HybridPlonk: SubLogarithmic Linear Time SNARKs from Improved Sumcheck

Sikhar Patranabis[1], Nitin Singh[2], and Sayani Sinha[3]

[1,2]IBM Research India
[3]IIT Kharagpur, India

Abstract

We present HybridPlonk – the first SNARK that simultaneously achieves linear-time prover, sublogarithmic proof size, provable security in the random oracle model, and also features an updatable setup. As a core technical contribution (possibly of independent interest), we reduce the communication complexity of the classical sumcheck protocol for multivariate polynomials from logarithmic to sublogarithmic, while retaining linear prover complexity. For degree d multivariate polynomials in μ variables which can be decomposed into ℓ multilinear polynomials, our protocol achieves $O(\ell + d\log(\log n))$ communication and $O(n)$ prover cost for $n = 2^\mu$. Our protocol leverages recently proposed multilinear polynomial commitment schemes (PCS) with linear-time prover and constant proof size.

Multivariate sumcheck is a key ingredient in the design of several prover-efficient SNARKs, such as HyperPlonk (Eurocrypt'23), Spartan (Crypto'20), Hyrax (S&P'18), Libra (Crypto'19), Gemini (Eurocrypt'22), Virgo (S&P'20) etc. All of these SNARKs incur $\Omega(\log n)$ proof size, with the smallest concrete proof sizes ranging from 5KB-10KB for circuits of size 2^{20}-2^{30}.

We compile a variant of HyperPlonk multilinear PIOP with our improved sumcheck to realize HybridPlonk. HybridPlonk achieves $O(n)$ prover, $O(\log\log n)$ proof size, and $O(\log n)$ verifier, while avoiding proof recursion and non-black-box use of cryptographic primitives. We implement HybridPlonk to show that it is efficient in practice. We compare HybridPlonk's performance with several state-of-the-art prover-efficient SNARKs. For circuits of sizes of upto 2^{30}, HybridPlonk achieves a proof size of ≈ 2.3 KB, which is $2.5 - 4\times$ smaller than the most compact prover-efficient SNARKs, while retaining comparable prover costs.

Contents

1 Introduction

A Succinct Non-interactive ARgument of Knowledge (SNARK) allows a prover to convince a verifier about the integrity of a computation such that the size of the proof and the work done by the verifier in checking the proof is much smaller than the size of the computation. Concretely, for an NP relation $\mathcal{R}$, a SNARK proof π corresponding to a statement $\mathbb{x}$ convinces a verifier (non-interactively) about the existence of witness $\mathbb{w}$ such that $\mathcal{R}(\mathbb{x}, \mathbb{w}) = 1$. Succinctness requires that the size of π and the verification complexity are both $O_\lambda(\text{polylog}(|\mathbb{x}| + |C|))$, where $|\mathbb{x}|$ is the size of the statement $\mathbb{x}$, $|C|$ is the number of gates in the circuit C that verifies the relation $\mathcal{R}(\mathbb{x}, \mathbb{w}) = 1$, and λ is the security parameter [1]. While initially introduced as theoretical concepts in the seminal contributions of Micali [Mic94] and Kilian [Kil92], SNARKs have since been refined in a long line of works [Gro10, Lip12, BCCT12, BCI$^+$13, GGPR13, PHGR13, BCG$^+$13, Lip13, BCTV14] towards practical adoption. More recently, SNARKs have seen real-world deployment in several niche applications such as decentralized payment systems like Zcash [BCG$^+$14] and Monero [NMT], decentralized file storage systems like Filecoin [Lab17], blockchain scaling solutions such as ZkRollups [rol21], using existing credentials for authentication on blockchain [BCJ$^+$24], and ensuring privacy and integrity of machine learning applications [LVA$^+$25, LKKO20, ZFZS20, SDP22].

Models for SNARKs. Efficient SNARKs typically require a Structured Reference String (SRS) to be generated at setup. Setup for SNARKs in the SRS model can be broadly classified into three categories: (i) *transparent* (setup only uses public randomness and verifier randomness consists of only *public coins*), (ii) *trusted* (require a randomized, circuit-dependent preprocessing phase with secret random coins), and (iii) *universal and updatable* (a *one-time* setup can be used to prove statements about any computation; moreover, there is a mechanism to update by contributing to the randomness of the SRS, and the SRS is trusted as long as at least one of the updates is honest). While the first practical SNARKs were in the trusted SRS model [PHGR13, GGPR13], several recent (pairing-based) SNARKs are in the updatable SRS model [MBKM19, GWC19, CHM$^+$20, Set20, BCHO22, CBBZ23, XZZ$^+$19, ZSCZ25, GPS25], which is also the model that we use for our construction in this paper.

SNARKs from PIOP and PCS. A common design methodology underlying several modern SNARKs [WTs$^+$18, GWC19, XZZ$^+$19, CHM$^+$20, Set20, SL20, BCHO22, XZS22, CBBZ23, GLS$^+$23] is to compile an information-theoretically secure Polynomial Interactive Oracle Proof (PIOP) using a cryptographic tool called a Polynomial Commitment Scheme (PCS). In a PIOP, the prover provides the verifier with oracle access to a set of polynomials, and the verifier queries these polynomial oracles at random challenge points to obtain evaluations, eventually outputting accept or reject. Compiling a PIOP using a PCS yields a public-coin succinct argument, where the polynomial oracles are realized via *commitments* to polynomials that a verifier can query for evaluations, together with proofs that the evaluations are indeed consistent with the commitments. The interactive argument is compiled using Fiat-Shamir [FS87] to obtain a SNARK in the ROM.

Univariate and Multilinear PCS. SNARKs based on the above approach can be broadly classified into two categories: those realized by compiling univariate PIOPs using univariate PCS (such as Sonic [MBKM19], Marlin [CHM$^+$20], Plonk [GWC19], and Pari [DMS24]), and those realized

[1] Throughout the paper, all of our asymptotics are stated for growing instance size, and have implicit dependence on a fixed security parameter λ. Hence, we simply write $O(\cdot)$ instead of $O_\lambda(\cdot)$ for simplicity of exposition.

by compiling multilinear PIOPs using multilinear PCS (such as Hyrax [WTs$^+$18], Libra [XZZ$^+$19], Spartan [Set20], Kopis [SL20], Xiphos [SL20], Gemini [BCHO22], Orion [XZS22], Brakedown [GLS$^+$23] and HyperPlonk [CBBZ23]). Univariate PCS-based SNARKs achieve constant proof size and constant verification complexity. However, their prover time 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). This becomes a bottleneck for applications requiring large prover computations [rol21, Lab17], and has sparked increased interest in designing *prover-efficient* SNARKs.

Prover-Efficient SNARKs. SNARKs based on multilinear PIOPs and multilinear PCS [WTs$^+$18, XZZ$^+$19, Set20, SL20, CBBZ23, BCHO22, XZS22, GLS$^+$23] are prover-efficient in the sense that they achieve $O(n)$ prover. However, they incur $O(\log n)$ proof size. With growing computation sizes reaching $\approx 2^{30}$, even the best known constructions incur concrete proof sizes $\approx$ 10 KB[2]. In comparison, Plonk [GWC19] achieves constant proof size of 0.61 KB. Testudo [CGG$^+$23] achieves constant proof size by recursively composing sumcheck verification and inner pairing-product verification with a Groth16 [Gro16] SNARK. 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. In this paper, we ask the following question:

Can we design a SNARK in the ROM with linear-time prover and sublogarithmic proof size?

Multivariate Sumcheck. Several prominent prover-efficient SNARKs [WTs$^+$18, XZZ$^+$19, Set20, ZXZS20, BCHO22, CBBZ23] are designed using PIOPs for *multivariate sumcheck*. These schemes inherit their logarithmic proof size from the logarithmic communication complexity of the classical sumcheck protocol for multivariate polynomials [LFKN90]. Consequently, though recent works [GPS25, EG25] have proposed multilinear PCS with linear-time prover and constant proof size [3], they cannot be used to compile multivariate sumcheck PIOPs to yield prover-efficient SNARKs with sublogarithmic proof size. The communication complexity of multivariate sumcheck is thus a major barrier towards achieving prover-efficient SNARKs with sublogarithmic proof size. This leads to the following question:

Can we design a multivariate sumcheck protocol with linear-time prover and sublogarithmic communication?

1.1 Our Contributions

We answer both of the above questions in the affirmative. We exhibit a new multivariate sumcheck protocol with linear-time prover and sublogarithmic communication. We then leverage this protocol to design HybridPlonk – the first SNARK in the updatable SRS setting that simultaneously achieves $O(n)$ prover time, sublogarithmic proof size of $O(\log(\log n))$, and $O(\log n)$ verifier, while also being provably secure in the ROM. We expand on our contributions below.

[2]Throughout this paper, we report proof sizes assuming a BLS12-381 curve-based implementation, unless specified otherwise.

[3]We consider multiexponentiation of size n to be $O(n)$ work, with implicit dependence on security parameter λ.

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]		$O(\log n)\,\mathbb{G}_T$	$O(\log n)\,\mathbb{G}_T$	24KB	
Kopis [SL20]		$O(\sqrt{n})\,\mathbb{G}_T$	$O(\log n)\,\mathbb{G}_T$	39KB	Transparent
Xiphos [SL20]		$O(\log n)\,\mathbb{G}_T$	$O(\log n)\,\mathbb{G}_T$	61KB	
Garuda [DMS24]		$O(\log n)\,\mathbb{P}$	$O(\log n)\,\mathbb{G}_1$	$\approx$ 10KB	Trusted
Spartan [Set20] + KZG [KZG10]		$O(\log^2 n)\,\mathbb{F}$, $O(\log n)\,\mathbb{G}_1$	$O(\log^2 n)\,\mathbb{F}$	50KB	
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 + HyperKZG [BCHO22]	$O(n)\,\mathbb{G}_1$	$O(\log n \log(\log n))\,\mathbb{F}$, $O(\log n)\,\mathbb{G}_1$	$O(\log n)\,\mathbb{G}_1$	6.1KB	
Libra [XZZ+19]		$O(d \log n)\,\mathbb{F}$, $O(\log n)\,\mathbb{G}_1$	$O(d \log n)\,\mathbb{G}_1$	–	Updatable
MicroSpartan [ZSCZ25]		$O(\log n)\,\mathbb{G}_1$	$O(\log n)\,\mathbb{G}_1$	6–7KB	
Samaritan [GPS25]		$O(\log n)\,\mathbb{F}$, $O(1)\,\mathbb{G}_1$	$O(\log n)\,\mathbb{F}$, $O(1)\,\mathbb{G}_1$	6.2KB	
HyperPlonk + Samaritan PCS [GPS25]		$O(\log n \log(\log n))\,\mathbb{F}$, $O(\log n)\,\mathbb{G}_1$	$O(\log n)\,\mathbb{F}$, $O(1)\,\mathbb{G}_1$	5KB	
HybridPlonk (this work)		$O(\log n)\,\mathbb{F}$, $O(\log n)\,\mathbb{G}_1$	$O(\log(\log n))\,\mathbb{F}$, $O(1)\,\mathbb{G}_1$	2.3KB	

Improved Multivariate Sumcheck. We leverage two recent works [GPS25, EG25] that have proposed multilinear polynomial commitment schemes to reduce the communication complexity of the classical sumcheck protocol for multivariate polynomials [LFKN90]. For degree d multivariate polynomials in μ variables which can be decomposed into ℓ multilinear polynomials, we exhibit a new multivariate sumcheck protocol with $O(\ell + d \log(\log n))$ communication for $n = 2^{\mu}$. Our protocol retains the $O(n)$ prover cost (where the precise constant depends on ℓ, d and the multivariate form). Thus we improve on the $O(\log n)$ communication inherent in all existing applications of the multivariate sumcheck protocol.

HybridPlonk. We present an optimized version of the HyperPlonk multilinear PIOP [CBBZ23] using our new multivariate sumcheck protocol. We choose HyperPlonk because it can be expressed entirely as a multilinear PIOP, and has the smallest concrete proof size among all prover-efficient SNARKs. We compile this optimized HyperPlonk PIOP using Samaritan PCS from [GPS25], which has the smallest concrete proof size among all multilinear PCS. The resulting SNARK, which we call HybridPlonk, is in the updatable SRS setting, and simultaneously achieves $O(n)$ prover time, sublogarithmic proof size of $O(\log(\log n))$, and $O(\log n)$ verifier. HybridPlonk also avoids proof recursion techniques and non-black-box usage of cryptographic primitives, and can be proven secure in the ROM. The concrete proof size of HybridPlonk is ≈ 2.3KB for circuit sizes up to 2^{30}, which is $2.5 - 4\times$ smaller than the concrete proof size for existing state of the art prover-efficient SNARKs.

We note that SNARKs with smaller proof size than HybridPlonk are based on univariate PCS, and are not prover-efficient as they inherently incur $O(n \log n)$ prover cost due to polynomial multiplications. We present a detailed comparison of HybridPlonk with existing SNARKs in Section 1.2 and Table 1.

Evaluation. We contribute performant implementations for Samaritan PCS from [GPS25] (the original paper does not report an implementation), as well as our scheme HybridPlonk, built on top of the popular Arkworks [ac22] ecosystem. Our implementation is anonymously available at: `https://anonymous.4open.science/r/HybridPlonk-F011`. In future, we aim to make these implementations available as part of the Arkworks cryptographic suite. We provide a thorough empirical evaluation of HybridPlonk and compare it with relevant recent works in Section 7.

1.2 Comparison with Related Work

Pairing-based Prover-Efficient SNARKs. Table 1 presents a detailed comparison of HybridPlonk against known pairing-based SNARKs with linear prover time. HybridPlonk is the only one with sublogarithmic proof size, and also achieves the smallest concrete proof size. We note that Dory [Lee21], Kopis [SL20], and Xiphos [SL20] support transparent setup (but incur significantly larger proof size and verifier overheads compared to HybridPlonk), while Garuda [DMS24] requires a trusted, circuit-dependent setup (unlike HybridPlonk, where setup is circuit-independent). All of the other SNARKs in Table 1 are in the updatable setup setting (same as HybridPlonk). See Section 7 for a detailed comparison of the concrete performance of these SNARKs with that of HybridPlonk.

