Shred-to-Shine Metamorphosis of (Distributed) Polynomial Commitments *

Weihan Li[†,**] Zongyang Zhang[†,‡] Sherman S. M. Chow[§] Yanpei Guo[¶]
Boyuan Gao[†] Xuyang Song[‡,‡‡] Yi Deng[‖] Jianwei Liu[†]

Abstract

Succinct non-interactive arguments of knowledge (SNARKs) rely on polynomial commitment schemes (PCSs) to verify polynomial evaluations succinctly. High-performance multilinear PCSs (MLPCSs) from linear codes reduce prover cost, and distributed MLPCSs cut it further by parallelizing commitment and opening across provers. Employing a fast Reed–Solomon interactive oracle proof of proximity (FRI), we propose PIP_{FRI}, an MLPCS that combines the linear-time proving of linear-time-encodable-code PCSs with the compact proofs and fast verification of Reed–Solomon (RS) PCSs. Reducing fast Fourier transform and hash overhead, PIP_{FRI} is 10× faster to prove than the RS-based DeepFold (USENIX Security '25) while keeping competitive proof size and verifier time. Measured against Orion (CRYPTO '22) from linear-time-encodable codes, PIP_{FRI} proves 3.5× faster and reduces proof size and verifier time by 15×. As a linearly scalable distributed variant, we propose $\text{DEPIP}_{\text{FRI}}$, which adds accountability and distributes a single polynomial across provers, enabling the first code-based distributed SNARK for general circuits. Notably, compared with DeVirgo (CCS '22), which lacks accountability and supports only multiple independent polynomials, $\text{DEPIP}_{\text{FRI}}$ improves prover time by 25× and inter-prover communication by 7×. We identify shred-to-shine as the key insight: partitioning a polynomial into independently handled fragments while maintaining proof size and verifier time. Hitting the pairing regime, this insight yields a group-based MLPCS with a 16× shorter structured reference string (SRS) and a 10× faster opening time than a multilinear variant of Kate–Zaverucha–Goldberg (TCC '13).

*This is the extended version of our paper at USENIX Security 2026. An earlier IACR ePrint 2025/1354 version used a different title.

[†]School of Cyber Science and Technology, Beihang University. This work was conducted in part at CUHK during Li's official visit. [**]Ant Group.
[‡]Corresponding author.
[§]Dept. of Information Engineering, Chinese University of Hong Kong. The CUHK-affiliated authors used AI tools to support manuscript crafting; no AI was used to generate new technical ideas, results, or proofs.

[¶]National University of Singapore. [‡‡]Anoma.
[‖]School of Cryptology, Xidian University.

1 Introduction

Capturing polynomial evaluation claims succinctly, polynomial commitment schemes (PCSs) [20, 22] let a prover commit to a polynomial f in μ variables with degree bound d and later prove that $f(\boldsymbol{x}) = y$ at a public point $\boldsymbol{x}$. Succinct non-interactive arguments of knowledge (SNARKs) [14, 15] use PCSs with sublinear proof size and verifier cost in polynomial size d^{μ}. Modern SNARK constructions follow the "PIOP (polynomial interactive oracle proof) [15] + PCS" framework, so overall efficiency depends on both the PIOP and the PCS.

Suited to multilinear polynomials (degree <2 per variable, $\mu > 1$), multilinear PCSs (MLPCSs) [24, 33] pair naturally with linear-prover-time multilinear PIOPs [14, 29] in SNARKs. High-performance MLPCSs from linear codes [21, 43, 44] operate over finite fields, avoid the more costly group operations of group-based PCSs [11, 22], and thereby enable faster proving. Explicitly, we treat prover time, proof size, and verifier time as the *core metrics*. Representative applications include verifiable machine learning [1, 17], blockchain scalability [17, 30], multi-client oblivious RAM [16], and multi-writer searchable encryption with integrity against malicious writers [34]. Most such applications prioritize prover time. Accordingly, proof size and verifier time still matter because they determine the verification-circuit size and thus the prover cost in recursive SNARKs [37]. Notwithstanding this, known schemes face a sharp tradeoff among the core metrics.

Current code-based MLPCSs do not simultaneously achieve *linear prover time* (with low constants) and *polylogarithmic (polylog) proof size and verifier time* (with low constants). Specifically, MLPCSs based on Reed–Solomon (RS) codes [21, 42, 44] offer polylog proof size and verifier time with several-hundred-KB proofs, but their prover time is optimally quasi-linear due to RS encoding via fast Fourier transforms (FFTs). MLPCSs from linear-time encodable codes, in contrast, attain linear prover time but either have square-root proof size and verifier cost [20], or incur several-MB proofs and 10× slower verifier time [39].

Distributed PCSs [25, 26, 35, 37] reduce prover time by

enabling multiple provers to commit and open collaboratively. They also enable distributed SNARKs via the "distributed PIOP [25, 26] + distributed PCS" paradigm. This paradigm accelerates proof generation when the prover can exploit distributed (parallel) computational resources. It has been deployed in practical systems such as cross-chain bridges [37]. A distributed PCS has *linear speedup* if, with ℓ provers, each prover does $1/\ell$ of the single-prover work. It has *full* linear speedup if proof size and verifier time also remain unchanged. In this setting, full linear speedup is the strongest scaling in ℓ one can hope for without increasing proof size or verifier time.

DeVirgo [37], the only code-based distributed (ML)PCS, achieves full linear speedup but has four drawbacks. First, it inherits quasi-linear opening complexity from its base non-distributed PCS Virgo [43]. Since Virgo is 10× slower in prover time than recent linear-opening PCSs [21,44], DeVirgo needs at least 8 provers to match their prover times, increasing deployment costs. Second, unlike non-code-based distributed MLPCSs [25,35], DeVirgo distributes work only across multiple independent polynomials (one per sub-prover), and it cannot shard a single polynomial across provers. This prevents its use in distributed SNARKs for general circuits, since state-of-the-art prover-efficient schemes [25, 35] use distributed multilinear PIOPs, which require distributed PCSs supporting a single polynomial and, more generally, a linear number of polynomials. Third, DeVirgo lacks accountability [26, 35], so an honest master prover cannot detect malicious sub-provers. Finally, its concrete amortized communication can be hundreds of MBs and is impractical for low-bandwidth networks.

Summarizing, this gap raises two questions.

1. Must code-based MLPCSs trade prover efficiency for proof size and verifier time?
2. Can we design an *accountable* code-based distributed PCS that supports *sharding a single polynomial* across provers?

1.1 Our Contribution

PIP: Shred-to-Shine MLPCS-to-MLPCS Upgrade. We propose PIP, a simple and general framework that upgrades an MLPCS to reduce prover time with at most small increases in proof size and verifier time. Conceptually, PIP provides a metamorphosis from an existing PCS to an improved PCS. Instantiating PIP with code-based and group-based MLPCSs gives two MLPCSs, PIP_{FRI} (based on Fast Reed–Solomon interactive oracle proof of proximity [4], FRI) and PIP_{KZG} (based on the Kate–Zaverucha–Goldberg PCS [22], KZG), respectively, with different tradeoffs. We instantiate PIP in the distributed setting via $\text{DEPIP}_{\text{FRI}}$, showing that the approach extends beyond the single-prover model.

PIP_{FRI}: Efficient FRI-based MLPCS with zero knowledge. Table 1 compares code-based MLPCSs. PIP_{FRI} has linear opening complexity, unlike quasi-linear Virgo [43] and HyperPlonk [14]. Relative to recent FRI-based linear-opening MLPCSs (*e.g.*, PolyFRIM [44], DeepFold [21]), PIP_{FRI} reduces FFT-based committing costs from $O(N \log N)$ to $O(N \log m)$ by splitting N into $m\ell$ for tunable m. Using smaller Merkle trees, it cuts hash-based committing and opening costs from $O(N)$ to $O(m)$. Its opening cost stays at $O(N)$, but the linear term is a single $\mathbb{F}_{\text{rlc}}$ pass, where the pass forms one random linear combination over $\mathbb{F}$. Table 2 reports micro-benchmarks showing 2× and 5× speedups in FFTs and Merkle trees. In our benchmarks, PIP_{FRI} outperforms Orion, a PCS from linear-time-encodable codes over RS-friendly fields, in prover time despite Orion's linear-time encoding.

We equip PIP_{FRI} with zero knowledge, which is essential for zk-SNARKs and is missing from several recent works [3, 14, 42, 44] (*cf.* Table 1).

$\text{DEPIP}_{\text{FRI}}$: Efficient code-based distributed PCS with workload generality and accountability. We propose $\text{DEPIP}_{\text{FRI}}$ with the following advantages (*cf.* Table 3):

1. **Distributed efficiency.** $\text{DEPIP}_{\text{FRI}}$ has *full linear speedup*, whereas DeDory [25] and concurrent works [40, 41] trade off proof size or verifier time to distribute the workload.
2. **Workload generality.** $\text{DEPIP}_{\text{FRI}}$ supports distributing *both* a single polynomial and multiple independent polynomials across provers. Single-polynomial support enables the first code-based distributed SNARK for *general* circuits, whereas DeVirgo [37] targets data-parallel circuits. This relies on the distributed FFT from PIP (*cf.* Section 1.2).
3. **Accountability.** We add mechanisms that make $\text{DEPIP}_{\text{FRI}}$ the first code-based scheme with accountability. Group-based distributed PCSs [26, 35] can often verify each sub-proof directly because it is generated locally. In code-based schemes, each sub-proof is generated jointly by multiple provers, so naïve checks do not isolate a malicious prover.
4. **Performance.** The amortized *prover* and *communication* costs for openings drop to $O(m/\ell)$, attaining optimal $1/\ell$ scaling in ℓ. This provides $O(\ell \log m)$-fold and $O(\ell)$-fold gains over DeVirgo. These gains combine PIP_{FRI}'s complexity improvements over Virgo with an improved distributed FRI sub-protocol. Since many code-based schemes rely on FRI [3, 21], our distributed FRI reduces opening complexity as a standalone component and can be reused in other FRI-based distributed PCSs.

Empirical Evaluation. Our implementations cover PIP_{FRI}, $\text{DEPIP}_{\text{FRI}}$, and (distributed) SNARKs built from them. PIP_{FRI} reduces prover (commit + open) time by 10× versus FRI-based DeepFold and by 3.5× versus Orion from linear-time codes, while keeping proof size and verifier time comparable to DeepFold and cutting both by 15× versus Orion. PIP_{FRI} is the first FRI-based PCS achieving faster prover time than PCSs from linear-time codes. It is memory-efficient among code-based PCSs with polylog proof size, using 10× less memory than DeepFold and 5× less memory than Orion. It is the *first* to support size-2^{29} polynomials on 32 GB RAM. Overall, PIP_{FRI} leads to SNARKs with 2–10× faster provers

Table 1: Comparisons of code-based MLPCSs for polynomials of size $N = m\ell$ with λ-bit security

Scheme	Commit	Open	Verify	Proof size	zk
Brakedown [20]	$O(N)\,\mathbb{F}/\mathsf{H}$	$O(N)\,\mathbb{F}/\mathsf{H}$	$O(\sqrt{N})\,\mathbb{F}/\mathsf{H}$	$O(\lambda\sqrt{N})\,\mathbb{F}+O(\log N)\,\mathsf{H}$	○
Orion [39]	$O(N)\,\mathbb{F}/\mathsf{H}$	$O(N)\,\mathbb{F}/\mathsf{H}$	$O(\sqrt{N})\,\mathbb{F}/\mathsf{H}$, no preprocessing	$O(\lambda\log^2 N)\,\mathsf{H}$	◐
HyperPlonk [14]	$O(N\log N)\,\mathbb{F}, O(N)\,\mathsf{H}$	$O(N\log N)\,\mathbb{F}, O(N)\,\mathsf{H}$	$O(\lambda\log N)\,\mathbb{F}, O(\lambda\log^2 N)\,\mathsf{H}$	$O(\lambda\log N)\,\mathbb{F}, O(\lambda\log^2 N)\,\mathsf{H}$	○
Virgo [43]	$O(N\log N)\,\mathbb{F}, O(N)\,\mathsf{H}$	$O(N\log N)\,\mathbb{F}, O(N)\,\mathsf{H}$	$O(\lambda\log N)\,\mathbb{F}, O(\lambda\log^2 N)\,\mathsf{H}$	$O(\lambda\log N)\,\mathbb{F}, O(\lambda\log^2 N)\,\mathsf{H}$	●
PolyFRIM [44]	$O(N\log N)\,\mathbb{F}, O(N)\,\mathsf{H}$	$O(N)\,\mathbb{F}/\mathsf{H}$	$O(\lambda\log N)\,\mathbb{F}, O(\lambda\log^2 N)\,\mathsf{H}$	$O(\lambda\log N)\,\mathbb{F}, O(\lambda\log^2 N)\,\mathsf{H}$	○
BaseFold [42]	$O(N\log N)\,\mathbb{F}, O(N)\,\mathsf{H}$	$O(N)\,\mathbb{F}/\mathsf{H}$	$O(\lambda\log N)\,\mathbb{F}, O(\lambda\log^2 N)\,\mathsf{H}$	$O(\lambda\log N)\,\mathbb{F}, O(\lambda\log^2 N)\,\mathsf{H}$	○
DeepFold [21]	$O(N\log N)\,\mathbb{F}, O(N)\,\mathsf{H}$	$O(N)\,\mathbb{F}/\mathsf{H}$	$O(\lambda\log N)\,\mathbb{F}, O(\lambda\log^2 N)\,\mathsf{H}$	$O(\lambda\log N)\,\mathbb{F}, O(\lambda\log^2 N)\,\mathsf{H}$	◐
PIP$_{\mathbf{FRI}}$	$O(N\log m)\,\mathbb{F}, O(m)\,\mathsf{H}, O(m)\,\mathsf{H}_\ell$	$O(N)\,\mathbb{F}_{\mathsf{rlc}}, O(m)\,\mathbb{F}/\mathsf{H}$	$O(\lambda\log m+\lambda\ell)\,\mathbb{F}, O(\lambda\log^2 m)\mathsf{H}$	$O(\lambda\log m+\lambda\ell)\,\mathbb{F}, O(\lambda\log^2 m)\mathsf{H}$	●

$\mathbb{F}$ is a field with large multiplicative cosets, and H a hash function. Both denote operation time or size, as context requires. H_ℓ: hashing $O(\ell)$ entries (versus H for a constant number); $\mathbb{F}_{\mathsf{rlc}}$: one-time random linear combination over $\mathbb{F}$; implementations use $\log N \le \ell < \log^2 N$, $m = N/\ell$. ○: No zero-knowledge (zk) variant explicitly provided; ◐: No zk implementation provided; ●: Both are given.

Table 2: Micro-benchmarks of PolyFRIM and PIP$_{\text{FRI}}$

Type	Term	Time	Term	Time
FFT	$O(N\log N)\,\mathbb{F}$	620 ms	$O(N\log m)\,\mathbb{F}$	315 ms
Build MT	$O(N)\,\mathsf{H}$	2,258 ms	$O(m)\,\mathsf{H}, O(m)\,\mathsf{H}_\ell$	468 ms
RLC	-	-	$O(N)\,\mathbb{F}_{\mathsf{rlc}}$	52 ms

MT: Merkle tree; RLC: random linear combination over field $\mathbb{F}$; Figures are for size-2^{20} polynomials over a 64-bit finite field.

than DeepFold and 10–20× smaller proof sizes than Orion.[1]

DEPIP$_{\text{FRI}}$ achieves accountability and *full* linear speedup, unlike DeDory [25] and two concurrent proposals, FRItata [40] and HyperFond [41], which sacrifice proof size or verifier time. Compared to DeVirgo, DEPIP$_{\text{FRI}}$ cuts prover time by 25× and communication by 7×. By supporting distribution of a single polynomial, the resulting distributed SNARK for general circuits runs 8× faster for the prover and 5× faster for the verifier than HyperPianist [25].

PIP$_{\text{KZG}}$: Faster group-based PCS with a shorter SRS. Instantiating PIP with a multilinear KZG variant (mKZG) [28] gives PIP$_{\text{KZG}}$ (*cf.* Table 4). PIP$_{\text{KZG}}$ reduces the SRS size, opening cost, and verifier time. In contrast to *univariate* Bünz *et al.* [13] with a sublinear-size SRS in the generic group model (GGM) [19], PIP$_{\text{KZG}}$ is proved secure in the algebraic group model (AGM).

Relative to mKZG, PIP$_{\text{KZG}}$ has a 16× shorter SRS and 2× faster proving and verification, at a 1.8× increase in proof size (<3KB in most cases). Compared with group-based Dory [24] with transparent setup, PIP$_{\text{KZG}}$ requires a trusted setup but is more efficient by using type-1 groups and avoiding expensive target-group operations, resulting in a 2× faster prover, a 4× faster verifier, and a 10× smaller proof size.

[1] In our commitment implementation, the parallel FFT is up to 17× faster than a conventional baseline (*e.g.*, github.com/arkworks-rs).

1.2 Technical Overview

Construction of PIP. Our framework PIP produces PCS instantiations that improve a base PCS. It embodies a "shred-to-shine" approach: shredding a large multilinear polynomial into smaller sub-polynomials, handling each fragment with an efficient proof method, eliding dominant per-fragment costs to keep proof size and verifier time controlled, and regrouping fragment proofs via batching so performance shines. The same idea also shines across code-based and group-based PCS frameworks by cutting dominant costs in practice, including FFTs, Merkle-tree proofs, and group operations.

Technically, we use *tensor products* to reduce validation of a multilinear polynomial of size $N = m\ell$ to ℓ checks on size-m sub-polynomials. Since these sub-polynomials share the same evaluation point, we use a *batch PCS* to amortize the expensive steps in evaluation proofs.

Construction of PIP$_{\text{FRI}}$. We build PIP$_{\text{FRI}}$ by instantiating PIP with PolyFRIM [44]. The resulting PIP$_{\text{FRI}}$ outperforms DeepFold [21], suggesting that PIP can upgrade an FRI-based PCS instantiation into a state-of-the-art one. To build PIP$_{\text{FRI}}$, we face two challenges. (1) A direct instantiation builds $O(\ell)$ separate Merkle trees for $O(\ell)$ sub-polynomials in the committing phase, undermining the gains. (2) Sending sub-polynomial evaluations to the verifier leaks additional information, so zero knowledge is not preserved by default.

For challenge (1), by analyzing the structural patterns of opened entries in PolyFRIM, we aggregate entries likely opened together into a single leaf. This lets PIP$_{\text{FRI}}$ use a *single* Merkle tree in the commitment phase, reducing prover hash cost from $O(\ell \cdot m)\,\mathsf{H}$ to $O(m)\,\mathsf{H}_\ell + O(m)\,\mathsf{H}$.

For challenge (2), we first build zk-PolyFRIM by merging the tensor-product foldings inherent to PolyFRIM with a masking-coefficient strategy from DeepFold. As it suffices to prove one target evaluation rather than all sub-evaluations in a batch PCS, we exploit the linearity of RS codes to build the virtual target polynomial without sending the sub-evaluations.

Construction of DEPIP$_{\text{FRI}}$. Supporting sharding a single

Table 3: Distributed MLPCSs with ℓ provers supporting only ℓ size-m parallel polynomials or a single size-$m\ell$ polynomial as well

Scheme	Trans.	Workload	$\mathcal{P}_i$: Commit	$\mathcal{P}_i$: Open	Comm.: Commit	Comm.: Open	$\mathcal{V}$ & $\lvert\pi\rvert$	Acct.	Circuit
DemKZG [25, 35]	no	S+B	$O(m\frac{\log\lvert\mathbb{F}\rvert}{\log m})$	$O(m\frac{\log\lvert\mathbb{F}\rvert}{\log m})$	$O(\log m\ell)$	$O(\log m\ell)$	$O(\log m\ell)$	yes	arbitrary
DeDory [25]	yes	S+B	$O(m\frac{\log\lvert\mathbb{F}\rvert}{\log m})$	$O(m\frac{\log\lvert\mathbb{F}\rvert}{\log m})$	$O(\log m\ell)$	$O(\log m\ell)$	$O(\log m^2\ell)$	no	arbitrary
DeVirgo [37]	yes	B	$O(m\log m)$	$O(m\log m)$	$O(m)$	$O(m)$	$O(\lambda\ell+\lambda\log^2 m)$	no	data-parallel
DePIP$_{\text{FRI}}$	**yes**	**S+B**	$O(m\log m)$	$\mathbf{O(m/\ell)}$	$O(m)$	$\mathbf{O(m/\ell)}$	$O(\lambda\ell+\lambda\log^2 m)$	**yes**	**arbitrary**

Trans.: transparent. Workload: S: shard one polynomial across provers; B: parallelize across independent polynomials; S+B = both. $\mathcal{P}_i/\mathcal{V}$: sub-prover/verifier complexity over a field. Note that $O(m)$ group multi-scalar exponentiations, as in DemKZG and DeDory, are equal to $O(m\frac{\log|\mathbb{F}|}{\log m})$ over a field where $\log|\mathbb{F}| = \omega(\log m)$ [20]. $|\pi|$: proof size. Comm.: amortized communication complexity. Acct.: accountability. Circuit: supported SNARK circuit type. The $O(\lambda\ell+\lambda\log^2 m)$ verifier complexity and proof size of DePIP$_{\text{FRI}}$ are smaller than $O(\lambda\log^2(m\ell))$ as ℓ, the prover number, is usually smaller than $\log^2(m\ell)$, where $m\ell$ can be 2^{30} potentially in distributed proofs.

