Code-based Distributed Polynomial Commitment Scheme with Linear Prover Time and Polylogarithmic Communication

Zesheng Li[1,2], Xinzhe Chen[3], Yuejia Cheng[4], Zixing Wang[5], Yihang Du[6], Xinxuan Zhang[1,2], and Yi Deng[7]

[1] State Key Laboratory of Cyberspace Security Defense, Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China
[2] School of Cyber Security, University of Chinese Academy of Sciences, Beijing, China
[3] National University of Singapore
[4] Sussex University
[5] School of Cyber Science and Technology, Shandong University
[6] College of Computer Science and Electronic Engineering, Hunan University
[7] Xidian University
lizesheng@iie.ac.cn
asleep@u.nus.edu
chengyuejia@foxmail.com
zixing@mail.sdu.edu.cn
202208060130@hnu.edu.cn
zhangxinxuan@iie.ac.cn
ydeng.cas@gmail.com

Abstract. We present the first fully distributed, code-based polynomial commitment scheme (PCS) that achieves linear prover time and polylogarithmic communication for multilinear polynomials. Our construction builds on Brakedown with proof composition. We design a distributed proof of correct linear encoding. Codeword validity is reduced to a sum-check relation over the sparse parity-check matrix, whereas systematic consistency is verified through multilinear evaluation checks. These techniques preserve linear prover complexity while reducing the communication size between sub-provers to polylogarithmic scale.
We implement our construction and evaluate it on multilinear polynomials ranging in size from 2^{22} to 2^{28}. The prover exhibits near-linear parallel scalability. Compared with existing evaluated distributed PCS implementations with 8 parties, it reduces communication by at least $13\times$ at every tested polynomial size, while its prover time remains within $1.31\times$ of the fastest implementation. These results demonstrate that our asymptotic communication improvements translate into substantial concrete savings while retaining efficient proving.

Keywords: Distributed · Polynomial Commitment · Scalable

1 Introduction

The Polynomial commitment scheme (PCS) is a cryptographic primitive that enables a prover to commit to a polynomial and subsequently convince a ver-

ifier that it evaluates correctly at a specified point. In recent years, as many modern SNARK constructions have adopted the "PIOP+PCS" paradigm, the study of polynomial commitment schemes (PCS) [7,26,8,15,1,2,27] has emerged as a central focus in SNARK research. While SNARKs are being deployed in an expanding range of applications such as ZCash [3] and zkRollup [5], their substantial memory requirements present a critical bottleneck.

To overcome this limitation, a feasible solution is to construct distributed proof systems. A prominent category includes pairing-based elliptic curve schemes such as DIZK [20], Pianist [13], Hekaton [17], HyperPianist [10], Cirrus [18], and Soloist [12]. Despite achieving sublinear verification costs and supporting general arithmetic circuits, these systems incur significant prover overhead due to computationally expensive group and pairing operations. Moreover, they are not transparent.

There are several code-based distributed SNARKs, which are transparent and post-quantum. DeVirgo [22] is the first code-based distributed SNARK. However, it only supports data-parallel circuits. Subsequent works have primarily focused on constructing code-based distributed polynomial commitment schemes (PCS) for multilinear polynomials, which are then compiled with an existing distributed PIOP. For example, FRIttata [23] and dePIP$_{\text{FRI}}$ [11] are FRI-based polynomial commitments, utilizing distinct techniques to optimize efficiency. Leveraging Reed-Solomon (RS) codes, LigeSIS [9] and Veloz [25] introduce code-based PCSs featuring polylogarithmic proof sizes and verification times that scale independently of the number of machines. While the Veloz [25] framework can be adapted to Brakedown to achieve linear prover time, these schemes suffer from excessive communication overhead between sub-provers, generally scaling beyond the square root of the circuit size. Furthermore, HyperFond [24] presents a distributed version of BaseFold [26], which is a code-based distributed PCS with polylogarithmic communication. UltraFold [19] further employs an interleaved Merkle-leaf layout and packed Merkle trees to achieve a distributed BaseFold construction whose proof size is independent of the number of machines. However, both constructions retain quasi-linear prover complexity. This naturally leads to the following question:

Can we design a transparent, post-quantum distributed PCS with a linear-time prover and polylogarithmic communication between sub-provers?

