLOG assumption for the bilinear group generator* BG *in the algebraic group model (AGM). For* μ*-variate polynomials, evaluation proof incurs* $O(n)$ $\mathbb{F}$*-operations and* $2n + o(n)$ *multi-exponentiations where* $n = 2^\mu$*. The verification incurs* $O(\mu)$ $\mathbb{F}$*-operations,* $O(1)$ $\mathbb{G}_1$*-operations and* 2 *pairing checks. The proof consists of* 7 $\mathbb{G}_1$*-elements and* 1 $\mathbb{F}$*-element. For BLS12-381 curve, the proof size is* 368 *bytes.*

4.4 Batched SamaritanPCS

We highlight excellent batching properties of the preceding construction with regard to the cryptographic operations incurred by the prover. In particular, to prove k evaluations $\widetilde{f}_1(\mathbf{z}^{(1)}) = v^{(1)}, \ldots, \widetilde{f}_k(\mathbf{z}^{(k)}) = v^{(k)}$, the prover only needs to compute $2n + O(k\sqrt{n})$ multi-exponentiations when using the preceding construction with $\ell = m = \sqrt{n}$. The communication also scales as $4k + 3$ $\mathbb{G}_1$ + k $\mathbb{F}$, which translates to additional 224 bytes for

each evaluation as against 368 bytes over the BLS12-381 curve. The key modified steps are as below:

- **Amortization Phase:** Here the prover starts by sending k polynomial commitments $\mathsf{cm}_v^{(i)}$ to polynomials $\hat{v}^{(i)}(X)$ for $i \in [k]$. Similarly in the final step it sends k polynomial commitments $\mathsf{cm}_p^{(i)}$ to polynomials $\hat{p}^{(i)}(X)$ for $i \in [k]$. Note that the prover incurs $k\sqrt{n}$ multi-exponentiations to compute the commitments and $O(kn)$ $\mathbb{F}$-operations to compute the polynomials.

- **Aggregation Phase:** Analogous to computation of polynomials $\hat{u}(X)$ and $\hat{b}(X)$ in the standalone scheme, the prover computes k polynomials $\hat{u}^{(i)}(X)$ and $\hat{b}^{(i)}(X)$ for $i \in [k]$. It sends commitments $\mathsf{cm}_u^{(i)}$ and $\mathsf{cm}_b^{(i)}$ to polynomials $\hat{u}^{(i)}(X)$ and $\hat{b}^{(i)}(X)$ respectively. On receiving the challenge $\beta \leftarrow \mathbb{F}$ from verifier, the prover computes k polynomials $\hat{t}^{(i)}(X)$, $i \in [k]$ following the computation in the previous section. However, instead of sending individual commitments to $\hat{t}^{(i)}(X)$, the prover computes the aggregate polynomial $\hat{t}(X) = \sum_{i=1}^{k} \beta^{7(i-1)} \hat{t}^{(i)}(X)$. It then computes $\hat{s}(X) = X^{D-n+1}\hat{t}(X)$. The prover sends commitments cm_t and cm_s to $\hat{t}$ and $\hat{s}$ respectively. Again, since the commitments to the polynomials $\hat{f}_i$ are known, the commitment cm_t can be computed in $O(k\sqrt{n})$ multi-exponentiations. The computation of the commitment cm_s requires n multi-exponentiations as before. The required polynomials can be computed in $O(kn)$ $\mathbb{F}$-operations.

- **Evaluation Phase:** The prover computes the linearized polynomial $\hat{q}(X)$ as a linear combination of $5k + 1$ committed polynomials $\hat{t}$ and $\{\hat{f}_i, \hat{p}_i, \hat{v}_i, \hat{u}_i, \hat{b}_i\}$, $i \in [k]$. The computation of evaluation proof $\hat{q}(\delta) = 0$ incurs $O(kn)$ $\mathbb{F}$-operations and n multi-exponentiations. The verifier can compute commitment to the polynomial $\hat{q}$ in $5k$ $\mathbb{G}_1$ operations and $O(k \log n)$ $\mathbb{F}$-operations (to compute the coefficients of linear combination). Two additional pairing checks are required as before.

Lemma 4.2. *The batched SamaritanPCS construction in this section is secure under the q-DLOG assumption for the bilinear group generator* BG *in the algebraic group model (AGM) and achieves the following asymptotics: to prove k evaluations of μ-variate multilinear polynomials the prover incurs $O(kn)$ $\mathbb{F}$-operations and $2n + O(k\sqrt{n})$ multi-exponentiations; the verifier incurs $O(k \log n)$ $\mathbb{F}$-operations, $O(k)$ $\mathbb{G}_1$-operations and 2 pairing checks, where $n = 2^\mu$. The proof consists of $(4k + 3)$ $\mathbb{G}_1$-elements and k $\mathbb{F}$-elements. For BLS12-381 curve, the proof size is $224k + 144$ bytes.*

5 LogSpartan: PIOP from Log-Up based Lookups

5.1 Oracle Composition Using Logarithmic Derivatives

The key technical challenge in proving the correct evaluation of MLE of sparse matrix M using Equation (3) is to prove correctness of the oracle $\widetilde{f_M} : B_\kappa \to \mathbb{F}$, defined by $\mathbf{y} \mapsto$

$\widetilde{eq}(\mathbf{r}_x\langle\, \widetilde{\mathsf{R}}_M(\mathbf{y})\,\rangle)$. The oracle $\widetilde{f}$ can be treated as composition of oracles $\widetilde{eq}(\mathbf{r}_x, \cdot) : B_\mu \to \mathbb{F}$, and $\widetilde{\mathsf{R}}_M(\cdot) : B_\kappa \to \mathbb{F}$, where the function $\langle\, \cdot\,\rangle$ maps the co-domain of $\widetilde{\mathsf{R}}_M$ to the domain of $\widetilde{eq}(\mathbf{r}_x, \cdot)$. We first present a PIOP for the oracle composition relation in isolation, before applying it to obtain a more efficient variant of Spartan PIOP.