Table 4: Group-based MLPCSs for polynomial of size $N = m\ell$

Scheme	SRS size	Commit	Open	Verifier	Proof size
mKZG	$O(N)\mathbb{G}$	$O(N)\mathbb{G}$	$O(N)\mathbb{G}$	$O(\log N)P$	$O(\log N)\mathbb{G}$
PIP$_{\text{KZG}}$	$O(m)\mathbb{G}$	$O(N)\mathbb{G}$	$O(N)\mathbb{G}_{\text{rlc}}$, $O(m)\mathbb{G}$	$O(\ell)\mathbb{G}$, $O(\log m)P$	$O(\ell)\mathbb{G}$, $O(\log m)\mathbb{G}$

$\mathbb{G}$: operations over group. $\mathbb{G}_{\text{rlc}}$: linear combination between group and field. P: pairing operations. Efficiency: $\mathbb{G}_{\text{rlc}} > \mathbb{G} > P$.

polynomial requires an efficient distributed FFT. Most FRI-based PCSs run linear-size FFTs, so a naïve distributed FFT becomes a bottleneck. Current approaches [36] distribute parallel FFTs but require cross-prover coordination across dependent steps. With a size-$m\ell$ vector spread across ℓ provers, this implies $O(m\log^2 m)$ amortized prover work and $O(m\ell)$ communication. Our shred-to-shine method avoids this coordination by reducing the FFT on a size-$m\ell$ polynomial to ℓ independent size-m FFTs. Each sub-prover then runs its sub-FFT locally, with $O(m\log m)$ work per prover and no communication for the FFT step. More broadly, shred-to-shine turns the size-$m\ell$ FFT into ℓ independent local FFTs by choosing a data layout aligned with tensor-product structure.

The opening phase also benefits from an improved distributed FRI. In round i, provers must compute the codeword $f_i|_{L_i}$ over domain L_i, where f_i is the folded size-$|L_i|$ polynomial for that round. In DeVirgo, each prover $\mathcal{P}_j$ holds a size-$2|L_i|$ polynomial $f_{i-1}^{(j)}$, computes the size-$|L_i|$ polynomial $f_i^{(j)}$ and its full codeword $f_i^{(j)}|_{L_i}$, and exchanges these to assemble $f_i|_{L_i}$. Since each $\mathcal{P}_j$ computes and sends an entire length-$|L_i|$ codeword, amortized prover work and communication are both $O(|L_i|)$. We instead assign $\mathcal{P}_j$ only the j-th segment of $f_i|_{L_i}$ of length $O(|L_i|/\ell)$. We develop a *distributed folded polynomial computation* that computes $f_i|_{L_i}$ jointly and directly, without first computing all $f_i^{(j)}|_{L_i}$ and then merging. This reduces amortized prover work and communication to $O(|L_i|/\ell)$. Combining the distributed FFT and the improved distributed FRI, we build DePIP$_{\text{FRI}}$ over PIP$_{\text{FRI}}$.

Finally, we equip DePIP$_{\text{FRI}}$ with **accountability**, overcoming the challenge of attributing faults in jointly generated proofs. We handle two failure modes, incorrect computation and inconsistent messaging. For the former, each sub-prover produces a local PolyFRIM sub-proof for its sub-polynomial, which uniquely determines the expected DePIP$_{\text{FRI}}$ proof under the shred-to-shine structure. The master prover $\mathcal{P}_0$ reconstructs the honest DePIP$_{\text{FRI}}$ proof and compares it with the received one. For the latter, we route inter-prover messages through $\mathcal{P}_0$ with commitments, so recipients see exactly what senders committed to and $\mathcal{P}_0$ can open and check consistency. In both cases, any discrepancy *pinpoints the exact* malicious sub-prover(s). Accountability preserves the sub-prover work and inter-prover communication asymptotically, while increasing $\mathcal{P}_0$'s cost from $O(\ell\log m)$ to $O(\ell\log^2 m)$, still below each sub-prover's $O(m\log m)$ cost for $m \gg \ell$.

1.3 Related Work

Code-based PCSs all rely on a low-degree test (LDT) to check that a committed vector is within a constant relative Hamming distance to a valid codeword. Standard LDTs include: 1) the direct LDT [2] underlies PCSs [8, 20, 39] from linear-time encodable codes, and 2) FRI(-like) LDTs for RS codes [4, 5, 42] underlie PCSs [21, 42–44]. The direct LDT applies to general constant-relative-distance linear codes but incurs proof size linear in the vector length. FRI-based LDTs yield polylogarithmic proofs but are specialized to RS codes.

PCSs from direct LDT. Brakedown [20] introduces a practical generalized Spielman (GS) code and an MLPCS with a linear-time prover relying on the direct LDT from Ligero [2]. It has square-root verifier complexity and proof size, which can reach tens of MBs. Orion [39] uses a code-based SNARK [43] to recursively prove the verification circuit of Brakedown, reducing the proof size to polylog. Nonetheless, the proofs remain several MBs and verifier complexity stays square-root without trusted preprocessing. Block *et al.* [8] modify the GS code in Brakedown into an expand-accumulate (EA) code with larger code distance and hence smaller proofs while leaving overall complexities unchanged. Blaze [10]

uses a repeat-accumulate-accumulate (RAA) code to tailor an MLPCS. Currently, it only works over binary fields.

PCSs from FRI. The univariate FRI-PCS [32] with linear opening complexity is not directly compatible with multilinear polynomials. Zeromorph [23], a generic MLPCS-from-univariate-PCS approach, requires quasi-linear openings when instantiated with FRI-PCS. Virgo [43] and HyperPlonk [14] build MLPCSs with $O(N \log N)$ openings due to committing size-$O(N)$ auxiliary polynomials.

PolyFRIM [44] refines FRI to avoid such committing, giving the first $O(N)$-opening MLPCS. BaseFold [42] treats FRI as a multilinear polynomial oracle and combines it with the multilinear sum-check, also resulting in $O(N)$ opening. This oracle approach weakens soundness, requiring more verifier queries and leading to proofs near 1 MB.[2] DeepFold [21] improves BaseFold's soundness and has concretely faster prover time and smaller proof size than PolyFRIM. Among these, only Virgo and DeepFold offer zero-knowledge variants.

Distributed PCSs enable multiple sub-provers, each handling a sub-polynomial, to jointly produce commitments and evaluation proofs. Multiple distributed PCSs target different polynomial forms using different primitives. Existing distributed MLPCSs include group-based DemKZG and DeDory [25], which can distribute single and multiple independent polynomials. In contrast, DeVirgo [37], the sole code-based distributed (ML)PCS, supports only the latter. When used for SNARKs, this would limit its usage to only data-parallel circuits, as current linear-prover distributed PIOPs for general circuits [25] involve a single linear-size polynomial.

Concurrent works of FRIttata [40] and HyperFond [41] propose distributed code-based PCSs and target code-based distributed SNARKs for general circuits. Against them, $\text{DePip}_{\text{FRI}}$ and our SNARK have lower prover time. They also do not achieve full linear speedup: the proof size or verifier time grows with the number of provers and is 3–1000× larger than ours in the compared regimes. Our shred-to-shine framework is compatible with any linear code, while HyperFond requires foldable linear codes [42], and FRIttata specializes to a specific PCS. Neither work targets accountability. We also contribute non-distributed instantiations via Pip_{FRI} and Pip_{KZG}. Appendix A provides a more detailed comparison.

2 Preliminaries

$[n]$ denotes $\{1,\dots,n\}$ and $[m,n]$ denotes $\{m,m+1,\dots,n\}$ for non-negative integers $m < n$. Lowercase letters, *e.g.*, f, $\hat{f}$, and $\tilde{f}$, denote polynomials over a finite field $\mathbb{F}$. Let $\tilde{f}(x_1,\dots,x_\mu) = \sum_{\boldsymbol{e}\in\{0,1\}^\mu} f_{\boldsymbol{e}} \cdot \prod_{j=1}^{\mu} x_j^{e_j}$ be multilinear, and let $N = 2^\mu$. Index coefficients as $(f_1,\dots,f_N)$ by $i = \sum_{j=1}^{\mu} e_j 2^{j-1}$ for $\boldsymbol{e} = (e_1,\dots,e_\mu)$, and define the twisted univariate polynomial as $\hat{f}(X) = \sum_{i\in[N]} f_i \cdot X^{i-1}$. Bold letters denote vectors over $\mathbb{F}$. For $\boldsymbol{x} \in \mathbb{F}^n$, x_i is the i-th entry of $\boldsymbol{x}$. For $\boldsymbol{a},\boldsymbol{b} \in \mathbb{F}^n$, $\langle \boldsymbol{a},\boldsymbol{b}\rangle$ is the inner product and $\boldsymbol{a}\otimes\boldsymbol{b}$ is the tensor product. We write $\otimes_{i=1}^{n} \boldsymbol{x_i}$ for $\boldsymbol{x_1}\otimes\cdots\otimes\boldsymbol{x_n}$. Let $N = d^\mu$ be the size parameter for a μ-variate polynomial with degree bound d. Let $L \subseteq \mathbb{F}$ be an evaluation domain. The Reed–Solomon code $\text{RS}[L,\rho] \subseteq \mathbb{F}^{|L|}$ is $\{ f|_L \colon \deg(f) < \rho|L| \}$, where $f|_L$ is the evaluation vector of f on L and $\rho \in (0,1)$ is the code rate.

A Merkle tree commits to a length-N vector with $O(N)$ prover time. For a query set I, opening and verification take $O(|I|\log N)$ time and the proof size is $O(|I|\log N)$. It comprises three algorithms. $\mathsf{rt} \leftarrow \mathsf{MT.Commit}(\boldsymbol{v})$ outputs the Merkle root of vector $\boldsymbol{v}$. Given I, $(\{v_i\}_{i\in I}, \mathsf{path}) \leftarrow \mathsf{MT.Open}(\mathsf{rt}, I, \boldsymbol{v})$ outputs the queried values and an authentication path. $\{0,1\} \leftarrow \mathsf{MT.Verify}(\mathsf{rt}, I, \{v_i\}_{i\in I}, \mathsf{path})$ accepts iff path is consistent with rt and $\{v_i\}_{i\in I}$. Our Merkle trees use collision- and preimage-resistant hash functions.

Argument of Knowledge (AoK). An interactive argument for an NP relation $\mathcal{R}$ includes a public parameter generation algorithm $\mathcal{G}$ and a pair of interactive machines $\langle \mathcal{P},\mathcal{V}\rangle$. $\mathsf{pp} \leftarrow \mathcal{G}(1^\lambda)$ generates the public parameter. $\mathcal{P}$ and $\mathcal{V}$ represent a PPT prover and verifier, respectively. $\mathcal{P}$ tries to convince $\mathcal{V}$ of the existence of $\mathtt{w}$ s.t. $(\mathtt{x},\mathtt{w}) \in \mathcal{R}$ for a statement $\mathtt{x}$ through interaction. An argument of knowledge further allows $\mathtt{w}$ to be efficiently extractable by an extractor.

Definition 2.1 (Argument of knowledge)**.** *$(\mathcal{G}, \langle \mathcal{P},\mathcal{V}\rangle)$ is an interactive argument of knowledge (AoK) for $\mathcal{R}$ if it satisfies:*

- ***Completeness.*** *For every* pp *and all* $(\mathtt{x},\mathtt{w}) \in \mathcal{R}$*, we have* $\Pr[\langle \mathcal{P}(\mathtt{w}), \mathcal{V}\rangle(\mathsf{pp},\mathtt{x}) = 1] = 1$.
- ***Knowledge Soundness.*** *For any PPT $\mathcal{P}^*$, there exists an expected polynomial time extractor $\mathcal{E}^{\mathcal{P}^*}$ with access to $\mathcal{P}^*$'s randomness s.t. for all* $\mathsf{pp} \leftarrow \mathcal{G}(1^\lambda)$ *and* $\mathtt{x}$*,* $\Pr\left[\langle \mathcal{P}^*(), \mathcal{V}\rangle(\mathsf{pp},\mathtt{x}) = 1 \wedge (\mathtt{x},\mathtt{w}) \notin \mathcal{R} \;\middle|\; \mathtt{w} \leftarrow \mathcal{E}^{\mathcal{P}^*}(\mathsf{pp},\mathtt{x})\right] \le \mathsf{negl}(\lambda)$*. $\mathcal{E}^{\mathcal{P}^*}$ means $\mathcal{E}$ has access to the randomness of $\mathcal{P}^*$.*

A public-coin interactive AoK can be transformed into non-interactive via the Fiat–Shamir transformation. An AoK is succinct with proof size sublinear to $|\mathtt{w}|$.

Polynomial Commitment Scheme (PCS) and Distribution. A PCS is specified by the following algorithms [20].

- $\mathsf{pp} \leftarrow \mathsf{Gen}(1^\lambda, d, \mu)$ takes security parameter λ, degree bound d, and arity μ, and outputs public parameters pp.
- $C \leftarrow \mathsf{Com}(\mathsf{pp}, f)$ outputs a commitment C to polynomial f.
- $b \leftarrow \mathsf{VerPoly}(\mathsf{pp}, C, f)$ verifies that C is a valid commitment to f and outputs $b \in \{0,1\}$.
- $\mathsf{Eval}(\mathsf{pp}, C, \boldsymbol{x}, y; f)$ is an (interactive) argument between a prover $\mathcal{P}$ and a verifier $\mathcal{V}$. $\mathcal{P}$ and $\mathcal{V}$ hold C, $\boldsymbol{x}$, and y, and $\mathcal{P}$ additionally holds f. $\mathcal{P}$ proves that C commits to a polynomial f with degree bound d in μ variables and that $f(\boldsymbol{x}) = y$, and $\mathcal{V}$ outputs $b \in \{0,1\}$.

A PCS satisfies the following security properties.

- **Completeness.** For any f with $\mathsf{pp} \leftarrow \mathsf{Gen}(1^\lambda, d, \mu)$, $C \leftarrow \mathsf{Com}(\mathsf{pp}, f)$, and $f(\boldsymbol{x}) = y$, $\Pr[\mathsf{Eval}(\mathsf{pp}, C, \boldsymbol{x}, y; f) = 1] = 1$.

[2] BaseFold offers an "FRI-like" LDT for foldable linear codes, which generalizes RS codes. PCS with this LDT is field-agnostic but less efficient.

- **Polynomial Binding.** For all $\mathsf{pp} \leftarrow \mathsf{Gen}(1^\lambda, d, \mu)$ and any PPT adversary $\mathcal{A}$, the following probability is $\mathsf{negl}(\lambda)$:

$$\Pr\left[\begin{array}{c} \mathsf{pp} \leftarrow \mathsf{Gen}(1^\lambda, d, \mu), (C, f_0, f_1) \leftarrow \mathcal{A}(\mathsf{pp}) \\ b_0 \leftarrow \mathsf{VerPoly}(\mathsf{pp}, C, f_0), b_1 \leftarrow \mathsf{VerPoly}(\mathsf{pp}, C, f_1) : \\ b_0 = b_1 = 1 \wedge f_0 \neq f_1 \end{array}\right].$$

- **Knowledge Soundness.** Eval is an AoK for the NP relation $\mathcal{R}_{\mathsf{Eval}}(\mathsf{pp})$ specified below given $\mathsf{pp} \leftarrow \mathsf{Gen}(1^\lambda, d, \mu)$: $\{(C, \boldsymbol{x}, y; f) : f \in \mathbb{F}[d^\mu] \wedge f(\boldsymbol{x}) = y \wedge \mathsf{VerPoly}(\mathsf{pp}, C, f) = 1\}$.

We say a PCS is zero knowledge [43] if it satisfies:

- **Honest-Verifier Zero Knowledge.** For all PPT adversaries $\mathcal{A}$, randomness $r_{\mathcal{A}}$, $\mathsf{pp} \leftarrow \mathsf{Gen}(1^\lambda, d, \mu)$, and size-$d^\mu$ polynomial f, there exists a simulator $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2)$ s.t. the experiments in Figure 1 are computationally indistinguishable, *i.e.*, $|\Pr[\mathsf{Real}_{\mathcal{A},f}(\mathsf{pp}) = 1] - \Pr[\mathsf{Ideal}_{\mathcal{A},\mathcal{S}^{\mathcal{A}}}(\mathsf{pp}) = 1]| \leq \mathsf{negl}(\lambda)$. If the difference is 0, we call it perfect zero knowledge.

$\mathsf{Real}_{\mathcal{A},f}(\mathsf{pp})$:	$\mathsf{Ideal}_{\mathcal{A},\mathcal{S}^{\mathcal{A}}}(\mathsf{pp})$:
1: $C \leftarrow \mathsf{Com}(\mathsf{pp}, f)$	1: $C \leftarrow \mathcal{S}_1(1^\lambda, \mathsf{pp}, r_{\mathcal{A}})$
2: $\boldsymbol{x} \leftarrow \mathcal{A}(C, \mathsf{pp})$	2: $\boldsymbol{x} \leftarrow \mathcal{A}(C, \mathsf{pp})$
3: $y \leftarrow f(\boldsymbol{x})$	3: $y \leftarrow f(\boldsymbol{x})$
4: $1 \leftarrow \langle \mathcal{P}(f), \mathcal{A} \rangle(\mathsf{pp}, C, \boldsymbol{x}, y)$	4: $1 \leftarrow \langle \mathcal{S}_2, \mathcal{A} \rangle(\mathsf{pp}, C, \boldsymbol{x}, y)$, given oracle access to y
5: $b \leftarrow \mathcal{A}$ and output b	5: $b \leftarrow \mathcal{A}$ and output b

Figure 1: Experiments for zero knowledge in PCSs

For distributed PCSs, we model a master prover that interacts with the verifier and coordinates sub-provers. In prior group-based distributed PCSs, sub-provers communicate only with the master prover. In our setting, sub-provers may also communicate with each other. Beyond the security properties of PCSs, we consider accountability [35]. An honest master prover can identify *all* sub-provers that produce incorrect proofs or send messages inconsistent with their local inputs.

FRI. *Fast Reed–Solomon interactive oracle proof of proximity* (FRI) [4] is a low-degree test for RS codes. Let L be a multiplicative coset of size M, and let $f|_L \in \mathbb{F}^M$ be the evaluation vector of f on L. FRI proves that $f|_L$ is δ-close to $\mathrm{RS}[L, \rho]$ in relative Hamming distance, for proximity parameter δ.

FRI has $\sigma = \log_2 M + 1$ rounds. In round $i \in [\sigma]$, the prover $\mathcal{P}$ sends a Merkle tree commitment to a codeword $f_{i-1}|_{L_{i-1}}$ of length $M/2^{i-1}$, where $f_0 = f$ and $L_0 = L$, or to f_σ when $i = \sigma$. The verifier $\mathcal{V}$ sends a random challenge α_i if $i \neq \sigma$. For $i \in [\sigma - 1]$, we have $L_i = \{x^2 \mid x \in L_{i-1}\}$ and $f_i(X) = g_{i-1}(X) + \alpha_i \cdot h_{i-1}(X)$, where g_{i-1}, h_{i-1} satisfy $f_{i-1}(X) = g_{i-1}(X^2) + X \cdot h_{i-1}(X^2)$. In the last round, $\mathcal{V}$ picks $q = O(\lambda)$ random queries in L and queries at most $2q$ evaluations on each $f_{i-1}|_{L_{i-1}}$ for consistency checks. Specifically, for query $\beta \in L$, $\mathcal{P}$ opens $\{f_{i-1}(\pm\beta^{2^{i-1}})\}_{i \in [\sigma]}$. $\mathcal{V}$ checks whether $(\pm\beta^{2^{i-1}}, f_{i-1}(\pm\beta^{2^{i-1}}))$ and $(\beta^{2^i}, f_i(\beta^{2^i}))$ are consistent with α_i, *i.e.*, these evaluation pairs lie on a common line. This shows the consistency of $\{f_i\}_{i \in [0,\sigma]}$.

Batch FRI [5] enables a single low-degree test for multiple RS codewords evaluated on the same domain. Each codeword is padded to a common maximum degree. $\mathcal{P}$ and $\mathcal{V}$ then run one FRI on a random linear combination of these codewords.

HyperPlonk, PolyFRIM, and Rolling Batch FRI. Given a size-$N = 2^\mu$ multilinear polynomial $\tilde{f}_0$ and an evaluation point $\boldsymbol{x}$, the prover $\mathcal{P}$ in HyperPlonk [14, §B] folds $\tilde{f}_0$'s twisted univariate polynomial f_0 to obtain and commit to $f_1, \ldots, f_{\mu+1}$, where f_i has size $N/2^i$ for $i \in [\mu + 1]$ and $f_{\mu+1}$ is a constant. This folding procedure mirrors FRI, except the folding parameter is x_i rather than the random challenge α_i. Next, $\mathcal{P}$ proves the low-degree properties of $f_0, \ldots, f_{\mu+1}$ via batch FRI. $\mathcal{V}$ also checks consistency as in FRI, replacing α_i with x_i.