1.1 Our Contribution

In this paper, we present the first fully distributed, code-based polynomial commitment scheme featuring linear-time prover computation alongside polylogarithmic communication, verification time, and proof size, and affirmatively answer the above question. A detailed comparison between our work and prior works is summarized in Table 1. The total communication size of our scheme is $O(\lambda M \log^2 \frac{N}{M})$, which is polylogarithmic in the polynomial size N, assuming the number of sub-provers M is independent of N. As for the security model, we assume that the distributed workers are trusted and use distribution solely for parallel acceleration.

We further implement the proposed PCS by instantiating the underlying polynomial commitment with DeepFold [8] and evaluate its concrete performance against existing distributed PCS implementations. Our experiments demonstrate near-linear parallel scalability. In the 8-party comparison, our scheme communicates only 2.28–4.75 MB across the tested range, at least 13× less than every evaluated baseline, while maintaining prover performance within 1.31× of the fastest implementation. At $N = 2^{28}$, it requires 22.7 s of prover time, 4.18 ms of verifier time, and 4.75 MB of communication between parties; the latter is 20.6×, 319×, 399×, and 1,583× smaller than HyperFond [24], LigeSIS [9], dPIP-FRI [11], and dFRIttata [23], respectively. These results confirm that our construction achieves its main design goal in practice: substantially reducing communication while retaining efficient proving.

Table 1. Comparison between our work and other code-based distributed polynomial commitments. Here N denotes the length of the committed multilinear polynomial (equivalently, its Boolean evaluation table), M denotes the number of sub-provers, and $T := \frac{N}{M}$. In FRIttata [23], K denotes the "Fold-and-Batch" folding parameter, where $K \in [1, T]$. In Veloz [25], K denotes the flexible parameter where $K \in [N^{\frac{1}{3}}, N^{\frac{1}{2}}]$ and is a power of two. "$\mathcal{P}_i$ Time" stands for the running time of each sub-prover, "Comm" stands for the total communication size between sub-provers, and "$\mathcal{V}$ Time" stands for the verification time. "Ours" stands for our distributed polynomial commitment scheme.

Scheme	$\mathcal{P}_i$ Time	Comm.	Proof Size	$\mathcal{V}$ Time
deVirgo [22]	$O(T \log T)$	$O(N)$	$O(\lambda \log^2 N)$	$O(\lambda \log^2 N)$
HyperFond [24]	$O(T \log T)$	$O(\lambda M \log^2 T)$	$O(\lambda M \log^2 T)$	$O(\lambda M \log^2 T)$
FRIttata [23]	$O(T \log T)$	$O(MK + \lambda \cdot M \log T \log \frac{T}{K})$	$O(\lambda \cdot (M \log T \log \frac{T}{K} + \log^2 K))$	$O(\lambda \cdot (M \log T \log \frac{T}{K} + \log^2 K))$
DePIP$_{\text{FRI}}$ [11]	$O(T \log T)$	$O(N)$	$O(\lambda \cdot M + \log^2 T)$	$O(\lambda \cdot M + \log^2 T)$
UltraFold [19]	$O(T \log N)$	$O(N)$	$O(\lambda \log^2 N)$	$O(\lambda \log^2 N)$
LigeSIS [9]	$O(T \log T)$	$O(\sqrt{\lambda M N})$	$O(\lambda \log^2 N)$	$O(\lambda \log^2 N)$
Veloz$_{\text{RS}}$ [25]	$O(T \log \frac{N}{\log N})$	$O(\lambda \cdot \frac{\log^2 N}{\log \log N} + M \cdot \frac{N}{\log N})$	$O(\lambda \cdot \frac{\log^2 N}{\log \log N})$	$O(\lambda \cdot \frac{\log^2 N}{\log \log N})$
Veloz$_{\text{Lin}}$ [25]	$O(T)$	$O(\lambda \cdot K + M \cdot \frac{N}{K})$	$O(\lambda \cdot K)$	$O(\lambda \cdot (K + \frac{N}{K}))$
Ours	$O(T)$	$O(\lambda M \log^2 T)$	$O(\lambda M \log^2 T)$	$O(\lambda M \log^2 T)$