Definition 5.1 (Oracle Composition)**.** *Let $\mu, \kappa \in \mathbb{N}$. We define the oracle relation $\mathcal{R}^{\mathsf{comp}}_{\mu,\kappa}$ as the set of pairs $(\mathbf{x}, \mathbf{w})$ with $\mathbf{x} = ([\![\widetilde{f}]\!], [\![\widetilde{g}]\!], [\![\widetilde{h}]\!])$ and $\mathbf{w} = (\widetilde{f}, \widetilde{g}, \widetilde{h})$ where $\widetilde{g}$, is a μ-variate multilinear polynomial and $\widetilde{f}, h$ are κ-variate multilinear polynomials satisfying $\widetilde{f}(\mathbf{y}) = \widetilde{g}(\langle\, \widetilde{h}(\mathbf{y})\,\rangle_\mu)$. Here $[\![\cdot]\!]$ denotes the oracle access to the polynomial.*

The oracle composition can be viewed as an indexed-lookup relation over the evaluation vectors of multilinear polynomials over their respective domains.

Definition 5.2 (Indexed Lookup)**.** *Let m, n be integers. We say that vectors $\mathbf{t} \in \mathbb{F}^n$, $\mathbf{a} \in \mathbb{F}^m$ and $\mathbf{v} \in \mathbb{F}^m$ satisfy* indexed lookup *relation denoted by $\mathbf{v} = \mathbf{t}[\,\mathbf{a}\,]$ if $v_i = t_{a_i}$ for all $i \in [m]$.*

We will use the following result on logarithmic derivatives of polynomials from [Hab22a] for proving indexed lookup relation.

Lemma 5.1 ([Hab22a])**.** *Let m, n be positive integers and let $\mathbb{F}$ be a field of characteristic $p > n$. Then, $\mathbf{t} \in \mathbb{F}^n$, $\mathbf{a} \in \mathbb{F}^m$ and $\mathbf{b} \in \mathbb{F}^m$ satisfy the indexed lookup relation in Definition 5.2 if and only if there exists vector $\mathbf{m} = (m_1, \ldots, m_n) \in \mathbb{F}^n$ such that*

$$\sum_{i=1}^{n} \frac{m_i}{X + iY + t_i} = \sum_{i=1}^{m} \frac{1}{X + a_i Y + v_i}$$

Identifying the vectors in Lemma 5.1 with the evaluations of multilinear polynomials, we see that the relation $\widetilde{f}(\mathbf{y}) = \widetilde{g}(\langle\, \widetilde{h}(\mathbf{y})\,\rangle)$ is implied by the existence of multilinear polynomial $\widetilde{\chi}$ (which interpolates the vector $(m_{\mathbf{y}})_{\mathbf{y} \in B_\mu}$ in Lemma 5.1) satisfying the identity of rational functions:

$$\sum_{\mathbf{y} \in B_\mu} \frac{\widetilde{\chi}(\mathbf{y})}{X + \widetilde{\mathsf{id}}(\mathbf{y})Y + \widetilde{g}(\mathbf{y})} = \sum_{\mathbf{y} \in B_\kappa} \frac{1}{X + \widetilde{h}(\mathbf{y})Y + \widetilde{f}(\mathbf{y})} \tag{13}$$

After the prover supplies the oracle $[\![\widetilde{\chi}]\!]$, the verifier can check the above identity probabilistically. It sends $\alpha, \beta \leftarrow \mathbb{F}$ to the prover, who then proves the claim:

$$\sum_{\mathbf{y} \in B_\mu} \frac{\widetilde{\chi}(\mathbf{y})}{\alpha + \beta\widetilde{\mathsf{id}}(\mathbf{y}) + \widetilde{g}(\mathbf{y})} = \sum_{\mathbf{y} \in B_\kappa} \frac{1}{\alpha + \beta\widetilde{h}(\mathbf{y}) + \widetilde{f}(\mathbf{y})} \tag{14}$$

The above identity involves summation over two distinct hypercubes B_μ and B_κ. For reasons of efficiency, it helps us to homogenize the above check to involve summation over a single hypercube B_ν, where $\nu = \max(\mu, \kappa)$. Let $\delta_\mu = \nu - \mu$ and $\delta_\kappa = \nu - \kappa$. We can

vacuously view the μ-variate and κ-variate multilinear polynomials as ν-variate multilinear polynomials. It is easily seen that the claim in Equation (14) is equivalent to the following homogenized identity:

$$\sum_{\mathbf{y}\in B_\nu}\left(2^{\delta_\kappa}\cdot\frac{\widetilde{\chi}(\mathbf{y})}{\alpha+\beta\widetilde{\mathsf{id}}(\mathbf{y})+\widetilde{g}(\mathbf{y})}-2^{\delta_\mu}\cdot\frac{1}{\alpha+\beta\widetilde{h}(\mathbf{y})+\widetilde{f}(\mathbf{y})}\right)=0 \tag{15}$$

5.2 LogSpartan: PIOP using Log-Derivative Based Lookups

We now use the techniques of the previous subsection to give an efficient evaluation proof for the sparse multilinear polynomials encoding the R1CS matrices. Consider the computation of $\widetilde{M}(\mathbf{r}_x,\mathbf{r}_y)$ in Equation (3). To establish the correctness of the claim $\widetilde{M}(\mathbf{r}_x,\mathbf{r}_y)=v_M$, the prover can send oracles $[\![\widetilde{f}_M]\!]$, $[\![\widetilde{g}_M]\!]$ for multilinear polynomials $\widetilde{f}_M : B_\kappa \to \mathbb{F}$ and $\widetilde{g}_M : B_\kappa \to \mathbb{F}$, and prove:

$$\sum_{\mathbf{k}\in B_\kappa}\widetilde{\mathsf{val}}_M(\mathbf{k})\cdot\widetilde{f}_M(\mathbf{k})\cdot\widetilde{g}_M(\mathbf{k})=v_M \tag{16}$$

$$\widetilde{f}_M(\mathbf{k})=\widetilde{eq}_\mu(\mathbf{r}_x,\langle\,\mathsf{R}_M(\mathbf{k})\,\rangle) \tag{17}$$

$$\widetilde{g}_M(\mathbf{k})=\widetilde{eq}_\mu(\mathbf{r}_y,\langle\,\mathsf{C}_M(\mathbf{k})\,\rangle) \tag{18}$$