Directly running batch FRI for $f_0, \ldots, f_{\mu+1}$ requires padding all polynomials to size N, leading to quasi-linear opening complexity for committing them. PolyFRIM [44] proposes a *rolling batch FRI* specialized to the LDTs of these polynomials. It observes that these codewords have the same code rate, and that the j-th FRI-folded polynomial derived from f_1 is evaluated over the same domain as f_{j+1}. It therefore combines their LDTs directly via random linear combination. Applying this observation recursively reduces the quasi-linear opening complexity to linear. We recall the rolling batch FRI in Protocol 1.

3 PIP: Shred-to-Shine MLPCS from MLPCS

We propose PIP, a PCS-improved-from-PCS (PIP) transformation that turns a base MLPCS into an upgraded MLPCS. The goal is to reduce dominant PCS costs, such as FFTs, Merkle-tree construction, and group multi-scalar multiplications. We reduce one evaluation claim for a size-N multilinear polynomial $\tilde{f}$, where $N = m\ell$, to ℓ independent claims for size-m sub-polynomials. Write the coefficients of $\tilde{f}$ as $(f_1, \ldots, f_N)$ and split them into ℓ consecutive blocks of size m. For $j \in [\ell]$, define a sub-polynomial $\tilde{f}_j$ with coefficient vector $\boldsymbol{f}_j = (f_{(j-1)m+1}, \ldots, f_{jm})$.

Without loss of generality, assume m and ℓ are powers of two. Let the evaluation point be $\boldsymbol{x} = (x_1, \ldots, x_{\log_2(m\ell)})$ and let $\tilde{f}(\boldsymbol{x}) = y$. Define a matrix $U \in \mathbb{F}^{\ell \times m}$ whose j-th row is $\boldsymbol{f}_j$. Define $\boldsymbol{v} = \otimes_{k \in [\log_2 m]}(1, x_k) \in \mathbb{F}^m$ and $\boldsymbol{w} = \otimes_{k \in [\log_2 \ell]}(1, x_{k+\log_2 m}) \in \mathbb{F}^\ell$. Then $y = \boldsymbol{w} U \boldsymbol{v}^\top$ [20, 33], or

$$y = \langle (\tilde{f}_1(\boldsymbol{v}), \ldots, \tilde{f}_\ell(\boldsymbol{v})), \boldsymbol{w} \rangle. \tag{1}$$

Equation (1) drives the construction. Instead of committing to $\tilde{f}$ directly, the prover commits to $\tilde{f}_1, \ldots, \tilde{f}_\ell$ and proves their claimed evaluations at the common point $\boldsymbol{v}$. The verifier checks these proofs and accepts if Equation (1) holds for the public vector $\boldsymbol{w}$. Since all $\tilde{f}_j$ share the same evaluation point, we open them via a batch PCS. Batch opening is supported by leading schemes under diverse assumptions, *e.g.*,

Protocol 1 (The Rolling Batch FRI [44]).
Inputs: *Secret codewords $f_0|_{L_0}, \ldots, f_\sigma|_{L_\sigma}$ such that $f_i|_{L_i} \in \mathrm{RS}[L_i, \rho]$ for all $i \in [0,\sigma]$. Public multiplicative cosets $\{L_i\}_{i\in[0,\sigma]}$ such that $L_{i+1} = \{x^2 \mid x \in L_i\}$ for all $i \in [0,\sigma]$. Without loss of generality, assume $\deg(f_\sigma) = 1$.*
Prover $\mathcal{P}$ and verifier $\mathcal{V}$ prove $f_i|_{L_i} \in \mathrm{RS}[L_i,\rho]$ for all $i \in [0,\sigma]$ as follows:

1. *$\mathcal{P} \to \mathcal{V}$: $\{C_i\}_{i\in[0,\sigma]}$ where $C_i \leftarrow \mathsf{MT.Commit}(f_i|_{L_i})$.*
2. *$\mathcal{P}$ sets $f'_0 = f_0$. For each $i \in [0,\sigma]$, $\mathcal{P}$ and $\mathcal{V}$ do:*
   (a) *$\mathcal{P}$: decomposes $f'_i(X)$ uniquely as $\ell_i(X^2) + X \cdot r_i(X^2)$.*
   (b) *$\mathcal{V} \to \mathcal{P}$: a random challenge $\alpha_i \in \mathbb{F}$.*
   (c) *$\mathcal{P} \to \mathcal{V}$: $\mathsf{MT.Commit}(p_{i+1}|_{L_{i+1}})$ when $i \neq \sigma$, or $f'_{\sigma+1}$ when $i = \sigma$, where $p_{i+1}(X) = \ell_i(X) + \alpha_i \cdot r_i(X)$.*
   (d) *$\mathcal{P}$: computes $f'_{i+1}(X) = \ell_i(X) + \alpha_i \cdot r_i(X) + \alpha_i^2 \cdot f_{i+1}(X)$, which equals $p_{i+1}(X) + \alpha_i^2 \cdot f_{i+1}(X)$.*
3. *$\mathcal{P}$ and $\mathcal{V}$ repeat the following steps $q = O(\lambda)$ times:*
   (a) *$\mathcal{V} \to \mathcal{P}$: a random challenge $\beta \in \mathbb{F}$.*
   (b) *$\mathcal{P}$: opens $\{f_i(\pm\beta^{2^i})\}_{i\in[0,\sigma]}$ and $\{p_i(\pm\beta^{2^i})\}_{i\in[\sigma]}$ via $\mathsf{MT.Open}$. $\mathcal{V}$ checks them via $\mathsf{MT.Verify}$. For each $i \in [\sigma]$, $\mathcal{V}$ checks that $(\beta^{2^{i-1}}, f'_{i-1}(\beta^{2^{i-1}}))$, $(-\beta^{2^{i-1}}, f'_{i-1}(-\beta^{2^{i-1}}))$, and $(\alpha_i, p_i(\beta^{2^i}))$ lie on a common line. Note: $f'_i(\pm\beta^{2^i})$ can be computed from $p_i(\pm\beta^{2^i})$ and $f_i(\pm\beta^{2^i})$.*

KZG [22], Bulletproofs [11], Dark [12], and FRI-PCS [32]. Let n denote the base PCS size parameter, and in our setting $n = m$. If the base PCS has opening time $t_\mathcal{P}(n)$, verifier time $t_\mathcal{V}(n)$, and proof size $\pi(n)$ for a size-n polynomial, then for ℓ committed polynomials batch PCS yields opening time $O(\ell) + t_\mathcal{P}(n)$, verifier time $O(\ell) + t_\mathcal{V}(n)$, and proof size $O(\ell) + \pi(n)$. Protocol 2 gives PIP with batch PCS, and Theorem 3.1 states the resulting advantages, with proofs in Appendix B.1.

Theorem 3.1 (PIP: Shred-to-shine MLPCS)**.** *Let $N = m\ell$. Protocol 2 is an MLPCS for size-N multilinear polynomials with the following properties.*
- *The committing phase decomposes into ℓ independent commitments to size-m polynomials, and these commitments can be computed in parallel.*
- *The prover opening time is $O(N) + t_\mathcal{P}(m)$. The verifier time is $O(\ell) + t_\mathcal{V}(m)$. The proof size is $O(\ell) + \pi(m)$.*

Moreover, under the base-PCS cost regimes below, PIP *shines.*
- *If the base committing time satisfies $t_\mathsf{C}(N) = O(N \log N)$, then the committing time of* PIP *is $O(N \log m)$.*
- *If the base opening time satisfies $t_\mathcal{P}(N) = O(N \log N)$, then setting $m = O(N/\log N)$ and $\ell = O(\log N)$ yields prover opening time $O(N)$.*
- *If the base verifier time and proof size satisfy $t_\mathcal{V}(N) = O(\log^c N)$ and $\pi(N) = O(\log^c N)$ for a constant $c \geq 1$, then setting $m = O(N/\log^c N)$ and $\ell = O(\log^c N)$ keeps the verifier time and proof size asymptotically unchanged.*
- *If the base PCS uses a size-$O(N)$ SRS, then* PIP *uses a size-$O(m)$ SRS.*

Protocol 2 (PIP). *Let $\mathsf{PC_b} = (\mathsf{Gen_b}, \mathsf{Com_b}, \mathsf{VerPoly_b}, \mathsf{Eval_b})$ be a batch MLPCS.*
- $\mathsf{pp} \leftarrow \mathsf{Gen}(1^\lambda, 2, \mu)$. *pp includes $(\mathbb{F}, \mu, m, \ell, \mathsf{pp_b})$, where $\mathsf{pp_b} = \mathsf{Gen_b}(1^\lambda, 2, \log m)$ and $m\ell = 2^\mu$.*
- $C \leftarrow \mathsf{Com}(\mathsf{pp}, \tilde{f})$. *Given $\tilde{f}$ and its coefficient $\tilde{f}$, $\mathcal{P}$ does:*
  1. *Partition $\tilde{f}$ into ℓ consecutive blocks of length m. For each $j \in [\ell]$, define $\tilde{f}_j$ as the size-m multilinear polynomial with coefficients $(f_{(j-1)m+1}, \ldots, f_{jm})$.*
  2. *Run $C_j \leftarrow \mathsf{Com_b}(\mathsf{pp_b}, \tilde{f}_j)$, $j \in [\ell]$. Output $(C_1, \ldots, C_\ell)$.*
- $b \leftarrow \mathsf{VerPoly}(\mathsf{pp}, C, \tilde{f})$. *$\mathcal{V}$ parses $\tilde{f}$ and runs $b_j \leftarrow \mathsf{VerPoly_b}(\mathsf{pp_b}, C_j, \tilde{f}_j)$, $j \in [\ell]$. Output 1 iff $b_j = 1, \forall j \in [\ell]$.*
- $b \leftarrow \mathsf{Eval}(\mathsf{pp}, C, \boldsymbol{x}, y; \tilde{f})$. *Given $\boldsymbol{x}$ and $y = \tilde{f}(\boldsymbol{x})$,*
  1. *$\mathcal{P}$ and $\mathcal{V}$: compute public vectors $\boldsymbol{v} = \otimes_{k\in[\log m]}(1, x_k)$ and $\boldsymbol{w} = \otimes_{k\in[\log \ell]}(1, x_{k+\log m})$.*
  2. *$\mathcal{P} \to \mathcal{V}$: $\boldsymbol{u} = (u_1, \ldots, u_\ell)$, where $u_j = \tilde{f}_j(x_1, \ldots, x_{\log m})$.*
  3. *$\mathcal{P}$ and $\mathcal{V}$: run the batch evaluation protocol $\mathsf{Eval_b}$ to prove $\tilde{f}_j(x_1, \ldots, x_{\log m}) = u_j$ for all $j \in [\ell]$.*
  4. *$\mathcal{V}$: accepts iff $\mathsf{Eval_b}$ outputs 1 and $\langle \boldsymbol{u}, \boldsymbol{w} \rangle = y$.*

Appendix B.1 provides the full proofs. Appendix B.2 extends multilinear PIP to the univariate setting and explains a general approach for zero knowledge.

3.1 Comparisons with Other PCSs

Prior PCSs use tensor-product reductions [14, 20] and multi-instance batching [9, 33]. PIP combines these two tools by reducing one size-N claim to ℓ evaluations at a common point and then proving them via a batch PCS. To our knowledge, prior PCSs do not combine them in this way. This yields a simple black-box construction compatible with diverse PCS instantiations supporting batch opening, with efficiency captured by Theorem 3.1. We compare PIP with Hyrax [33] and representative code-based PCSs [7, 20, 39].

Comparison with Hyrax. Hyrax is a PIP-like MLPCS framework, but it relies on commitment homomorphism. It arranges the coefficients of $\tilde{f}$ into a matrix U and writes the claim as $y = \boldsymbol{w} U \boldsymbol{v}^\top$ for public $\boldsymbol{v}, \boldsymbol{w}$. Committing to the rows of U via group-based vector commitments lets the verifier homomorphically derive a commitment to $\boldsymbol{u} = \boldsymbol{w}U$. An inner-product argument (IPA) [11] then proves $\langle \boldsymbol{u}, \boldsymbol{v} \rangle = y$.

In contrast, PIP exploits the tensor-product structure of $\boldsymbol{v}$ to compute $U\boldsymbol{v}^\top$, *i.e.*, the ℓ row evaluations at the common point $\boldsymbol{v}$, and then checks their aggregation by $\boldsymbol{w}$. This avoids assuming commitment homomorphism and thereby extends

to code-based PCSs. Instantiating PIP with an IPA-based PCS [11] recovers Hyrax, so Hyrax is a special case of PIP.

Comparison with Code-based PCSs. Brakedown also uses $y = \boldsymbol{w}U\boldsymbol{v}^\top$, but relies on row encoding and a direct LDT, where the direct LDT lets the verifier check that a claimed $\boldsymbol{u} = \boldsymbol{w}U$ is consistent with the encoded matrix via a few column queries, and hence validate $\langle \boldsymbol{u}, \boldsymbol{v}\rangle = y$. The proof therefore contains $\boldsymbol{u}$ and the opened columns, yielding $O(\sqrt{N})$ proof size and verifier cost (and $O(N)$ prover time) in its standard parameterization. In contrast, PIP replaces row encoding and direct LDT with a tensor-product reduction plus batch opening, so an FRI-based instantiation inherits polylogarithmic proof size and verifier time from the chosen base PCS.

Orion employs an FRI-based recursive SNARK [43] to prove the size-$O(\sqrt{N})$ verification circuit of Brakedown. Concretely, the recursive SNARK forces Orion to encode the columns of U, producing double-dimension codewords whose Merkle openings lead to several-MB proofs [14]. The prover also incurs overhead from proving a large verification circuit that depends on the (randomly) queried matrix columns [14]. Rather than a recursive SNARK, PIP uses PCS-level recursion (recursive PCSs); with an FRI-based instantiation it avoids double-dimension codewords. This makes PIP-based PCSs potentially more efficient than Orion.

Ligero++ is a SNARK for general circuits and is PCS-like since polynomial evaluation is a linear constraint. It encodes a witness matrix U row-wise and reduces checking to inner-product queries on public vectors and sampled columns, proved via the FRI-based IPA Virgo [43]. As in Brakedown, the induced double-dimension codewords make Merkle openings large. PIP uses recursive PCSs rather than IPA-style recursion, which unlocks a wider choice of inner PCS instantiations (including faster FRI-based ones) than fixing Virgo [43]. The prover time gap can exceed $10\times$ [44].

3.2 PIP_{KZG}: Instantiation of PIP with mKZG

With mKZG as PC_b, Protocol 2 yields PIP_{KZG} with a slight modification such that the commitment homomorphism removes the need to send $\boldsymbol{u}$, further reducing the proof size. As shown in Table 4, PIP_{KZG} improves over mKZG by shifting partial expensive group exponentiations and pairings into cheaper linear combinations between the group and field. This comes at a larger proof size. Appendix C presents the design and evaluation of PIP_{KZG}.

4 PIP_{FRI}: Instantiation of PIP with Batch and Zero-Knowledge PolyFRIM

This section builds PIP_{FRI} by combining PIP and PolyFRIM, before which we build batch and zero-knowledge PolyFRIM.

4.1 Batch and Zero-Knowledge PolyFRIM

Batch PolyFRIM. Given multilinear $\tilde{f}^{(1)}, \ldots, \tilde{f}^{(\ell)}$, our goal is to prove $\tilde{f}^{(j)}(\boldsymbol{x}) = y_j$ for all $j \in [\ell]$. In the i-th FRI-like round, folded $\hat{f}_i^{(1)}, \ldots, \hat{f}_i^{(\ell)}$ use the same folding parameter x_i. By RS linearity, we batch the ℓ openings into one for $\sum_{j=1}^{\ell} \gamma^j \tilde{f}^{(j)}(\boldsymbol{x})$ for a random γ. If $\tilde{f}^{(j)}(\boldsymbol{x}) \neq y_j$ for some j, $\sum_{j=1}^{\ell} \gamma^j \tilde{f}^{(j)}(\boldsymbol{x}) = \sum_{j=1}^{\ell} \gamma^j y_j$ holds with probability at most $\ell/|\mathbb{F}|$. PolyFRIM also requires low-degree tests for $\hat{f}^{(1)}, \ldots, \hat{f}^{(\ell)}$, which we batch via batch FRI [5] on their random linear combination.

A direct batch PolyFRIM yields proof size and verification time $O((\lambda+\ell)\log N + \log^2 N)$. The $O(\ell \log N)$ term comes from opening ℓ Merkle trees for the size-$O(N)$ codewords of $\hat{f}^{(1)}, \ldots, \hat{f}^{(\ell)}$ in the first round. This scales poorly in ℓ.

To remove this term, we use an *optimized Merkle tree commitment* that packs same-location entries across codewords into one leaf. We denote this commitment by $\mathsf{MT.Com}([\hat{f}^{(1)}|_{L_0}, \ldots, \hat{f}^{(\ell)}|_{L_0}])$. In PolyFRIM (and in FRI), the verifier queries the same set of positions in each codeword in a given round. Thus, if position i is queried in $\hat{f}^{(j)}|_{L_0}$ for any $j \in [\ell]$, then position i is queried in $\hat{f}^{(j')}|_{L_0}$ for all $j' \in [\ell]$. Therefore, the prover can pack $\{(\hat{f}^{(j)}|_{L_0})_i\}_{j\in[\ell]}$ into one Merkle leaf and open a single tree. With this packing rule, the first-round opening cost drops from ℓ trees to one, giving $O(\lambda \log N + \ell + \log^2 N)$ proof size and verification time.

For a size-N multilinear polynomial $\tilde{f}$, PolyFRIM can leak information from queries to $\hat{f}$ in the first round and from queries to folded polynomials in later rounds. We mask $\tilde{f}$ in two steps. First, we adopt the padding technique from DeepFold [21]. We extend $\tilde{f}$ to a masked polynomial $\tilde{f}'$ by padding the coefficient vector with random values so that $\tilde{f}(\boldsymbol{x}) = \tilde{f}'(\boldsymbol{x}, 0)$ for $\boldsymbol{x} = (x_1, \ldots, x_\mu)$. Since PolyFRIM processes this public vector one variable at a time, an extra round handles the appended variable fixed to 0.

Second, we add an independent random masking polynomial $\tilde{s}$. The prover commits to $\tilde{f}'$ and to $\tilde{s}$, claims $y = \tilde{f}(\boldsymbol{x})$ and $y_s = \tilde{s}(\boldsymbol{x})$, and runs PolyFRIM on $\hat{f}' + \alpha \cdot \hat{s}$ for a verifier-chosen random α. The verifier checks the claimed evaluation $y + \alpha \cdot y_s$. In the first round, opened evaluations are randomized by α, and in later rounds $\hat{s}$ prevents leakage.

We use batch zk-PolyFRIM as the sub-protocol to build PIP_{FRI}. Due to page limitations, we omit the formal scheme.

4.2 Construction of ZK-PIP_{FRI}

This subsection presents zk-PIP_{FRI}. A naïve instantiation of Protocol 2 using batch zk-PolyFRIM is not zero knowledge. The reason is that Protocol 2 sends $\boldsymbol{u} = (\tilde{f}_1(\boldsymbol{x'}), \ldots, \tilde{f}_\ell(\boldsymbol{x'}))$. These sub-polynomial evaluations are linear functions of $\tilde{f}$'s coefficients and thus leak information about $\tilde{f}$.

In zk-PIP_{FRI}, we avoid proving (and sending) all entries of $\boldsymbol{u}$, and we prove only the target evaluation $\tilde{f}(\boldsymbol{x}) = y$. We use RS linearity to define a virtual target polynomial and a virtual

evaluation that the verifier can check without seeing $\boldsymbol{u}$. Let $\tilde{f}_1,\ldots,\tilde{f}_\ell$ be the size-m sub-polynomials of $\tilde{f}$ as in Protocol 2. Let $\boldsymbol{x'}=(x_1,\ldots,x_{\log m})$ and let $\boldsymbol{w}=\otimes_{k\in[\log\ell]}(1,x_{k+\log m})$. We mask each $\tilde{f}_j$ by extending it to a size-$2m$ polynomial $\tilde{f}^{(j)}$ such that $\tilde{f}^{(j)}(\boldsymbol{x'},0)=\tilde{f}_j(\boldsymbol{x'})$. By Equation (1), this gives $\tilde{f}(\boldsymbol{x})=\sum_{j=1}^{\ell}w_j\cdot\tilde{f}^{(j)}(\boldsymbol{x'},0)$. Since $\boldsymbol{w}$ is public, we define the virtual polynomial $\tilde{f}'=\sum_{j=1}^{\ell}w_j\cdot\tilde{f}^{(j)}$. It suffices to prove the single claim $\tilde{f}'(\boldsymbol{x'},0)=\tilde{f}(\boldsymbol{x})$ via zk-PolyFRIM.