Non-Pairing-based Prover-Efficient SNARKs. Several recent works have proposed (plausibly post-quantum) SNARKs with transparent setup and linear prover from hash-based assumptions (e.g., FRI [BBHR18], Orion [XZS22], Brakedown [GLS$^+$23], BaseFold [ZCF24], STIR [ACFY24], WHIR [ACFY25], and Blaze [BCF$^+$25b]) and lattice-based assumptions (e.g., LaBRADOR [BS23], Greyhound [NS24] and [CMNW24]). Asymptotically, all of these constructions incur superlogarithmic proof sizes and verification overheads. In comparison, HybridPlonk incurs sublogarithmic proof size and logarithmic verification overhead (but is not post-quantum due to its reliance on pairings). The concrete proof sizes of known hash-based SNARKs [BBHR18, XZS22, GLS$^+$23, ZCF24, ACFY24, ACFY25, BCF$^+$25b] are greater than 200KB for $n = 2^{20}$ (two orders of magnitude larger than HybridPlonk's proof size of 2.2KB). The concrete proof sizes of state-of-the-art lattice-based SNARKs [BS23, NS24, CMNW24] are around 50KB for $n = 2^{20}$ (nearly 22$\times$ larger than HybridPlonk), but more crucially, their verifier overhead is $O(\sqrt{n})$, which is substantially larger (both asymptotically and concretely) than that of HybridPlonk.

SNARKs with SubLogarithmic Proof Size. Several early SNARK constructions [Gro10, Lip12, BCCT12, BCI$^+$13, GGPR13, PHGR13, BCG$^+$13, Lip13, BCTV14] have constant proof size and verification complexity. These are realized via linear PCPs compiled using cryptographic tools such as linear-only encodings (e.g., exponentiation in a bilinear group). All of these SNARKs incur at least $O(n \log n)$ prover time. More recently, SNARKs with constant proof size and verification complexity have been achieved by compiling univariate PIOPs using univariate PCS. Some prominent examples are Sonic [MBKM19], Marlin [CHM$^+$20], Plonk [GWC19], and Pari [DMS24]. These incur $O(n \log n)$ prover time. HybridPlonk belongs to an alternative family of SNARKs that trade off proof size (and verification time) for more efficient $O(n)$ proof generation. In particular, while the aforementioned SNARKs have concretely smaller proof sizes and verification time than HybridPlonk, they incur significantly larger prover overheads (see Table 3 for a concrete comparison of the prover time between Plonk and HybridPlonk).

Recent Improvements to Sumcheck. A recent work [LMN25] proposed an $O(\log(\log n))$-round multivariate sumcheck PIOP with $O(n)$ prover work. However, they use Zeromorph [KT24] PCS to compile their PIOP into an argument, which results in a logarithmic sized argument. They do not report compilation to a generic SNARK. Since, the PIOP in [LMN25] involves multivariate polynomials of degree > 1, it is not immediately clear if the recent multilinear PCS such as Samaritan [GPS25] or Mercury [EG25] can be used to compile their PIOP to an argument. Concurrent to our work, two recent works [ARZR25, BCF^{+}25a] propose $O(\log^* n)$-round [4] multilinear sumcheck PIOPs with $O(n)$ prover work. While [ARZR25] works specifically over binary fields, [BCF^{+}25a] generalizes to fields of large prime order. However, these works do not describe a generic SNARK constructions based on the corresponding sumcheck PIOPs. In particular, the authors of [BCF^{+}25a] only briefly sketch how one could compile their sumcheck protocol for products of multilinear polynomials using PST [PST13]-like commitments with a *trusted setup* (instead of updatable) to achieve an argument system with $O(\log^* n)$ proof size and $O(n)$ prover. They also do not discuss generic SNARKs beyond sumcheck involving product of two multilinear polynomials. Achieving a practical SNARK using their techniques is an interesting future work.

2 Preliminaries

We present preliminary background material in this section.

Notation. We denote the set of integers $\{1, \dots, n\}$ 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 $\mathbf{y} = (y_1, \dots, y_\mu) \in B_\mu$, 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 $\langle\, \cdot \,\rangle_\mu : [\,2^\mu\,] \to B_\mu$ are inverses of each other. We will drop the subscript μ when it is clear from the context.

2.1 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.

[4]The concrete value of both $\log^* n$ and $\log(\log n)$ (rounded to the nearest integer) is 5 for $n = 2^{32}$.

Definition 2.1 (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 2.2 (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 $\mathsf{poly}(\lambda, \log |w|)$.

2.2 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 public parameters pp, commitment C, degree d, evaluation point x, and claimed evaluation v as common input. 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 or outputs 0 indicating rejecting the proof.

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

Definition 2.3 (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 $(x_1, \ldots, x_n) \in \mathbb{F}^n$,*

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

Definition 2.4 (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 2.5 (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:*

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

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.

2.3 The KZG PCS

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

Definition 2.6 (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* $\mathsf{BG}(1^\lambda)$*.*

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

Definition 2.7 (q-DLOG Assumption)**.** *The q-DLOG assumption with respect to* $\mathcal{G}$ *holds if for all* λ *and for all PPT* $\mathcal{A}$*, the following probability is negligible in* λ*:*

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

KZG is shown to be evaluation binding assuming the hardness of q-DLOG (Definition 2.7) 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).

2.4 Polynomial IOP

A modular approach for designing efficient succinct arguments is to (i) first, construct an Interactive Oracle Proof (IOP) that is an information-theoretic protocol in an idealized model, (ii) compile the information-theoretic protocol via a cryptographic compiler to obtain a cryptographic argument system. In a Polynomial IOP (PIOP), the prover provides oracle access to a set of polynomials, and the verifier accepts or rejects by checking certain identities over the polynomials output by the prover and possibly public polynomials known to the verifier. A PIOP is compiled into a succinct argument of knowledge by using a *polynomial commitment scheme* to realize the polynomial oracles. Many recent constructions of zkSNARKs [BFS20, CHM$^+$20, GWC19] follow this approach where the information theoretic object is a a PIOP and the cryptographic compiler is a polynomial commitment scheme.

Definition 2.8 (Polynomial IOP). *A polynomial IOP is a public-coin interactive proof for a relation $\mathcal{R} = (\mathbb{x}, \mathbb{w})$. $\mathcal{R}$ is an oracle relation such that $\mathbb{x}$ 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 2.1 ([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.*

Indexed Relations. An indexed relation $\mathcal{R}$ is a set of triples $(\mathbb{i}, \mathbb{x}, \mathbb{w})$ where $\mathbb{i}$ is the index, $\mathbb{x}$ is the instance, and $\mathbb{w}$ is the witness. Typically, an NP relation $((C, \mathbb{x}); \mathbb{w})$ given by $C(\mathbb{x}, \mathbb{w}) = 1$ for an arithmetic circuit C is expressed as an indexed relation with index $\mathbb{i}$ encoding the description of the circuit C. A PIOP for indexed relation $\mathcal{R}$ is specified by indexer $\mathcal{I}$, prover $\mathcal{P}$ and verifier $\mathcal{V}$ which operate as follows:

- *Offline Phase*: The indexer $\mathcal{I}$ receives the index $\mathbb{i}$ as input and outputs a set of polynomials encoding $\mathbb{i}$.
- *Online Phase*: The prover $\mathcal{P}(\mathbb{i}, \mathbb{x}, \mathbb{w})$ interacts with the verifier $\mathcal{V}([\![\mathbb{i}]\!], \mathbb{x})$, i.e, the verifier has oracle access to the polynomials output by $\mathcal{I}$ in addition to any oracles in the statement $\mathbb{x}$.

Lemma 2.2 ([CHM$^+$20, BFS20, CBBZ23]). *Let $\mathcal{R}$ be a relation over $\mathbb{F}$. Let PIOP $= (\mathcal{I}, \mathcal{P}, \mathcal{V})$ 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.*

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* $\mathcal{R}$*. If the PIOP is for an indexed relation, the resulting 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.

2.5 Algebraic Preliminaries

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 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 $\mathbf{x} \in \mathbb{F}^\mu$ and $\mathbf{y} \in \mathbb{F}^\mu$. The polynomials $\{\widetilde{eq}_\mu(x, \langle\, i\,\rangle) : i \in [2^\mu]\}$ are linearly independent over $\mathbb{F}$ and form the Lagrange basis polynomials for the set B_μ. For $\mathbf{x}, \mathbf{y} \in B_\mu$, $\widetilde{eq}_\mu(\mathbf{x}, \mathbf{y}) = 1$ if $\mathbf{x} = \mathbf{y}$, and is 0 otherwise. For a function $f : B_\mu \to \mathbb{F}$, the (unique) polynomial $\widetilde{f}(\mathbf{x}) = \sum_{i=1}^{2^\mu} f(\langle\, i\,\rangle)\widetilde{eq}_\mu(\mathbf{x}, \langle\, i\,\rangle)$ 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.
Multivariate Sumcheck Protocol. Let $f(X_1, \ldots, X_\mu) \in \mathbb{F}[X_1, \ldots, X_\mu]$. Consider the claim:

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

The above claim takes time $O(|B_\mu|)$ to verify. The sumcheck protocol [LFKN90] 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. We recall the classical sumcheck protocol from [LFKN90] for multilinear polynomials.

- **Common Input**: The verifier is given oracle access $[\![\widetilde{f}]\!]$ to the multilinear polynomial $\widetilde{f} \in \mathbb{F}^{\leq 1}[X_1, \ldots, X_\mu]$ and the claimed sum $v = \sum_{\mathbf{x} \in B_\mu} f(\mathbf{x})$. The prover and verifier interact in μ rounds $1, \ldots, \mu - 1$ as:
- As its i^{th} message, the prover $\mathcal{P}$ sends univariate polynomial $g_1(X_i)$ such that $g_i(X_i) = \sum_{\mathbf{x}' \in B_{\mu-i}} f(\mathbf{r}, X_i, \mathbf{x}')$, where $\mathbf{r} = (r_1, \ldots, r_{i-1})$ is the vector of challenges sent by the verifier in the previous rounds. Note that $\mathbf{r}$ is not defined for $i = 1$, and so we ignore it for $i = 1$.
- $\mathcal{V}$ checks $g_i(1) + g_i(0) = g_{i-1}(r_i)$ for $i > 1$ and $g_1(1) + g_1(0) = v$ for $i = 1$. The verifier aborts if the check fails.
- If $i < \mu$, $\mathcal{V}$ sends $r_i \leftarrow \mathbb{F}$ to the prover.
- For $i = \mu$, the verifier samples $r_\mu \leftarrow \mathbb{F}$ and checks $g_{\mu-1}(r_\mu) = \widetilde{f}(r_1, \ldots, r_\mu)$ by querying the oracle $[\![f]\!]$.

The above interactive oracle proof is turned into an interactive proof of knowledge by using a polynomial commitment scheme to commit to the polynomial $\widetilde{f}$ and using evaluation proof to answer the final oracle query. Ignoring the cost of the polynomial commitment scheme, the above protocol has communication complexity $O(\mu)$, with the prover computing $O(2^\mu)$ $\mathbb{F}$-operations.

3 Background and Overview

In this section, we review the prior work crucially relevant to our work and provide a high-level overview of our techniques.

3.1 HyperPlonk

HyperPlonk [CBBZ23] is a multilinear PIOP which is an adaptation of Plonk [GWC19], which is a popular univariate PIOP. HyperPlonk leverages sumcheck protocol to achieve efficient prover compared to Plonk. We describe the variant of HyperPlonk for gates with 3 wires, and 5 selectors. Let $\mu \in \mathbb{N}$ and let $n = 2^\mu$ be the number of gates. We specify the HyperPlonk circuit $\mathcal{C}$ as below:

- Vectors $\mathbf{q}_M, \mathbf{q}_L, \mathbf{q}_R, \mathbf{q}_O$ and $\mathbf{q}_C$ in $\mathbb{F}^n$ denote the selectors.
- Permutation $\sigma : [3n] \to [3n]$ denotes the wiring constraints.
- The vector $\mathbf{z} = (z_1, \dots, z_{3n})$ denotes the wires of the circuit. Without loss of generality, we assume $z_1, \dots, z_k$ denote the k public input wires.

We say that $\mathbf{z} \in \mathbb{F}^{3n}$ satisfies $(\mathcal{C}, \mathbf{x})$ for $\mathbf{x} \in \mathbb{F}^k$ if it satisfies:

1. Input Constraints: $z_i = x_i$ for $i \in [k]$.
2. Gate Constraints: $\mathbf{q}_{M,i} \cdot z_i \cdot z_{i+n} + \mathbf{q}_{L,i} \cdot z_i + \mathbf{q}_{R,i} \cdot z_{i+n} + \mathbf{q}_{O,i} \cdot z_{i+2n} + \mathbf{q}_{C,i} = 0$ for all $i \in [n]$.
3. Wiring Constraints: $z_{\sigma(i)} = z_i$ for $i \in [3n]$.

The HyperPlonk indexer outputs the following multilinear polynomials for the circuit $\mathcal{C}$:

- For $X \in \{M, L, R, O, C\}$ it outputs multilinear polynomial $\widetilde{q}_X$ satisfying $\widetilde{q}_X(\langle i \rangle) = \mathbf{q}_{X,i}$ for $i \in [n]$.
- Multilinear polynomials $\widetilde{\sigma}_j$ satisfying $\widetilde{\sigma}_j(\langle i \rangle) = \sigma((j-1)n + i)$ for $j \in [3]$.
- Multilinear polynomials $\widetilde{\mathsf{id}}_j$, $j \in [3]$ for the identity permutation satisfying $\widetilde{\mathsf{id}}_j(\langle i \rangle) = i + (j-1)n$.

We note that the polynomials for identity permutation can be computed by the verifier in $O(\log n)$ $\mathbb{F}$-operations, so indexer need not output them. We now describe the online phase of the HyperPlonk PIOP. To show knowledge of $\mathbf{w} \in \mathbb{F}^{3n}$ satisfying $(\mathcal{C}, \mathbf{x})$ the prover encodes the witness as multilinear polynomials $\widetilde{w}_1, \widetilde{w}_2$ and $\widetilde{w}_3$ satisfying $\widetilde{w}_1(\langle i \rangle) = z_i$, $\widetilde{w}_2(\langle i \rangle) = z_{i+n}$, $\widetilde{w}_3(\langle i \rangle) = z_{i+2n}$. As a technical point, the prover sends z_i as 0 for $i \in [k]$ to accommodate public inputs. It then sends oracles $[\![\widetilde{w}_1]\!]$, $[\![\widetilde{w}_2]\!]$ and $[\![\widetilde{w}_3]\!]$. The verifier computes the multilinear polynomial $\widetilde{I}$ for the public inputs satisfying $\widetilde{I}(\langle i \rangle) = x_i$ for $i \in [k]$ and 0 elsewhere. It adjusts the oracle $[\![\widetilde{w}_1]\!]$ sent by the prover to $[\![\widetilde{w}_1]\!] + [\![\widetilde{I}]\!]$. Henceforth, we will use $[\![\widetilde{w}_1]\!]$ to denote this "adjusted" oracle.