Now, Equation 16 above is a simple application of the sum-check protocol. The latter two identities are instances of oracle composition discussed in the last subsection. First, we introduce virtual oracles to reduce the two instances of oracle composition to a single instance. We define the virtual oracles $\widetilde{h}_M : B_{\kappa+1}\to\mathbb{F}$, $\widetilde{u}_M : B_{\kappa+1}\to\mathbb{F}$ and $\widetilde{T} : B_{\mu+1}\to\mathbb{F}$ which correspond to concatenated oracles $\widetilde{f}_M||\widetilde{g}_M$, $\widetilde{\mathsf{R}_M}||\widetilde{\mathsf{C}_M}$ and $\widetilde{eq}_\mu(\mathbf{r}_x,\cdot)||\widetilde{eq}_\mu(\mathbf{r}_y,\cdot)$ respectively. The virtual oracles are defined as follows:

$$\begin{aligned}\widetilde{h}_M(\mathbf{k},k_{\kappa+1})&=(1-k_{\kappa+1})\widetilde{f}_M(\mathbf{k})+k_{\kappa+1}\widetilde{g}_M(\mathbf{k})\\ \widetilde{u}_M(\mathbf{k},k_{\kappa+1})&=(1-k_{\kappa+1})\widetilde{\mathsf{R}_M}(\mathbf{k})+k_{\kappa+1}(\widetilde{\mathsf{C}_M}(\mathbf{k})+2^\mu)\\ \widetilde{T}(\mathbf{y},y_{\mu+1})&=(1-y_{\mu+1})\widetilde{eq}_\mu(\mathbf{r}_x,\mathbf{y})+y_{\mu+1}\widetilde{eq}_\mu(\mathbf{r}_y,\mathbf{y})\end{aligned}$$

In terms of the virtual oracles, the oracle composition instances in Equations 17 and 18 are equivalent to the oracle composition $\widetilde{h}_M(\mathbf{k})=\widetilde{T}(\langle\widetilde{u}_M(\mathbf{k})\rangle_{\mu+1})$. By the previous discussion, the above is achieved by the prover sending the oracle $[\![\widetilde{\chi}_M]\!]$ for a multilinear polynomial $\widetilde{\chi}_M : B_\nu\to\mathbb{F}$ for $\nu=\max(\mu+1,\kappa+1)$ satisfying the homogenized identity:

$$\sum_{\mathbf{y}\in B_\nu}\left(2^{\delta_\kappa}\cdot\frac{\widetilde{\chi}_M(\mathbf{y})}{\alpha+\beta\widetilde{\mathsf{id}}(\mathbf{y})+\widetilde{T}(\mathbf{y})}-2^{\delta_\mu}\cdot\frac{1}{\alpha+\beta\widetilde{h}_M(\mathbf{y})+\widetilde{u}_M(\mathbf{y})}\right)=0 \tag{19}$$

Finally, to verify evaluations $\widetilde{M}(\mathbf{r}_x,\mathbf{r}_y)=v_M$ for all $M\in\{A,B,C\}$, the verifier uses random challenges $\Gamma_A,\Gamma_B,\Gamma_C\leftarrow\mathbb{F}$. The relation in Equation 16 is verified for all the

matrices using the sum-check:

$$\sum_{\mathbf{k}\in B_\kappa}\sum_{M\in\{A,B,C\}}\Gamma_M\cdot\widetilde{\mathsf{val}}_M(\mathbf{k})\cdot\widetilde{f}_M(\mathbf{k})\cdot\widetilde{g}_M(\mathbf{k})=\Gamma_A v_A+\Gamma_B v_B+\Gamma_C v_C. \tag{20}$$

To prove relation in Equation 19 for all $M\in\{A,B,C\}$, the prover shows the following:

$$\sum_{\mathbf{y}\in B_\nu}\sum_{M\in\{A,B,C\}}\Gamma_M\cdot\left(\frac{2^{\delta_\kappa}\cdot\widetilde{\chi}_M(\mathbf{y})}{\alpha+\beta\widetilde{\mathsf{id}}(\mathbf{y})+\widetilde{T}(\mathbf{y})}-\frac{2^{\delta_\mu}}{\alpha+\beta\widetilde{h}_M(\mathbf{y})+\widetilde{u}_M(\mathbf{y})}\right)=0 \tag{21}$$

To prove the above, the prover sends the oracle $[\![s]\!]$ for a multilinear polynomial $s:B_\nu\to\mathbb{F}$ satisfying the identity:

$$\widetilde{s}(\mathbf{y})=\sum_{M\in\{A,B,C\}}\Gamma_M\cdot\left(\frac{2^{\delta_\kappa}\cdot\widetilde{\chi}_M(\mathbf{y})}{\alpha+\beta\widetilde{\mathsf{id}}(\mathbf{y})+\widetilde{T}(\mathbf{y})}-\frac{2^{\delta_\mu}}{\alpha+\beta\widetilde{h}_M(\mathbf{y})+\widetilde{u}_M(\mathbf{y})}\right) \tag{22}$$

The identity in Equation 21 follows from the sum-check $\sum_{\mathbf{y}\in B_\nu}\widetilde{s}(\mathbf{y})=0$ while the well-formedness of the oracle $[\![s]\!]$ reduces to a zero-check PIOP for a degree 5 multilinear polynomial (obtained by clearing the denominators in Equation 22). This results in a sum-check of degree 6 using standard techniques for zero-check PIOP. Finally, to enable batching, we can transform the sum-check in Equation 20 to a sum-check over the hypercube B_ν as:

$$\sum_{\mathbf{y}\in B_\nu}\left(\sum_{M\in\{A,B,C\}}\Gamma_M\cdot\widetilde{\mathsf{val}}_M(\mathbf{y})\cdot\widetilde{f}_M(\mathbf{y})\cdot\widetilde{g}_M(\mathbf{y})\right)=2^{\delta_\kappa}\big(\Gamma_A v_A+\Gamma_B v_B+\Gamma_C v_C\big)$$

where we vacuously view oracles over B_κ as oracles over B_ν. The complete Spartan PIOP using logarithmic derivatives appears in Figure 3.