Protocol 3 gives the formal algorithms for zk-PIP_{FRI}. In Step 3, the prover commits to the FRI-like folded polynomials $\hat{f}_1,\ldots,\hat{f}_{\sigma+1}$ as in zk-PolyFRIM for the input polynomial $\tilde{f}'+\alpha\tilde{s}$ and its twisted univariate form. In Step 4, the verifier checks the folding consistency. In Step 5, the verifier checks the low-degree properties of $\{\hat{f}_i\}_{i\in[\sigma+1]}$ and of the input polynomials $\hat{f}^{(1)},\ldots,\hat{f}^{(\ell)}$.

Theorem 4.1. *Protocol 3 is a zk-PCS with committing complexity $O(N\log m)\,\mathbb{F}+O(m)\,\mathsf{H}_\ell+O(m)\,\mathsf{H}$. It has opening complexity $O(N)\,\mathbb{F}_{\mathsf{rlc}}+O(m)\,\mathbb{F}+O(m)\,\mathsf{H}$. Its verifier complexity and proof size are $O(\lambda\log m+\lambda\ell+\lambda\log^2 m)$.*

Complexity Analysis. *Commit.* Commitment generation runs ℓ FFTs on length-m vectors, costing $O(\ell m\log m)=O(N\log m)\,\mathbb{F}$. It also builds one Merkle tree with $O(m)$ leaves, each packing ℓ entries, costing $O(m)\,\mathsf{H}_\ell+O(m)\,\mathsf{H}$.

Open. In round i, $\mathcal{P}$ derives $\hat{f}_i|_{L_i}$ and its Merkle commitment from $\hat{f}_{i-1}|_{L_{i-1}}$ in $O(|L_{i-1}|)$ time. Summed over the $\log m$ rounds, this is $O(\sum_i|L_i|)=O(m)$ field and hash operations. $\mathcal{P}$ also runs a rolling batch FRI over the folded polynomials, which adds another $O(\sum_{i=0}^{\log m-1}m/2^i)=O(m)$ field and hash operations. Finally, forming $\hat{f}_0$ from $\hat{f}^{(1)},\ldots,\hat{f}^{(\ell)}$ costs a one-time $O(N)\,\mathbb{F}_{\mathsf{rlc}}$ random linear combination.

Proof size and verify. $\mathcal{V}$ needs to open two Merkle trees at $O(\lambda)$ positions per round over $\log m$ rounds. This contributes to a proof size of $O(\lambda\log m)\,\mathbb{F}$ and $O(\lambda\log^2 m)\,\mathsf{H}$. In the first round, $\mathcal{V}$ additionally opens $O(\lambda)$ positions of the single packed Merkle tree with $O(m)$ leaves, contributing to a proof size of $O(\lambda\ell)\,\mathbb{F}$ and $O(\lambda\log m)\,\mathsf{H}$. The verifier complexity is of the same order for processing the transcript linearly.

Security Analysis. Appendix D proves completeness, binding, knowledge soundness, and zero knowledge.

Zero-knowledge intuition. First-round queries to each $\hat{f}^{(j)}$ are randomized by same-degree masking. From the second round on, folded-polynomial queries are randomized by $\hat{s}(X)$.

5 DePIP$_{\text{FRI}}$: Distributed Version of PIP$_{\text{FRI}}$

We introduce DePIP$_{\text{FRI}}$, a distributed PIP$_{\text{FRI}}$ that improves generality, accountability, and efficiency via a distributed FRI with lower opening communication and computation.

Protocol 3 (zk-PIP$_{\text{FRI}}$).

- $\mathsf{pp}\leftarrow\mathsf{Gen}(1^\lambda,2,\mu)$: pp *includes* $\mathbb{F}$, $\ell=O(\log N)$, $m=2^\mu/\ell$, *and a multiplicative coset L_0 of size $O(m)$. Fix a code rate $\rho\in(0,1)$ and choose $|L_0|=2m/\rho$.*
- $C\leftarrow\mathsf{Com}(\mathsf{pp},\tilde{f})$: *Given $\tilde{f}$ as input, the prover $\mathcal{P}$ does:*
  1. *Split $\tilde{f}$'s coefficients into ℓ size-m vectors, as in Protocol 2. Add m random entries to each. Denote these size-$2m$ twisted univariate polynomials by $\hat{f}^{(1)},\ldots,\hat{f}^{(\ell)}$.*
  2. *Pick a random size-$2m$ polynomial $\tilde{s}$. Compute $C\leftarrow\mathsf{MT.Com}([\hat{f}^{(1)}|_{L_0},\ldots,\hat{f}^{(\ell)}|_{L_0},\hat{s}|_{L_0}])$ obtained via the optimized Merkle tree commitment in Section 4.1.*
- $b\leftarrow\mathsf{VerPoly}(\mathsf{pp},C,\hat{f}^{(1)}|_{L_0},\ldots,\hat{f}^{(\ell)}|_{L_0},\hat{s}|_{L_0},\tilde{f})$: *Decode $\hat{f}^{(1)}|_{L_0},\ldots,\hat{f}^{(\ell)}|_{L_0}$ to obtain $\hat{f}_1,\ldots,\hat{f}_\ell$. Output 1 iff $C=\mathsf{MT.Com}([\hat{f}^{(1)}|_{L_0},\ldots,\hat{f}^{(\ell)}|_{L_0},\hat{s}|_{L_0}])\wedge\tilde{f}=\sum_{j\in[\ell]}w_j\tilde{f}^{(j)}$.*
- $b\leftarrow\mathsf{Eval}(\mathsf{pp},C,\boldsymbol{x},y,y_s;(\hat{f}^{(1)},\ldots,\hat{f}^{(\ell)},\hat{s}))$: *Let $\sigma=\log m$.*
  1. *$\mathcal{P}\to\mathcal{V}$: claimed evaluations $y=\tilde{f}(\boldsymbol{x})$ and $y_s=\tilde{s}(\boldsymbol{x})$.*
  2. *$\mathcal{V}\to\mathcal{P}$: a random challenge $\alpha\in\mathbb{F}$ for zero knowledge.*
  3. *$\mathcal{P}\to\mathcal{V}$: $\{C_i\}_{i\in[\sigma]},\hat{f}_{\sigma+1}$. For $i=1,\ldots,\sigma+1$:*
     - *a. Uniquely write $\hat{f}_{i-1}(X)=\hat{g}_{i-1}(X^2)+X\cdot\hat{h}_{i-1}(X^2)$.*
     - *b. Set $\hat{f}_i(X)\leftarrow\hat{g}_{i-1}(X)+x_i\cdot\hat{h}_{i-1}(X)$, where $x_{\sigma+1}=0$.*
     - *c. If $i\le\sigma$, set $L_i\leftarrow\{x^2:x\in L_{i-1}\}$ and compute $C_i\leftarrow\mathsf{MT.Com}(\hat{f}_i|_{L_i})$. Set $\hat{f}_{\sigma+1}$ as the constant $y+\alpha y_s$.*
  4. ***Consistency check:** Repeat below for $q=O(\lambda)$ times:*
     - *a. $\mathcal{V}\to\mathcal{P}$: a random challenge $\beta\in L_0$.*
     - *b. $\mathcal{P}\to\mathcal{V}$: via $\mathsf{MT.Open}$, open from C $\{\hat{f}^{(j)}(\pm\beta)\}_{j\in[\ell]}$ and $\hat{s}(\pm\beta)$, and open from $\{C_i\}_{i\in[\sigma]}$ $\{\hat{f}_i(\pm\beta^{2^i})\}_{i\in[\sigma]}$.*
     - *c. $\mathcal{V}$: verify all openings by $\mathsf{MT.Verify}$. Let $\beta_i:=\beta^{2^i}$ for $i\in[0,\sigma]$. Set $\hat{f}_0(\pm\beta_0):=\sum_{j\in[\ell]}w_j\cdot\hat{f}^{(j)}(\pm\beta_0)+\alpha\hat{s}(\pm\beta_0)$. For each $i\in[\sigma]$, check that $(\pm\beta_{i-1},\hat{f}_{i-1}(\pm\beta_{i-1}))$ and $(x_i,\hat{f}_i(\beta_i))$ lie on a common line. Also check that $(\pm\beta_\sigma,\hat{f}_\sigma(\pm\beta_\sigma))$ and $(x_{\sigma+1},\hat{f}_{\sigma+1})$ lie on a common line.*
  5. ***Validity check:** $\mathcal{P}$ and $\mathcal{V}$ invoke rolling batch FRI to prove the low-degree properties, i.e., $\forall i\in[\sigma+1]$, $\hat{f}_{i-1}|_{L_{i-1}}\in\mathrm{RS}[L_{i-1},\rho]$. $\hat{f}_0$ is proved via a batch FRI on $\hat{f}^{(1)},\ldots,\hat{f}^{(\ell)}$ and $\hat{s}$. The FRI uses $q'=O(\lambda)$ queries over $\{L_{i-1}\}$, reusing the consistency-check queries.*
  6. *$\mathcal{V}$ outputs 1 iff all checks pass and $\hat{f}_{\sigma+1}=y+\alpha y_s$.*

5.1 Improved Distributed FRI

We formalize and improve the distributed FRI underlying DeVirgo [37]. Let $\mathcal{P}_0$ be the master prover and $\mathcal{P}_1,\ldots,\mathcal{P}_\ell$ be sub-provers, where $\mathcal{P}_j$ holds $f^{(j)}$. Given a coset L and code rate ρ, distributed FRI proves $f^{(j)}|_L\in\mathrm{RS}[L,\rho]$ for all $j\in[\ell]$. The key idea is a single batch FRI on $\sum_{j\in[\ell]}\alpha^{j-1}\cdot f^{(j)}(X)$, which requires distributed commitment and opening across all rounds. We use i for rounds, and $j,k\in[\ell]$ for sub-provers.

Distributed commitment generates a Merkle commitment to $f^{(1)}|_L, \ldots, f^{(\ell)}|_L$, where $\mathcal{P}_j$ holds $f^{(j)}|_L$. Let $d = |L|/\ell$. For each $k \in [\ell]$, each $\mathcal{P}_j$ sends the segment $f^{(j)}|_L[(k-1)d+1], \ldots, f^{(j)}|_L[kd]$ to $\mathcal{P}_k$. In DeVirgo, $\mathcal{P}_k$ builds a local Merkle tree whose leaves are these $d\ell$ field elements. We use the Merkle tree packing rule in Section 4.1. For a fixed position h, the ℓ entries $\{f^{(j)}|_L[h]\}_{j\in[\ell]}$ are queried together, or none is queried at all. Thus, each leaf of $\mathcal{P}_k$ packs the same-index entries across the ℓ codewords, *i.e.*, $\{f^{(j)}|_L[h]\}_{j\in[\ell]}$ for all $h \in [(k-1)d+1, kd]$. Each $\mathcal{P}_k$ commits to a local sub-tree with d packed leaves and sends its sub-root $\mathsf{H}^{(k)}$ to $\mathcal{P}_0$. Then, $\mathcal{P}_0$ aggregates $\{\mathsf{H}^{(k)}\}_{k\in[\ell]}$ into a global tree with m leaves.

Distributed openings require provers to compute the i-th round polynomial $f_i(X) = g_{i-1}(X) + \alpha_i h_{i-1}(X)$ for $i \in [\log m]$. Let $f_0(X) = \sum_{j\in[\ell]} \alpha^{j-1} \cdot f^{(j)}(X)$. These polynomials satisfy $f_{i-1}(X) = g_{i-1}(X^2) + X \cdot h_{i-1}(X^2)$, where the odd-even decomposition is unique, and α_i is the i-th round challenge. In the first round, $f_0(X)$ can be computed via the distributed commitment above. DeVirgo does not describe the remaining rounds but assumes the same method can be applied. However, applying the first-round method above in later rounds is less efficient, as detailed below.

A less efficient approach. Sub-provers locally compute sub-codewords, exchange segments, and commit. Fix a round i and let α_i be the (common) challenge. For each $j \in [\ell]$, let $f_{i-1}^{(j)}(X) = g_{i-1}^{(j)}(X^2) + X \cdot h_{i-1}^{(j)}(X^2)$ be the unique odd-even decomposition. Define the local folded polynomial $f_i^{(j)}(X) = g_{i-1}^{(j)}(X) + \alpha_i \cdot h_{i-1}^{(j)}(X)$. Let the target folded polynomial be $f_i(X) = g_{i-1}(X) + \alpha_i \cdot h_{i-1}(X)$. Assuming $f_{i-1}(X) = \sum_{j\in[\ell]} f_{i-1}^{(j)}(X)$, uniqueness implies $g_{i-1}(X) = \sum_{j\in[\ell]} g_{i-1}^{(j)}(X)$ and $h_{i-1}(X) = \sum_{j\in[\ell]} h_{i-1}^{(j)}(X)$. Thus, $f_i(X) = \sum_{j\in[\ell]} f_i^{(j)}(X)$.

Let the i-th domain be L_i and let $d_i = |L_i|/\ell$. Each $\mathcal{P}_j$ sends the segment $f_i^{(j)}|_{L_i}[(k-1)d_i+1], \ldots, f_i^{(j)}|_{L_i}[kd_i]$ to $\mathcal{P}_k$. Then $\mathcal{P}_k$ aggregates $f_i|_{L_i}[h] = \sum_{j\in[\ell]} f_i^{(j)}|_{L_i}[h]$ for all $h \in [(k-1)d_i+1, kd_i]$ and builds a commitment $\mathsf{H}_i^{(k)}$. Finally, the sub-provers jointly obtain a commitment to the i-th round codeword $f_i|_{L_i}$.

Complexity. For $i > 1$, each $\mathcal{P}_j$ computes $f_i^{(j)}|_{L_i}$ locally, which costs $O(|L_i|)$ time. Each $\mathcal{P}_j$ additionally aggregates $d_i = |L_i|/\ell$ positions of $f_i|_{L_i}$, which costs $O(|L_i|/\ell)$ time. Each $\mathcal{P}_j$ receives $(\ell-1)d_i = (\ell-1)|L_i|/\ell$ field elements from other sub-provers, so the total communication per prover is $O(|L_i|)$.

Our distributed opening. To build the distributed Merkle tree commitment in round i, $\mathcal{P}_j$ must output its segment of $f_i|_{L_i}$ of length $d_i = |L_i|/\ell$, namely $f_i|_{L_i}[(j-1)d_i+1], \ldots, f_i|_{L_i}[jd_i]$. In the less efficient approach, $\mathcal{P}_j$ derives these d_i entries from all ℓ local contributions per entry via $\{f_i^{(k)}|_{L_i}[(j-1)d_i+1], \ldots, f_i^{(k)}|_{L_i}[jd_i]\}_{k\in[\ell]}$, which costs $d_i \cdot \ell$ field elements.

We build an optimal distributed method using the folding locality of FRI. For any $a \in [|L_i|]$, there exists a function F such that $f_i|_{L_i}[a] = F(f_{i-1}|_{L_{i-1}}[a], f_{i-1}|_{L_{i-1}}[a+|L_{i-1}|/2], \alpha_i)$.

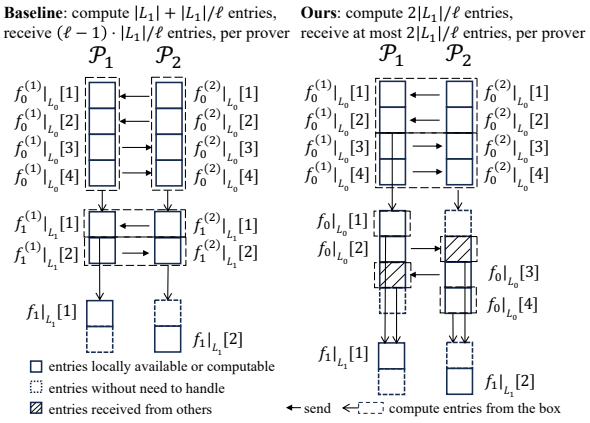


Figure 2: Comparison of two approaches in the first-round distributed opening for $\ell = 2$ and $|L_0| = 4$ (so $|L_1| = 2$)

Hence, as long as the vector $f_{i-1}|_{L_{i-1}}$ is *collectively held* by the provers, $\mathcal{P}_k$ can compute its d_i outputs (*e.g.*, $f_i|_{L_i}[a]$ for some a) from the corresponding $2d_i$ predecessor entries (*e.g.*, $f_{i-1}|_{L_{i-1}}[a]$ and $f_{i-1}|_{L_{i-1}}[a+|L_{i-1}|/2]$), fetched from the provers holding them. It suffices to justify collective holding for $i = 1$, since later rounds repeat the same step. In the distributed commitment, $\mathcal{P}_k$ holds $\{f_0^{(j)}|_{L_0}[(k-1)d_0+1], \ldots, f_0^{(j)}|_{L_0}[kd_0]\}_{j\in[\ell]}$, and can compute $f_0|_{L_0}[h] = \sum_{j\in[\ell]} \alpha^{j-1} \cdot f_0^{(j)}|_{L_0}[h]$ for all $h \in [(k-1)d_0+1, kd_0]$. Thus, $f_0|_{L_0}$ is evenly distributed, with $\mathcal{P}_k$ holding its own segment.

Complexity. In round i, $\mathcal{P}_j$ computes $2d_i = 2|L_i|/\ell$ entries in $O(|L_i|/\ell)$ time, instead of $O(|L_i| + |L_i|/\ell)$. It receives at most $2|L_i|/\ell$ field elements, instead of $O(|L_i|)$.

Figure 2 illustrates the first round of distributed opening. In the beginning, $\mathcal{P}_1$ holds $f_0^{(1)}|_{L_0}$ and $f_0^{(2)}|_{L_0}[1], f_0^{(2)}|_{L_0}[2]$; $\mathcal{P}_2$ holds $f_0^{(2)}|_{L_0}$ and $f_0^{(1)}|_{L_0}[3], f_0^{(1)}|_{L_0}[4]$. To compute and commit $f_1|_{L_1}$, the baseline computes $|L_1| + |L_1|/\ell = 3$ entries and receives $(\ell-1)\cdot|L_1|/\ell = 1$ entries per sub-prover. Our method computes $|L_0|/\ell = 2|L_1|/\ell = 2$ entries and receives at most $2|L_1|/\ell = 2$ entries per sub-prover. Here, "at most" accounts for local availability of the required predecessors (*e.g.*, $\mathcal{P}_1$ holds $f^{(1)}|_{L_0}[1]$ and $f^{(2)}|_{L_0}[1]$ and can compute $f|_{L_0}[1] = f^{(1)}|_{L_0}[1] + \alpha f^{(2)}|_{L_0}[1]$). For $\ell \geq 4$, $(\ell-1)|L_1|/\ell > 2|L_1|/\ell$, so communication and computation strictly decrease.

Protocol 4 formalizes distributed FRI from our distributed commitment and opening. It includes distributed commitment (Step 1), distributed opening for the first $\log(m/\ell)$ rounds (Step 3), and non-distributed opening for the last $\log \ell$ rounds (Step 4). After $\log(m/\ell)$ rounds, each sub-prover holds a codeword of length ℓ, so they send these codewords to $\mathcal{P}_0$, who completes the remaining rounds non-distributedly.