3.2 HyperPlonk PIOP

We summarize the key steps in HyperPlonk PIOP [CBBZ23] below.

- The prover sends the witness oracles $[\![\widetilde{w}_i]\!]$, $i \in [3]$ as above.
- The verifier sends challenges $\beta, \gamma \leftarrow \mathbb{F}$.
- On receiving the challenges, the prover computes $\mu + 1$ variate oracle $\widetilde{v}$ satisfying the following for all $\mathbf{x} \in B_\mu$.

$$\begin{aligned}\widetilde{v}(0, \mathbf{x}) &= \prod_{i=1}^{3} \left(\frac{\widetilde{w}_i(\mathbf{x}) + \beta \cdot \widetilde{\mathsf{id}}_i(\mathbf{x}) + \gamma}{\widetilde{w}_i(\mathbf{x}) + \beta \cdot \widetilde{\sigma}_i(\mathbf{x}) + \gamma} \right). \\ \widetilde{v}(1, \mathbf{x}) &= \widetilde{v}(\mathbf{x}, 0) \cdot \widetilde{v}(\mathbf{x}, 1).\end{aligned}$$

- The prover computes μ-variate polynomials $\widetilde{v}_0$ and $\widetilde{v}_1$where $\widetilde{v}_b(\mathbf{X}) = \widetilde{v}(b, \mathbf{X})$ for $b \in \{0, 1\}$.
- The prover also computes μ-variate polynomials $\widetilde{u}_0$ and $\widetilde{u}_1$ where $\widetilde{u}_b(\mathbf{X}) = \widetilde{v}(\mathbf{X}, b)$ for $b \in \{0, 1\}$.
- The verifier sends batching challenge $\xi \leftarrow \mathbb{F}$ and a zero-check challenge $\tau \leftarrow \mathbb{F}^\mu$.
- The prover and verifier execute the sumcheck PIOP:

$$\sum_{\mathbf{x} \in B_\mu} \widetilde{eq}(\mathbf{x}, \tau) \cdot \widetilde{G}(\mathbf{x}) = 0,$$

where $\widetilde{G}$ is defined as:

$$\begin{aligned}\widetilde{G}(\mathbf{x}) = {} & \widetilde{q}_M(\mathbf{x})\widetilde{w}_1(\mathbf{x})\widetilde{w}_2(\mathbf{x}) + \widetilde{q}_L(\mathbf{x})\widetilde{w}_1(\mathbf{x}) + \widetilde{q}_R(\mathbf{x})\widetilde{w}_2(\mathbf{x}) \\ & + \widetilde{q}_O(\mathbf{x})\widetilde{w}_3(\mathbf{x}) + \widetilde{q}_C(\mathbf{x}) \\ & + \xi \cdot \widetilde{v}_0(\mathbf{x}) \prod_{i=1}^{3} (\widetilde{w}_i(\mathbf{x}) + \beta\widetilde{\sigma}_i(\mathbf{x}) + \gamma) \\ & - \xi \cdot \prod_{i=1}^{3} (\widetilde{w}_i(\mathbf{x}) + \beta\widetilde{\mathsf{id}}_i(\mathbf{x}) + \gamma) \\ & + \xi^2 \cdot (\widetilde{v}_1(\mathbf{x}) - \widetilde{u}_0(\mathbf{x}) \cdot \widetilde{u}_1(\mathbf{x})) . \end{aligned} \tag{1}$$

As noted in [CBBZ23], the oracles $[\![\widetilde{u}_b]\!]$ for $b \in \{0, 1\}$ can be simulated using the oracles $[\![\widetilde{v}_0]\!]$ and $[\![\widetilde{v}_1]\!]$. We will however require the prover to explicitly commit to oracles $[\![\widetilde{u}_b]\!]$ instead of simulating them as it helps us to reduce the number of multilinear oracle queries.

3.3 Multilinear PCS with Constant Proof Size

We recall recent constructions [GPS25, EG25] of multilinear polynomial commitment schemes with constant proof size. The constructions use the following isomorphism between μ-variate multilinear polynomials and univariate polynomials $\mathbb{F}[X]$ of degree $< n$ for $n = 2^\mu$. Define the isomorphism of the $\mathbb{F}$-vector spaces as:

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

For $\widetilde{f} \in \mathbb{F}^{\leq 1}[X_1, \ldots, X_\mu]$, we use $\hat{f}$ to denote the univariate polynomial $\varphi(\widetilde{f})$. We associate a multilinear polynomial $\widetilde{f}$ with the univariate polynomial $\hat{f}$ according to the above isomorphism. Subsequently, we treat the KZG commitment C to the polynomial $\hat{f}$ as the commitment to the multilinear polynomial $\widetilde{f}$. The constructions in [GPS25, EG25] show elegant protocols to prove evaluations $\widetilde{f}(\mathbf{z}) = v$ for $\mathbf{z} \in \mathbb{F}^\mu$ and $v \in \mathbb{F}$ with constant proof size and $O_\lambda(n)$ prover cost. Specifically, we will use the Samaritan PCS from [GPS25] which achieves constant proof size of 368 bytes over the BLS12-381 curve.

3.4 Overview of Our Techniques

Our aim is to retain the prover efficiency of sumcheck without incurring $O(\log n)$ rounds of communication. For the sake of illustration consider the claim $\sum_{\mathbf{x} \in B_\mu} \widetilde{a}(\mathbf{x})\widetilde{b}(\mathbf{x}) = v$ for μ-variate multilinear polynomials $\widetilde{a}$ and $\widetilde{b}$. Let $n = 2^\mu$, then using the sumcheck protocol (see Section 2), the prover can prove the claim in $O(\log n)$ rounds of communication while doing $O(n)$ work. We ask:

Is there a way we can reduce the communication rounds?

A possible approach is to use the *univariate sumcheck* protocol introduced in [BCR+19], where instead of considering multilinear polynomial evaluations over the hypercube, we consider univariate polynomial evaluations over a multiplicative subgroup of $\mathbb{F}$ of size n. Specifically, letting $\mathbb{H} = \{1, \omega, \ldots, \omega^{n-1}\}$ be the multiplicative subgroup of $\mathbb{F}$ of size n with $\omega \in \mathbb{F}$ as a generator, we define univariate polynomials $\hat{a}, \hat{b} \in \mathbb{F}^{<n}[X]$ such that $\hat{a}(\omega^{i-1}) = \widetilde{a}(\langle i \rangle)$ and $\hat{b}(\omega^{i-1}) = \widetilde{b}(\langle i \rangle)$. Then, the equivalent claim to prove is $\sum_{h \in \mathbb{H}} \hat{a}(h)\hat{b}(h) = v$. The univariate sumcheck protocol [BCR+19], when used with the KZG PCS allows the identity to be checked using $O(1)$ communication. However, the prover work is $O(n \log n)$ due to the need to compute polynomial product $\hat{a}(X) \cdot \hat{b}(X)$. The next question we ask is:

Can we reduce communication while ensuring that the prover work remains $O(n)$?

We give an affirmative answer to this question, achieving communication of $O(\log \log n)$ with linear prover effort. At a high level we achieve this by using a hybrid of the two sumcheck techniques. We initially proceed with the multilinear sumcheck protocol for $O(\log \log n)$ rounds, at the end of which we are left with a claim over hypercube of size $m = n/\log n$, involving multilinear polynomials of size m. By identifying the multilinear polynomials of size m with univariate polynomials of degree $< m$, we derive an equivalent univariate sumcheck claim. Using the univariate sumcheck protocol, and the KZG PCS, we can then prove the claim with $O(1)$ communication and $O(m \log m)$ prover work, which is $O(n)$ since $m = n/\log n$. The approach is detailed as *improved sumcheck* in Section 4. We crucially use Samaritan PCS [GPS25] to show that the multilinear polynomials of size m are suitable restrictions of the original multilinear polynomials of size n. Another technical challenge is to show that the polynomials involved in the univariate sumcheck are consistent with the multilinear polynomials in the reduced sumcheck instance, which requires us to show that a pair of polynomials share the same coefficients in the power basis and Lagrange basis respectively. We present a PIOP for this in Section 5. Finally in Section 6, we present several optimizations in applying the above blueprint to the HyperPlonk PIOP presented in Section 3.

4 Improved Sumcheck

Let $\widetilde{G}(X_1, \ldots, X_\ell)$ be a multivariate polynomial of degree d over $\mathbb{F}$. Let $\widetilde{f}_1, \ldots, \widetilde{f}_\ell$ be μ-variate multilinear polynomials. For *improved sumcheck*, we consider a sumcheck instance of the form:

$$\sum_{\mathbf{x} \in B_\mu} \widetilde{G}(\widetilde{f}_1(\mathbf{x}), \ldots, \widetilde{f}_\ell(\mathbf{x})) = v. \tag{2}$$

Let $n = 2^\mu$ denote the size of the hypercube. Choose $\rho = \log\log n$, and set $\kappa = \mu - \rho$. Let $m = 2^\kappa$. We observe that $m = n/\log n$. To prove the claim in Equation (2), the prover and verifier initially proceed with the first ρ rounds of the classical multivariate sumcheck protocol described in Section 2. Assuming $\mathbf{r} = (r_1, \ldots, r_\rho)$ is the set of challenges sent by the verifier so far, the reduced claim is of the form

$$\sum_{\mathbf{x} \in B_\kappa} \widetilde{G}(\widetilde{f}_1(\mathbf{r}, \mathbf{x}), \ldots, \widetilde{f}_\ell(\mathbf{r}, \mathbf{x})) = \bar{v}, \tag{3}$$

for some $\bar{v} \in \mathbb{F}$. At this stage, the prover commits to κ-variate multilinear polynomials $\widetilde{p}_i, i \in [\ell]$ defined by $\widetilde{p}_i(\mathbf{x}) = \widetilde{f}_i(\mathbf{r}, \mathbf{x})$ for $\mathbf{x} \in B_\kappa$. Let $\hat{p}_1(X), \ldots, \hat{p}_\ell(X)$ denote the corresponding univariate polynomials of degree less than m, such that j^{th} coefficient of $\hat{p}_i$ is $\widetilde{p}_i(\langle j \rangle)$, $\forall i \in [\ell], j \in [m]$. Our idea is to verify the reduced claim using a univariate PIOP over the much smaller domain of size m. Specifically, we choose the domain to be the set $\mathbb{H}$ consisting of the m roots of unity $\{1, \omega, \ldots, \omega^{m-1}\}$. To this end, the prover computes polynomials $\hat{q}_i(X)$, $i \in [\ell]$ such that evaluations of $\hat{q}_i$ over the set $\mathbb{H}$ match the monomial coefficients of polynomial $\hat{p}_i$. Let $\{\mu_k^{\mathbb{H}}(X)\}_{k=1}^m$ denote the Lagrange basis polynomials for the set $\mathbb{H}$ satisfying $\mu_k^{\mathbb{H}}(\omega^{j-1}) = \delta_{kj}$. We define linear operator ifft on the set $\mathbb{F}^{<m}[X]$ as:

$$\mathsf{ifft}\left(\sum_{i=1}^m a_i X^{i-1}\right) = \sum_{i=1}^m a_i \mu_i^{\mathbb{H}}(X). \tag{4}$$

In terms of the above operator, we have $\hat{q}_i = \mathsf{ifft}(\hat{p}_i)$ for all $i \in [\ell]$. Assuming that the polynomials $\hat{p}_i$ and $\hat{q}_i$ are honest, we can reduce the sumcheck claim in Equation (3) to a univariate sumcheck claim as below:

$$\begin{aligned} &\sum_{\mathbf{x} \in B_\kappa} \widetilde{G}(\widetilde{p}_1(\mathbf{x}), \ldots, \widetilde{p}_\ell(\mathbf{x})) = \bar{v} \\ \Leftrightarrow &\sum_{i \in [m]} \widetilde{G}(\widetilde{p}_1(\langle i \rangle), \ldots, \widetilde{p}_\ell(\langle i \rangle)) = \bar{v} \\ \Leftrightarrow &\sum_{i \in [m]} \widetilde{G}(\hat{q}_1(\omega^{i-1}), \ldots, \hat{q}_\ell(\omega^{i-1})) = \bar{v}. \end{aligned} \tag{5}$$

The final claim uses the fact that $\hat{q}_i(\omega^{j-1}) = \widetilde{p}_i(\langle j \rangle)$ for all $i \in [\ell]$ and $j \in [m]$. Let $\hat{G}(X) = \widetilde{G}(\hat{q}_1(X), \ldots, \hat{q}_\ell(X))$. Then the final claim is essentially equivalent to the claim $\sum_{i\in[m]} \hat{G}(\omega^{i-1}) = \bar{v}$. Now, we can use the *univariate sumcheck* to establish this claim. As shown in [BCR+19], it requires the prover to send polynomial oracles $\hat{g}$ and $\hat{h}$, with $\deg(\hat{h}) \le m-2$ such that:

$$\hat{G}(X) = (X^m - 1) \cdot \hat{g}(X) + X \cdot \hat{h}(X) + \bar{v}/m. \tag{6}$$

The prover commits to polynomials $\hat{g}(X)$ and $\hat{h}(X)$. To check the identity, the verifier queries the polynomials $\hat{q}_i, i \in [\ell]$, $\hat{g}$, and $\hat{h}$ at a random point and enforces the degree bound on $\hat{h}$. The above polynomials can be computed and committed in $O(\mathcal{C}(\widetilde{G}, d, \ell) \cdot m \log m) = O(\mathcal{C}(\widetilde{G}, d, \ell) \cdot n)$ $\mathbb{F}$ operations and $O(dm + \ell m) = O((d+\ell)n/\log n)$ $\mathbb{G}_1$ operations, where $\mathcal{C}(\widetilde{G}, d, \ell)$ is a constant dependent on the multivariate form $\widetilde{G}$, d and ℓ. Note that most $\mathbb{F}$-operations are incurred in computing polynomial products corresponding to high degree terms in $\widetilde{G}$. In a typical application we expect d to be a small constant (e.g., $d \le 5$) and thus computing each term of $\widetilde{G}$ takes $\approx O(m \log m)$ operations. The entire computation of $\hat{G}$ then takes $O(||\widetilde{G}|| m \log m)$ operations, which is $O(||\widetilde{G}|| n)$ where $||\widetilde{G}||$ denotes the number of monomial terms in $\widetilde{G}$.