Lemma 5.2. *Given integers $n,\mu\in\mathbb{N}$ with $n=2^\mu$, the interactive oracle protocol in Figure 3 is complete and knowledge sound PIOP for the relation $\mathcal{R}_n^{\mathsf{R1CS}}$.*

5.3 LogSpartan: Alternate PIOP with Smaller Proof

In this subsection, we present an alternate PIOP to the one in Section 5.2. This alternative version has a smaller proof size but incurs larger prover computation. Again, we consider the computation of $\widetilde{M}(\mathbf{r}_x,\mathbf{r}_y)$ in Equation (3). To establish the claim $\widetilde{M}(\mathbf{r}_x,\mathbf{r}_y)=v_M$, the prover can send oracles $[\![\widetilde{f}_M]\!]$, $[\![\widetilde{g}_M]\!]$ for multilinear polynomials $\widetilde{f}_M:B_\kappa\to\mathbb{F}$ and $\widetilde{g}_M:B_\kappa\to\mathbb{F}$, and prove the following:

$$\sum_{\mathbf{k}\in B_\kappa}\widetilde{\mathsf{val}}_M(\mathbf{k})\cdot\widetilde{f}_M(\mathbf{k})\cdot\widetilde{g}_M(\mathbf{k})=v_M \tag{23}$$

$$\widetilde{f}_M(\mathbf{k})=\widetilde{eq}_\mu(\mathbf{r}_x,\langle\,\mathsf{R}_M(\mathbf{k})\,\rangle) \tag{24}$$

$$\widetilde{g}_M(\mathbf{k})=\widetilde{eq}_\mu(\mathbf{r}_y,\langle\,\mathsf{C}_M(\mathbf{k})\,\rangle) \tag{25}$$

Common Input : κ-variate oracles for $n \times n$ sparse matrices $(\widetilde{\mathsf{val}}_M, \widetilde{\mathsf{R}}_M, \widetilde{\mathsf{C}}_M)_{M \in \{A,B,C\}}$ for $n = 2^\mu$. It holds that $\widetilde{M}(\mathbf{x}, \mathbf{y}) = \sum_{\mathbf{z} \in B_\kappa} \widetilde{\mathsf{val}}_M(\mathbf{z}) \cdot \widetilde{eq}_\mu(\mathbf{x}, \langle \widetilde{\mathsf{R}}_M(\mathbf{z}) \rangle) \cdot \widetilde{eq}_\mu(\mathbf{y}, \langle \widetilde{\mathsf{C}}_M(\mathbf{z}) \rangle)$ for $M \in \{A, B, C\}$.
Prover's Input : $\mathbf{z} \in \mathbb{F}^n$, for $n = 2^\mu$.

1. $\mathcal{P}$ sends ML extension $\widetilde{z}$ of $\mathbf{z}$ as oracle $[\![\widetilde{z}]\!]$.
2. $\mathcal{V}$ sends $\mathbf{t} \leftarrow \mathbb{F}^\mu$.
3. $\mathcal{P}$ and $\mathcal{V}$ execute the following sum-check:
$$\sum_{\mathbf{x} \in B_\mu} \widetilde{eq}(\mathbf{t}, \mathbf{x}) \left[\left(\sum_{\mathbf{y} \in B_\mu} \widetilde{A}(\mathbf{x}, \mathbf{y}) \widetilde{z}(\mathbf{y}) \right) \left(\sum_{\mathbf{x} \in B_\mu} \widetilde{B}(\mathbf{x}, \mathbf{y}) \widetilde{z}(\mathbf{y}) \right) - \left(\sum_{\mathbf{y} \in B_\mu} \widetilde{C}(\mathbf{x}, \mathbf{y}) \widetilde{z}(\mathbf{y}) \right) \right] = 0$$
4. After μ rounds of above sum-check, with $\mathbf{r}_x = (r_1, \ldots, r_\mu) \in \mathbb{F}^\mu$ as $\mathcal{V}$'s challenges and t_μ as μ^{th} polynomial sent by $\mathcal{P}$, the sum-check reduces to:
$$\widetilde{eq}(\mathbf{t}, \mathbf{r}_x) \left[\left(\sum_{\mathbf{y} \in B_\mu} \widetilde{A}(\mathbf{r}_x, \mathbf{y}) \widetilde{z}(\mathbf{y}) \right) \left(\sum_{\mathbf{y} \in B_\mu} \widetilde{B}(\mathbf{r}_x, \mathbf{y}) \widetilde{z}(\mathbf{y}) \right) - \left(\sum_{\mathbf{y} \in B_\mu} \widetilde{C}(\mathbf{r}_x, \mathbf{y}) \widetilde{z}(\mathbf{y}) \right) \right] = t_\mu(r_\mu)$$
5. $\mathcal{P}$ sends v_A, v_B and v_C
6. $\mathcal{V}$ checks: $\widetilde{eq}(\mathbf{t}, \mathbf{r}_x)(v_A \cdot v_B - v_C) = t_\mu(r_\mu)$. It aborts if the check fails, else it sends $\rho_A, \rho_B, \rho_C \leftarrow \mathbb{F}$.
7. $\mathcal{P}$ and $\mathcal{V}$ execute the following sum-check:
$$\sum_{\mathbf{y} \in B_\mu} (\rho_A \widetilde{A}(\mathbf{r}_x, \mathbf{y}) + \rho_B \widetilde{B}(\mathbf{r}_x, \mathbf{y}) + \rho_C \widetilde{C}(\mathbf{r}_x, \mathbf{y})) \widetilde{z}(\mathbf{y}) = \rho_A v_A + \rho_B v_B + \rho_C v_C$$
8. After μ rounds of the above sum-check, let $\mathbf{r}_y = (r'_1, \ldots, r'_\mu)$ be $\mathcal{V}$'s challenges, and t'_μ be the final polynomial sent by $\mathcal{P}$. Subsequently, $\mathcal{P}$ sends purported evaluations $\bar{v}_A, \bar{v}_B, \bar{v}_C$ of $\widetilde{A}$, $\widetilde{B}$ and $\widetilde{C}$ at $(\mathbf{r}_x, \mathbf{r}_y)$.
9. $\mathcal{V}$ queries $[\![\widetilde{z}]\!]$ at $\mathbf{r}_y$ to obtain $\bar{v}_Z$.
10. $\mathcal{V}$ checks $\bar{v}_Z(\rho_A \bar{v}_A + \rho_B \bar{v}_B + \rho_C \bar{v}_C) = t'_\mu(r'_\mu)$. It aborts on failure.
11. $\mathcal{P}$ and $\mathcal{V}$ execute PIOP in Section 5.2 to show $\widetilde{A}(\mathbf{r}_x, \mathbf{r}_y) = \bar{v}_A$, $\widetilde{B}(\mathbf{r}_x, \mathbf{r}_y) = \bar{v}_B$ and $\widetilde{C}(\mathbf{r}_x, \mathbf{r}_y) = \bar{v}_C$.
12. $\mathcal{V}$ accepts if all the checks are satisfied and above PIOP accepts.