1.2 Technique Overview

We now provide an overview of the techniques used in our distributed polynomial commitment scheme. Our starting point is the Brakedown [7] polynomial commitment, which is transparent, post-quantum, and features a linear-time prover. Brakedown employs a linear-time encoding scheme Enc. To prove the evaluation $f(\mathbf{x}_1, \mathbf{x}_2) = v$ for a μ-variate multilinear polynomial f, its evaluations are arranged into a matrix $M_f \in \mathbb{F}^{n_1 \times n_2}$, where $n_1 n_2 = 2^\mu = N$. In the commit phase, the prover encodes each row of M_f to obtain $M'_f \in \mathbb{F}^{n_1 \times cn_2}$ and commits to M'_f via a Merkle tree. The evaluation phase consists of a proximity test and a consistency check. In the proximity test, the verifier sends a random

vector $\mathbf{a} \in \mathbb{F}^{n_1}$, and the prover responds with the linear combination vectors $F_1 = \mathbf{a}^T M_f$, $F_2 = \mathbf{e}_{row}^T M_f$, where $\mathbf{e}_{row} = \otimes_{i=0}^{\log n_1 - 1}(1 - \mathbf{x}_1[i], \mathbf{x}_1[i])$. The verifier samples a random index set $\mathcal{I} = \{idx_1, \ldots, idx_\ell\}$; the prover returns the corresponding columns $R \in \mathbb{F}^{n_1 \times \ell}$ of M'_f with Merkle proofs. The verifier checks the proofs and that $E_1[idx_i] = \sum_{j \in [n_1]} \mathbf{a}[j] R[j, i], E_2[idx_i] = \sum_{j \in [n_1]} \mathbf{e}_{row}[j] R[j, i]$, for each $i \in [\ell]$, where $E_1 = \mathsf{Enc}(F_1), E_2 = \mathsf{Enc}(F_2)$. In the consistency check, the verifier checks that $v = F_2 \mathbf{e}_{col}$, where $\mathbf{e}_{col} = \otimes_{i=0}^{\log n_2 - 1}(1 - \mathbf{x}_2[i], \mathbf{x}_2[i])$.

Subsequent works [15] and [16] optimize the scheme via proof composition: f is arranged into a wider matrix $M_f \in \mathbb{F}^{B \times \frac{N}{B}}$ and the linearly combined vectors E_1, E_2, F_1, F_2 are committed. The verification of $E_1 = \mathsf{Enc}(F_1)$ and $E_2 = \mathsf{Enc}(F_2)$ is embedded into an arithmetic circuit and proven using the sumcheck protocol, reducing proof size and verification time ($O(\log^2 N)$ for [15] and $O(\frac{N}{B} + \log^2 B)$ for [16]). Details are provided in Subsection 3.1.

In the distributed setting involving M sub-provers (one of the M sub-provers additionally acts as the master prover), two main challenges arise: (1) how to generate commitments to the linearly combined vectors E_1, E_2, F_1, F_2, and (2) how to efficiently prove relations expressed as arithmetic circuits, i.e. $E_1 = \mathsf{Enc}(F_1), E_2 = \mathsf{Enc}(F_2)$. We outline how to address these two challenges.

Committing to linearly combined vectors. In the Brakedown protocol with proof composition, the prover commits to F_1, F_2, as well as to their respective encoding vectors $E_1 = \mathsf{Enc}(F_1), E_2 = \mathsf{Enc}(F_2)$. We illustrate the process using F_1 as an example; the procedures for F_2, E_1 and E_2 are similar to that for F_1.