Theorem 5.1. *Protocol 4 is a distributed FRI. In the committing and opening phases, respectively, the sub-prover complexity is $O(m \log m)$ and $O(m/\ell)$, and the amortized commu-*

Protocol 4 (Distributed FRI). *Let $\mathcal{P}_0$ be the master prover and $\mathcal{P}_1,\dots,\mathcal{P}_\ell$ the sub-provers.*
Secret inputs: ℓ size-m polynomials $f^{(1)},\dots,f^{(\ell)}$.
Public inputs: a size-$O(m)$ multiplicative coset L_0.
For $i\in[0,\log m]$, define $L_i=\{x^2 \mid x\in L_{i-1}\}$ and $d_i=|L_i|/\ell$.
To prove $f^{(j)}|_{L_0}\in \mathrm{RS}[L_0,\rho]$ for all $j\in[\ell]$, $\mathcal{P}$ and $\mathcal{V}$ run:

1. ***(Commit)***
   (a) *$\mathcal{P}_j$: computes $f^{(j)}|_{L_0}$ and sends to each $\mathcal{P}_k$ the block $f^{(j)}|_{L_0}[(k-1)d_0+1],\dots,f^{(j)}|_{L_0}[kd_0]$.*
   (b) *$\mathcal{P}_j\to\mathcal{P}_0$: a Merkle sub-root $\mathsf{H}_0^{(j)}$ with d_0 leaves under the packing rule (Section 4.1), where the leaf at index $h\in[(j-1)d_0+1, jd_0]$ contains $f^{(1)}|_{L_0}[h]\|\cdots\|f^{(\ell)}|_{L_0}[h]$.*
   (c) *$\mathcal{P}_0\to\mathcal{V}$: root C obtained by aggregating $\{\mathsf{H}_0^{(j)}\}_{j\in[\ell]}$.*
2. ***(Challenge)*** *$\mathcal{V}\to\mathcal{P}_0$: random $\alpha\in\mathbb{F}$ for batch FRI. $\mathcal{P}_0$ forwards α to all $\mathcal{P}_j$.*
3. ***(Open)*** *For $i\in[\sigma]$, $\sigma=\log(m/\ell)$:*
   (a) *$\mathcal{V}\to\mathcal{P}_0$: random $\alpha_i\in\mathbb{F}$. $\mathcal{P}_0$ assigns α_i to each $\mathcal{P}_j$.*
   (b) *$\mathcal{P}_0\to\mathcal{V}$: a Merkle tree commitment C_i to $f_i|_{L_i}$.*
      i. *$\mathcal{P}_j$: computes its segment $\{f_i|_{L_i}[h]\}_{h=(j-1)d_i+1}^{jd_i}$ by the FRI folding rule $f_i|_{L_i}[a]=F(f_{i-1}|_{L_{i-1}}[a], f_{i-1}|_{L_{i-1}}[a+|L_{i-1}|/2],\alpha_i)$. For $i=1$, each $\mathcal{P}_j$ first forms its segment of $f_0|_{L_0}[n]=\sum_{k\in[\ell]}\alpha^k\cdot f^{(k)}|_{L_0}[n]$ for the needed indices n.*
      ii. *$\mathcal{P}_j\to\mathcal{P}_0$: a Merkle sub-root $\mathsf{H}_i^{(j)}$ over its segment. If $i=\sigma$, $\mathcal{P}_j$ also sends its segment values so that $\mathcal{P}_0$ holds the full $f_i|_{L_i}$.*
      iii. *$\mathcal{P}_0$: aggregates $\{\mathsf{H}_i^{(j)}\}_{j\in[\ell]}$ into C_i*
4. ***(Local Open)*** *$\mathcal{P}_0$ and $\mathcal{V}$ proceed non-distributedly for $i\in[\sigma+1,\log m+1]$:*
   (a) *$\mathcal{V}\to\mathcal{P}_0$: random challenge $\alpha_i\in\mathbb{F}$.*
   (b) *$\mathcal{P}_0\to\mathcal{V}$: $\mathcal{P}_0$ locally folds $f_{i-1}|_{L_{i-1}}$ into $f_i|_{L_i}$ and commits via a Merkle tree, or sends $f_{i=(\log m+1)}$.*
5. ***(Verification)*** *Repeat the following for $q=O(\lambda)$ times:*
   (a) *$\mathcal{V}\to\mathcal{P}_0$: random query $\beta\in L_0$. This means that $\mathcal{V}$ needs to query $f_i(\pm\beta^{2^i})$ for all $i\in[0,\log m]$.*
   (b) *$\mathcal{P}_j\to\mathcal{P}_0$: For $i\in[\log(m/\ell)]$, if $f_i(\pm\beta^{2^i})$ corresponds to root $\mathsf{H}_i^{(j)}$, $\mathcal{P}_j$ sends values and tree path to $\mathcal{P}_0$.*
   (c) *$\mathcal{P}_0\to\mathcal{V}$: For last $\log\ell$ rounds, $\mathcal{P}_0$ opens C_i herself. Otherwise, $\mathcal{P}_0$ forwards the above values and tree path.*
   (d) *$\mathcal{V}$: verifies Merkle trees by $\mathsf{MT.Verify}$. Check if $2f_i(\beta^{2^i})=f_{i-1}(\beta^{2^{i-1}})+f_{i-1}(-\beta^{2^{i-1}})+\alpha_i/\beta^{2^{i-1}}\cdot(f_{i-1}(\beta^{2^{i-1}})-f_{i-1}(-\beta^{2^{i-1}}))$. Accept iff all pass.*

nication complexity (per sub-prover) is $O(m)$ and $O(m/\ell)$. The master prover complexity is $O(\ell\log m)$. The proof size and verifier complexity are $O(\lambda\log m+\lambda\ell+\lambda\log^2 m)$.

Proof. **Complexity.** We first analyze the costs.

Prover. Sub-prover $\mathcal{P}_j$ runs an FFT to compute $f^{(j)}|_{L_0}$ in $O(m\log m)$ time. In round i, $\mathcal{P}_j$ receives at most $2|L_i|/\ell$ entries, computes its $|L_i|/\ell$ entries of $f_i|_{L_i}$, and builds or opens a local Merkle tree with $|L_i|/\ell$ leaves. Summed over rounds, this costs $O(\sum_i|L_i|/\ell)=O(m/\ell)$. Each sub-prover spends $O(m\log m)$ time in committing and $O(m/\ell)$ time in opening.

For the master prover $\mathcal{P}_0$, receiving ℓ sub-roots and aggregating them costs $O(\ell)$ time per round. Over $O(\log m)$ rounds, this is $O(\ell\log m)$. In the last $\log\ell$ rounds, $\mathcal{P}_0$ acts as a non-distributed prover on size-ℓ codewords, costing $O(\ell\log\ell)\subseteq O(\ell\log m)$. Overall, $\mathcal{P}_0$ costs $O(\ell\log m)$.

Proof size and verifier complexity. In the first round, the verifier receives λ queried leaves from an optimized Merkle tree with $O(m)$ leaves, where each leaf contains ℓ field elements. This contributes $O(\lambda\ell)\,\mathbb{F}+O(\lambda\log m)\,\mathsf{H}$ to both proof size and verification. In each subsequent round, the verifier receives $O(\lambda)$ field elements and verifies their Merkle paths, costing $O(\lambda)\,\mathbb{F}+O(\lambda\log m)\,\mathsf{H}$ per round. Over $O(\log m)$ rounds, this totals $O(\lambda\log m)\,\mathbb{F}+O(\lambda\log^2 m)\,\mathsf{H}$. Hence, the total proof size and verifier complexity follow.

Communication. Each sub-prover sends ℓ segments of size $O(m/\ell)$ in round 1, so the total communication is $O(m\ell)$, *i.e.*, amortized $O(m)$ per sub-prover. In round i, each sub-prover receives at most $2|L_i|/\ell$ entries, for amortized communication $O(\sum_i|L_i|/\ell)=O(m/\ell)$ over all opening rounds. The master prover receives ℓ sub-roots per round, for total $O(\ell\log m)$.

Security. As with several other distributed PCSs [25, 26, 37], we assume each sub-prover is honest here. We discuss the accountability of DePip$_{\mathrm{FRI}}$ in Section 5.3. From the verifier's view, the transcript is the same as that of a batch FRI [5] (up to the packed Merkle commitment). Hence, the security properties directly follow from the batch FRI. □

5.2 Construction of DePip$_{\mathrm{FRI}}$

We build DePip$_{\mathrm{FRI}}$ by applying the shred-to-shine method of Pip$_{\mathrm{FRI}}$ in the distributed setting. We split a size-N multilinear polynomial $\tilde{f}$ into ℓ size-m polynomials $\tilde{f}^{(1)},\dots,\tilde{f}^{(\ell)}$. This replaces the FFT of $\hat{f}$ by ℓ local FFTs for $\hat{f}^{(1)},\dots,\hat{f}^{(\ell)}$, avoiding a single hard-to-distribute linear-size FFT [36].

Distributed committing. Let $\mathcal{P}_1,\dots,\mathcal{P}_\ell$ be the sub-provers, where $\mathcal{P}_j$ holds $\tilde{f}^{(j)}$. Each $\mathcal{P}_j$ runs an FFT to obtain $\hat{f}^{(j)}|_{L_0}$. Then committing is exactly Step 1 of Protocol 4.

Distributed opening. We realize the opening of DePip$_{\mathrm{FRI}}$ by reusing the distributed committing and opening of distributed FRI (Steps 1, 3, 4) and then extending the resulting batch FRI into a distributed rolling batch FRI. Step 3 of Pip$_{\mathrm{FRI}}$ follows FRI folding, except that the folding parameters are the evaluation-point coordinates rather than random challenges. Accordingly, the sub-provers mimic Steps 3 and 4 of Protocol 4 to generate, commit, and open $\hat{f}_1,\dots,\hat{f}_\sigma$, where

Protocol 5 (DEPIP$_{\text{FRI}}$). *Let $\mathcal{P}_0$ be the master prover and $\mathcal{P}_1,\dots,\mathcal{P}_\ell$ the sub-provers.*

- $\mathsf{pp} \leftarrow \mathsf{Gen}(1^\lambda, 2, \mu)$: pp *includes $\mathbb{F}$, power-of-two integers $\ell = O(\log N)$ and $m = N/\ell$, and a multiplicative coset L_0. Let $\sigma = \log m$. For $i \in [\sigma]$, define $L_i = \{x^2 \mid x \in L_{i-1}\}$.*
- $C \leftarrow \mathsf{Com}(\mathsf{pp}, \tilde{f}, \tilde{f}^{(1)}, \dots, \tilde{f}^{(\ell)})$: *Given $\tilde{f}$, split it into ℓ size-m polynomials $\tilde{f}^{(1)},\dots,\tilde{f}^{(\ell)}$ as in Protocol 2. Assign $\tilde{f}^{(j)}$ to $\mathcal{P}_j$. The sub-provers invoke Step 1 of Protocol 4 to commit to $\tilde{f}^{(1)},\dots,\tilde{f}^{(\ell)}$ and obtain C.*
- $b \leftarrow \mathsf{VerPoly}(\mathsf{pp}, C, [\hat{f}^{(1)}|_{L_0}, \dots, \hat{f}^{(\ell)}|_{L_0}], \tilde{f})$. *This algorithm is the same as in* PIP$_{\text{FRI}}$*, and we omit it.*
- $b \leftarrow \mathsf{Eval}(\mathsf{pp}, C, \boldsymbol{x}, y; (\hat{f}^{(1)}, \dots, \hat{f}^{(\ell)}))$.
  1. *$\mathcal{P}_0$ and the sub-provers run Steps 3–4 of Protocol 4 to generate $\hat{f}_1,\dots,\hat{f}_\sigma$ and their commitments, and to derive the constant $\hat{f}_{\sigma+1}$. Each $\hat{f}_i$ is computed as in Step 3 of Protocol 3, with the replacements below.*
     (a) $\alpha^j \mapsto w_j$ for $j \in [\ell]$, where $\boldsymbol{w} = \otimes_{k=1}^{\log \ell}(1, x_{k+\log m})$.
     (b) $\alpha_i \mapsto x_i$ for $i \in [\sigma]$.
     At the end, $\mathcal{P}_j$ holds the j-th part of each $\hat{f}_i|_{L_i}$.
  2. *$\mathcal{P}_0$ and the sub-provers prove low degree as follows.*
     (a) Run Steps 3–4 of Protocol 4 on $\{\hat{f}^{(j)}\}_{j\in[\ell]}$.
     (b) Extend the same steps into a distributed rolling batch FRI on $\hat{f}_1,\dots,\hat{f}_\sigma$. In round i, $\mathcal{P}_j$ holds the j-th part of $\hat{f}_{i-1}|_{L_{i-1}}$ and $f'_{i-1}|_{L_{i-1}}$. For $i = 1$, set $f'_0|_{L_0} = \sum_{j=1}^{\ell} \alpha^j \cdot \hat{f}^{(j)}|_{L_0}$. Each $\mathcal{P}_j$ folds f'_{i-1} to obtain its part of $f_i|_{L_i}$, then computes its part of $f'_i|_{L_i}$ via Equation (2).
  3. *The verifier performs the same consistency and low-degree checks as in* PIP$_{\text{FRI}}$*. Merkle openings follow Step 5 of Protocol 4.*

$\sigma = \log m$. Thereafter, they run a batch FRI over $\hat{f}^{(1)},\dots,\hat{f}^{(\ell)}$ and a rolling batch FRI over $\hat{f}_1,\dots,\hat{f}_\sigma$.

We next extend the distributed batch FRI on $\hat{f}^{(1)},\dots,\hat{f}^{(\ell)}$ to a distributed rolling batch FRI that also incorporates $\hat{f}_1,\dots,\hat{f}_\sigma$. After the distributed folding produces $\hat{f}_1|_{L_1},\dots,\hat{f}_\sigma|_{L_\sigma}$, each $\mathcal{P}_j$ holds the j-th segment of $\hat{f}_i|_{L_i}$ of length $d_i = |L_i|/\ell$ for all $i \in [\sigma]$. In round i, distributed batch FRI also gives $\mathcal{P}_j$ the j-th segment of $f_i|_{L_i}$, where f_i is the i-th folded codeword for $\hat{f}^{(1)},\dots,\hat{f}^{(\ell)}$. To run rolling batch FRI, the sub-provers additionally compute $f'_{i+1}|_{L_{i+1}}$ such that

$$\begin{aligned} f'_{i+1}(X) &= g_i(X) + \alpha_i \cdot h_i(X) + \alpha_i^2 \cdot \hat{f}_{i+1}(X) \\ &= f_{i+1}(X) + \alpha_i^2 \cdot \hat{f}_{i+1}(X), \end{aligned} \tag{2}$$

where $f_i(X)$ is uniquely decomposed as $g_i(X^2) + X \cdot h_i(X^2)$ and $f_{i+1}(X) = g_i(X) + \alpha_i \cdot h_i(X)$. Since $\mathcal{P}_j$ can compute its segment of $f_{i+1}|_{L_{i+1}}$ and already holds its segment of $\hat{f}_{i+1}|_{L_{i+1}}$, it can also compute its segment of $f'_{i+1}|_{L_{i+1}}$. This adds only $O(|L_i|/\ell)$ work per round over distributed FRI, leaving the overall complexity unchanged.

With the distributed rolling batch FRI, DEPIP$_{\text{FRI}}$ is built in Protocol 5. For simplicity, we give a non-zk version. Extending it to zero knowledge is natural. In Com, each prover holds a sub-polynomial of the target $\tilde{f}$. They invoke the distributed commitment of distributed FRI on the twisted univariate representations of these sub-polynomials.

In Step 1 of Eval, the provers run a variant of distributed FRI to generate $\hat{f}_1,\dots,\hat{f}_{\sigma+1}$ and commit to $\hat{f}_1,\dots,\hat{f}_\sigma$. The folding parameters are derived from the evaluation point, as in PIP$_{\text{FRI}}$ and PolyFRIM [44]. In Step 2, the provers invoke a generalized distributed rolling batch FRI to prove the low-degree properties of the input and folded polynomials.

In Step 3, the verifier checks the folded-polynomial consistency via a variant of distributed FRI. The verifier also checks the low-degree properties of the input and folded polynomials.

Theorem 5.2. *Protocol 5 is a distributed MLPCS. The complexities are the same as Protocol 4 in Theorem 5.1.*

Security and Complexity. Assuming each sub-prover is honest, DEPIP$_{\text{FRI}}$ is equivalent to PIP$_{\text{FRI}}$ from the verifier's view. The security properties follow from Theorem 4.1. DEPIP$_{\text{FRI}}$ invokes distributed FRI twice, and one instance is generalized to a distributed rolling batch FRI without affecting complexities. The complexity claim follows from Theorem 5.1, with an $O(m \log m)$ sub-prover complexity, an $O(m\ell)$ communication complexity in total, and a proof size or verifier complexity of $O(\lambda \log m + \lambda\ell + \lambda \log^2 m)$.

5.3 Achieving Accountability

Section 5.2 assumes honest sub-provers. We next add accountability so an honest $\mathcal{P}_0$ can flag malicious sub-provers. Our goal is to identify all such sub-provers.

In DEPIP$_{\text{FRI}}$, all messages to the verifier go through $\mathcal{P}_0$. Thus, $\mathcal{P}_0$ can locally check (i) Merkle openings and (ii) algebraic consistency of queried triples, *e.g.*, checking in round i (Step 4 of Protocol 3) whether $\hat{f}_i(\beta^{2^i}) = F(\hat{f}_{i-1}(-\beta^{2^{i-1}}), \hat{f}_{i-1}(\beta^{2^{i-1}}), x_i)$.[3] Merkle-path failures are easy to attribute because each sub-prover opens its own subtree. We thus focus on algebraic inconsistencies.

Even if $\mathcal{P}_0$ detects an inconsistent triple, it may not identify the culprit because the three queried values may come from different sub-provers. Figure 3 illustrates three representative cases. In Case (1), malicious $\mathcal{P}_{j_1}$ commits and opens $\hat{f}_2(\beta^4)$ but sends a fake value $\hat{f}_2^*(\beta^4)$ to an honest recipient. In Case (2), malicious $\mathcal{P}_{j_4}$ computes a wrong $\hat{f}_1(\beta^2)$. In Case (3), even if the middle value is correct, a wrong input value in the first round can still evade a naive "exclude all inconsistent holders" rule. In short, malice manifests in two ways: (M-I) Sending inconsistent entries to different recipients (Case (1)), or (M-II) Incorrect computation (Cases (2)–(3)).

To address M-I, we route inter-prover messages through $\mathcal{P}_0$. Any entry that $\mathcal{P}_{j_1}$ sends to $\mathcal{P}_{j_2}$ is first sent to $\mathcal{P}_0$, who forwards

[3] $\mathcal{P}_0$ can also check the cross-round consistency conditions in the rolling batch FRI (*cf.* Step 5 in Protocol 3). We omit similar checks for brevity.

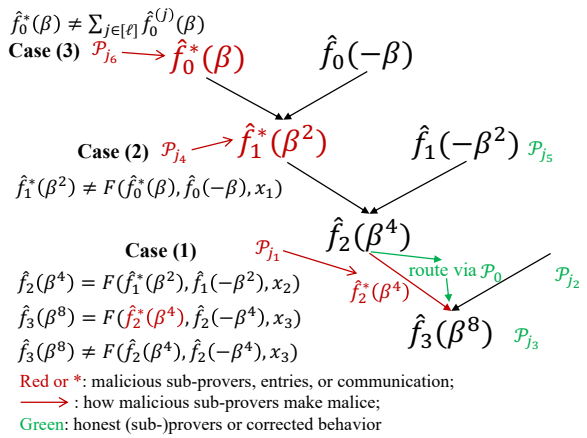


Figure 3: Three malice patterns in distributed opening: Red marks malicious provers/values ($*$ = incorrect), and green marks honest provers and messages routed via $\mathcal{P}_0$.

it to $\mathcal{P}_{j_2}$ (green arrows in Case (1) of Figure 3). Then $\mathcal{P}_0$ checks that each forwarded value matches the committed and opened value, so each message is bound to a single value.

We address M-II by requiring each sub-prover to additionally send a PolyFRIM sub-proof for its sub-polynomial.[4] These sub-proofs determine the expected DePIP$_{\text{FRI}}$ proof under the shred-to-shine structure. Using the received sub-proofs, $\mathcal{P}_0$ constructs an honest reference proof and compares it with the assembled DePIP$_{\text{FRI}}$ proof. Any mismatch identifies the malicious sub-prover(s).

The concrete modifications are described as follows.

1. For each $j \in [\ell]$, sub-prover $\mathcal{P}_j$ runs Com of PolyFRIM on $\tilde{f}^{(j)}$ and sends the commitment to $\mathcal{P}_0$. Sub-prover $\mathcal{P}_j$ also sends the claimed value $\tilde{f}^{(j)}(\boldsymbol{v})$ to $\mathcal{P}_0$. Master prover $\mathcal{P}_0$ sets the target claim as $\tilde{f}(\boldsymbol{x}) = \sum_{j\in[\ell]} w_j \cdot \tilde{f}^{(j)}(\boldsymbol{v})$. See Equation (1) for $\boldsymbol{v}$ and $\boldsymbol{w}$. All provers invoke Com of DePIP$_{\text{FRI}}$.
2. Run the following two protocols in parallel.
   (a) All provers invoke Eval of DePIP$_{\text{FRI}}$ with verifier-sampled challenges. All inter-prover messages $\mathcal{P}_j \to \mathcal{P}_k$ are routed as $\mathcal{P}_j \to \mathcal{P}_0 \to \mathcal{P}_k$. The master prover $\mathcal{P}_0$ receives the full DePIP$_{\text{FRI}}$ transcript.
   (b) For each $j \in [\ell]$, sub-prover $\mathcal{P}_j$ invokes Eval of PolyFRIM on $\tilde{f}^{(j)}$ with $\mathcal{P}_0$ as the verifier. The challenges in (b) are identical to those in (a).
3. The master prover $\mathcal{P}_0$ checks all routed messages against the committed and opened values. Any mismatch identifies the sender as malicious.
4. If all sub-proofs in (b) pass, then for each query point used in DePIP$_{\text{FRI}}$, $\mathcal{P}_0$ obtains the corresponding *sub* evaluations from each $\tilde{f}^{(j)}$. Using Equation (1), $\mathcal{P}_0$ reconstructs the expected evaluations of the *aggregated* objects in DePIP$_{\text{FRI}}$ (*e.g.*, $\hat{f}_0(\pm\beta)$) as the weighted sum $\sum_{j\in[\ell]} w_j \cdot \hat{f}_0^{(j)}(\pm\beta)$. Comparing the reconstructed values with the values appearing in the DePIP$_{\text{FRI}}$ transcript identifies any sub-prover that contributed incorrect entries.

[4] We can use PIP$_{\text{FRI}}$ for sub-proofs, but this complicates the description.

Security. We argue accountability by handling M-I and M-II. For M-I, routing all inter-prover messages through $\mathcal{P}_0$ and binding them to Merkle openings ensures a sender cannot equivocate between recipients. Thus, any detected inconsistency cannot be blamed on inconsistent forwarding. For M-II, $\mathcal{P}_0$ verifies PolyFRIM sub-proofs under the same challenges as DePIP$_{\text{FRI}}$. If a sub-proof fails, $\mathcal{P}_0$ flags the corresponding sub-prover. If all sub-proofs verify, then with high probability the queried sub-evaluations are correct, and Equation (1) lets $\mathcal{P}_0$ reconstruct the expected aggregated evaluations. Any mismatch with the DePIP$_{\text{FRI}}$ transcript identifies the sub-prover(s) that supplied incorrect entries.