Figure 3: LogSpartan: Multilinear PIOP using Logarithmic Derivatives

Now, Equation (23) above is a simple application of the sum-check protocol. The latter two identities are instances of oracle composition discussed in Section 5.1. In a departure from the approach in Section 5.2, we combine the resulting six instances of oracle composition (two instances for each $M \in \{A, B, C\}$) into a single instance of oracle composition by

defining virtual oracles over $B_{\mu+1}$ and $B_{\kappa+3}$ as follows:

$$\begin{aligned}
\widetilde{T}(\mathbf{y}, y_{\mu+1}) &= (1-y_{\mu+1})\widetilde{eq}_{\mu}(\mathbf{r}_x, \mathbf{y}) + y_{\mu+1}\widetilde{eq}_{\mu}(\mathbf{r}_y, \mathbf{y}) \\
\widetilde{D}(\mathbf{k}, \mathbf{k}') &= \widetilde{eq}(\langle 1 \rangle, \mathbf{k}') \cdot \widetilde{\mathsf{R}}_A(\mathbf{k}) + \widetilde{eq}(\langle 2 \rangle, \mathbf{k}') \cdot \widetilde{\mathsf{R}}_B(\mathbf{k}) + \widetilde{eq}(\langle 3 \rangle, \mathbf{k}') \cdot \widetilde{\mathsf{R}}_C(\mathbf{k}) \\
&\quad + \widetilde{eq}(\langle 4 \rangle, \mathbf{k}') \cdot (2^{\mu} + \widetilde{\mathsf{C}}_A(\mathbf{k})) + \widetilde{eq}(\langle 5 \rangle, \mathbf{k}') \cdot (2^{\mu} + \widetilde{\mathsf{C}}_B(\mathbf{k})) \\
&\quad + \widetilde{eq}(\langle 6 \rangle, \mathbf{k}') \cdot (2^{\mu} + \widetilde{\mathsf{C}}_C(\mathbf{k})) + \widetilde{eq}(\langle 7 \rangle, \mathbf{k}') \cdot \widetilde{\mathsf{R}}_A(\mathbf{k}) + \widetilde{eq}(\langle 8 \rangle, \mathbf{k}') \cdot \widetilde{\mathsf{R}}_A(\mathbf{k}) \\
\widetilde{h}(\mathbf{k}, \mathbf{k}') &= \widetilde{eq}(\langle 1 \rangle, \mathbf{k}') \cdot \widetilde{f}_A(\mathbf{k}) + \widetilde{eq}(\langle 2 \rangle, \mathbf{k}') \cdot \widetilde{f}_B(\mathbf{k}) + \widetilde{eq}(\langle 3 \rangle, \mathbf{k}') \cdot \widetilde{f}_C(\mathbf{k}) \\
&\quad + \widetilde{eq}(\langle 4 \rangle, \mathbf{k}') \cdot \widetilde{g}_A(\mathbf{k}) + \widetilde{eq}(\langle 5 \rangle, \mathbf{k}') \cdot \widetilde{g}_B(\mathbf{k}) + \widetilde{eq}(\langle 6 \rangle, \mathbf{k}') \cdot \widetilde{g}_C(\mathbf{k}) \\
&\quad + \widetilde{eq}(\langle 7 \rangle, \mathbf{k}') \cdot \widetilde{f}_A(\mathbf{k}) + \widetilde{eq}(\langle 8 \rangle, \mathbf{k}') \cdot \widetilde{f}_A(\mathbf{k}) \qquad (26)
\end{aligned}$$

In the above, we have $\mathbf{k}' = (k_{\kappa+1}, k_{\kappa+2}, k_{\kappa+3})$, while $\widetilde{eq}(\langle i \rangle, \mathbf{k}')$, $i \in [8]$ are the basis polynomials for the 3-dimensional hypercube B_3. The polynomial $\widetilde{T}$ interpolates the table obtained by concatenating $(\widetilde{eq}(\mathbf{r}_x, \mathbf{y}))_{\mathbf{y} \in B_\mu}$ and $(\widetilde{eq}(\mathbf{r}_y, \mathbf{y}))_{\mathbf{y} \in B_\mu}$. The polynomial $\widetilde{D}$ interpolates the vector of lookup indices obtained by concatenating evaluation vectors of following index polynomials

$$(\widetilde{\mathsf{R}}_A, \widetilde{\mathsf{R}}_B, \widetilde{\mathsf{R}}_C, \widetilde{\mathsf{C}}_A, \widetilde{\mathsf{C}}_B, \widetilde{\mathsf{C}}_C, \widetilde{\mathsf{R}}_A, \widetilde{\mathsf{R}}_A)$$

Note that the polynomials $\widetilde{\mathsf{C}}_M$, $M \in \{A, B, C\}$ are translated by 2^{μ}, as the corresponding table $\widetilde{eq}(\mathbf{r}_y, \cdot)$ starts at the 2^{μ}-th location inside $\widetilde{T}$. Finally, the polynomial $\widetilde{h}$ interpolates the claimed output of the lookup. While we only need to aggregate six pairs of lookups, we canonically extend the vectors to length $2^{\kappa+3}$ by arbitrarily padding the vectors with the first pair. With concatenated oracles $\widetilde{T}, \widetilde{D}$ and $\widetilde{h}$, the identities (24) and (25) for $M \in \{A, B, C\}$ can be checked by the homogenized sum-check over the hypercube of dimension $\max(\mu + 1, \kappa + 3)$.

5.4 Samaritan

Compiling LogSpartan PIOP (Lemma 5.2) using SamaritanPCS (Theorem 4.2), we obtain Samaritan: a SNARK with linear proving cost, logarithmic proof-size and verification. Let $\mathbf{a}_\nu \in \mathbb{F}^\nu$ be the verifier's challenges in the sub-protocol in Step 11 of Figure 3, involving sum-check over the hypercube B_ν, with $\nu = \max(\mu + 1, \kappa + 1)$. We assume that standard techniques of using random linear combination to batch several sum-check instances over the common hypercube to a single instance is used to reduce sum-check instances. Then, the verifier needs evaluation proofs for the following oracle queries. We omit oracle queries for polynomials that the verifier can compute in $O(\log n)$ $\mathbb{F}$-operations.