The matrix M_f is partitioned row-wise into M sub-matrices $M_f^{(0)}, \ldots, M_f^{(M-1)} \in \mathbb{F}^{\frac{B}{M} \times \frac{N}{B}}$, and the randomly sampled vector $\mathbf{a} \in \mathbb{F}^B$ is divided into M segments $\mathbf{a}^{(0)}, \ldots, \mathbf{a}^{(M-1)} \in \mathbb{F}^{\frac{B}{M}}$. The i-th sub-prover, holding $M_f^{(i)}$ and $\mathbf{a}^{(i)}$, computes the partial sum $F_1^{(i)} = \sum_{k \in \{0, \ldots, \frac{B}{M} - 1\}} \mathbf{a}^{(i)}[k] M_f^{(i)}[k]$, such that the full vector is obtained as $F_1 = \sum_{i \in \{0, \ldots, M-1\}} F_1^{(i)}$. We demonstrate that for a linearly combined polynomial $\sum_i a_i f_i$, it suffices to commit to and open each f_i individually using an arbitrary polynomial commitment scheme. In the distributed setting, each f_i is committed and opened by the i-th sub-prover. In the case of committing to F_1 in a distributed manner, we have $f_i = F_1^{(i)}$ and $a_i = 1$ for all i. For the Merkle-tree hash commitments, each sub-prover generates a commitment and a corresponding opening proof for the matrix or vector it holds. The verifier checks each proof individually, and the overall verification succeeds only if every proof is valid.

When $B = \Theta(M \log \frac{N}{M})$, the prover time can achieve linear complexity. Compared to the underlying polynomial commitment, our method introduces an $O(M)$-factor overhead in proof size and verification time.

For more details, please refer to Subsection 3.2.

Proving the circuit relation The core task is to prove that $E_1, E_2 \in \mathbb{F}^{cl}$ are valid encodings of $F_1, F_2 \in \mathbb{F}^l$ under a systematic code with rate $1/c$. For simplicity, we explain how to prove the encoding algorithm $E = Enc(F)$. This requires

establishing two conditions: (1) Codeword validity: $E \cdot H = 0$, where H is a sparse parity-check matrix; and (2) Systematic consistency: the first l entries of E are equal to F. As proven in [15], the parity-check matrix used in Brakedown's linear code is sparse; therefore, H is sparse.

For condition (1), we express the check as a sumcheck relation $\sum \tilde{E}(b) \cdot \tilde{H}_u(b) = 0$, where $\tilde{H}_{\mathbf{u}}(b) = \tilde{H}(b, \mathbf{u})$. When we attempt to apply the distributed sumcheck protocol in HyperPianist [10], an issue arises because the i-th prover holds the $\tilde{E}^{(i)}$ such that $\tilde{E} = \sum_{i \in \{0,\dots M-1\}} \tilde{E}^{(i)}$, rather than the i-th partial polynomial of $\tilde{E}$ as required by the distributed sumcheck protocol in HyperPianist [10]. A key observation is that every sub-prover $\mathcal{P}_i$ holds the parity-check matrix H. This is feasible because H is sparse, containing only $O(\frac{cN}{B})$ non-zero entries. Consequently, each sub-prover can locally compute the evaluations of $\tilde{H}_u$ efficiently. In the k-th round of sumcheck, the i-th sub-prover computes $\tilde{E}_k^{(i)}(X_k, x) = \tilde{E}^{(i)}(r_1, \dots, r_{k-1}, X_k, x)$,$\tilde{H}_{\mathbf{u},k}(X_k, x) := \tilde{H}_{\mathbf{u}}(r_1, \dots, r_{k-1}, X_k, x)$, and then $Q_k^{(i)}(X_k) \leftarrow \sum_{x \in \{0,1\}^{\log \frac{cN}{B} - k}} \tilde{E}_k^{(i)}(X_k, x) \cdot \tilde{H}_{\mathbf{u},k}(X_k, x)$. The master prover $\mathcal{P}_0$ then aggregates the partial results from sub-provers to $Q_k(X_k)$. At the final stage, the "query" for $\tilde{E}$ is handled by our distributed polynomial commitment scheme, while the query for $\tilde{H}_u$ is handled using the distributed BaseFold protocol.