4.1 Checking Consistency of Polynomials

The above reduction requires that the polynomials $\hat{p}_i$ and $\hat{q}_i$ are correctly computed from the original polynomials $\widetilde{f}_i$ and initial challenges $\mathbf{r} = (r_1, \ldots, r_\rho)$. First, we wish to show that $\widetilde{p}_i(\cdot) = \widetilde{f}_i(\mathbf{r}, \cdot)$ for all $i \in [\ell]$. The verifier uses a challenge α to batch the checks into an aggregated check:

$$\sum_{i=1}^{\ell} \alpha^{i-1} \widetilde{p}_i(\cdot) = \sum_{i=1}^{\ell} \alpha^{i-1} \widetilde{f}_i(\mathbf{r}, \cdot).$$

The verifier can homomorphically compute commitments to the polynomials $\widetilde{p} = \sum_{i=1}^{\ell} \alpha^{i-1} \widetilde{p}_i$ and $\widetilde{f} = \sum_{i=1}^{\ell} \alpha^{i-1} \widetilde{f}_i$. Thereafter it samples $\mathbf{y} \leftarrow \mathbb{F}^\kappa$ and queries $\widetilde{p}$ and $\widetilde{f}$ at points $\mathbf{y}$ and $(\mathbf{r}, \mathbf{y})$ respectively. It accepts if the two evaluations are equal. Using the multilinear PCS in [GPS25, EG25], the preceding check incurs $O(m+n)$ cost to the prover and results in constant communication.

Next, we wish to check that for all $i \in [\ell]$, $\hat{q}_i = \mathsf{ifft}(\hat{p}_i)$. Again, using the linearity of ifft operation, the verifier can batch the checks into a single check. We can use the challenge α used previously. The verifier can compute the commitment to the polynomials $\hat{q} = \sum_{i=1}^{\ell} \alpha^{i-1} \hat{q}_i$ and $\hat{p} = \sum_{i=1}^{\ell} \alpha^{i-1} \hat{p}_i$ (note that the commitment to $\hat{p}$ is the same as the commitment to $\widetilde{p}$ as computed in the previous subsection). It now needs to check that $\hat{q} = \mathsf{ifft}(\hat{p})$, given the commitments to both the univariate polynomials. We present a PIOP to check this claim in Section 5.

5 PIOP for Basis Translation

In this section, we exhibit a polynomial IOP for showing $\hat{q} = \mathsf{ifft}(\hat{p})$ given commitments to the univariate polynomials $\hat{p}$ and $\hat{q}$ in $\mathbb{F}^{<m}[X]$. As before, $\widetilde{p}$ and $\widetilde{q}$ will denote the corresponding multilinear polynomials defined over the hypercube B_κ for $\kappa = \log m$. Let $\mathbf{p} = (p_1, \ldots, p_m)$ and $\mathbf{q} = (q_1, \ldots, q_m)$ denote the coefficient vectors of polynomials $\hat{p}$ and $\hat{q}$ (and equivalently evaluations of $\widetilde{p}$ and $\widetilde{q}$) respectively.

We will find it convenient to check the equality $\hat{q} = \mathsf{ifft}(\hat{p})$ using the multilinear representation of the polynomials. Note that the equality implies $p_i = \hat{q}(\omega^{i-1})$ for $i \in [m]$. Let W denote the $m \times m$ FFT matrix with $W_{ij} = \omega^{(i-1)(j-1)}$. Let $\widetilde{W}$ denote the multilinear polynomial over 2κ variables such that $\widetilde{W}(\langle\, i\,\rangle, \langle\, j\,\rangle) = \omega^{(i-1)(j-1)}$. In other words, $\widetilde{W}$ is a multilinear extension of the FFT matrix W viewed as a function $B_{2\kappa} \to \mathbb{F}$. The coefficient vectors $\mathbf{q}$ and $\mathbf{p}$ of polynomials $\widetilde{q}$ and $\widetilde{p}$ satisfy the following matrix product.

$$(p_1, \ldots, p_m)^T = (q_1, \ldots, q_m)^T \cdot W.$$

The equivalent identity over multilinear polynomials that needs to be checked is:

$$\widetilde{p}(\mathbf{Y}) = \sum_{\mathbf{x} \in B_\kappa} \widetilde{q}(\mathbf{x}) \cdot \widetilde{W}(\mathbf{x}, \mathbf{Y}). \tag{7}$$

where $\mathbf{Y}$ denotes the block of variables $Y_1, \ldots, Y_\kappa$. Before proceeding, we state a key technical result from ([EG25], Claim 4.1) in Lemma 5.1, and its variant in Lemma 5.2.

Lemma 5.1. *For polynomials $\hat{p} = \sum_{i=1}^m p_i X^{i-1}$ and $\hat{q} = \sum_{i=1}^m q_i X^{i-1}$ in $\mathbb{F}[X]$, let $\langle \hat{p}, \hat{q} \rangle$ denote the sum $\sum_{i=1}^m p_i q_i$. Then, $\langle \hat{p}, \hat{q} \rangle = v$ if and only if there exists a polynomial $\hat{S} \in \mathbb{F}[X]$, computable in $O(m \log m)$ field operations such that:*

$$\begin{aligned}&\hat{p}(X) \cdot \hat{q}(1/X) + \hat{p}(1/X) \cdot \hat{q}(X) \\ &\quad = 2v + X \cdot \hat{S}(X) + (1/X) \cdot \hat{S}(1/X)\end{aligned}$$

The following variant of Lemma 5.1 will also be useful.

Lemma 5.2. *For polynomials $\hat{p} = \sum_{i=1}^m p_i X^{i-1}$ and $\hat{q} = \sum_{i=1}^m q_i X^{i-1}$, let $\hat{r} = \hat{p} \circ \hat{q}$ be the polynomial $\sum_{i=1}^m r_i X^{i-1}$ with $r_i = p_i q_i$ for all $i \in [m]$. Then, with overwhelming probability over the choice of $\gamma \leftarrow \mathbb{F}$, the polynomials $\hat{p}, \hat{q}$ and $\hat{r}$ in $\mathbb{F}^{<m}[X]$ satisfy $\hat{r} = \hat{p} \circ \hat{q}$ if and only if $\langle \hat{p}(\gamma X), \hat{q}(X) \rangle = \hat{r}(\gamma)$. Equivalently, there exists a polynomial $\hat{S}$, computable in $O(m \log m)$ field operations such that:*

$$\begin{aligned}&\hat{p}(\gamma X) \cdot \hat{q}(1/X) + \hat{p}(\gamma/X) \cdot \hat{q}(X) \\ &\quad = 2\hat{r}(\gamma) + X \cdot \hat{S}(X) + (1/X) \cdot \hat{S}(1/X)\end{aligned}$$

Proof. The proof essentially follows from batching the checks $r_i = p_i q_i$ using the challenge γ. □

We now explicitly specify the multilinear extension $\widetilde{W}$ of the FFT matrix as below:

$$\widetilde{W}(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^m \sum_{j=1}^m \omega^{(i-1)\cdot(j-1)} \cdot \widetilde{eq}(\langle\, i\,\rangle, \mathbf{x}) \cdot \widetilde{eq}(\langle\, j\,\rangle, \mathbf{y}). \tag{8}$$

To check the multilinear polynomial identity in Equation (7), the verifier samples $\mathbf{y} \leftarrow \mathbb{F}^\kappa$ and asks the prover to prove:

$$\widetilde{p}(\mathbf{y}) = \sum_{\mathbf{x} \in B_\kappa} \widetilde{q}(\mathbf{x}) \cdot \widetilde{W}(\mathbf{x}, \mathbf{y}).$$

At this stage, the prover computes the multilinear polynomial $\widetilde{t}(\mathbf{X})$ given by $\widetilde{t}(\mathbf{X}) = \widetilde{W}(\mathbf{X}, \mathbf{y})$ and sends a commitment to $\widetilde{t}$ to the verifier. Thereafter, the statement reduces to $\sum_{\mathbf{x} \in B_\kappa} \widetilde{q}(\mathbf{x}) \cdot \widetilde{t}(\mathbf{x}) =$

$\widetilde{p}(\mathbf{y})$. The claimed sum $v_p = \widetilde{p}(\mathbf{y})$ can be obtained by querying the polynomial $\widetilde{p}$. The preceding summation over the hypercube is succinctly captured by the associated univariate polynomials $\hat{q}$ and $\hat{t}$ as $\langle \hat{q}, \hat{t} \rangle = v_p$. The inner product of the polynomials (defined as the inner product of their coefficient vectors) can be proved using Lemma 5.1. Two important questions remain: (i) How does the prover compute polynomial $\widetilde{t}$ in time $O(n)$ and (ii) How does it convince the verifier that $\widetilde{t}(\mathbf{X})$ is indeed $\widetilde{W}(\mathbf{X}, \mathbf{y})$? We answer the first question in Lemma 5.3 below.

Lemma 5.3. *The multilinear polynomial $\widetilde{t}(\mathbf{x})$ as defined above can be computed in $O(n)$ $\mathbb{F}$-operations.*

Proof. Let $e_j = \widetilde{eq}(\langle\, j\,\rangle, \mathbf{y})$ for $j \in [m]$. Using standard techniques, e_j for all $j \in [m]$ can be computed in $O(m)$ time. Now, for $i \in [m]$, from Equation 8, we have $\widetilde{t}(\langle\, i\,\rangle) = \sum_{j=1}^{m} e_j \cdot \left(\omega^{(i-1)}\right)^{j-1}$. If we define $\hat{e}(X)$ to be the polynomial $\sum_{j=1}^{m} e_j \cdot X^{j-1}$, we see that $\widetilde{t}(\langle\, i\,\rangle) = \hat{e}(\omega^{i-1})$ for all $i \in [m]$. Thus, $\widetilde{t}(\langle\, i\,\rangle)$ can be computed for all $i \in [m]$ in time $O(m \log m) = O(n)$ using the FFT algorithm. $\square$

Proving correctness of $\widetilde{t}$. Next, we turn to the task of showing that $\widetilde{t}(\mathbf{x}) = \widetilde{W}(\mathbf{x}, \mathbf{y})$. Let $e_j = \widetilde{eq}(\langle\, j\,\rangle, \mathbf{y})$ for $j \in [m]$ be the coefficients of the polynomial $\hat{e}(X)$ used in the proof of Lemma 5.3. Let $\mathbf{e}$ denote the vector $(e_1, \ldots, e_m)$. Let $t_1, \ldots, t_m$ be the evaluations of the polynomial $\widetilde{t}(\mathbf{x})$ (and equivalently coefficients of $\hat{t}(X)$). Let θ_i denote the vector $(1, \omega^{i-1}, \omega^{2(i-1)}, \ldots, \omega^{(m-1)(i-1)})$ for $i \in [m]$. Then, one needs to prove that: $t_i = \langle \theta_i, \mathbf{e} \rangle$ for all $i \in [m]$. The verifier can batch these checks into a single check by sending a challenge $\gamma \leftarrow \mathbb{F}$ and requiring the prover to prove:

$$\sum_{i=1}^{m} t_i \cdot \gamma^{i-1} = \Big\langle \sum_{i=1}^{m} \gamma^{i-1} \cdot \theta_i, \mathbf{e} \Big\rangle, \tag{9}$$

or equivalently: $\hat{t}(\gamma) = \langle \mathbf{a}, \mathbf{e} \rangle$ for $\mathbf{a} = \sum_{i=1}^{m} \gamma^{i-1} \theta_i$. Now, routine calculation shows that for $j \in [m]$,

$$a_j = \sum_{i=1}^{m} \gamma^{i-1} \cdot \omega^{(i-1)(j-1)} = \frac{(\gamma\omega^{j-1})^m - 1}{\gamma\omega^{j-1} - 1} = \frac{\gamma^m - 1}{\gamma\omega^{j-1} - 1}.$$

Thus, the vector $\mathbf{a}$ is given by:

$$\mathbf{a} = (\gamma^m - 1) \cdot \left(\frac{1}{\gamma - 1}, \frac{1}{\gamma\omega - 1}, \ldots, \frac{1}{\gamma\omega^{m-1} - 1} \right). \tag{10}$$

At this stage, the prover commits to polynomial $\hat{A}(X)$, where $\hat{A}(X) = \sum_{i=1}^{m} (\gamma\omega^{i-1} - 1)^{-1} X^{i-1}$. The check $\hat{t}(\gamma) = \langle \mathbf{a}, \mathbf{e} \rangle$ is equivalent to the following check on polynomials:

$$\langle \hat{A}, \hat{e} \rangle = \hat{t}(\gamma)/(\gamma^m - 1). \tag{11}$$

The only thing that remains is to prove that the polynomials $\hat{A}$ and $\hat{e}$ have the claimed coefficients. In other words, we need to show that the i^{th} coefficients of $\hat{A}$ and $\hat{e}$ are $1/(\gamma\omega^{i-1} - 1)$ and $\widetilde{eq}(\langle\, i\,\rangle, \mathbf{y})$ respectively. Fortunately, the polynomial $\hat{e}(X)$ has a simple description, which enables the verifier to query it efficiently (in $O(\log m)$ effort). Specifically, we have:

$$\hat{e}(X) = \prod_{i=1}^{\kappa} (y_i X^{2^{i-1}} + (1 - y_i)). \tag{12}$$

One can observe that the coefficient of X^{i-1} in $\hat{e}(X)$ is precisely $\widetilde{eq}(\langle i \rangle, \mathbf{y})$. Thus, it only remains to show the correct form of the polynomial $\hat{A}$. Unlike $\hat{e}$, the polynomial $\hat{A}$ does not directly admit a succinct description for the verifier. However, it turns out that the polynomial $\hat{B}(X)$ whose coefficients are inverses of the coefficients of $\hat{A}$, i.e, the polynomial $\hat{B}(X) = \sum_{i=1}^{m}(\gamma\omega^{i-1} - 1)X^{i-1}$ has a succinct description. First, we notice that the polynomial $\Psi_\omega(X)$ given below has powers of ω $(1, \omega, \omega^2, \ldots, \omega^{m-1})$ as its coefficient vector.

$$\Psi_\omega(X) = \prod_{i=1}^{\kappa}(1 + \omega^{2^{i-1}} X^{2^{i-1}}). \tag{13}$$