- Polynomials From Setup: $\{\widetilde{\mathsf{val}}_M(\mathbf{a}_\nu), \widetilde{\mathsf{R}}_M(\mathbf{a}_\nu), \widetilde{\mathsf{C}}_M(\mathbf{a}_\nu)\}$, $M \in \{A, B, C\}$.
- Round 1: $\widetilde{z}(\mathbf{r}_y)$.

- Round 2: $\{\widetilde{\chi}_M(\mathbf{a}_\nu), \widetilde{f}_M(\mathbf{a}_\nu), \widetilde{g}_M(\mathbf{a}_\nu)\}$, $M \in \{A, B, C\}$.
- Round 3: $\widetilde{s}(\mathbf{a}_\nu)$.

Efficiency. The prover complexity is dominated by multilinear polynomial commitments. We commit to polynomials $\{\widetilde{\chi}_M, \widetilde{f}_M, \widetilde{g}_M\}$ for $M \in \{A, B, C\}$ in addition to the witness polynomial $\widetilde{z}$ and the polynomial $\widetilde{s}$ for the zero-check. Of these, the polynomials $\widetilde{\chi}_M$, $M \in \{A, B, C\}$ and witness polynomial $\widetilde{z}$ have "short" coefficients (typically less than 32 bits), while the remaining polynomials have arbitrary coefficients. Thus the prover effort is dominated by $8n$ multi-exponentiations involving arbitrary elements and $7n$ multi-exponentiations with short elements (where $n = 2^\mu$ and $\kappa = \mu$). The proof size is dominated by the three sum-checks of degrees 2, 3 and 6 respectively resulting in $2\log n + 3\log n + 6\log n = 11\log n$ $\mathbb{F}$-elements. The verification complexity is $O(\log n)$ $\mathbb{F}$-operations in addition to $O(1)$ group operations and a pairing check.

Theorem 5.1. *Let* mPC = (Setup, Com, Open, Prove, Verify) *be a multilinear polynomial commitment scheme in Section 4.3. Then the SNARK obtained by compiling the PIOP in Figure 3 using* mPC *satisfies:*

$$\begin{array}{lcl} \textit{Prover time } (\mathrm{t}_{\mathcal{P}}) & = & 10m \text{ v-MSM} + 7m \text{ v-MSM (short)} + O(n)\,\mathbb{F} \\ \textit{Proof size } (|\pi|) & = & (11\log m + 8)\,\mathbb{F} + 25\,\mathbb{G}_1 \\ \textit{Verifier time } (\mathrm{t}_{\mathcal{V}}) & = & O(\log m)\,\mathbb{F} + O(1)\,\mathbb{G}_1 + O(1)\,\mathbb{P} \end{array}$$

In the above, we have $m = \max(n, K)$, where K is an upper bound on the number of non-zero entries in matrices A, B and C.

The prover cost in Theorem 5.1 additionally accounts for $2m$ v-MSMs incurred by SamaritanPCS to answer two oracle queries.

For $n, K = 2^\mu$ for BLS12-381 curve, where $\mathbb{F}$-elements are 32 bytes and $\mathbb{G}_1$-elements are 48 bytes, our concrete proof size is given by $352\mu + 1456$ bytes, which is 8.3 KB for $\mu = 20$.

Efficiency with Alternate PIOP. We also obtain an alternative version of Samaritan by combining the alternative PIOP in Section 5.3 with SamaritanPCS (Section 4.3). This alternative version has the following efficiency parameters. The prover effort is dominated by commitments to oracles $\{[\![f_M]\!], [\![g_M]\!]\}$ for $M \in \{A, B, C\}$ of size n each, multiplicity oracle $[\![\chi]\!]$ of size $2n$, and an oracle of size $8n$ to interpolate the rational function in Equation 15 (which incurs $6n$ multi-exponentiations due to first and last $2n$ positions being identical). In addition, the prover computes commitment to the witness using n multi-exponentiations and incurs $2n$ multi-exponentiations in proving oracle queries. Thus, in total the prover incurs $17n$ multi-exponentiations out of which $3n$ are for short exponents. The proof size is reduced compared to the construction in Section 5.2, as the zero-check

PIOP for implementing the homogenized sum-check in Equation 15 results in a sum-check of degree 3, instead of that of degree 6. This reduces proof-size from $11 \log n + O(1)$ to $8 \log n + O(1)$. Concrete proof size for $n = 2^{20}$ realized over the BLS12-381 curve is around 6.2 KB.

Comparison with Other SNARKs. Recent work MicroSpartan [ZSCZ24] reports proof sizes around 6.1 KB (over BN254 curve). For the Plonkish constraint systems, the proof size for HyperPlonk + PST is $224\mu + 1168$ bytes which is 5.5 KB for $\mu = 20$, for HyperPlonk + KZG + Gemini it is $288\mu + 1168$ bytes which works out to 6.9 KB. Finally, compiling the HyperPlonk PIOP [CBBZ23] with SamaritanPCS, we obtain a SNARK with proof size $\approx$ 5 KB. We emphasize that the above benchmarks are only meant for broad reference, as Samaritan and HyperPlonk target different constraint systems.

Extension to Zero-Knowledge SNARKs. Though we focus on SNARKs in this paper, transformations from existing works [CHM$^+$20, CBBZ23] can be applied to obtain a zero-knowledge version of Samaritan. In particular, Lemma 3.3 can be extended to account for zero-knowledge as follows:

Lemma 5.3 ([CHM$^+$20, BFS20, CBBZ23])**.** *Let $\mathcal{R}$ be a relation over $\mathbb{F}$. Let $PIOP = (I, P, V)$ be a zero-knowledge PIOP over $\mathbb{F}$ for $\mathcal{R}$ with negligible soundness error, and $\mathsf{PC} = (\mathsf{Setup}, \mathsf{Com}, \mathsf{Open}, \mathsf{Eval})$ be a polynomial commitment scheme over $\mathbb{F}$ that satisfies completeness, binding, extractability with negligible extraction error, hiding and a zero-knowledge evaluation protocol. Then there exists a compiler that compiles the zkPIOP using PC to obtain a public-coin zero-knowledge argument of knowledge $\Pi = (\mathsf{Setup}, \mathcal{P}, \mathcal{V})$ for $\mathcal{R}$ with negligible knowledge error. If the PIOP is holographic, the argument system is a preprocessing argument system.*

At a high level, the transformation consists of two steps: (i) apply a compiler that transforms multivariate PIOPs into ones that are zero knowledge [CBBZ23, XZZ$^+$19] by masking the oracle polynomials (ii) compile the zero-knowledge PIOP using a PCS that satisfies hiding and zero-knowledge evaluation (in addition to completeness, binding and extraction). By instantiating our univariate-to-multilinear transformation with a univariate PCS that is hiding and ZK [CHM$^+$20], we obtain a multilinear PCS with hiding and ZK. Standard techniques [CBBZ23, XZZ$^+$19] can be applied to obtain a zero-knowledge version of our PIOP. Like in [CBBZ23], the queries to the PIOP has to be restricted to ensure that there exists at least one dimension where each query point has a distinct value. Given zkPIOP together with a hiding and zkPCS, we have both ingredients necessary to invoke Lemma 5.3, and obtain a zkSNARK.