For condition (2), it is enough to prove that $\tilde{E}(0^{\log c}, x) = \tilde{F}(x)$ for any $x \in \{0,1\}^{\log l}$. This reduces to querying the values of $\tilde{E}$ and $\tilde{F}$ at specific points. These queries are instantiated directly using the evaluation phases of our distributed polynomial commitment scheme, ensuring linear prover time and polylogarithmic verification costs.

For more details, please refer to Subsection 3.3.

2 Preliminary

Notation Let λ denote the security parameter. A function $f(\lambda)$ is *negligible* (denoted by $\mathsf{negl}(\lambda)$) if $\forall c > 0$, $f(\lambda) = o(\lambda^{-c})$. For $n \in \mathbb{N}$, define $[n] := \{0, 1, \dots, n-1\}$, and $[a : b] = \{a, a+1, \dots, b\}$. Let $\mathbb{F}$ be a finite field with $|\mathbb{F}| = \Omega(2^\lambda)$. $\mathcal{F}_\mu^{(\leq d)}(X)$ denotes the set of μ-variate polynomials of degree at most d over $\mathbb{F}$.

Let v be a vector and M a matrix. Denote by $v[i]$ the i-th entry of v, and by $M[i, j]$ the entry in the i-th row and j-th column of M. Let $M[i, :]$ represent the row vector corresponding to the i-th row of M, and $M[:, j]$ the column vector corresponding to the j-th column of M.

Let $bin : [2^d] \to \{0,1\}^d$ be a function that outputs the binary representation of a number, i.e., $\mathbf{b} = bin(c)$ satisfies $c = \sum_{i \in [d]} 2^{d-1-i} \cdot \mathbf{b}[i]$, and let $bin^{-1} : \{0,1\}^d \to [2^d]$ denote its inverse mapping.

For simplicity, we assume that all vector dimensions used for multilinear extensions are powers of two. Otherwise, the corresponding vectors and matrices are padded with zeros to the next admissible dimension.

Definition 1 (Argument of Knowledge [4]). *An interactive protocol* $\Pi = (\mathsf{Setup}, \mathcal{I}, \mathcal{P}, \mathcal{V})$ *is an argument of knowledge for an indexed relation* $\mathcal{R}$ *with knowledge error* $\delta : \mathbb{N} \to [0,1]$ *if the following properties hold, where given an index* $\mathtt{i}$*, common input* $\mathtt{x}$ *and prover witness* $\mathtt{w}$*, the deterministic indexer outputs* $(\mathsf{vk}, \mathsf{pk}) \leftarrow \mathcal{I}(\mathsf{pp}, \mathtt{i})$ *and the output of the verifier is denoted by the random variable* $\langle \mathcal{P}(\mathsf{pk}, \mathtt{x}, \mathtt{w}), \mathcal{V}(\mathsf{vk}, \mathtt{x}) \rangle$*:*

- ***Completeness****: For all* $(\mathtt{i}, \mathtt{x}, \mathtt{w}) \in \mathcal{R}$

$$\Pr\left[\langle \mathcal{P}(\mathsf{pk}, \mathtt{x}, \mathtt{w}), \mathcal{V}(\mathsf{vk}, \mathtt{x}) \rangle = 1 \;\middle|\; \begin{array}{c} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda) \\ (\mathsf{vk}, \mathsf{pk}) \leftarrow \mathcal{I}(\mathsf{pp}, \mathtt{i}) \end{array}\right] = 1.$$