Now, we can write $\hat{B}(X)$ as:

$$\hat{B}(X) = \sum_{i=1}^{m}(\gamma\omega^{i-1} - 1)X^{i-1} = \gamma\Psi_\omega(X) - \frac{X^m - 1}{X - 1}. \tag{14}$$

The verifier can query polynomial $\hat{B}$ in $O(\log m)$ effort. We then use the polynomial $\hat{B}$ to establish correctness of the polynomial $\hat{A}$ by proving $\hat{A} \circ \hat{B} = 1 + X + \cdots + X^{m-1} = (X^m - 1)/(X - 1)$ invoking Lemma 5.2. Concretely, the verifier sends the challenge $\delta \leftarrow \mathbb{F}$ to which the prover responds with commitment to polynomial $\hat{S}_1$ such that:

$$\begin{aligned} &\hat{A}(X) \cdot \hat{B}(\delta/X) + \hat{A}(1/X) \cdot \hat{B}(\delta X) \\ &\qquad = 2 \cdot \frac{\delta^m - 1}{\delta - 1} + X \cdot \hat{S}_1(X) + (1/X) \cdot \hat{S}_1(1/X). \end{aligned} \tag{15}$$

The above identity can be checked by querying the polynomials $\hat{A}, \hat{B}$ and $\hat{S}$ at a random point, where the prover provides evaluations of $\hat{A}$ and $\hat{S}_1$, while the verifier computes the evaluation of $\hat{B}$ at the point itself. Having established the correctness of the polynomial $\hat{A}$, the correctness of polynomial $\hat{t}$ can be verified by checking the identity in Equation 11. Using Lemma 5.1, this is accomplished by the prover sending commitment to polynomial $\hat{S}_2(X)$ satisfying:

$$\begin{aligned} &\hat{A}(X) \cdot \hat{e}(1/X) + \hat{A}(1/X) \cdot \hat{e}(X) \\ &\qquad = \frac{2 \cdot \hat{t}(\gamma)}{\gamma^m - 1} + X \cdot \hat{S}_2(X) + (1/X) \cdot \hat{S}_2(1/X). \end{aligned} \tag{16}$$

Again, the identity is checked by querying the polynomials $\hat{A}, \hat{S}_2$ at points determined by a random evaluation challenge, polynomial $\hat{t}$ at γ whereas the queries to $\hat{e}$ can be computed by the verifier efficiently. Finally, having established the correctness of $\hat{t}(X)$, we use one more instance of the check in Lemma 5.1 to prove $\langle \hat{q}(X), \hat{t}(X) \rangle = v_p$, where v_p was the answer to the polynomial query $\widetilde{p}(\mathbf{y})$. Concretely, the prover sends commitment to polynomial $\hat{S}_3(X)$ satisfying:

$$\begin{aligned} &\hat{q}(X) \cdot \hat{t}(1/X) + \hat{q}(1/X) \cdot \hat{t}(X) \\ &\qquad = 2v_p + X \cdot \hat{S}_3(X) + (1/X) \cdot \hat{S}_3(1/X). \end{aligned} \tag{17}$$

The verifier can check the above identity as before. This completes the description of the protocol to check $\hat{q} = \mathsf{ifft}(\hat{p})$. In fact, as an optimization, using an additional challenge $\varepsilon \leftarrow \mathbb{F}$, the checks in

Equations 15, 16 and 17 can be batched into a single check, requiring the prover to send commitment to a single polynomial $\hat{S}(X)$ satisfying:

$$\begin{aligned}
&\hat{A}(X) \cdot \hat{B}(\delta/X) + \hat{A}(1/X) \cdot \hat{B}(\delta X) \\
&\quad + \varepsilon\big(\hat{A}(X) \cdot \hat{e}(1/X) + \hat{A}(1/X) \cdot \hat{e}(X)\big) \\
&\qquad + \varepsilon^2\big(\hat{q}(X) \cdot \hat{t}(1/X) + \hat{q}(1/X) \cdot \hat{t}(X)\big) \\
&= 2 \cdot \big((\delta^m - 1)/(\delta - 1) + \varepsilon \cdot \hat{t}(\gamma)/(\gamma^m - 1) + \varepsilon^2 v_p\big) \\
&\qquad\qquad + X \cdot \hat{S}(X) + (1/X) \cdot \hat{S}(1/X).
\end{aligned}$$

Clearly, the computational effort of the prover is $O(m \log m)$ $\mathbb{F}$-operations and $O(m)$ $\mathbb{G}_1$ operations. For $m = n/\log n$, these costs are $O(n)$. The communication is easily seen to be $O(1)$, as we make a constant number of univariate polynomial queries, each with constant communication.

Theorem 5.1. *Given commitments to polynomials $\hat{p}$ and $\hat{q}$ of degree at most m, there exists an argument of knowledge to check that $\hat{q} = \mathsf{ifft}(\hat{p})$ where the prover cost is $O(m \log m)$ $\mathbb{F}$ operations and $O(m)$ $\mathbb{G}_1$ operations, the verification cost is $O(\log m)$ $\mathbb{F}$-operations and $O(1)$ group operations, while the argument size is $O(1)$.*

Theorem 5.2. *Given a multivariate polynomial $\widetilde{\mathbf{G}}$ with ℓ variables and total degree d, and μ-variate multilinear polynomials $\widetilde{f}_1, \ldots, \widetilde{f}_\ell$, there exists an argument of knowledge in the algebraic group model (AGM) for the relation*

$$\sum_{\mathbf{x} \in B^\mu} \widetilde{\mathbf{G}}(\widetilde{f}_1(\mathbf{x}), \ldots, \widetilde{f}_\ell(\mathbf{x})) = v,$$

where (i) the prover incurs $\mathcal{C}(\widetilde{\mathbf{G}}, d, \ell) \cdot n$ $\mathbb{F}$-operations, $O(n + \ell n/\log n)$ $\mathbb{G}_1$ operations for some constant $\mathcal{C}(\widetilde{\mathbf{G}}, d, \ell)$ dependent on the multivariate form $\widetilde{\mathbf{G}}$, d and ℓ, (ii) the verifier incurs $O(\ell + \log n)$ $\mathbb{F}$-operations and $O(\ell)$ $\mathbb{G}_1$-operations while (iii) the proof size is $O(\ell + d \log\log n)$.

Proof. The proof follows from observing that polynomial IOP for the sumcheck relation in Sections 4 and 5 is sound, i.e, the verifier accepts with negligible probability if the sumcheck relation is not satisfied. Using Samaritan PCS from [GPS25], which is extractable in the AGM to realize the polynomial oracles, we obtain a knowledge sound argument using Lemma 2.2. □

Remark. Recently, the security of the KZG [KZG10] PCS underlying the construction of constant sized multilinear PCS such as Samaritan [GPS25] and Mercury [EG25] has been proven from falsifiable assumptions [LPS24], which are shown hold under weaker models than the AGM. Plausibly, the multilinear polynomial commitment schemes can also be shown to be secure under these assumptions.

6 HybridPlonk

In this section, we compile HyperPlonk PIOP [CBBZ23] from Section 3.2 with sumcheck protocol from Theorem 5.2 to construct a new SNARK with sub-logarithmic proof size and linear time prover. We also incorporate several optimizations to reduce the concrete proof size. We call the resulting SNARK as HybridPlonk. We describe the key steps below.

6.1 Reducing the number of commitments

We first order the polynomials involved in the sumcheck in Equation 1.

$$\widetilde{\mathcal{O}} := \big(\widetilde{eq}, \widetilde{\mathsf{id}}_1, \widetilde{\mathsf{id}}_2, \widetilde{\mathsf{id}}_3, \widetilde{q}_M, \widetilde{q}_L, \widetilde{q}_R, \widetilde{q}_O, \widetilde{q}_C, \widetilde{\sigma}_1, \widetilde{\sigma}_2, \widetilde{\sigma}_3, \widetilde{w}_1, \widetilde{w}_2, \widetilde{w}_3, \widetilde{v}_0, \widetilde{v}_1, \widetilde{u}_0, \widetilde{u}_1\big). \tag{18}$$

Let $\mathbf{r} = (r_1, \ldots, r_\rho)$ be the vector of sumcheck challenges in the first ρ rounds. At this stage, the protocol requires the prover to send commitments to κ-variate polynomials $\widetilde{p}_i = \widetilde{\mathcal{O}}_i(\mathbf{r}, \cdot)$, where we use $\widetilde{\mathcal{O}}_i$ to denote the i^{th} polynomial in $\widetilde{\mathcal{O}}$. In addition, the prover is required to send commitments to polynomials $\hat{q}_i$ such that $\hat{q}_i = \mathsf{ifft}(\hat{p}_i)$ for $i \in [\ell]$. Recall from Section 4 that our eventual univariate sumcheck is defined using the polynomials $\hat{q}_i$, $i \in [\ell]$. The polynomials $\widetilde{p}_i$, $i \in [\ell]$ were essentially "intermediataries" to ensure that evaluations of $\hat{q}_i$ at the m roots of unity are identical to the the evaluations of polynomial $\widetilde{\mathcal{O}}_i$ on the hypercube B_κ. Towards this, we checked:

$$\widetilde{\mathcal{O}}_i(\mathbf{r}, \cdot) = \widetilde{p}_i(\cdot) \equiv \hat{p}_i = \mathsf{ifft}(\hat{q}_i).$$

As an optimization, we realize that we need not commit to individual polynomials $\widetilde{p}_i$ for all $i \in [\ell]$. On receiving challenge α from the verifier, the prover sends commitment only to the polynomial $\hat{p} = \sum_{i=1}^{\ell} \alpha^{i-1}\hat{p}_i$. Next the prover sends $\mathbf{y} \leftarrow \mathbb{F}^\kappa$ and asks the prover to prove:

$$\sum_{i=1}^{\ell} \alpha^{i-1}\hat{q}_i = \mathsf{ifft}(\hat{p}) \quad \text{and} \quad \widetilde{p}(\mathbf{y}) = \sum_{i=1}^{\ell} \alpha^{i-1}\widetilde{\mathcal{O}}_i(\mathbf{r}, \mathbf{y}). \tag{19}$$

6.2 Reducing multilinear queries

For the sake of understanding the trade-offs, let us assume we simulate oracles $[\![\widetilde{u}_b]\!]$ in terms of oracles $[\![\widetilde{v}_0]\!]$ and $[\![\widetilde{v}_1]\!]$ as in original HyperPlonk [CBBZ23]. We note that:

$$\begin{aligned}
\widetilde{u}_b(X_1, \ldots, X_\mu) &= \widetilde{v}(X_1, \ldots, X_\mu, b) \\
&= (1 - X_1)\widetilde{v}(0, X_2, \ldots, X_\mu, b) + X_1\widetilde{v}(1, X_2, \ldots, X_\mu, b) \\
&= (1 - X_1)\widetilde{v}_0(X_2, \ldots, X_\mu, b) + X_1\widetilde{v}_1(X_2, \ldots, X_\mu, b).
\end{aligned}$$

If we separate out the simulated polynomials $\widetilde{u}_0$, $\widetilde{u}_1$, the second check in Equation 19 reduces to:

$$\begin{aligned}
\widetilde{p}(\mathbf{y}) &= \sum_{i=1}^{\ell-2} \alpha^{i-1}\widetilde{\mathcal{O}}_i(\mathbf{r}, \mathbf{y}) + \alpha^{\ell-2}\widetilde{u}_0(\mathbf{r}, \mathbf{y}) + \alpha^{\ell-1}\widetilde{u}_1(\mathbf{r}, \mathbf{y}) \\
&= \widetilde{h}(\mathbf{r}, \mathbf{y}) + \alpha^{\ell-2}\big((1 - r_1) \cdot \widetilde{v}_0(\mathbf{r}', \mathbf{y}, 0) + r_1 \cdot \widetilde{v}_1(\mathbf{r}', \mathbf{y}, 0)\big) \\
&\quad + \alpha^{\ell-1}\big((1 - r_1) \cdot \widetilde{v}_0(\mathbf{r}', \mathbf{y}, 1) + r_1 \cdot \widetilde{v}_1(\mathbf{r}', \mathbf{y}, 1)\big),
\end{aligned}$$

where $\widetilde{h} = \sum_{i=1}^{\ell-2} \widetilde{\mathcal{O}}_i$ and $\mathbf{r}' = (r_2, \ldots, r_\rho)$. To check the above equality we need 4 multilinear queries, namely to:

- Polynomials $\widetilde{p}$ and $\widetilde{h}$ at $\mathbf{y}$ and $(\mathbf{r}, \mathbf{y})$ respectively.
- Polynomial $(1 - r_1)\widetilde{v}_0 + r_1\widetilde{v}_1$ at points $(r_2, \ldots, r_\rho, \mathbf{y}, b)$ for $b \in \{0, 1\}$.