References

[BBB+18] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *2018 IEEE Symposium on Security and Privacy*, pages 315–334. IEEE Computer Society Press, May 2018.

[BCCT12] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, pages 326–349, 2012.

[BCF+25] Martijn Brehm, Binyi Chen, Ben Fisch, Nicolas Resch, Ron D. Rothblum, and Hadas Zeilberger. Blaze: Fast SNARKs from interleaved RAA codes. In Serge Fehr and Pierre-Alain Fouque, editors, *EUROCRYPT 2025, Part IV*, volume 15604 of *LNCS*, pages 123–152. Springer, Cham, May 2025.

[BCG+13] Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, Eran Tromer, and Madars Virza. SNARKs for C: Verifying program executions succinctly and in zero knowledge. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 90–108. Springer, Berlin, Heidelberg, August 2013.

[BCHO22] Jonathan Bootle, Alessandro Chiesa, Yuncong Hu, and Michele Orrù. Gemini: Elastic SNARKs for diverse environments. In Orr Dunkelman and Stefan Dziembowski, editors, *EUROCRYPT 2022, Part II*, volume 13276 of *LNCS*, pages 427–457. Springer, Cham, May / June 2022.

[BCI+13] Nir Bitansky, Alessandro Chiesa, Yuval Ishai, Rafail Ostrovsky, and Omer Paneth. Succinct non-interactive arguments via linear interactive proofs. In Amit Sahai, editor, *TCC 2013*, volume 7785 of *LNCS*, pages 315–333. Springer, Berlin, Heidelberg, March 2013.

[BCTV14] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. Succinct non-interactive zero knowledge for a von neumann architecture. In Kevin Fu and Jaeyeon Jung, editors, *USENIX Security 2014*, pages 781–796. USENIX Association, August 2014.

[BFS20] Benedikt Bünz, Ben Fisch, and Alan Szepieniec. Transparent SNARKs from DARK compilers. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part I*, volume 12105 of *LNCS*, pages 677–706. Springer, Cham, May 2020.

[BSS08] Eli Ben-Sasson and Madhu Sudan. Short pcps with polylog query complexity. *SIAM Journal on Computing*, 38(2):551–607, 2008.

[CBBZ23] Binyi Chen, Benedikt Bünz, Dan Boneh, and Zhenfei Zhang. HyperPlonk: Plonk with linear-time prover and high-degree custom gates. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part II*, volume 14005 of *LNCS*, pages 499–530. Springer, Cham, April 2023.

[CGG+23] Matteo Campanelli, Nicolas Gailly, Rosario Gennaro, Philipp Jovanovic, Mara Mihali, and Justin Thaler. Testudo: Linear time prover SNARKs with constant size proofs and square root size universal setup. In Abdelrahaman Aly and Mehdi Tibouchi, editors, *LATINCRYPT 2023*, volume 14168 of *LNCS*, pages 331–351. Springer, Cham, October 2023.

[CHM+20] Alessandro Chiesa, Yuncong Hu, Mary Maller, Pratyush Mishra, Psi Vesely, and Nicholas P. Ward. Marlin: Preprocessing zkSNARKs with universal and updatable SRS. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part I*, volume 12105 of *LNCS*, pages 738–768. Springer, Cham, May 2020.

[CMNW24] Valerio Cini, Giulio Malavolta, Ngoc Khanh Nguyen, and Hoeteck Wee. Polynomial commitments from lattices: Post-quantum security, fast verification and transparent setup. In Leonid Reyzin and Douglas Stebila, editors, *CRYPTO 2024, Part X*, volume 14929 of *LNCS*, pages 207–242. Springer, Cham, August 2024.

[EFG22] Liam Eagen, Dario Fiore, and Ariel Gabizon. cq: Cached quotients for fast lookups. Cryptology ePrint Archive, Report 2022/1763, 2022.

[EG25] Liam Eagen and Ariel Gabizon. MERCURY: A multilinear polynomial commitment scheme with constant proof size and no prover FFTs. Cryptology ePrint Archive, Paper 2025/385, 2025.

[FKL18] Georg Fuchsbauer, Eike Kiltz, and Julian Loss. The algebraic group model and its applications. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part II*, volume 10992 of *LNCS*, pages 33–62. Springer, Cham, August 2018.

[FS87] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *CRYPTO'86*, volume 263 of *LNCS*, pages 186–194. Springer, Berlin, Heidelberg, August 1987.

[GGPR13] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct NIZKs without PCPs. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 626–645. Springer, Berlin, Heidelberg, May 2013.

[GLS+23] Alexander Golovnev, Jonathan Lee, Srinath T. V. Setty, Justin Thaler, and Riad S. Wahby. Brakedown: Linear-time and field-agnostic SNARKs for R1CS. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part II*, volume 14082 of *LNCS*, pages 193–226. Springer, Cham, August 2023.

[GNS24] Chaya Ganesh, Vineet Nair, and Ashish Sharma. Dual polynomial commitment schemes and applications to commit-and-prove SNARKs. In Bo Luo, Xiaojing Liao, Jun Xu, Engin Kirda, and David Lie, editors, *ACM CCS 2024*, pages 884–898. ACM Press, October 2024.

[Gro10] Jens Groth. Short pairing-based non-interactive zero-knowledge arguments. In Masayuki Abe, editor, *ASIACRYPT 2010*, volume 6477 of *LNCS*, pages 321–340. Springer, Berlin, Heidelberg, December 2010.

[Gro16] Jens Groth. On the size of pairing-based non-interactive arguments. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 305–326. Springer, Berlin, Heidelberg, May 2016.