- δ-***Knowledge Soundness****: There exists a PPT extractor* $\mathcal{E}$ *such that given oracle access to any pair of PPT adversarial algorithms* $(\mathcal{A}_1, \mathcal{A}_2)$*, the following holds:*

$$\Pr\left[\begin{array}{c} \langle \mathcal{A}_2(\mathtt{i}, \mathtt{x}, \mathsf{st}), \mathcal{V}(\mathsf{vk}, \mathtt{x}) \rangle = 1 \\ \wedge \\ (\mathtt{i}, \mathtt{x}, \mathtt{w}) \notin \mathcal{R} \end{array} \;\middle|\; \begin{array}{c} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda) \\ (\mathtt{i}, \mathtt{x}, \mathsf{st}) \leftarrow \mathcal{A}_1(\mathsf{pp}) \\ (\mathsf{vk}, \mathsf{pk}) \leftarrow \mathcal{I}(\mathsf{pp}, \mathtt{i}) \\ \mathtt{w} \leftarrow \mathcal{E}^{\mathcal{A}_1, \mathcal{A}_2}(\mathsf{pp}, \mathtt{i}, \mathtt{x}) \end{array}\right] \leq \delta(|\mathtt{i}| + |\mathtt{x}|).$$

  An interactive protocol has ***knowledge soundness*** *property if the knowledge error* δ *is negligible in* λ*.*

An interactive argument of knowledge protocol can be made non-interactive by the Fiat-Shamir transformation [6] if it is public coin.

- ***Public coin****: An interactive protocol is public-coin if* $\mathcal{V}$*'s messages are chosen uniformly at random.*

If the protocol further satisfies succinctness, then it is a Succinct Non-interactive Argument of Knowledge(SNARK).

- ***Succinctness****: The proof size is* $|\pi| = \mathsf{poly}(\lambda, \log|\mathcal{C}|)$ *and the verification time is* $\mathsf{poly}(\lambda, \log|\mathcal{C}|)$*, where* $\mathcal{C}$ *is the arithmetic circuit representing the relation* $\mathcal{R}$*.*

Definition 2 (Polynomial Commitment [26]). *A polynomial commitment scheme is a tuple of PPT algorithms* $(\mathsf{Gen}, \mathsf{Commit}, \mathsf{Open}, \mathsf{Eval})$ *where:*

- $\mathsf{Gen}(1^\lambda) \to \mathsf{pp}$ *takes the security parameter* λ*, and outputs the public parameter* pp*.*
- $\mathsf{Commit}(\mathsf{pp}, f) \to \mathsf{com}_f$ *takes the public parameter* pp *and a multilinear polynomial* f*, outputs a public commitment* com_f*;*
- $\mathsf{Open}(\mathsf{pp}, \mathsf{com}_f, f) \to b$ *takes the public parameter* pp *and the public commitment* com_f*, outputs a bit* b*;*
- $\mathsf{Eval}(\mathsf{pp}, \mathsf{com}_f, \boldsymbol{x}, y; f) \to b$ *is a protocol between the prover* $\mathcal{P}$ *and verifier* $\mathcal{V}$ *with public parameter* pp*, the commitment* com_f*, an evaluation point* $\boldsymbol{x} \in \mathbb{F}^\mu$*, the value* y *and the multilinear polynomial* f*.* $\mathcal{P}$ *secretly holds a multilinear polynomial* f *and wants to convince* $\mathcal{V}$ *that* com_f *is a commitment to* f *and* $f(\boldsymbol{x}) = y$*. At the end of the protocol,* $\mathcal{V}$ *outputs a bit* b*. We write* $\mathsf{Eval} = (\mathsf{Eval.Prove}, \mathsf{Eval.Verify})$ *for its prover and verifier algorithms.*

A polynomial commitment satisfies the following properties:

- ***Completeness.*** *For any $\mu \in \mathbb{N}$, polynomial $f \in \mathcal{F}_\mu^{(\leq 1)}$ and $\boldsymbol{x} \in \mathbb{F}^\mu$,*

$$Pr\left[\mathsf{Eval}(\mathsf{pp}, \mathsf{com}_f, \boldsymbol{x}, y; f) = 1 \wedge f(\boldsymbol{x}) = y \;\middle|\; \begin{array}{c} \mathsf{pp} \leftarrow \mathsf{Gen}(1^\lambda) \\ \mathsf{com}_f \leftarrow \mathsf{Commit}(\mathsf{pp}, f) \end{array}\right] = 1$$