Complexity. Each sub-prover additionally produces one PolyFRIM sub-proof on a size-m instance, matching its existing $O(m \log m)$ work. Master prover $\mathcal{P}_0$ verifies ℓ sub-proofs, increasing its work from $O(\ell \log m)$ to $O(\ell \log^2 m)$. Routing messages via $\mathcal{P}_0$ preserves asymptotic communication and adds only constant-factor overhead to the existing exchanges.

6 Implementation and Evaluation

We implement PIP$_{\text{FRI}}$ and DePIP$_{\text{FRI}}$ in Rust using the arkworks ecosystem. Non-distributed benchmarks run on an AMD Ryzen 3900X CPU (12 cores, 32 GB RAM). Distributed benchmarks run on an Intel Xeon Platinum 8255C CPU (24 cores). In distributed runs, each sub-prover is pinned to a single core, using 2–16 cores total. We report averages over 10 executions. We do not parallelize within a prover.

6.1 Evaluation of PCSs

Our comparison includes code-based MLPCSs Virgo, Orion, PolyFRIM, and DeepFold. We exclude the schemes below.

- HyperPlonk [14] and BaseFold [42] are dominated by DeepFold [21] in our benchmark setting.
- Brakedown [20] and Block *et al.* [8] are field-agnostic linear-time schemes, and are reported or estimated to be worse than Orion on RS-friendly fields, which we tested.
- Blaze [10] runs over binary fields. We run over prime fields.

We also include group-based mKZG [14] and Hyrax. mKZG has $O(\log N)$ proof size and verification but requires a trusted setup. Hyrax is transparent with $O(\sqrt{N})$ proof size and verification. We omit Dory [24], as it performs worse than mKZG.

Parameters. We use the Goldilocks field[5] $\mathbb{F}_p$ with $p = 2^{64} -$

[5] All our benchmarks, except Orion in C++, use arkworks in Rust. All code-based benchmarks use $\mathbb{F}_p$, except Orion uses $\mathbb{F}_{q^2}$ where $q = 2^{61} - 1$. Basic operations over $\mathbb{F}_{q^2}$ (*e.g.*, FFTs) can be about 70% slower than over

$2^{32}+1$, as in plonky2[6]. The number of sub-polynomials ℓ is set to the nearest power-of-two to $\log_2(4N)$, typically set to 64 or 128. For most schemes, the RS code rate is set to $\rho = 1/8$, as in PolyFRIM and DeepFold. Additionally, we implement a PIP_{FRI} variant with $\rho = 1/2$ to trade proof size and verification for prover time. We set the security level to $\lambda = 100$ bits. For FRI-based schemes, we set the repetition parameter (q' in Protocol 3) to $-\lambda/\log_2\rho$, following prior work [5, 21, 30, 37, 39, 43, 44]. This targets conjectured λ-bit security for list polynomial binding, and DEEP-FRI [6] can reduce the residual list to one with slight overhead. For Orion's direct LDT, we (re-)set the queried column number to $-\lambda/\log_2(1-d/3)$ [18, 20], where d is the GS code distance.

Performance of PIP_{FRI}. We target faster prover time with minimal impact on proof size and verifier time. Figure 4 compares *non-zk* MLPCSs for polynomial sizes from 2^{18} to 2^{27}. Prover time is committing time plus opening time. Missing points are due to out-of-memory, except Orion, which we could not run beyond polynomial size 2^{23}. We benchmark all schemes at $\mu = 22$, and only PIP_{FRI} and Hyrax also run at $\mu = 27$. At $\mu = 27$, $\text{PIP}_{\text{FRI-1/8}}$ has a 115 s prover time, a 2.8 ms verifier time, and a 358 KB proof size.

Against mKZG, $\text{PIP}_{\text{FRI-1/8}}$ is 10–15× faster in prover time and 5× faster in verifier time, despite the worse complexity. Its proof can be 100× larger. However, mKZG needs a trusted setup. Compared with Hyrax, $\text{PIP}_{\text{FRI-1/8}}$ is 10× faster in prover time and 30–300× faster in verifier time. For small μ, Merkle openings make the proofs of $\text{PIP}_{\text{FRI-1/8}}$ larger, but for $\mu \geq 27$ its proofs become smaller due to lower complexity.

Against Virgo and PolyFRIM, $\text{PIP}_{\text{FRI-1/8}}$ is 20× faster in prover time, 1.1–1.6× faster in verifier time, and 1.4–1.8× smaller in proof size. Compared with DeepFold, $\text{PIP}_{\text{FRI-1/8}}$ is 5× faster in committing and 40× faster in opening, for a 14× faster prover time overall. Its proof size and verifier time are competitive, ranging from 5% better to 7% worse.

Relative to Orion (linear prover complexity), $\text{PIP}_{\text{FRI-1/8}}$ has only 5% slower prover time, but 30× better proof size and verifier time. With code rate 1/2, $\text{PIP}_{\text{FRI-1/2}}$ has a 3.5× faster prover time, with 15× better proof size and verifier time. To our knowledge, this is the first FRI-based PCS with a competitive or faster prover than Orion.

Figure 4(e) compares memory costs, which are crucial for large-scale SNARKs. For example, zkLLM [31] targets models with about 13 billion parameters, suggesting $\mu \approx 34$. In memory use, $\text{PIP}_{\text{FRI-1/8}}$ matches Orion, and both use 4× less memory than mKZG and more than 10× less than Virgo, PolyFRIM, and DeepFold. $\text{PIP}_{\text{FRI-1/2}}$ uses 3× less memory than $\text{PIP}_{\text{FRI-1/8}}$ due to shorter RS codewords, and about 30% less than Hyrax. With 32 GB RAM, $\text{PIP}_{\text{FRI-1/2}}$ is the only PCS in Figure 4 that supports $\mu = 29$. This advantage persists as μ grows, improving headroom for larger instances.

$\mathbb{F}_p$. We account for this arithmetic gap when interpreting Orion's timings.

[6] https://github.com/0xPolygonZero/plonky2/blob/main/plonky2

Table 5: Comparison of several code-based (zk-)PCSs

Scheme	Size	$\mathcal{P}$ (s)	$\mathcal{V}$ (ms)	π (KB)	Size	$\mathcal{P}$	$\mathcal{V}$	π
WHIR [3]	2^{18}	0.84	0.8	72	2^{20}	3.8	0.9	85
PIP_{FRI}		0.18	1.1	114		0.8	1.5	156
zk-FRI-PCS	2^{18}	4.8	2.3	242	2^{20}	22.1	2.9	306
zk-Virgo		23.2	1.8	212		102.3	2.1	265
zk-DeepFold		5.1	1.3	137		22.5	1.6	169
zk-PIP_{FRI}		0.4	1.3	132		1.8	1.6	175

Further comparisons with WHIR. This comparison favors WHIR in two ways. First, WHIR replaces FRI as a sub-protocol, so its verifier and proof-size gains come from a lower-complexity LDT. In principle, the same upgrade pattern in PIP can be instantiated with WHIR's LDT in place of FRI. Second, WHIR reports a faster FRI baseline than our implementation, likely due to different Merkle-tree engineering. We view these gaps as implementation-dependent rather than inherent to PIP_{FRI}. Overall, these factors indicate that PIP_{FRI} can achieve better time performance in practice.

Performance of zk-PCSs. We expect PIP_{FRI}'s advantages to persist with zero knowledge. Table 5 compares zk-PIP_{FRI} with zk-PCSs including Virgo and DeepFold. We omit Orion due to lack of a compatible implementation. For zk-DeepFold, we estimate performance for μ variables from reported results at $\mu+1$ variables [21], since masking doubles the instance size. We also implement a univariate zk-FRI-PCS baseline via the folklore method of running FRI-PCS [32] twice. Table 5 shows that zk-PIP_{FRI} is 10–50× faster than zk-FRI-PCS and zk-Virgo [43] in prover time, with 1.3–1.8× faster verification and 1.5–1.8× smaller proofs. It is also 10× faster than zk-DeepFold while matching its proof size and verifier cost.

Notably, (zk-)PIP_{FRI} outperforms (zk-)FRI-PCS in all three metrics. Although both rely on FRI with comparable total instance size, PIP replaces one large instance with multiple smaller instances that can be committed and opened more efficiently. This may seem counterintuitive: FRI-based PCSs invoke FRI on a comparable total codeword mass across rounds. We improve constants via shred-to-shine: we run more but smaller FRI instances and aggregate them with our packed Merkle commitment, reducing per-instance overhead.

6.2 Evaluation of SNARKs

Figure 5 reports SNARK performance for combinations of PCSs and PIOPs in Spartan [29] and HyperPlonk [14], which are state-of-the-art linear-prover PIOPs for R1CS and Plonkish. Spartan offers two PIOPs. We use the linear-verifier variant, since the sublinear-verifier implementation is integrated with the Hyrax PCS. Accordingly, Spartan-based schemes in Figure 5 have linear verifier time.

With HyperPlonk, PIP_{FRI} proves an R1CS instance with 2^{23} constraints in 100 s. It is 2× faster than DeepFold, with only 3% worse proof size and verifier time. Against Orion,

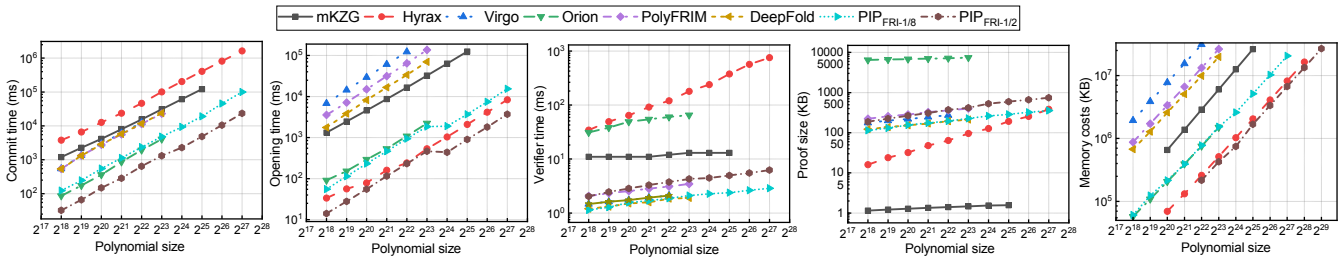


Figure 4: Performance comparison of PIP_{FRI}, mKZG, Hyrax, and several other code-based MLPCSs

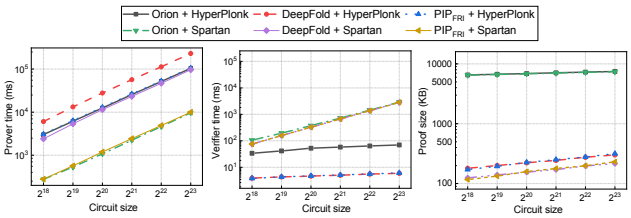


Figure 5: SNARKs from code-based PCSs and varied PIOPs

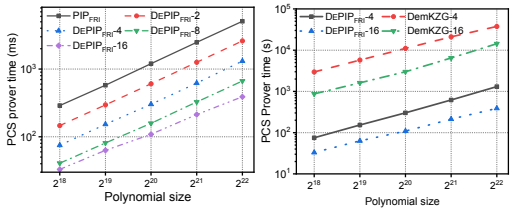


Figure 6: (a) Times for PIP_{FRI}/$\text{DEPIP}_{\text{FRI}}$ with 2–16 provers; (b) Times for $\text{DEPIP}_{\text{FRI}}$/DemKZG with 4 or 16 provers

PIP_{FRI}+HyperPlonk has a 2% worse prover time, but is 10–20× better in proof size and verifier time.

With Spartan, PIP_{FRI} proves a 2^{23}-constraint instance in 10 s. It is 10× faster than DeepFold, while keeping proof size and verifier time close. Compared with Orion, PIP_{FRI}+Spartan reduces proof size by 20×, with similar prover time and slightly faster verification. Overall, PIP_{FRI} improves existing SNARK instantiations by reducing prover time without materially worsening the other metrics.

Our goal is full linear speedup for the prover time of PIP_{FRI}. Figure 6(a) shows scaling from 2 to 16 provers. At size 2^{22}, $\text{DEPIP}_{\text{FRI}}$ reaches 0.66 s with 8 provers. Across these settings, proof size and verifier time remain close to PIP_{FRI}. See Figure 6(a) and Table 6. Figure 6(b) and Table 6 compare $\text{DEPIP}_{\text{FRI}}$ with DemKZG [25] and DeVirgo[7]. We omit DeDory [25], which is less efficient than DemKZG in our reported metrics. With the same number of provers, $\text{DEPIP}_{\text{FRI}}$ is 40× faster than DemKZG in prover time, and 20× faster than DeVirgo (estimated). Code-based $\text{DEPIP}_{\text{FRI}}$ trades off

[7]We could not obtain an open-source implementation of DeVirgo. We estimate its performance from Virgo, assuming full linear speedup, and estimate communication from our distributed Merkle tree implementation.

Table 6: Performance of distributed PCSs/SNARKs

Scheme	$\mathcal{P}$ (s)	$\mathcal{V}$ (ms)	π (KB)	Per comm. (KB)
DemKZG [25, 35]	14.3	17.0	2.3	1.7
DeVirgo [37]	7.7	2.2	213	110,000
$\text{DEPIP}_{\text{FRI}}$	0.39	2.1	198	15,000
Accountable **$\text{DEPIP}_{\text{FRI}}$**	2.05	2.1	198	18,775
DemKZG + HyperPianist	35.7	17.4	10.7	16
$\text{DEPIP}_{\text{FRI}}$ + HyperPianist	4.3	2.5	206	46,000

Figures are for size-2^{22} polynomials or circuits with 16 provers. We did not count the time for communication.

with group-based DemKZG, with faster verification but larger proof size and communication. Notably, $\text{DEPIP}_{\text{FRI}}$ reduces the amortized communication of DeVirgo by 7×, while keeping proof size and verifier time competitive.

For accountable $\text{DEPIP}_{\text{FRI}}$, prover time increases by 6× due to local PCS proofs on sub-polynomials, but remains 3× faster than non-accountable DeVirgo. Amortized communication increases by about 20% due to routing entries via $\mathcal{P}_0$, and has a limited impact in our measurements.

Table 6 compares distributed SNARKs that use the HyperPianist PIOP and either DemKZG or $\text{DEPIP}_{\text{FRI}}$. $\text{DEPIP}_{\text{FRI}}$ delivers an 8× faster prover and a 7× faster verifier. It also yields a *general* SNARK with 46 MB amortized communication, versus the reported 125 MB for a *non-general* SNARK from DeVirgo. This gap grows when a PIOP requires multiple distributed PCSs, as HyperPianist uses three [25] while the DeVirgo SNARK uses one.

6.3 Applications to Machine Learning

PIP_{FRI} has low prover time and low memory overhead, making it suitable for large-scale computations. Kaizen [1] uses Virgo as its code-based PCS, where PCS commitment reportedly exceeds 50% of the prover time. Replacing the PCS with PIP_{FRI} reduces prover time by 36%–42% (*cf.*, Table 7). For 4 batches of LeNet ($N = 4$), PIP_{FRI} reduces PCS time from 27.8 s to 5.7 s, reducing the PCS share from 54.2% to 19.5%.

Table 7: Prover times of Kaizen by Virgo or PIP_{FRI}

Batch Size	$N = 2$		$N = 4$	
Kaizen [1] for LeNet	Virgo	PIP_{FRI}	Virgo	PIP_{FRI}
PCS (s)	15.6	3.4	27.8	5.7
Others (s)	17.8		23.5	
Total (s)	33.4	21.2	51.3	29.2

7 Conclusion and Future Work

We presented PIP, a prover-efficiency upgrade framework for multilinear polynomial commitments. The core idea is to shred a large multilinear polynomial into sub-polynomials, make batch-MLPCS commitments to them, and combine their evaluations to answer the target query. We instantiated PIP in both group-based and code-based settings as PIP_{KZG} and PIP_{FRI}. We further built $\text{DEPIP}_{\text{FRI}}$, a distributed PIP_{FRI} that supports general circuits and provides accountability to identify misbehaving sub-provers. Our evaluation shows improved prover-side performance with competitive proof size and verifier time in relevant regimes, and it demonstrates end-to-end gains when integrated into SNARK constructions.

Future work includes extending PIP to additional PCS families and sub-protocols beyond the instantiations studied here. Another direction is to combine PIP with alternative low-degree tests to reduce proof size and verifier time without sacrificing prover efficiency.

Acknowledgments

Zongyang Zhang is supported in part by the National Cryptologic Science Fund of China (2025NCSF02022), the National Natural Science Foundation of China (62372020), and the Fundamental Research Funds for the Central Universities.

Sherman Chow is supported in part by the Research Grants Council of Hong Kong under the Collaborative Research Fund (C5097-25GF) and the General Research Fund (14210825).

Yi Deng is supported in part by the National Key Research and Development Program of China (2023YFB4503203), the Strategic Priority Research Program of the Chinese Academy of Sciences (XDB0690200), and the National Natural Science Foundation of China (62372447).

References

[1] Kasra Abbaszadeh, Christodoulos Pappas, Dimitrios Papadopoulos, and Jonathan Katz. Zero-knowledge proofs of training for deep neural networks. In *CCS*, 2024.

[2] Scott Ames, Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. Ligero: Lightweight sublinear arguments without a trusted setup. In *CCS*, 2017.

[3] Gal Arnon, Alessandro Chiesa, Giacomo Fenzi, and Eylon Yogev. WHIR: Reed-Solomon proximity testing with super-fast verification. In *EUROCRYPT IV*, 2025.

[4] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Fast Reed-Solomon interactive oracle proofs of proximity. In *ICALP*, 2018.

[5] Eli Ben-Sasson, Dan Carmon, Yuval Ishai, Swastik Kopparty, and Shubhangi Saraf. Proximity gaps for Reed-Solomon codes. In *FOCS*, 2020.

[6] Eli Ben-Sasson, Lior Goldberg, Swastik Kopparty, and Shubhangi Saraf. DEEP-FRI: Sampling outside the box improves soundness. In *ITCS*, 2020.

[7] Rishabh Bhadauria, Zhiyong Fang, Carmit Hazay, Muthuramakrishnan Venkitasubramaniam, Tiancheng Xie, and Yupeng Zhang. Ligero++: A new optimized sublinear IOP. In *CCS*, 2020.

[8] Alexander R. Block, Zhiyong Fang, Jonathan Katz, Justin Thaler, Hendrik Waldner, and Yupeng Zhang. Field-agnostic SNARKs from expand-accumulate codes. In *CRYPTO Part X*, 2024.

[9] Dan Boneh, Justin Drake, Ben Fisch, and Ariel Gabizon. Halo infinite: Proof-carrying data from additive polynomial commitments. In *CRYPTO Part I*, 2021.

[10] Martijn Brehm, Binyi Chen, Ben Fisch, Nicolas Resch, Ron D. Rothblum, and Hadas Zeilberger. Blaze: Fast SNARKs from interleaved RAA codes. In *EUROCRYPT Part IV*, 2025.

[11] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Gregory Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *S&P*, 2018.

[12] Benedikt Bünz, Ben Fisch, and Alan Szepieniec. Transparent SNARKs from DARK compilers. In *EUROCRYPT Part I*, 2020.

[13] Benedikt Bünz, Mary Maller, Pratyush Mishra, Nirvan Tyagi, and Psi Vesely. Proofs for inner pairing products and applications. In *ASIACRYPT Part III*, 2021.

[14] Binyi Chen, Benedikt Bünz, Dan Boneh, and Zhenfei Zhang. HyperPlonk: Plonk with linear-time prover and high-degree custom gates. In *EUROCRYPT II*, 2023.

[15] Alessandro Chiesa, Yuncong Hu, Mary Maller, Pratyush Mishra, Psi Vesely, and Nicholas P. Ward. Marlin: Preprocessing zkSNARKs with universal and updatable SRS. In *EUROCRYPT Part I*, 2020.

[16] Sherman S. M. Chow, Katharina Fech, Russell W. F. Lai, and Giulio Malavolta. Multi-client oblivious RAM with poly-logarithmic communication. In *ASIACRYPT Part II*, 2020.

[17] Mingshu Cong, Sherman S. M. Chow, Siu-Ming Yiu, and Tsz Hon Yuen. Scalable zkSNARKs for matrix computation: A generic framework for verifiable deep learning. In *ASIACRYPT Part V*, 2025.

[18] Thomas den Hollander and Daniel Slamanig. A crack in the firmament: Restoring soundness of the Orion proof system and more. In *ASIACRYPT Part V*, 2025.

[19] Georg Fuchsbauer, Eike Kiltz, and Julian Loss. The algebraic group model and its applications. In *CRYPTO Part II*, 2018.