We next show how to reduce the number of multilinear queries to just one. First, instead of simulating oracles $[\![\widetilde{u}_b]\!]$ in terms of oracles $[\![\widetilde{v}_0]\!]$ and $[\![\widetilde{v}_1]\!]$, we modify the HyperPlonk PIOP to have prover send $[\![\widetilde{u}_b]\!]$ as independent oracles. This allows us to include all the polynomials in $\widetilde{\mathcal{O}}$ in the linear combination $\widetilde{h}$, and we can simply query $[\![\widetilde{h}]\!]$ at $(\mathbf{r}, \mathbf{y})$. However, we need to ensure that the polynomials $\widetilde{u}_0$ and $\widetilde{u}_1$ satisfy:

$$\begin{aligned}&\widetilde{u}_b(X_1, \ldots, X_\mu) \\ &\quad = (1 - X_1)\widetilde{v}_0(X_2, \ldots, X_\mu, b) + X_1\widetilde{v}_1(X_2, \ldots, X_\mu, b).\end{aligned} \tag{20}$$

To prove the above, we define $(\mu + 1)$-variate polynomial:

$$\begin{aligned}&\widetilde{v}(X_1, \ldots, X_{\mu+1}) \\ &\quad = (1 - X_1)\widetilde{v}_0(X_2, \ldots, X_{\mu+1}) + X_1\widetilde{v}_1(X_2, \ldots, X_{\mu+1}).\end{aligned} \tag{21}$$

Then we have $\widetilde{v}_b(\mathbf{X}) = \widetilde{v}(b, \mathbf{X})$ and $\widetilde{u}_b(\mathbf{X}) = \widetilde{v}(\mathbf{X}, b)$ for $b \in \{0, 1\}$. Interpreting the preceding relations combinatorially, the $2n$ coefficients of $\widetilde{v}$ contain the n coefficients of $\widetilde{v}_0$ at *even* positions and the n coefficients of $\widetilde{v}_1$ at *odd* positions. In terms of their corresponding univariate polynomials, this is equivalent to $\hat{v}(X) = \hat{v}_0(X^2) + X \cdot \hat{v}_1(X^2)$. How do coefficients of $\widetilde{u}_0$ and $\widetilde{u}_1$ occur within coefficients of $\widetilde{v}$? We observe that since $\widetilde{u}_b$ is determined by restricting the most significant bit, coefficients of $\widetilde{u}_0$ constitute the first n coefficients of $\widetilde{v}$, while coefficients of $\widetilde{u}_1$ constitute the latter n coefficients of $\widetilde{v}$. In terms of univariate polynomials, this is equivalent to $\hat{v}(X) = \hat{u}_0(X) + X^n \cdot \hat{u}_1(X)$. Equating, the correctness of polynomials $\widetilde{u}_b$, $b \in \{0, 1\}$ is checked using the univariate identity:

$$\hat{u}_0(X) + X^n \cdot \hat{u}_1(X) = \hat{v}_0(X^2) + X \cdot \hat{v}_1(X^2). \tag{22}$$

We record the above polynomial identity to be checked by the verifier. We still need to query the κ-variate polynomial $\widetilde{p}$ at $\mathbf{y}$, and a linear combination of multilinear polynomials in $\widetilde{\mathcal{O}}$ at $(\mathbf{r}, \mathbf{y})$. To restrict the number of multilinear queries to just one, we use the following trick. We define the μ-variate polynomial $\widetilde{p}_{ext}$ that vacuously extends $\widetilde{p}$ to B_μ as $\widetilde{p}_{ext}(X_1, \ldots, X_\mu) = \widetilde{p}(X_{\rho+1}, \ldots, X_\mu)$, i.e, $\widetilde{p}_{ext}$ essentially ignores the first ρ variables. Thus, we can obtain the query $\widetilde{p}(\mathbf{y})$ as $\widetilde{p}_{ext}(\mathbf{r}, \mathbf{y})$. Let $\hat{p}_{ext}(X)$ be the univariate polynomial of degree $< n$ corresponding to μ-variate polynomial $\widetilde{p}_{ext}$. We need to relate it to the polynomial $\hat{p}(X)$ of degree at most m corresponding to $\widetilde{p}$. It can be seen that if $(p_1, \ldots, p_m)$ are the evaluations of $\widetilde{p}$ over B_κ, the evaluations of $\widetilde{p}_{ext}$ over B_μ are:

$$\underbrace{p_1, \ldots, p_1}_{s \text{ times}}, \underbrace{p_2, \ldots, p_2}_{s \text{ times}}, \ldots, \underbrace{p_m, \ldots, p_m}_{s \text{ times}}$$

where $s = 2^\rho$. This is so because the evaluation of $\widetilde{p}_{ext}$ is independent of the first ρ variables. The relation describing the corresponding univariate polynomials $\hat{p}_{ext}(X)$ and $\hat{p}(X)$ is seen to be:

$$\begin{aligned}\hat{p}_{ext}(X) &= \sum_{i=1}^{m} p_i X^{s(i-1)}(1 + X + \cdots + X^{s-1}) \\ &= \hat{p}(X^s) \cdot (X^s - 1)/(X - 1).\end{aligned} \tag{23}$$

Thus, we require the prover to send the commitment to polynomial $\hat{p}_{ext}$, and the verifier checks the identity in Equation 23. To summarize, after sending the commitments to the polynomials in the modified HyperPlonk PIOP, the prover sends commitments to polynomials $\hat{q}_i$, $i \in [\ell]$. Thereafter, on receiving the challenge $\alpha \leftarrow \mathbb{F}$, the prover commits to the polynomial $\widetilde{p}(\mathbf{Y}) = \sum_{i=1}^{\ell} \alpha^{i-1} \cdot \widetilde{\mathcal{O}}_i(\mathbf{r}, \mathbf{Y})$ and the "extended" polynomial $\widetilde{p}_{ext}$. The prover and the verifier then check the following:

- Check $\sum_{i=1}^{\ell} \alpha^{i-1}\hat{q}_i = \mathsf{ifft}(\hat{p})$ using PIOP in Section 5.
- Check $\hat{p}_{ext}(X) = (X^s - 1) \cdot \hat{p}(X^s)/(X-1)$.
- Check $\widetilde{p}_{ext}(\mathbf{r}, \mathbf{y}) = \sum_{i=1}^{\ell} \alpha^{i-1}\widetilde{\mathcal{O}}_i(\mathbf{r}, \mathbf{y})$. This is equivalent to checking that the polynomial $\widetilde{p}_{ext} - \sum_{i=1}^{\ell} \alpha^{i-1}\widetilde{\mathcal{O}}_i$ is 0 at $(\mathbf{r}, \mathbf{y})$, which requires a single query to the polynomial whose commitment can be computed by the verifier.
- Check the univariate polynomial identity in Equation 22 holds.
- They check the univariate sumcheck instance determined by the polynomials $\hat{q}_i$, $i \in [\ell]$.

In the next Section, we unroll the above steps to provide the full protocol for HybridPlonk.

6.3 HybridPlonk: Unrolled

We describe the protocol as consisting of three phases. The first phase is called the *sumcheck phase* where classical multivariate sumcheck protocol is used to reduce the original sumcheck instance to a *univariate* sumcheck instance. In the next phase, the prover and verifier interact to prove the univariate sumcheck instance. The final phase is the *query phase*, where the verifier queries the oracles sent in the prior phases to check the polynomial identities output in the previously. During the first two phases, we will use the terminology, "$\mathcal{P}$ and $\mathcal{V}$ output the polynomial identity $\mathcal{P}(X) = 0$" to denote the polynomial identities that will be checked by the verifier in the query phase. The protocol follows.

- **Sumcheck Phase**:
  1. $\mathcal{P}$ sends commitments to witness polynomials $\widetilde{w}_1$, $\widetilde{w}_2$ and $\widetilde{w}_3$.
  2. $\mathcal{V}$ sends challenges $\beta, \gamma \leftarrow \mathbb{F}$.
  3. $\mathcal{P}$ computes the multilinear polynomial $\widetilde{v}$ as described in Section 6.2 and sends commitments to $\widetilde{v}_b(\mathbf{X}) = \widetilde{v}(b, \mathbf{X})$ and $\widetilde{u}_b(\mathbf{X}) = \widetilde{v}(\mathbf{X}, b)$ for $b \in \{0,1\}$.
  4. $\mathcal{P}$ and $\mathcal{V}$ output $\hat{u}_0(X) + X^n \cdot \hat{u}_1(X) = \hat{v}_0(X^2) + X \cdot \hat{v}_1(X^2)$ with $\deg(\hat{u}_b)$, $\deg(\hat{v}_b) < n$ for $b \in \{0,1\}$.
  5. $\mathcal{V}$ sends batching challenge $\xi \leftarrow \mathbb{F}$ and zero-check challenge $\tau \leftarrow \mathbb{F}^{\mu}$.
  6. $\mathcal{P}$ and $\mathcal{V}$ define the sumcheck: $\sum_{\mathbf{x} \in B_\mu} \widetilde{eq}(\mathbf{x}, \tau) \cdot \widetilde{G}(\mathbf{x}) = 0$, where $\widetilde{G}$ is defined by:

$$\begin{aligned} \widetilde{G} := {} & \widetilde{q}_M \cdot \widetilde{w}_1 \cdot \widetilde{w}_2 + \widetilde{q}_L \cdot \widetilde{w}_1 + \widetilde{q}_R \cdot \widetilde{w}_2 + \widetilde{q}_O \cdot \widetilde{w}_3 + \widetilde{q}_C \\ & + \xi \cdot \Big(\widetilde{v}_0 \cdot \prod_{i=1}^{3} (\widetilde{w}_i + \beta \cdot \widetilde{\sigma}_i + \gamma) - \prod_{i=1}^{3} (\widetilde{w}_i + \beta \cdot \widetilde{\mathsf{id}}_i + \gamma) \Big) \\ & + \xi^2 \cdot \big(\widetilde{v}_1 - \widetilde{u}_0 \cdot \widetilde{u}_1 \big). \end{aligned}$$

  7. $\mathcal{P}$ and $\mathcal{V}$ execute the first ρ rounds of the above sumcheck, with verifier sending challenges $\mathbf{r} = (r_1, \ldots, r_\rho)$.

8. $\mathcal{P}$ and $\mathcal{V}$ define the reduced sumcheck instance $\sum_{\mathbf{x}\in B_\kappa} \widetilde{eq}_\tau(\mathbf{r},\mathbf{x})\cdot\widetilde{G}(\mathbf{r},\mathbf{x}) = \bar{v}$ for some $\bar{v}\in\mathbb{F}$. To simplify notation, we define following vectors of polynomials:

$$\begin{aligned}\widetilde{\mathcal{O}} &= \big(\widetilde{eq}, \widetilde{\mathsf{id}}_1, \widetilde{\mathsf{id}}_2, \widetilde{\mathsf{id}}_3, \widetilde{q}_M, \widetilde{q}_L, \widetilde{q}_R, \widetilde{q}_O, \widetilde{q}_C, \widetilde{\sigma}_1, \widetilde{\sigma}_2,\\ &\quad \widetilde{\sigma}_3, \widetilde{w}_1, \widetilde{w}_2, \widetilde{w}_3, \widetilde{v}_0, \widetilde{v}_1, \widetilde{u}_0, \widetilde{u}_1\big),\\ \hat{\mathcal{O}} &= \big(\hat{eq}, \hat{\mathsf{id}}_1, \hat{\mathsf{id}}_2, \hat{\mathsf{id}}_3, \hat{q}_M, \hat{q}_L, \hat{q}_R, \hat{q}_O, \hat{q}_C, \hat{\sigma}_1, \hat{\sigma}_2,\\ &\quad \hat{\sigma}_3, \hat{w}_1, \hat{w}_2, \hat{w}_3, \hat{v}_0, \hat{v}_1, \hat{u}_0, \hat{u}_1\big),\\ \bar{\mathcal{O}} &= \big(\bar{eq}, \bar{\mathsf{id}}_1, \bar{\mathsf{id}}_2, \bar{\mathsf{id}}_3, \bar{q}_M, \bar{q}_L, \bar{q}_R, \bar{q}_O, \bar{q}_C, \bar{\sigma}_1, \bar{\sigma}_2,\\ &\quad \bar{\sigma}_3, \bar{w}_1, \bar{w}_2, \bar{w}_3, \bar{v}_0, \bar{v}_1, \bar{u}_0, \bar{u}_1\big).\end{aligned} \tag{24}$$

Denoting i^{th} polynomial in vectors $\widetilde{\mathcal{O}}$, $\hat{\mathcal{O}}$ and $\bar{\mathcal{O}}$ as $\widetilde{\mathcal{O}}_i$, $\hat{\mathcal{O}}_i$ and $\bar{\mathcal{O}}_i$ respectively, the verifier enforces the following constraints:
   - $\hat{\mathcal{O}}_i(X)$ is the univariate polynomial corresponding to $\widetilde{\mathcal{O}}_i(\mathbf{r},\mathbf{X})$.
   - $\bar{\mathcal{O}}_i = \mathsf{ifft}(\hat{\mathcal{O}}_i)$, i.e, $\bar{\mathcal{O}}_i(\omega^{j-1})$ gives the coefficient of X^{j-1} in $\hat{\mathcal{O}}_i$.
9. Subject to above constraints, $\mathcal{P}$ and $\mathcal{V}$ define the *univariate* sumcheck instance: $\sum_{i=1}^m \bar{eq}(\omega^{i-1})\cdot\bar{G}(\omega^{i-1}) = \bar{v}$, where the polynomial $\bar{G}(X)$ is defined as:

$$\begin{aligned}\bar{G} := {} & \bar{q}_M\cdot\bar{w}_1\cdot\bar{w}_2 + \bar{q}_L\cdot\bar{w}_1 + \bar{q}_R\cdot\bar{w}_2 + \bar{q}_O\cdot\bar{w}_3 + \bar{q}_C\\ &+ \xi\cdot\Big(\bar{v}_0\cdot\prod_{i=1}^{3}(\bar{w}_i+\beta\cdot\bar{\sigma}_i+\gamma) - \prod_{i=1}^{3}(\bar{w}_i+\beta\cdot\bar{\mathsf{id}}_i+\gamma)\Big)\\ &+ \xi^2\cdot\big(\bar{v}_1 - \bar{u}_0\cdot\bar{u}_1\big).\end{aligned}$$

Next, we describe the *univariate* sumcheck phase. The prover needs to prove that the polynomial $\bar{eq}(X)\cdot\bar{G}(X)$ sums to $\bar{v}$ over the set $\mathbb{H}$. To this end, the prover sends commitments to polynomials $\hat{g}(X)$ and $\hat{h}(X)$ with $\deg(\hat{h}) \le m-2$ satisfying:

$$\bar{eq}(X)\bar{G}(X) = (X^m-1)\hat{g}(X) + X\cdot\hat{h}(X) + \bar{v}/m. \tag{25}$$

So far, we have not required the prover to explicitly commit to all the polynomials defining the above identity, though all the polynomials are implicitly defined by the commitments sent in the sumcheck phase (linked via ifft to committed polynomials). Following known optimization techniques, we ask the prover for evaluations of the polynomials upfront, and later batch the checking of evaluations. This deferred verification allows us to optimize communication.

To start this phase, the verifier sends evaluation point $z \leftarrow \mathbb{F}$ and the prover sends evaluations $V_{eq} = \bar{eq}(z)$, $V_{w,1} = \bar{w}_1(z)$, $V_{w,2} = \bar{w}_2(z)$, $V_{w,3} = \bar{w}_3(z)$, $V_{\sigma,1} = \bar{\sigma}_1(z)$, $\mathsf{V}_{\sigma,2} = \bar{\sigma}_2(z)$, $V_{\sigma,3} = \bar{\sigma}_3(z)$, $V_{\mathsf{id},1} = \bar{\mathsf{id}}_1(z)$, $V_{\mathsf{id},2} = \bar{\mathsf{id}}_2(z)$, $V_{\mathsf{id},3} = \bar{\mathsf{id}}_3(z)$ and $V_{u,0} = \bar{u}_0(z)$. The preceding evaluations allow us to linearize the identity in Equation 25, reducing it to $G'(z) = 0$ for a polynomial G' which can be expressed as a linear combination of committed polynomials. Subsequently, the verifier checks the correctness of all the claimed evaluations which can be efficiently batched. We detail the steps in the univariate sumcheck phase below.