- ***Binding.*** *For any $\mu \in \mathbb{N}$, and any PPT adversary $\mathcal{A}$,*

$$Pr\left[b_0 = b_1 = 1 \wedge f_0 \neq f_1 \;\middle|\; \begin{array}{c} \mathsf{pp} \leftarrow \mathsf{Gen}(1^\lambda) \\ (\mathsf{com}, f_0, f_1) \leftarrow \mathcal{A}(\mathsf{pp}) \\ b_0 \leftarrow \mathsf{Open}(\mathsf{pp}, \mathsf{com}, f_0) \\ b_1 \leftarrow \mathsf{Open}(\mathsf{pp}, \mathsf{com}, f_1) \end{array}\right] \leq \mathsf{negl}(\lambda)$$

- ***Knowledge soundness.*** *For any PPT adversary $\mathcal{P}^*$, there exists a PPT extractor $\mathcal{E}$ with access to $\mathcal{P}^*$'s messages during the protocol,*

$$Pr\left[\begin{array}{c} \mathsf{pp} \leftarrow \mathsf{Gen}(1^\lambda) \\ (y^*, \boldsymbol{x}^*, \mathsf{com}^*) \leftarrow \mathcal{P}^*(\mathsf{pp}) \\ f^* \leftarrow \mathcal{E}^{\mathcal{P}^*(\cdot)}(\mathsf{pp}) \\ \mathsf{Eval}(\mathsf{pp}, \mathsf{com}^*, \boldsymbol{x}^*, y^*; f^*) = 1 \end{array}\right] \approx Pr\left[\begin{array}{c} \mathsf{pp} \leftarrow \mathsf{Gen}(1^\lambda) \\ (y^*, \boldsymbol{x}^*, \mathsf{com}^*) \leftarrow \mathcal{P}^*(\mathsf{pp}) \\ f^* \leftarrow \mathcal{E}^{\mathcal{P}^*(\cdot)}(\mathsf{pp}) \\ ((\mathsf{com}^*, \boldsymbol{x}^*, y^*); f^*) \in \mathcal{R}_{Eval} \end{array}\right]$$

where $\mathcal{R}_{Eval} = \{((\mathsf{com}^*, \boldsymbol{x}^*, y^*); f) | f(\boldsymbol{x}^*) = y^* \wedge \mathsf{Open}(\mathsf{pp}, \mathsf{com}^*, f) = 1\}$.

Definition 3 (Polynomial Interactive Oracle Proof [4]). *A public-coin polynomial interactive oracle proof (PIOP) is a public-coin interactive proof for a polynomial oracle relation $\mathcal{R} = (\mathtt{i}; \mathtt{x}; \mathtt{w})$, where $\mathtt{i}$ and $\mathtt{x}$ can contain oracles to n-variate polynomials over some field $\mathbb{F}$. These oracles can be queried at arbitrary points in $\mathbb{F}^n$ to evaluate the polynomial at these points. We denote an oracle to a polynomial f by $[[f]]$.*

Definition 4 (Multilinear Extension). *For any $f : \{0,1\}^\mu \to \mathbb{F}$, there exists a unique multilinear polynomial $\tilde{f} : \mathbb{F}^\mu \to \mathbb{F}$ such that for all $\boldsymbol{x} \in \{0,1\}^\mu$, $\tilde{f}(\boldsymbol{x}) = f(\boldsymbol{x})$. The polynomial $\tilde{f}$ is called the multilinear extension (MLE) of f, and can be expressed as $\tilde{f}(\boldsymbol{X}) = \sum_{\boldsymbol{x} \in \{0,1\}^\mu} f(\boldsymbol{x}) \cdot \mathsf{eq}(\boldsymbol{x}, \boldsymbol{X})$, where $\mathsf{eq}(\boldsymbol{x}, \boldsymbol{X}) := \prod_{i=0}^{\mu-1} (\boldsymbol{x}[i]\boldsymbol{X}[i] + (1 - \boldsymbol{x}[i])(1 - \boldsymbol{X}[i]))$.*