[20] Alexander Golovnev, Jonathan Lee, Srinath T. V. Setty, Justin Thaler, and Riad S. Wahby. Brakedown: Linear-time and field-agnostic SNARKs for R1CS. In *CRYPTO Part II*, 2023.

[21] Yanpei Guo, Xuanming Liu, Kexi Huang, Wenjie Qu, Tianyang Tao, and Jiaheng Zhang. DeepFold: Efficient multilinear polynomial commitment from Reed-Solomon code and its application to zero-knowledge proofs. In *Usenix Security*, 2025.

[22] Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. Constant-size commitments to polynomials and their applications. In *ASIACRYPT*, 2010.

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

[24] Jonathan Lee. Dory: Efficient, transparent arguments for generalised inner products and polynomial commitments. In *TCC Part II*, 2021.

[25] Chongrong Li, Yun Li, Pengfei Zhu, Wenjie Qu, and Jiaheng Zhang. HyperPianist: Pianist with linear-time prover via fully distributed HyperPlonk. In *S&P*, 2025.

[26] Tianyi Liu, Tiancheng Xie, Jiaheng Zhang, Dawn Song, and Yupeng Zhang. Pianist: Scalable zkRollups via fully distributed zero-knowledge proofs. In *S&P*, 2024.

[27] Ueli M. Maurer. Abstract models of computation in cryptography. In *IMACC*, 2005.

[28] Charalampos Papamanthou, Elaine Shi, and Roberto Tamassia. Signatures of correct computation. In *TCC*, 2013.

[29] Srinath T. V. Setty. Spartan: Efficient and general-purpose zkSNARKs without trusted setup. In *CRYPTO Part III*, 2020.

[30] StarkWare Team. ethSTARK documentation version 1.2. IACR ePrint 2021/582.

[31] Haochen Sun, Jason Li, and Hongyang Zhang. zkLLM: Zero knowledge proofs for large language models. In *CCS*, 2024.

[32] Alexander Vlasov and Konstantin Panarin. Transparent polynomial commitment scheme with polylogarithmic communication complexity. IACR ePrint 2019/1020.

[33] Riad S. Wahby, Ioanna Tzialla, abhi shelat, Justin Thaler, and Michael Walfish. Doubly-efficient zkSNARKs without trusted setup. In *S&P*, 2018.

[34] Jiafan Wang and Sherman S. M. Chow. Unus pro omnibus: Multi-client searchable encryption via access control. In *NDSS*, 2024.

[35] Wenhao Wang, Fangyan Shi, Dani Vilardell, and Fan Zhang. Cirrus: Performant and accountable distributed SNARK. IACR ePrint 2024/1873.

[36] Howard Wu, Wenting Zheng, Alessandro Chiesa, Raluca Ada Popa, and Ion Stoica. DIZK: a distributed zero knowledge proof system. In *Usenix Security*, 2018.

[37] Tiancheng Xie, Jiaheng Zhang, Zerui Cheng, Fan Zhang, Yupeng Zhang, Yongzheng Jia, Dan Boneh, and Dawn Song. zkBridge: Trustless cross-chain bridges made practical. In *CCS*, 2022.

[38] Tiancheng Xie, Jiaheng Zhang, Yupeng Zhang, Charalampos Papamanthou, and Dawn Song. Libra: Succinct zero-knowledge proofs with optimal prover computation. In *CRYPTO Part III*, 2019.

[39] Tiancheng Xie, Yupeng Zhang, and Dawn Song. Orion: Zero knowledge proof with linear prover time. In *CRYPTO Part IV*, 2022.

[40] Hua Xu, Mariana Gama, Emad Heydari Beni, and Jiayi Kang. FRIttata: Distributed proof generation of FRI-based SNARKs. IACR ePrint 2025/1285.

[41] Yuanzhuo Yu, Mengling Liu, Yuncong Zhang, Shi-Feng Sun, Tianyi Ma, Man Ho Au, and Dawu Gu. HyperFond: A transparent and post-quantum distributed SNARK with polylogarithmic communication. IACR ePrint 2025/1349.

[42] Hadas Zeilberger, Binyi Chen, and Ben Fisch. BaseFold: Efficient field-agnostic polynomial commitment schemes from foldable codes. In *CRYPTO Part X*, 2024.

[43] Jiaheng Zhang, Tiancheng Xie, Yupeng Zhang, and Dawn Song. Transparent polynomial delegation and its applications to zero knowledge proof. In *S&P*, 2020.

[44] Zongyang Zhang, Weihan Li, Yanpei Guo, Kexin Shi, Sherman S. M. Chow, Ximeng Liu, and Jin Dong. Fast RS-IOP multivariate polynomial commitments and verifiable secret sharing. In *Usenix Security*, 2024.

A Comparisons with Concurrent Work

FRIttata [40] proposes a "fold-and-batch" distributed FRI, sharing our goal of distributed code-based PCSs while focusing on empirical communication savings. As summarized in Table 8, we improve master prover complexity, proof size, and verifier complexity, with the same *asymptotic* total communication complexity. For concrete efficiency (*cf.* Table 9) of our distributed FRI, our prover is 3× faster, our verifier is 3–7× faster, and our proof is 2–5× smaller. However, our concrete total communication is <2× higher.

FRIttata does not achieve full linear speedup, with verifier time and proof size worsening as the number of provers grows. Our prover efficiency comes from the improved distributed opening with $O(m/\ell)$ sub-prover and amortized communication, which could also accelerate theirs.

We build distributed code-based PCSs and SNARKs for general circuits via an MLPCS and the multilinear PIOP in HyperPianist [25]. Instead, FRIttata uses a bivariate PCS with the bivariate PIOP in Pianist [26], which is incompatible with our multilinear technique. Our faster FRI and linear-time PIOP yield a faster prover for distributed PCS and SNARK.[8]

Accountability is also not in their scope. Their PCS distributes only bivariate polynomials of the form $f(X,Y) = \sum_i f_i(X)L_i(Y)$ by assigning *independent* $f_i(X)$ to provers. In contrast, shred-to-shine enables distributed FFTs and supports (single) univariate, bivariate, and multilinear polynomials. It also speeds up the prover of *non-distributed* FRI-PCS by 10× (Table 5). Its benefits extend to other PCSs as well, *e.g.*, FRIttata's or mKZG.

HyperFond [41] builds a distributed MLPCS over the non-distributed BaseFold [42]. DePip$_{\text{FRI}}$ matches the HyperFond PCS in prover complexity (Table 10) while improving proof size and verifier complexity at the expense of higher communication complexity. HyperFond does not achieve *full* linear speedup, and it cannot simultaneously optimize proof size/verifier complexity and communication complexity. HyperFond does not report concrete PCS timings, but its prover time can be estimated. Our non-distributed Pip$_{\text{FRI}}$ already has a 10× faster prover time than DeepFold and hence BaseFold on RS-friendly fields. With *optimal* full linear speedup, our distributed DePip$_{\text{FRI}}$ would inherit this prover-time advantage over HyperFond.

[8] FRIttata did not report figures for their distributed PCSs and SNARKs.

Table 8: Efficiency of distributed FRIs in FRIttata and ours

	$\mathcal{P}_i$	$\mathcal{P}_0$	$\mathcal{V}/\pi$	Total Comm.
FRIttata	$O(m\log m)^*$	$O(m\ell)$	$O(\ell\log m+\log^2 m)$	$O(m\ell)$
Ours	$O(m\log m)$	$O(\ell\log m)$	$O(\lambda\ell+\log^2 m)$	$O(m\ell)$

*FRIttata **assume** that $\mathcal{P}_i$ initially holds the polynomial evaluations, hence not needing FFTs. Ours starts from FFTs and features a quasi-linear complexity. Operations other than FFTs also cost $O(m)$ time.

Table 9: Performance of distributed FRIs for size-2^{25} polynomials in FRIttata and ours, with 4 or 16 provers (#$\mathcal{P}_i$)

	#$\mathcal{P}_i$	$\mathcal{P}$ (s)	$\mathcal{V}$ (ms)	π (KB)	Total Comm.
FRIttata	4	30	10	700	1.07 GB
Ours	4	9.7	3.8	283	1.50 GB
FRIttata	16	10	30	1,500	1.07 GB
Ours	16	2.6	3.8	271	1.90 GB

Table 10: Complexity of the HyperFond PCS and DePip$_{\text{FRI}}$

	$\mathcal{P}_i$	$\mathcal{V}/\pi$	Total Comm.
HyperFond [41]	$O(m\log m)$	$O(\ell\log^2 m)$	$O(\ell\log^2 m)$
DePip$_{\text{FRI}}$	$O(m\log m)$	$O(\log^2 m\ell)$	$O(m\ell)$

Table 11: Performance of distributed SNARKs for size-2^{23} circuits in HyperFond and ours, with 4 or 16 provers (#$\mathcal{P}_i$)

	#$\mathcal{P}_i$	$\mathcal{P}$ (s)	$\mathcal{V}$ (ms)	π (KB)	Total/Per Comm.
HyperFond [41]	4	8	379	68,800	51.6/17.2 MB
Ours	4	33	3.5	232	1176/294 MB
HyperFond [41]	16	2	837	219,000	206/14 MB
Ours	16	8.4	3.8	228	1428/90 MB

*Each prover has 64 vCPUs in HyperFond and 2 vCPUs in ours.

The distributed encoding of the HyperFond PCS splits the coefficient vector $\boldsymbol{f}$ and lets prover $\mathcal{P}_i$ encode $\boldsymbol{f}^{(i)}$. To recover the codeword $\mathsf{Enc}(\boldsymbol{f})$ of $\boldsymbol{f}$, HyperFond uses a specific foldable-linear-code property by linearly combining $\mathsf{Enc}(\boldsymbol{f}^{(1)}),\ldots,\mathsf{Enc}(\boldsymbol{f}^{(\ell)})$. This approach does not apply to other codes like RS codes that lack this property. Our shred-to-shine method is more general and compatible with any linear codes by transforming f into sub-polynomials $f^{(1)},\ldots,f^{(\ell)}$ and reconstructing the evaluation of f from sub-evaluations.

Table 11 shows that our SNARK outperforms HyperFond with 100–200× faster verifier times and 300–1,000× smaller proof sizes, but 7–20× larger communication.

Two key factors contribute to HyperFond's reported prover time being 4× faster than ours. First, each prover in HyperFond uses 64 vCPUs, while ours operates on a single core with 2 vCPUs. Second, HyperFond uses a more efficient PIOP than both HyperPianist and ours. While we are propos-

ing improved PCSs rather than a new PIOP, we observe that, thanks to DePip$_{\text{FRI}}$, the proportion of PCS time in our distributed SNARK (comprising PIOP + PCS) drops from 85% to 25% compared to HyperPianist. Using HyperFond's PIOP is expected to significantly improve our prover time.

B Proofs and Analysis for Pip Framework

B.1 Proof of Theorem 3.1

Proof. For Equation (1), define $d_v = \log m$, $d_w = \log \ell$, and $d = d_v + d_w$. Define $b_k(i) \in \{0,1\}$ as the k-th bit of the binary representation of i, for $i \in [N]$. Define $c_k(a) \in \{0,1\}$ as the k-th bit of the binary representation of a, for $a \in [m]$. Define $d_k(j) \in \{0,1\}$ as the k-th bit of the binary representation of j, for $j \in [\ell]$. By definition of $\boldsymbol{v}$ and $\boldsymbol{w}$, we have $v_a = \prod_{k=1}^{d_v} x_k^{c_k(a)}$ and $w_j = \prod_{k=1}^{d_w} x_{k+d_v}^{d_k(j)}$. For each $j \in [\ell]$, we rewrite $\tilde{f}_j(\boldsymbol{v})$ as $\tilde{f}_j(\boldsymbol{v}) = \sum_{a=1}^{m} f_{(j-1)\cdot m+a} \cdot \prod_{k=1}^{d_v} x_k^{c_k(a)}$.

By definition of multilinear evaluation, $y = \tilde{f}(\boldsymbol{x}) = \sum_{i=1}^{N} f_i \cdot \prod_{k=1}^{d} x_k^{b_k(i)}$. Write $i = (j-1)\cdot m + a$, and expand as $y = \sum_{j=1}^{\ell}\sum_{a=1}^{m} f_{(j-1)\cdot m+a} \cdot \prod_{k=1}^{d} x_k^{b_k(i)}$. For $1 \le k \le d_v$, we have $b_k(i) = c_k(a)$. For $d_v + 1 \le k \le d$, we have $b_k(i) = d_{k-d_v}(j)$. Hence, $\prod_{k=1}^{d} x_k^{b_k(i)} = \left(\prod_{k=1}^{d_v} x_k^{c_k(a)}\right) \cdot \left(\prod_{k=1}^{d_w} x_{k+d_v}^{d_k(j)}\right) = v_a \cdot w_j$. Substituting, $y = \sum_{j=1}^{\ell}\sum_{a=1}^{m} f_{(j-1)\cdot m+a} \cdot v_a \cdot w_j$. Reindexing yields $y = \sum_{j=1}^{\ell} w_j \left(\sum_{a=1}^{m} f_{(j-1)\cdot m+a} \cdot v_a\right) = \sum_{j=1}^{\ell} w_j \cdot \tilde{f}_j(\boldsymbol{v})$.

Completeness and Binding. Completeness follows from the completeness of PC_ℓ applied to each committed sub-polynomial. For binding, suppose there exist $\tilde{f} \neq \tilde{f}'$ such that $\mathsf{VerPoly}(\mathsf{pp}, C, \tilde{f}) = \mathsf{VerPoly}(\mathsf{pp}, C, \tilde{f}') = 1$. Then there exists $j \in [\ell]$ with $\tilde{f}_j \neq \tilde{f}'_j$ such that $\mathsf{VerPoly}(\mathsf{pp}_\ell, C_j, \tilde{f}_j) = \mathsf{VerPoly}(\mathsf{pp}_\ell, C_j, \tilde{f}'_j) = 1$. This contradicts the binding of PC_ℓ.

Knowledge Soundness. Let $\mathcal{E}_\ell$ be the extractor for Eval_ℓ with expected running time $\mathsf{poly}(m)$. The extractor $\mathcal{E}$ runs $\mathcal{E}_\ell$ sequentially to obtain $\tilde{f}_j$ consistent with the claimed evaluation u_j, for all $j \in [\ell]$. $\mathcal{E}$ then concatenates the coefficients of $\{\tilde{f}_j\}_{j\in[\ell]}$ to output $\tilde{f}$ such that $\tilde{\boldsymbol{f}} = (\tilde{\boldsymbol{f}}_1, \ldots, \tilde{\boldsymbol{f}}_\ell)$. The running time is $\ell \cdot \mathsf{poly}(m) = \mathsf{poly}(N)$. Knowledge soundness follows.

Complexity. The ℓ commitments $C_1, \ldots, C_\ell$ can be generated independently and in parallel. The total commitment complexity is $\ell \cdot t_{\mathsf{C}}(m)$, which is $O(N \log m)$ when $t_{\mathsf{C}}(m) = O(m \log m)$. The opening complexity, verifier complexity, and proof size follow by substituting m and ℓ into the batch PCS bounds. If the proof size for a size-N polynomial is $\log^c N$, then $O(\log^c(N/\log^c N) + \log^c N) = O(\log^c N)$. The verifier complexity is similar. As the prover handles size-m polynomials, the size of SRS, if required, is m. □

B.2 Further Discussions

This subsection (i) extends multilinear Pip to the univariate case and (ii) discusses a generic approach to add zero-knowledge.

Extending to the Univariate Case. Protocol 2 is designed for multilinear polynomials and can be adapted to univariate ones. For any x, the vector $(1, x, \ldots, x^{N-1})$ can be expressed as a tensor product $\otimes_{i=1}^{\log N}(1, x^{2^{i-1}})$. Given a size-$N$ univariate polynomial $f(X)$, we form an equivalent multilinear polynomial $\tilde{f}(X_1, \ldots, X_\mu)$ by setting $X_i = X^{2^{i-1}}$ for each $i \in [\mu]$ (so $X_1 = X$ and $X_\mu = X^{N/2}$).

Adding Zero Knowledge. Protocol 2 is not zero-knowledge because $\boldsymbol{u}$ in Step 2 of Eval leaks information about $\tilde{f}$. We present a generic masking method for univariate polynomials $\hat{f}$. Without loss of generality, let the size be $N = m\ell$. Write $\hat{f}(X) = \sum_{i=1}^{\ell} X^{m(i-1)} \hat{f}_i(X)$. The prover samples random $r_1, \ldots, r_{\ell-1}$ and sets $r_0 = r_\ell = 0$. For each $i \in [\ell]$, define the masked polynomial $\hat{f}'_i(X) = \hat{f}_i(X) + r_i X^m - r_{i-1}$. The prover commits to each $\hat{f}'_i$ and sends the masked vector $\boldsymbol{u}' = (\hat{f}'_1(\alpha), \ldots, \hat{f}'_\ell(\alpha))$ instead. Correctness holds for every α because $\sum_{i=1}^{\ell} X^{m(i-1)} \hat{f}_i(X) = \sum_{i=1}^{\ell} X^{m(i-1)} \hat{f}'_i(X)$, and hence $\hat{f}(\alpha) = \langle \boldsymbol{u}', (1, \alpha^m, \ldots, \alpha^{m(\ell-1)}) \rangle$. Each $\hat{f}'_i$ increases the degree bound by 1, and this can be proven directly by LDTs [4] or PCS-based LDTs [15] with minimal overhead. Generalizing zero knowledge to all MLPCSs is left for future work.

C Pip$_{\text{KZG}}$: Instantiation of Pip with mKZG

We recall the background of group-based PCSs.

Group-based PCSs start from the KZG commitment [22]. Its multilinear version mKZG [28] has logarithmic proof size and verifier time in the polynomial size. (m)KZG requires a linear-size SRS, which can reach hundreds of GB for large-scale computations [36] and raises concerns for discrete-logarithm instantiations [13]. Bünz *et al.* [13] reduce the SRS to $O(\sqrt{N})$, but their security relies on the GGM [27], which is stronger than the AGM [19] assumed in (m)KZG-based SNARKs. Dory [24] removes trusted setup and uses a sub-linear common reference string. However, its prover and verifier rely on target-group operations, which are reported to be about 4× slower [24] than the source-group operations used by mKZG.

C.1 Construction

We construct Pip$_{\text{KZG}}$ in Protocol 6 by instantiating Protocol 2 with mKZG $(\mathsf{Gen}_b, \mathsf{Com}_b, \mathsf{Eval}_b)$. Let $\tilde{f}$ be a size-$m\ell$ multilinear polynomial partitioned into ℓ sub-polynomials $\tilde{f}^{(1)}, \ldots, \tilde{f}^{(\ell)}$, each of size m. Thanks to mKZG's homomorphism, $\mathcal{P}$ sends only the commitments $\mathsf{Com}(\tilde{f}^{(1)}), \ldots, \mathsf{Com}(\tilde{f}^{(\ell)})$ and avoids sending the sub-polynomial evaluations $\tilde{f}^{(1)}(\boldsymbol{x}), \ldots, \tilde{f}^{(\ell)}(\boldsymbol{x})$. $\mathcal{V}$ constructs a virtual commitment to $\tilde{f}$ as $\prod_{i\in[\ell]} \mathsf{Com}(\tilde{f}^{(i)})^{w_i}$.

Protocol 6 (PIP_{KZG}: sublinear-SRS PCS from pairing).

- $(\mathsf{pp},\mathsf{srs}) \leftarrow \mathsf{Gen}(1^\lambda, 2, \mu)$: $\mathsf{pp} = (m,\ell)$ *satisfying* $m\ell = N$, *and* $\mathsf{srs} = \mathsf{Gen}_b(1^\lambda, 2, \log m)$.
- $C \leftarrow \mathsf{Com}(\mathsf{pp}, \tilde{f}, \mathsf{srs})$: *Given* $\tilde{f}$, *the prover* $\mathcal{P}$ *does:*
  1. *Arrange* $\tilde{f}$*'s coefficient vector* $\boldsymbol{f}$ *as an* $m \times \ell$ *matrix with column* $U[j] = (f_{(j-1)\cdot m+1}, f_{(j-1)\cdot m+2}, \ldots, f_{j\cdot m})$*; treat it as a coefficient vector of a multilinear polynomial* $\tilde{f}_j$.
  2. *Output* $C \leftarrow (C_1, \ldots, C_\ell)$, *where* $C_j \leftarrow \mathsf{Com}_b(\mathsf{srs}, \tilde{f}_j)$.
- $b \leftarrow \mathsf{Eval}(\mathsf{pp}, \mathsf{srs}, C, x, y; \tilde{f})$: *Given the evaluation point* $\boldsymbol{x}$ *and evaluation* $y = \tilde{f}(\boldsymbol{x})$, $\mathcal{P}$ *and the verifier* $\mathcal{V}$ *do:*
  1. $\mathcal{P} \to \mathcal{V}$: $C = (C_1, \ldots, C_\ell)$.
  2. $\mathcal{V}$: *computes public vectors* $\boldsymbol{v} = \otimes_{k\in[\log m]}(1, x_k)$, $\boldsymbol{w} = \otimes_{k\in[\log \ell]}(1, x_{k+\log m})$, *and via homomorphism* $C' = \prod_{j\in[\ell]} C_j^{w_j}$, *an mKZG commitment to* $\tilde{f}' = \sum_{j\in[\ell]} w_j \cdot \tilde{f}_j$.
  3. $\mathcal{P}$ *and* $\mathcal{V}$: *return* $\mathsf{Eval}_b(C', \tilde{f}', \mathsf{srs}; y, (x_1, \ldots, x_{\log m}))$ *to prove the virtual evaluation* $y = \tilde{f}'(x_1, \ldots, x_{\log m})$.