- **Univariate Sumcheck Phase**:

1. $\mathcal{P}$ sends commitments to polynomials $\hat{g}(X)$ and $\hat{h}(X)$.

2. $\mathcal{V}$ sends evaluation point $z \leftarrow \mathbb{F}$.
3. $\mathcal{P}$ sends evaluations V_{eq}, $V_{w,1}$, $V_{w,2}$, $V_{w,3}$, $V_{\sigma,1}$, $\mathsf{V}_{\sigma,2}$, $V_{\sigma,3}$, $V_{\mathsf{id},1}$, $V_{\mathsf{id},2}$, $V_{\mathsf{id},3}$ and $V_{u,0}$, as discussed earlier.
4. $\mathcal{V}$ defines the polynomial $G'(X)$ as:

$$\begin{aligned}
&G'(X) := \\
&V_{eq} \cdot \big(V_{w,1} \cdot V_{w,2} \cdot \bar{q}_M + V_{w,1} \cdot \bar{q}_L + V_{w,2} \cdot \bar{q}_R + V_{w,3} \cdot \bar{q}_O + \bar{q}_C\big) \\
&+ V_{eq} \cdot \xi \cdot \big(\bar{v}_0 \cdot \prod_{i=1}^{3}(V_{w,i} + \beta V_{\sigma,i} + \gamma) - \prod_{i=1}^{3}(V_{w,i} + \beta V_{\mathsf{id},i} + \gamma)\big) \\
&+ V_{eq} \cdot \xi^2 \cdot \big(\bar{v}_1 - V_{u,0} \cdot \bar{u}_1\big) - (z^m - 1) \cdot \hat{g} - z \cdot \hat{h} - \bar{v}/m. \qquad (26)
\end{aligned}$$

We note that $G'(X)$ is a linear combination of polynomials $\bar{q}_M$, $\bar{q}_L$, $\bar{q}_R$, $\bar{q}_O$, $\bar{q}_C$, $\bar{v}_1$, $\bar{u}_1$, $\hat{g}$ and $\hat{h}$.
5. $\mathcal{V}$ sends challenge $\alpha \leftarrow \mathbb{F}$ to batch evaluation claims.
6. $\mathcal{P}$ and $\mathcal{V}$ output the evaluation claim $\bar{F}(z) = V$ for:

$$\begin{aligned}
\bar{F}(X) &= G'(X) + \alpha \cdot \bar{eq}(X) + \alpha^2 \cdot \bar{w}_1(X) + \alpha^3 \bar{w}_2(X) \\
&\quad + \alpha^4 \bar{w}_3(X) + \alpha^5 \bar{\sigma}_1(X) + \alpha^6 \bar{\sigma}_2(X) + \alpha^7 \bar{\sigma}_3(X) \\
&\quad + \alpha^8 \bar{\mathsf{id}}_1(X) + \alpha^9 \bar{\mathsf{id}}_2(X) + \alpha^{10} \bar{\mathsf{id}}_3(X) + \alpha^{11} \bar{u}_0(X), \\
V &= \alpha \cdot V_{eq} + \alpha^2 \cdot V_{w,1} + \alpha^3 \cdot V_{w,2} + \alpha^4 \cdot V_{w,3} \\
&\quad + \alpha^5 \cdot V_{\sigma,1} + \alpha^6 \cdot V_{\sigma,2} + \alpha^7 \cdot V_{\sigma,3} + \alpha^8 \cdot V_{\mathsf{id},1} \\
&\quad + \alpha^9 \cdot V_{\mathsf{id},2} + \alpha^{10} \cdot V_{\mathsf{id},3} + \alpha^{11} \cdot V_{u,0}.
\end{aligned}$$

7. $\mathcal{V}$ computes $a_1, \ldots, a_\ell, b_1, b_2 \in \mathbb{F}$ be such that $\bar{F} = \sum_{i=1}^{\ell} a_i \bar{\mathcal{O}}_i + b_1 \hat{g} + b_2 \hat{h} - \bar{v}/m$ in time $O(\ell + \log n)$. Let $\bar{H}(\mathbf{X})$ denote the polynomial $\sum_{i=1}^{\ell} a_i \bar{\mathcal{O}}_i$ and $\widetilde{H}(\mathbf{X})$ denote the corresponding multilinear polynomial.
8. $\mathcal{P}$ sends commitments to polynomials $\bar{H}(X)$ and $\hat{K}(X) = \sum_{i=1}^{\ell} a_i \hat{\mathcal{O}}_i(X)$, which are also commitments to the corresponding multilinear polynomials $\widetilde{H}(X)$ and $\widetilde{K}(\mathbf{X}) = \sum_{i=1}^{\ell} a_i \widetilde{\mathcal{O}}_i(\mathbf{r}, \mathbf{X})$. It also sends commitment to μ-variate extension $\widetilde{K}_{ext}$ of $\widetilde{K}$.
9. $\mathcal{V}$ sends $\mathbf{y} \leftarrow \mathbb{F}^{\kappa}$.
10. $\mathcal{P}$ sends $V_y = \widetilde{K}_{ext}(\mathbf{r}, \mathbf{y})$.
11. $\mathcal{P}$ and $\mathcal{V}$ output the following claims:

$$\sum_{\mathbf{x} \in B_\kappa} \widetilde{H}(\mathbf{x}) \widetilde{W}(\mathbf{x}, \mathbf{y}) = \widetilde{K}(\mathbf{y}) = V_y, \quad (\bar{H} = \mathsf{ifft}(\hat{K})) \qquad (27)$$

$$\widetilde{K}_{ext}(\mathbf{r}, \mathbf{y}) = \sum_{i=1}^{m} a_i \widetilde{\mathcal{O}}_i(\mathbf{r}, \mathbf{y}) = V_y, \qquad (28)$$

$$\hat{K}_{ext}(X) = \hat{K}(X^s) \cdot (X^s - 1)/(X - 1), \deg(\hat{K}) < m. \qquad (29)$$

Let $\eta, \delta \leftarrow \mathbb{F}$ denote the challenges sent by $\mathcal{V}$ in the sub-protocol (Section 5) to prove the first claim. Also, let $\epsilon \leftarrow \mathbb{F}$ be an additional challenge sent by $\mathcal{V}$ to accommodate the optimization discussed in Section 5. Then $\mathcal{P}$ sends commitments to $\hat{t}(X), \hat{A}(X)$ and $\hat{S}(X)$ such that,

$$\begin{aligned}
&\bar{H}(X) \cdot \hat{t}(1/X) + \bar{H}(1/X) \cdot \hat{t}(X) \\
&\quad + \epsilon(\hat{A}(X) \cdot \hat{e}(1/X) + \hat{A}(1/X) \cdot \hat{e}(X)) \\
&\quad\quad + \epsilon^2(\hat{A}(X) \cdot \hat{B}(\delta/X) + \hat{A}(1/X) \cdot \hat{B}(\delta X)) \\
&= 2 \cdot (V_y + \epsilon \cdot \hat{t}(\eta)/(\eta^m - 1) + \epsilon^2 \cdot (\delta^m - 1)/(\delta - 1)) \\
&\quad\quad + X \cdot \hat{S}(X) + (1/X) \cdot \hat{S}(1/X).
\end{aligned}$$

The challenge ϵ is also used to (i) batch the multilinear evaluations $\widetilde{K}_{ext}(\mathbf{r}, \mathbf{y}) = V_y$, $\sum_{i=1}^{\ell} a_i \widetilde{\mathcal{O}}_i(\mathbf{r}, \mathbf{y}) = V_y$ and (ii) batch the degree checks on polynomials $\hat{u}_0, \hat{u}_1, \hat{v}_0, \hat{v}_1, \hat{K}, \hat{h}$.

12. $\mathcal{P}$ sends commitments to polynomials $\hat{d}(X) = \hat{u}_0(X) + \epsilon\hat{u}_1(X) + \epsilon^2\hat{v}_0(X) + \epsilon^3\hat{v}_1(X) + \epsilon^4 X^{n-m}\hat{K}(X) + \epsilon^5 X^{n-m+1}\hat{h}(X)$ and $\hat{D}(X) = X^{D-n+1}\hat{d}(X)$ to enforce degree bounds on the polynomials $\hat{u}_0$, $\hat{u}_1$, $\hat{v}_0$, $\hat{v}_1$, $\hat{K}$ and $\hat{h}(X)$. Here D denotes the maximum degree supported by the KZG SRS.
13. $\mathcal{V}$ checks identities in query phase. It accepts if all the identities hold, otherwise it rejects.

- **Query Phase**:

1. $\mathcal{V}$ checks the following polynomial identities:

$$\hat{u}_0(X) + X^n \cdot \hat{u}_1(X) = \hat{v}_0(X^2) + X \cdot \hat{v}_1(X^2) \tag{A}$$

$$\hat{K}_{ext}(X) = \hat{K}(X^s) \cdot (X^s - 1)/(X - 1) \tag{B}$$

$$\begin{aligned}
&\bar{H}(X) \cdot \hat{t}(1/X) + \bar{H}(1/X) \cdot \hat{t}(X) \\
&\quad + \epsilon\big(\hat{A}(X) \cdot \hat{e}(1/X) + \hat{A}(1/X) \cdot \hat{e}(X)\big) \\
&\quad + \epsilon^2\big(\hat{A}(X) \cdot \hat{B}(\delta/X) + \hat{A}(1/X) \cdot \hat{B}(\delta X)\big) \\
&\quad - X \cdot \hat{S}(X) + (1/X) \cdot \hat{S}(1/X) \\
&\quad = 2\left(V_y + \epsilon\frac{\hat{t}(\eta)}{\eta^m - 1} + \epsilon^2\frac{\delta^m - 1}{\delta - 1}\right)
\end{aligned} \tag{C}$$

$$\begin{aligned}
\hat{d}(X) &= \hat{u}_0(X) + \epsilon\hat{u}_1(X) + \epsilon^2\hat{v}_0(X) \\
&\quad + \epsilon^3\hat{v}_1(X) + \epsilon^4 X^{n-m}\hat{K}(X) + \epsilon^5 X^{n-m+1}\hat{h}(X)
\end{aligned} \tag{D}$$

$$\sum_{i=1}^{\ell} a_i\widetilde{\mathcal{O}}_i(\mathbf{r}, \mathbf{y}) + \epsilon\widetilde{K}_{ext}(\mathbf{r}, \mathbf{y}) = (1 + \epsilon) \cdot V_y \tag{E}$$

$$\bar{F}(z) = V \tag{F}$$

2. $\mathcal{V}$ checks (E) using one multilinear query, while to check others it sends evaluation challenge $r \leftarrow \mathbb{F}$.
3. $\mathcal{P}$ sends evaluations: $W_1 = (\hat{v}_0 + r\hat{v}_1)(r^2)$, $W_2 = \hat{K}(r^s)$, $W_3 = \bar{H}(1/r)$, $W_4 = \hat{t}(1/r)$, $W_5 = \hat{A}(1/r)$, $W_6 = \hat{S}(1/r)$.

4. $\mathcal{V}$ now checks the following linearized polynomial identities:

$$\hat{u}_0(r) + r^n \cdot \hat{u}_1(r) = W_1, \tag{A}$$

$$\hat{K}_{ext}(r) = W_2 \cdot (r^s - 1)/(r-1), \tag{B}$$

$$\begin{aligned} &W_4 \cdot \bar{H}(r) + W_3 \cdot \hat{t}(r) \\ &\quad + \epsilon\big(\hat{A}(r) \cdot \boxed{\hat{e}(1/r)} + W_5 \cdot \boxed{\hat{e}(r)}\big) \\ &\quad + \epsilon^2\big(\hat{A}(r) \cdot \boxed{\hat{B}(\delta/r)} + W_5 \cdot \boxed{\hat{B}(\delta r)}\big) \\ &\quad - r \cdot \hat{S}(r) - (1/r) \cdot W_6 \\ &\quad = 2\left(V_y + \epsilon\frac{\hat{t}(\eta)}{\eta^m - 1} + \epsilon^2 \frac{\delta^m - 1}{\delta - 1}\right), \end{aligned} \tag{C}$$

$$\begin{aligned} \hat{d}(r) &= \hat{u}_0(r) + \epsilon\hat{u}_1(r) + \epsilon^2\hat{v}_0(r) + \epsilon^3\hat{v}_1(r) \\ &\quad + \epsilon^4 r^{n-m} \cdot \hat{K}(r) + \epsilon^5 r^{n-m+1} \cdot \hat{h}(r). \end{aligned} \tag{D}$$

5. $\mathcal{V}$ sends another batching challenge $\varphi \leftarrow \mathbb{F}$ to reduce above identities to checking $\hat{L}(r) = 0$ for a polynomial $\hat{L}(X)$ whose commitment can be computed by the verifier. The "boxed" polynomials can be efficiently evaluated by the verifier.
6. $\mathcal{P}$ sends proof Π_{eval} to show the correctness of evaluations $W_1, \ldots, W_6$ in addition to evaluations $\bar{F}(z)$, $\hat{L}(r)$ and $\hat{t}(\eta)$. Using Theorem 3 in [BDFG21], Π_{eval} consists of 2 $\mathbb{F}$ elements and 1 $\mathbb{G}_1$ element.

Degree Check: $\mathcal{V}$ checks $e([D(X)]_1, [1]_2) = e([d(X)]_1, [X^{D-n+1}]_2)$ to enforce degree bound of $n-1$ on $\hat{d}(X)$ and consequently the degree bounds on polynomials constituting $\hat{d}(X)$.

6.4 Proof Size of HybridPlonk

We compute the proof size of the protocol described in Section 6.3.

- The prover commitments to 7 multilinear polynomials in the sumcheck phase, in addition to the communication during the sumcheck itself. This contributes $7\,\mathbb{G}_1 + |\pi_{sumcheck}|$ to the proof size.
- The prover sends 10 commitments and 12 field elements in the univariate sumcheck phase, which contributes $9\,\mathbb{G}_1 + 12\,\mathbb{F}$ to the proof size.
- The prover sends 1 group element and 8 field elements in the query phase. In addition it incurs communication cost of the multilinear query $|\pi_{mlpcs}|$.