[GWC19] Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru. PLONK: Permutations over Lagrange-bases for oecumenical noninteractive arguments of knowledge. Cryptology ePrint Archive, Report 2019/953, 2019.

[Hab22a] Ulrich Haböck. Multivariate lookups based on logarithmic derivatives. Cryptology ePrint Archive, Report 2022/1530, 2022.

[Hab22b] Ulrich Haböck. A summary on the FRI low degree test. Cryptology ePrint Archive, Report 2022/1216, 2022.

[HMW+25] Yuncong Hu, Pratyush Mishra, Xiao Wang, Jie Xie, Kang Yang, Yu Yu, and Yuwen Zhang. DFS: Delegation-friendly zkSNARK and private delegation of provers. USENIX SEcurity 2025 (to appear), 2025.

[Kil92] Joe Kilian. A note on efficient zero-knowledge proofs and arguments. In *Proceedings of the twenty-fourth annual ACM symposium on Theory of computing*, pages 723–732, 1992.

[KT24] Tohru Kohrita and Patrick Towa. Zeromorph: Zero-knowledge multilinear-evaluation proofs from homomorphic univariate commitments. *J. Cryptol.*, 37(4):38, 2024.

[KZG10] Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. Constant-size commitments to polynomials and their applications. In Masayuki Abe, editor, *ASIACRYPT 2010*, volume 6477 of *LNCS*, pages 177–194. Springer, Berlin, Heidelberg, December 2010.

[Lab17] Protocol Labs. Filecoin: A decentralized storage network. `https://filecoin.io/filecoin.pdf`, 2017.

[Lee21] Jonathan Lee. Dory: Efficient, transparent arguments for generalised inner products and polynomial commitments. In Kobbi Nissim and Brent Waters, editors, *TCC 2021, Part II*, volume 13043 of *LNCS*, pages 1–34. Springer, Cham, November 2021.

[LFKN90] Carsten Lund, Lance Fortnow, Howard J. Karloff, and Noam Nisan. Algebraic methods for interactive proof systems. In *31st FOCS*, pages 2–10. IEEE Computer Society Press, October 1990.

[Lip12] Helger Lipmaa. Progression-free sets and sublinear pairing-based non-interactive zero-knowledge arguments. In Ronald Cramer, editor, *TCC 2012*, volume 7194 of *LNCS*, pages 169–189. Springer, Berlin, Heidelberg, March 2012.

[Lip13] Helger Lipmaa. Succinct non-interactive zero knowledge arguments from span programs and linear error-correcting codes. In Kazue Sako and Palash Sarkar, editors, *ASIACRYPT 2013, Part I*, volume 8269 of *LNCS*, pages 41–60. Springer, Berlin, Heidelberg, December 2013.

[MBKM19] Mary Maller, Sean Bowe, Markulf Kohlweiss, and Sarah Meiklejohn. Sonic: Zero-knowledge SNARKs from linear-size universal and updatable structured reference strings. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 2111–2128. ACM Press, November 2019.

[Mic94] Silvio Micali. Cs proofs. In *Proceedings 35th Annual Symposium on Foundations of Computer Science*, pages 436–453. IEEE, 1994.

[NS24] Ngoc Khanh Nguyen and Gregor Seiler. Greyhound: Fast polynomial commitments from lattices. In Leonid Reyzin and Douglas Stebila, editors, *CRYPTO 2024, Part X*, volume 14929 of *LNCS*, pages 243–275. Springer, Cham, August 2024.

[PH23] Shahar Papini and Ulrich Haböck. Improving logarithmic derivative lookups using GKR. Cryptology ePrint Archive, Report 2023/1284, 2023.

[PHGR13] Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. Pinocchio: Nearly practical verifiable computation. In *2013 IEEE Symposium on Security and Privacy*, pages 238–252. IEEE Computer Society Press, May 2013.

[PST13] Charalampos Papamanthou, Elaine Shi, and Roberto Tamassia. Signatures of correct computation. In Amit Sahai, editor, *TCC 2013*, volume 7785 of *LNCS*, pages 222–242. Springer, Berlin, Heidelberg, March 2013.

[rol21] An incomplete guide to rollups. `https://vitalik.ca/general/2021/01/05/rollup.html`, 2021.

[Set20] Srinath Setty. Spartan: Efficient and general-purpose zkSNARKs without trusted setup. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part III*, volume 12172 of *LNCS*, pages 704–737. Springer, Cham, August 2020.

[SL20] Srinath Setty and Jonathan Lee. Quarks: Quadruple-efficient transparent zkSNARKs. Cryptology ePrint Archive, Report 2020/1275, 2020.

[ST25] Srinath Setty and Justin Thaler. Twist and shout: Faster memory checking arguments via one-hot addressing and increments. Cryptology ePrint Archive, Paper 2025/105, 2025.

[WTs+18] Riad S. Wahby, Ioanna Tzialla, abhi shelat, Justin Thaler, and Michael Walfish. Doubly-efficient zkSNARKs without trusted setup. In *2018 IEEE Symposium on Security and Privacy*, pages 926–943. IEEE Computer Society Press, May 2018.

[XZS22] Tiancheng Xie, Yupeng Zhang, and Dawn Song. Orion: Zero knowledge proof with linear prover time. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part IV*, volume 13510 of *LNCS*, pages 299–328. Springer, Cham, August 2022.

[XZZ+19] Tiancheng Xie, Jiaheng Zhang, Yupeng Zhang, Charalampos Papamanthou, and Dawn Song. Libra: Succinct zero-knowledge proofs with optimal prover computation. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part III*, volume 11694 of *LNCS*, pages 733–764. Springer, Cham, August 2019.

[ZBK+22] Arantxa Zapico, Vitalik Buterin, Dmitry Khovratovich, Mary Maller, Anca Nitulescu, and Mark Simkin. Caulk: Lookup arguments in sublinear time. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *ACM CCS 2022*, pages 3121–3134. ACM Press, November 2022.

[ZSCZ24] Jiaxing Zhao, Srinath Setty, Weidong Cui, and Greg Zaverucha. MicroNova: Folding-based arguments with efficient (on-chain) verification. IEEE S&P 2025 (to appear), 2024.

[ZXZS20] Jiaheng Zhang, Tiancheng Xie, Yupeng Zhang, and Dawn Song. Transparent polynomial delegation and its applications to zero knowledge proof. In *2020 IEEE Symposium on Security and Privacy*, pages 859–876. IEEE Computer Society Press, May 2020.