Theorem C.1. *Protocol 6 is an MLPCS secure in the AGM. Its SRS size is* $O(m)$*, committing complexity is* $O(N)$*, opening complexity is* $O(m)\,\mathbb{G} + O(N)\,\mathbb{F}_{\mathsf{rlc}}$*, verifier complexity is* $O(\log m)\,P + O(\ell)\,\mathbb{G}$*, and proof size is* $O(\ell + \log m)$.

Proof of Theorem C.1. **Security.** Completeness holds by Theorem 3.1. We argue knowledge soundness in the AGM by defining an efficient extractor $\mathcal{E}$ and showing that, for any algebraic adversary $\mathcal{A}$ and any $m\ell = \mathsf{poly}(\lambda)$, the probability that $\mathcal{A}$ wins is $\mathsf{negl}(\lambda)$ over the randomness of $\mathcal{A}$, $\mathcal{V}$, and Gen [9]. An algebraic adversary $\mathcal{A}$ is a PPT algorithm that, whenever it outputs an element $A \in \mathbb{G}$, it also outputs a vector $\boldsymbol{a}$ over $\mathbb{F}$ such that $A = \langle \boldsymbol{a}, \mathsf{srs} \rangle$. Concretely, the game is as follows [9]:

1. Given $m\ell$ and srs, $\mathcal{A}$ outputs C.
2. $\mathcal{A}$ outputs $\boldsymbol{x}$ and y.
3. $\mathcal{A}$ plays the part of $\mathcal{P}$ in Eval with inputs C, $\boldsymbol{x}$, and y.
4. $\mathcal{E}$, given $\mathcal{A}$'s prior messages, outputs $\tilde{f}$ after Step 1 in Eval.
5. $\mathcal{A}$ wins if $\mathcal{V}$ outputs 1 in Eval and $\tilde{f}(\boldsymbol{x}) \neq y$.

Under the AGM, when $\mathcal{A}$ outputs $C_1, \ldots, C_\ell$, it also outputs witnesses that explain each C_i as a linear combination of SRS elements. In Protocol 6, this yields polynomials $\tilde{f}'_1, \ldots, \tilde{f}'_\ell$, and $\mathcal{P}$ and $\mathcal{V}$ invoke mKZG on the size-m polynomial $\tilde{f}' = \sum_{j\in[\ell]} w_j \cdot \tilde{f}'_j$. mKZG satisfies evaluation binding [15, 38], so no efficient adversary can open the same commitment to different values at the same point. Applying evaluation binding to the commitment of $\tilde{f}'$ implies knowledge soundness.

Complexity. The SRS size, committing complexity, and proof size follow from Theorem 3.1 and the complexities of mKZG. For opening, the prover computes the target size-m polynomial $\tilde{f}'$ and then runs mKZG opening on $\tilde{f}'$. The former costs $O(N)\,\mathbb{F}_{\mathsf{rlc}}$ and the latter costs $O(m)\,\mathbb{G}$. For verification, $\mathcal{V}$ first commits to the target polynomial, costing $O(\ell)\,\mathbb{G}$, and then runs mKZG verification for a size-m polynomial, costing $O(\log m)\,\mathcal{P}$. The claimed complexities follow. □

C.2 Evaluation of PIP_{KZG} and SNARKs

Methodology. We adapt the open-source mKZG implementation from HyperPlonk [14] to implement PIP_{KZG}. We set $\ell = 16$ and $m = N/\ell$ for size-N polynomials. We also use the open-source Dory implementations from HyperPianist [25] for comparison. All PCSs use the BN254 elliptic curve and the arkworks ecosystem in Rust.

Performance. Figure 7 compares mKZG, Dory, and PIP_{KZG} on multilinear polynomials of sizes from 2^{16} to 2^{24}. We only succeeded in running Dory when the polynomial size exceeded 2^{20}. Relative to mKZG, PIP_{KZG} has a 10× shorter SRS, a 1.7× faster prover, and up to a 2× faster verifier. Committing times are nearly the same, while PIP_{KZG} opening is 10× faster. PIP_{KZG} proofs are 1.5–1.8× larger than mKZG, but for a size-2^{24} polynomial the proof is only 2.2 KB. In SNARKs from multilinear PIOPs (e.g., Spartan and HyperPlonk), this gap shrinks after accounting for the PIOP proof size (Figure 8).

Compared with transparent Dory, PIP_{KZG} requires trusted setup and has a larger SRS. However, PIP_{KZG} has a 2× faster prover, a 4× faster verifier, and a 10× smaller proof size. This is because PIP_{KZG} runs over the more efficient type-1 group rather than the target group, as in Dory.

Figure 8 reports SNARKs using mKZG or PIP_{KZG} and PIOPs in Spartan [29] or HyperPlonk [14]. The SNARK "PIP_{KZG}+ HyperPlonk" is competitive with "mKZG + HyperPlonk" in prover time, verifier time, and proof size. This is because HyperPlonk uses three PCS committing algorithms but only one opening algorithm, and mKZG and PIP_{KZG} have nearly the same committing times. However, the SRS is determined by the underlying PCS, so "PIP_{KZG}+ HyperPlonk" has a 10× shorter SRS than "mKZG + HyperPlonk". With Spartan, "PIP_{KZG}+ Spartan" has a 1.5× faster prover than "mKZG + Spartan", a 1.1× larger proof size, and a competitive verifier time due to the linear-verifier PIOP.

D Security Proof of zk-PIP_{FRI}

Completeness follows from the completeness of rolling batch FRI, the linearity of the RS code, and the equivalence between multilinear evaluation and the FRI-like folding in Step 3 of Protocol 3. We state this equivalence as Lemma 1 (distilled from PolyFRIM [44]).

Lemma 1 ([44]). *For any size-*$m = 2^{\sigma+1}$ *multilinear polynomial* $\tilde{f}_0$ *and any evaluation point* $\boldsymbol{x} = (x_1, \ldots, x_\sigma, x_{\sigma+1})$*, if Step 3 of Protocol 3 is followed to fold* $\tilde{f}_0$ *to obtain the constant* $\hat{f}_{\sigma+1}$*, then* $\hat{f}_{\sigma+1} = \tilde{f}_0(x_1, \ldots, x_\sigma, x_{\sigma+1})$.

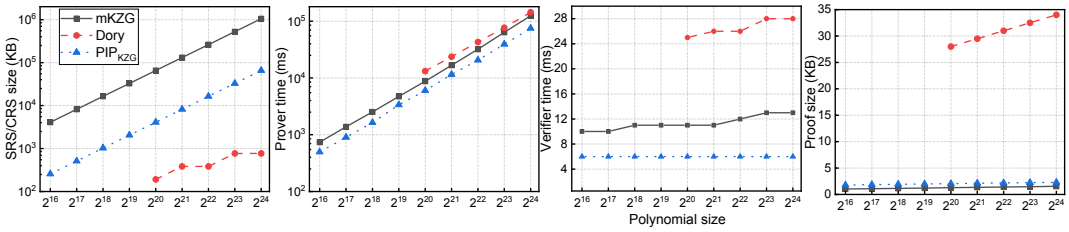


Figure 7: Performance of mKZG, Dory, and PIP_{KZG}

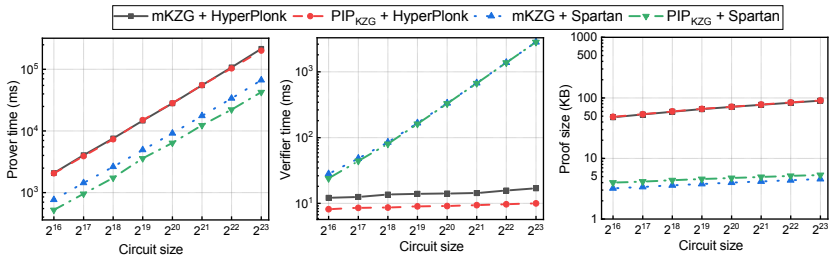


Figure 8: Performance of SNARKs constructed by mKZG or PIP_{KZG} and PIOPs in Spartan or HyperPlonk

Based on Lemma 1, we obtain the claimed evaluation used in our protocol by instantiating the folding point as required by Protocol 2. By construction of Protocol 2, we have $\sum_{j\in[\ell]} w_j \cdot \tilde{f}^{(j)}(\boldsymbol{x}) = \langle \boldsymbol{w}, \boldsymbol{u} \rangle = \tilde{f}(x_1, \ldots, x_\mu) = y$. Completeness follows.

Polynomial Binding. The Merkle commitment fixes the committed vectors $\hat{f}^{(1)}|_{L_0}, \ldots, \hat{f}^{(\ell)}|_{L_0}$ uniquely under collision resistance. Assume the proximity parameter $\delta = (1-\rho)/2$ is within the unique decoding radius. Then each committed vector determines a unique polynomial $\hat{f}^{(j)}$ consistent with it (within δ), and hence determines a unique target polynomial $\tilde{f}$. Polynomial binding follows.

Knowledge Soundness. PIP_{FRI} is an argument of knowledge in the random oracle model. Using extractability of Merkle-tree commitments, an extractor can obtain the committed Merkle leaves and thus the committed RS codewords [43,44]. The extractor decodes $\hat{f}^{(1)}|_{L_0}, \ldots, \hat{f}^{(\ell)}|_{L_0}$ (e.g., via Berlekamp–Welch) to obtain ℓ polynomials. It then parses coefficients, drops the right half of each, and reconstructs the multilinear polynomial $\sum_{j=1}^{\ell} w_j \cdot \tilde{f}^{(j)}$. Suppose the extracted $\tilde{f}$ satisfies $\tilde{f}(\boldsymbol{x}) \neq y$. We bound the acceptance probability by a case analysis.

• Case 1. Some $\hat{f}_i$ for $i \in [0, \mu+1]$ has size $> 2^{\mu+1-i}$. We argue that the soundness error in this case is $|L_0|/|\mathbb{F}| + ((1+\rho)/2)^{q'} + \text{negl}(\lambda)$. This follows from the soundness of rolling batch FRI as analyzed in PolyFRIM. Intuitively, the only difference is that rolling batch FRI starts from a single input codeword, whereas our LDT starts from multiple input codewords. Still, by the generalized correlated agreement over span spaces [5], if $\sum_{i\in[\ell]} \alpha^i \cdot \boldsymbol{u_i}$ is close to a codeword $\boldsymbol{v} \in \text{RS}[L_0, \rho]$ for vectors $\boldsymbol{u_1}, \ldots, \boldsymbol{u_\ell} \in \mathbb{F}^{|L_0|}$ and random challenge α, then with high probability each $\boldsymbol{u_i}$ is close to some codeword in $\text{RS}[L_0, \rho]$. Here, "close" refers to Hamming distance. Further, the agreement positions between each vector and its corresponding codeword are the same.

• Case 2. The degrees of $\{\hat{f}_i\}$ are at most $2^{\mu+1-i}$ for all $i \in [0, \mu+1]$, and the evaluation proofs are all valid. As $\tilde{f}(\boldsymbol{x}) \neq y$, the equality $\tilde{f}(\boldsymbol{x}) + \alpha \cdot \tilde{s}(\boldsymbol{x}, 0) = y + \alpha \cdot y_s$ holds with probability at most $1/|\mathbb{F}|$ over uniform α. Otherwise, there exists some i such that $\hat{f}_i(X) \neq \hat{g}_{i-1}(X) + x_i \cdot \hat{h}_{i-1}(X)$, yet the equality holds at the verifier's query β. By the Schwartz–Zippel lemma, the soundness error is bounded by $2m/|L_0|$ since the degree of $\hat{f}_i$ is at most $2m$. For q independent queries, this error is $(2m/|L_0|)^q$.

Using the union bound argument, the soundness error is $\text{negl}(\lambda) + |L_0|/|\mathbb{F}| + (2^{\mu+1}/|L_0|)^q + ((1+\rho)/2)^{q'}$. By appropriate choices of parameters, this error can be negligible.[9]

If the code rate is large, the protocol may require a large q and thus a large proof. We can instead sample β from $\mathbb{F}$ (rather than L_0) in Protocol 3. To complete the consistency check, the prover and verifier run a batched FRI-PCS for $\{\hat{f}_i(\pm\beta^{2^i})\}_{i\in[0,\sigma]}$. We batch this FRI-PCS with the rolling batch FRI, incurring only slight additional overhead. In this case, the soundness error reduces to $2m/|\mathbb{F}|$.

For the soundness of Case 1, we adapt some notation from PolyFRIM. Define functions $\{\text{back}_i\}_{i\in[0,\sigma+1]}$ and sets $\{\text{err}_i\}_{i\in[0,\sigma+1]}$. Define $\text{back}_0(A_0) = A_0$ for $A_0 \subseteq L_0$, and $\text{back}_i(A_i) = \text{back}_{i-1}(A_{i-1})$ for $A_i = \{x^2 | x \in A_{i-1}\}$ and $i \geq 1$.

[9] We prove polynomial binding and knowledge soundness in the unique decoding radius for simplicity, *i.e.*, with proximity parameter $\delta = (1-\rho)/2$. It extends to the list decoding radius via the techniques of DEEP-FRI [6] with little efficiency loss. In the list decoding radius, many polynomials may be close to the committed codeword. This list can be reduced to size one [6,21].

- $C \leftarrow \mathcal{S}_1\left(1^\lambda, \mathsf{pp}\right)$

1. Randomly pick a size-N multilinear polynomial $\tilde{f}'$, split it as ℓ size-m polynomials, and add m random entries to the right end of each polynomial's coefficients. Denote these size-$2m$ univariate polynomials as $\hat{f}'^{(1)}, \ldots, \hat{f}'^{(\ell)}$.
2. Randomly select a size-$2m$ polynomial $\hat{s}'$.
3. Run $C \leftarrow \mathsf{MT.Com}([\hat{f}'^{(1)}|_{L_0}, \ldots, \hat{f}'^{(\ell)}|_{L_0}, \hat{s}'|_{L_0}])$.

- $\langle \mathcal{S}_2, \mathcal{A} \rangle(\mathsf{pp}, C, \boldsymbol{x})$

1. Send $(y, \hat{s}'(\boldsymbol{x}, 0))$ to $\mathcal{A}$ given oracle access to $y = \tilde{f}(\boldsymbol{x})$.
2. Receive α from $\mathcal{A}$.
3. Given the query set of both consistency and validity checks $B = \{\beta_j\}_{j \in [Q]}$, $\mathcal{S}_2$ generates a size-m polynomial $\tilde{f}'_1$ subject to the following constraints. Let $\hat{f}'_1$ be the corresponding univariate polynomial of $\tilde{f}'_1$. Let $\hat{f}'_0 \leftarrow \sum_{i=1}^{\ell} w_i \hat{f}'^{(i)}$ for $\boldsymbol{w} = \otimes_{k \in [\log \ell]}(1, x_{k+\log m})$.
   - For $\beta_j \in B$, set

$$\tilde{f}'_1(\beta_j^2) = \frac{\hat{f}'_0(\beta_j) + \alpha \hat{s}'(\beta_j) + \hat{f}'_0(-\beta_j) + \alpha \hat{s}'(-\beta_j)}{2} + x_1 \cdot \frac{\hat{f}'_0(\beta_j) + \alpha \hat{s}'(\beta_j) - \hat{f}'_0(-\beta_j) - \alpha \hat{s}'(-\beta_j)}{2\beta_j}.$$

   - $\tilde{f}'_1(x_2, x_3, \ldots, x_{\log m}, 0) = y$.

   $\mathcal{S}_2$ opens $\hat{f}'^{(1)}, \ldots, \hat{f}'^{(\ell)}$ to open $\hat{f}'_0$ honestly, sets $\hat{f}'_1$ as the second-round polynomial, and acts honestly thereafter.

Figure 9: Simulator $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2)$ of zk-PIP$_{\mathrm{FRI}}$

Define $\mathrm{err}_0 = \emptyset$. For $i \in [\sigma + 1]$, let the set E be $\{x \in L_{i-1} \colon \ell_i(x^2) + \alpha r_i(x^2) + \alpha_i^2 \hat{f}_i(x^2) \neq \hat{f}'_i(x^2)\}$, where $\hat{f}'_i(X) = \ell_i(X^2) + X \cdot r_i(X^2)$. Define $\mathrm{err}_i = \mathrm{err}_{i-1} \cup \mathrm{back}_{i-1}(E)$. For $i \in [\sigma + 1]$, suppose that $\hat{h}^{(i)}|_{L_i} \in \mathrm{RS}[L_i, \rho]$ and set $A_i = \{x \in L_i | \hat{f}'_i(x) = \hat{h}^{(i)}(x)\}$. Define $\varepsilon_i(\cdot) \colon L_i \to \{0, 1\}$ such that $\varepsilon_i(A_i) = |\mathrm{back}_i(A_i) \cap \mathrm{err}_i|/|L_0| + (1 - |A_i|/|L_i|)$. By definition, $|\mathrm{back}_i(A_i)| = 2^i |A_i|$, $A_i \subseteq L_i$, and $\mathrm{err}_i \subseteq L_i$. Intuitively, $\varepsilon_i(A_i)$ refers to the probability that the verifier finds an inconsistency in the i-th round. Further, $\varepsilon_{\sigma+1}(A_{\sigma+1})$ refers to the probability that the verifier finds an inconsistency and rejects with a single query in Step 5 of Protocol 3. Suppose:

$$\Pr[\varepsilon_{\sigma+1}(A_{\sigma+1}) \geq (1 - \rho)/2] \geq 1 - |L_0|/|\mathbb{F}|. \tag{3}$$

That is, with probability at least $1 - |L_0|/|\mathbb{F}|$, the verifier can catch an inconsistency of about $\delta = (1 - \rho)/2$ if it picks only one query. For q' queries, the probability that a malicious prover cheats successfully will be $(1 - \delta)^{q'} = ((1 + \rho)/2)^{q'}$.

According to PolyFRIM [44, Theorem 3.1], to prove Inequality (3), it suffices to prove $\Pr[\varepsilon_1(A_1) \geq \delta] \geq 1 - |L_0|/|\mathbb{F}|$ in three cases below: (a) For all $i \in [1, \sigma]$, $\Delta(\hat{f}'_i|_{L_i}, \mathrm{RS}[L_i, \rho]) \leq \delta$; (b) For all $i \geq 2$, $\Delta(\hat{f}_i|_{L_i}, \mathrm{RS}[L_i, \rho]) \leq \delta$; (c) At least one of $\Delta(\hat{f}'_0|_{L_0}, \mathrm{RS}[L_0, \rho]) > \delta$ and $\Delta(\hat{f}_1|_{L_1}, \mathrm{RS}[L_1, \rho]) > \delta$ holds.

In the cases above, (a) and (b) are good ones. We only need to focus on (c). Here, $\hat{f}'_0(X) = \sum_{i=1}^{\ell} \alpha_0^{i-1} \hat{f}^{(i)}(X)$ and $\hat{f}'_1(X) = \hat{f}'_0(X) + \alpha_0^{\ell} \hat{f}_1(X)$. By Lemma 2, for random α_0, with a probability of $1 - |L_0|/|\mathbb{F}|$, it holds that $\Delta(\hat{f}'_1|_{L_1}, \mathrm{RS}[L_1, \rho]) \geq \delta$. The soundness of Case 1 hence follows.

Lemma 2 (The generalized correlated agreement [5])**.** *For $\delta < (1 - \rho)/2$, $\{\hat{f}_i\}_{i \in [n-1]}$, multiplicative coset L, and code rate ρ, if $\Pr_{\alpha \leftarrow_{\$} \mathbb{F}}[\Delta(\sum_{i=0}^{n-1} \alpha^i \cdot \hat{f}_i|_L, \mathrm{RS}[L, \rho]) \leq \delta] \geq |L|/|\mathbb{F}|$, where $\Delta(\boldsymbol{a}, \boldsymbol{b})$ is the relative Hamming distance between $\boldsymbol{a}$ and $\boldsymbol{b}$, there exists $L' \subset L$ and $\hat{p}_i$ with the same degree bounds as $\hat{f}_i$ s.t. $|L'|/|L| \geq 1 - \delta$ and $\hat{f}_i|_{L'} = \hat{p}_i|_{L'}$ for $i \in [0, n-1]$.*

Zero Knowledge. Figure 9 defines the simulator $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2)$. Each $\hat{f}'^{(j)}$ is formed by appending m random coefficients, which masks evaluations on any query set B of size $|B| = O(\lambda) \ll m$. After the first round, the masking polynomial $\hat{s}'$ further randomizes the opened values, so the opened evaluations are statistically independent of the witness conditioned on the transcript shape. The simulator does not need to know α in advance. The simulator runs efficiently because $\tilde{f}'_1$ can be obtained by solving a linear system with $|B| + 1$ constraints and m variables.