The total proof size is thus $|\pi_{sumcheck}| + |\pi_{mlpcs}| + 18\,\mathbb{G}_1 + 20\,\mathbb{F}$. For the BLS12-381 curve, using SamaritanPCS as the multilinear polynomial commitment scheme with proof size of 368 bytes, and 5 rounds of sumcheck (assuming $\log\log n = 5$), with each prover message in sumcheck consisting of 5 $\mathbb{F}$ elements, the total communication is $18\,\mathbb{G}_1 + 45\,\mathbb{F}$ + 368 bytes, which works out to 2.67 KB. Finally, we can reduce the proof size to 2.27 KB, by using the optimization in [CBBZ23, Section 3.1] which reduces the communication during the sumcheck rounds from $5\,\mathbb{F}$ to $1\,\mathbb{G}_1 + 1\,\mathbb{F}$.

7 Implementation and Evaluation

We experimentally evaluate the performance of HybridPlonk and compare its proof size, proving and verification time against other prover efficient SNARKs featuring an updatable setup. Specifically, we compare against Spartan [Set20] instantiated with KZG-based PCS, MicroSpartan [ZSCZ25] – a recent variant of Spartan with smaller proof sizes, and HyperPlonk [CBBZ23].

We implement HybridPlonk in Rust using Arkworks [ac22]. We use Samaritan PCS [GPS25] to compile the PIOP described in Section 6. We also provide the first RUST implementation of Samaritan PCS, which might be of independent interest. Our implementation is anonymously available at `https://anonymous.4open.science/r/HybridPlonk-F011`.

For comparison, we use open-source implementations of Spartan[5], HyperPlonk[6], and MicroSpartan[7]. The benchmarks in Table 3 were obtained using above implementations. All of these implementations (except MicroSpartan, which uses BN254 curve) use BLS12-381 as the associated elliptic curve. We note that Spartan variants target R1CS backend, which is different from the Plonkish constraint system supported by HybridPlonk and Hyperplonk, and thus tradeoffs in a specific application must account for the suitability of the backend as well. We run all the experiments on a high end developer laptop: System 76 Adder Workstation with Intel Core i9 processor running Ubuntu 22.04 with 100 GB RAM and boosted clock speed up to 5 GHz. All the benchmarks are reported for single-threaded implementation. We summarize our evaluation below.

Proof Size. Table 2 shows the proof size of compared schemes with varying number of gates/-constraints. For sake of illustration, we include Plonk [GWC19], which has constant proof length of $(9\mathbb{G}_1 + 6\mathbb{F})$, but does not feature a linear time prover. Spartan and MicroSpartan proof sizes are taken from Figure 8 of [Set20] and Table 5 of [ZSCZ25] respectively. Note that Spartan is implemented over Arkworks' BLS12-381 curve, while MicroSpartan is implemented over Halo2curves' BN256 curve where the size of a $\mathbb{G}_1$ element is smaller. HyperPlonk PIOP incurs communication of $(\log n + 5)\mathbb{G}_1 + (4\log n + 29)\mathbb{F}$ [CBBZ23]. While reporting the proof sizes for HyperPlonk variants, we add to this the communication incurred by the respective polynomial commitment scheme (PCS). Finally, our HybridPlonk has the proof size of $18\mathbb{G}_1 + 20\mathbb{F} + \log\log n(\mathbb{G}_1 + \mathbb{F}) + 368$ bytes. To report concrete proof sizes for BLS12-381 curve, we use $|\mathbb{G}_1| = 48$ bytes and $|\mathbb{F}| = 32$ bytes. For BN254 curve, we have $|\mathbb{G}_1| = |\mathbb{F}| = 32$ bytes. We can see that HybridPlonk has around $3\times$ shorter proofs than existing schemes at $n = 2^{24}$, with the gap expected to widen to $4\times$ for $n = 2^{30}$.

Prover Time. Table 3 summarizes the performance of the prover in the compared schemes. All the SNARKs in the table show $O(n)$ prover complexity through experimental evaluation as expected. All the compared schemes have similar proving times, with HybridPlonk exhibiting roughly 10% faster prover over the two Spartan variants. Although we do not report proving times for the variant of HyperPlonk with Samaritan PCS, we expect the performance to be similar to the one reported for HyperKZG based instantiation.

Verifier Time. The verification time for Spartan was ≈ 160 milliseconds. For all other schemes, verification time was under 20 ms. This is expected as all other schemes feature $O(\log n)$ verifier,

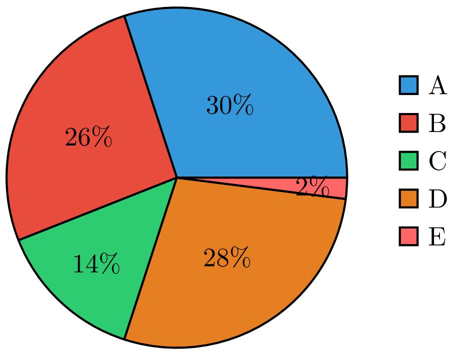
[5]https://github.com/arkworks-rs/spartan
[6]https://github.com/EspressoSystems/hyperplonk
[7]https://github.com/microsoft/Nova

Table 2: Proof sizes for different SNARKs with varying number of gates

Number of gates	2^{16}	2^{18}	2^{20}	2^{22}	2^{24}
Plonk[GWC19]	0.62 KB	0.62 KB	0.62 KB	0.62 KB	0.62 KB
Spartan[Set20]	71.6 KB	98 KB	142 KB	$>$ 142 KB	$>$ 142 KB
HyperPlonk(HyperKZG)[CBBZ23]	5.14 KB	5.64 KB	6.14 KB	6.54 KB	7.04 KB
HyperPlonk (Samaritan PCS)	4.01 KB	4.51 KB	5.01 KB	5.51 KB	6.01 KB
MicroSpartan[ZSCZ25]	$\approx$ 5 KB	5.63 KB	6.08 KB	6.53 KB	$\approx$ 7 KB
HybridPlonk	2.19 KB	2.27 KB	2.27 KB	2.27 KB	2.27 KB

Table 3: Prover time for different SNARKs with varying number of gates

Number of gates	2^{16}	2^{18}	2^{20}	2^{22}	2^{24}
Spartan[Set20]	2.8s	8.3s	28.5s	108.6s	409.7s
HyperPlonk (HyperKZG)[CBBZ23]	2.7s	10s	37.7s	141s	554s
MicroSpartan[ZSCZ25]	2.2s	7.4s	26.6s	101s	396s
HybridPlonk	1.6s	5.7s	21.8s	88s	357s



🟦	A: Multivariate sumcheck for $\log\log n$ rounds
🟥	B: Committing to multilinear and univariate polynomials
🟩	C: Eval proofs for multilinear and univariate polynomials
🟧	D: Computing polynomials
🟥	E: Others

Figure 1: HybridPlonk prover time breakdown across significant operations involved for a circuit with 2^{20} gates

whereas Spartan has $O(\log^2 n)$ verifier when instantiated with KZG commitments. MicroSpartan [ZSCZ25] was observed to have the fastest average verification time of ≈ 3 ms, while HybridPlonk and HyperPlonk averaged ≈ 11 ms.

Cost Breakdown of Prover Time in HybridPlonk: We also tabulate the break-up of different operations involved in HybridPlonk's proof computation. Figure 1 gives a cost breakdown of significant operations involved in the prover algorithm of HybridPlonk. Category A corresponds to the cost for $\log \log n$ multivariate sumcheck rounds. Category B includes the time required to commit to the seven multilinear polynomials and ten univariate polynomials, as described in Section 6.3. Category C includes the prover time required to generate evaluation proofs of two univariate polynomials (KZG PCS) and one multilinear polynomial (Samaritan PCS). Category D includes all the field operations involved in computing various univariate polynomials. All the costs are given with respect to the prover for 2^{20} gates.

8 Extensions and Future Work

We leave it as an interesting open question to adapt our improved sumcheck protocol to achieve sublogarithmic proof sizes for other SNARKs such as Spartan [Set20] and Gemini [BCHO22]. These SNARKs involve sparse representation of matrices over quadratic sized domain, so our methods do not directly apply. We also believe that our improved multivariate sumcheck protocol is of independent interest, and could have applications beyond SNARKs, such as in lookup arguments [STW24].

References

[ac22] arkworks contributors. `arkworks` zksnark ecosystem, 2022.

[ACFY24] Gal Arnon, Alessandro Chiesa, Giacomo Fenzi, and Eylon Yogev. STIR: Reed-solomon proximity testing with fewer queries. In Leonid Reyzin and Douglas Stebila, editors, *CRYPTO 2024, Part X*, volume 14929 of *LNCS*, pages 380–413. Springer, Cham, August 2024.

[ACFY25] Gal Arnon, Alessandro Chiesa, Giacomo Fenzi, and Eylon Yogev. WHIR: Reed-solomon proximity testing with super-fast verification. In Serge Fehr and Pierre-Alain Fouque, editors, *EUROCRYPT 2025, Part IV*, volume 15604 of *LNCS*, pages 214–243. Springer, Cham, May 2025.

[ARZR25] Noor Athamnah, Noga Ron-Zewi, and Ron D. Rothblum. Linear prover IOPs in log star rounds. Cryptology ePrint Archive, Paper 2025/1269, 2025.

[BBHR18] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Fast reed-solomon interactive oracle proofs of proximity. In Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella, editors, *ICALP 2018*, volume 107 of *LIPIcs*, pages 14:1–14:17. Schloss Dagstuhl, July 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$^+$25a] Anubhav Baweja, Alessandro Chiesa, Elisabetta Fedele, Giacomo Fenzi, Pratyush Mishra, Tushar Mopuri, and Andrew Zitek-Estrada. Time-space trade-offs for sumcheck. Cryptology ePrint Archive, Paper 2025/1473, 2025.

[BCF$^+$25b] 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.

[BCG$^+$14] Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *2014 IEEE Symposium on Security and Privacy*, pages 459–474. IEEE Computer Society Press, May 2014.

[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.

[BCJ$^+$24] Foteini Baldimtsi, Konstantinos Kryptos Chalkias, Yan Ji, Jonas Lindstrøm, Deepak Maram, Ben Riva, Arnab Roy, Mahdi Sedaghat, and Joy Wang. zkLogin: Privacy-preserving blockchain authentication with existing credentials. In Bo Luo, Xiaojing Liao, Jun Xu, Engin Kirda, and David Lie, editors, *ACM CCS 2024*, pages 3182–3196. ACM Press, October 2024.

[BCR$^+$19] Eli Ben-Sasson, Alessandro Chiesa, Michael Riabzev, Nicholas Spooner, Madars Virza, and Nicholas P. Ward. Aurora: Transparent succinct arguments for R1CS. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part I*, volume 11476 of *LNCS*, pages 103–128. Springer, Cham, May 2019.

[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.

[BDFG21] Dan Boneh, Justin Drake, Ben Fisch, and Ariel Gabizon. Halo infinite: Proof-carrying data from additive polynomial commitments. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part I*, volume 12825 of *LNCS*, pages 649–680, Virtual Event, August 2021. Springer, Cham.

[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.

[BS23] Ward Beullens and Gregor Seiler. LaBRADOR: Compact proofs for R1CS from module-SIS. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part V*, volume 14085 of *LNCS*, pages 518–548. Springer, Cham, August 2023.

[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.

[DMS24] Michel Dellepere, Pratyush Mishra, and Alireza Shirzad. Garuda and pari: Faster and smaller snarks via equifficient polynomial commitments. *IACR Cryptol. ePrint Arch.*, page 1245, 2024.

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

[GPS25] Chaya Ganesh, Sikhar Patranabis, and Nitin Singh. Samaritan: Linear-time prover SNARK from new multilinear polynomial commitments. *ASIACRYPT 2025 (to appear)*, 2025.

[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.

[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.

[LKKO20] Seunghwa Lee, Hankyung Ko, Jihye Kim, and Hyunok Oh. vCNN: Verifiable convolutional neural network. Cryptology ePrint Archive, Report 2020/584, 2020.

[LMN25] Christophe Levrat, Tanguy Medevielle, and Jade Nardi. A divide-and-conquer sumcheck protocol. *IACR Communications in Cryptology*, 2(1), 2025.

[LPS24] Helger Lipmaa, Roberto Parisella, and Janno Siim. Constant-size zk-SNARKs in ROM from falsifiable assumptions. In Marc Joye and Gregor Leander, editors, *EUROCRYPT 2024, Part VI*, volume 14656 of *LNCS*, pages 34–64. Springer, Cham, May 2024.

[LVA$^+$25] Hidde Lycklama, Alexander Viand, Nikolay Avramov, Nicolas Küchler, and Anwar Hithnawi. Artemis: Efficient commit-and-prove snarks for zkml, 2025.

[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.

[NMT] Shen Noether, Adam Mackenzie, and Monero Core Team. Ring confidential transactions. `https://lab.getmonero.org/pubs/MRL-0005.pdf`.

[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.

[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.

[SDP22] Nitin Singh, Pankaj Dayama, and Vinayaka Pandit. Zero knowledge proofs towards verifiable decentralized AI pipelines. In Ittay Eyal and Juan A. Garay, editors, *FC 2022*, volume 13411 of *LNCS*, pages 248–275. Springer, Cham, May 2022.

[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.

[STW24] Srinath T. V. Setty, Justin Thaler, and Riad S. Wahby. Unlocking the lookup singularity with Lasso. In Marc Joye and Gregor Leander, editors, *EUROCRYPT 2024, Part VI*, volume 14656 of *LNCS*, pages 180–209. Springer, Cham, May 2024.

[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.

[ZCF24] Hadas Zeilberger, Binyi Chen, and Ben Fisch. BaseFold: Efficient field-agnostic polynomial commitment schemes from foldable codes. In Leonid Reyzin and Douglas Stebila, editors, *CRYPTO 2024, Part X*, volume 14929 of *LNCS*, pages 138–169. Springer, Cham, August 2024.

[ZFZS20] Jiaheng Zhang, Zhiyong Fang, Yupeng Zhang, and Dawn Song. Zero knowledge proofs for decision tree predictions and accuracy. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *ACM CCS 2020*, pages 2039–2053. ACM Press, November 2020.

[ZSCZ25] Jiaxing Zhao, Srinath T. V. Setty, Weidong Cui, and Greg Zaverucha. Micronova: Folding-based arguments with efficient (on-chain) verification. In Marina Blanton, William Enck, and Cristina Nita-Rotaru, editors, *IEEE Symposium on Security and Privacy, SP 2025, San Francisco, CA, USA, May 12-15, 2025*, pages 1964–1982. IEEE, 2025.

[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.