We generalize the definition of MLE to vectors and matrices. For a vector $v \in \mathbb{F}^\ell$, the MLE of v is the multilinear polynomial $\tilde{v} : \mathbb{F}^{\log \ell} \to \mathbb{F}$ such that for all $\boldsymbol{x} \in \{0,1\}^{\log \ell}$, $\tilde{v}(\boldsymbol{x}) = v[bin^{-1}(\boldsymbol{x})]$. For the matrix $M \in \mathbb{F}^{m \times n}$, the MLE of M is the multilinear polynomial $\tilde{M} : \mathbb{F}^{\log m + \log n} \to \mathbb{F}$ such that for all $\boldsymbol{x} \in \{0,1\}^{\log m}$ and $\boldsymbol{y} \in \{0,1\}^{\log n}$, $\tilde{M}(\boldsymbol{x}, \boldsymbol{y}) = M[bin^{-1}(\boldsymbol{x}), bin^{-1}(\boldsymbol{y})]$.

Definition 5 (Sumcheck Relation for High Degree Polynomials [14,4]). *The sumcheck relation for high-degree multivariate polynomials $\mathcal{R}_{SUM}$ is the set of tuples $(\mathtt{x}; \mathtt{w}) = ((v, [[f]]); f)$, where $f \in \mathcal{F}_\mu^{(\leq d)}$, and $\sum_{x \in \{0,1\}^\mu} f(x) = v$.*

The sumcheck protocol has been shown to satisfy completeness and knowledge soundness, as established in [4]. Furthermore, a distributed variant of the sumcheck protocol is introduced in [10]. Additional details regarding the sumcheck protocol are provided in Appendix A.

3 Distributed Transparent PCS with Linear Prover Time

3.1 Brakedown with Proof Composition

Our starting point is Brakedown [7] polynomial commitment scheme, which employs a linear-time encoding scheme $Enc : \mathbb{F}^n \to \mathbb{F}^{cn}$, where c is the reciprocal of the rate of the code. Below, we summarize the Brakedown procedure for proving that $f(\mathbf{s}) = v$, where f is a μ-variate multilinear polynomial. Let $\mathbf{s} = \mathbf{s}_1||\mathbf{s}_2$, where $\mathbf{s}_1 \in \mathbb{F}^{\log n_1}$ and $\mathbf{s}_2 \in \mathbb{F}^{\log n_2}$.

In the commitment phase, the evaluations of f are arranged into a matrix $M_f \in \mathbb{F}^{n_1 \times n_2}$, where $n_1 n_2 = 2^\mu = N$. The prover computes an encoded matrix $M'_f \in \mathbb{F}^{n_1 \times cn_2}$, where each row $M'_f[i:]$ equals $Enc(M_f[i:])$ for all $i \in [n_1]$. Then the prover computes the Merkle commitment of M'_f as the commitment.

The evaluation phase consists of two main steps. The first step is the proximity test: the verifier sends a randomly sampled vector $\mathbf{a} \in \mathbb{F}^{n_1}$ to the prover, which responds with the linearly combined vectors $F_1, F_2 \in \mathbb{F}^{n_2}$ to the verifier, where $F_1[j] = \sum_{i \in [n_1]} \mathbf{a}[i] M_f[i,j]$ and $F_2[j] = \sum_{i \in [n_1]} \mathsf{eq}(\mathbf{s}_1, bin(i)) M_f[i,j]$. The verifier then samples a random index set $\mathcal{I} \subseteq [cn_2]$ of size $\ell = \Theta(\lambda)$ and sends it to the prover. Let $\mathcal{I} = \{idx_1, \ldots, idx_\ell\}$. The prover returns the columns of M'_f indexed by $\mathcal{I}$, denoted by $R \in \mathbb{F}^{n_1 \times \ell}$, along with the corresponding Merkle proofs. The verifier checks that: (1) all Merkle proofs are valid, and (2) $E_1[idx_i] = \sum_{j \in [n_1]} \mathbf{a}[j] R[j,i]$ for each $i \in [\ell]$, where $E_1 = Enc(F_1)$. This process is repeated, replacing F_1 with F_2. The second step is the evaluation check, in which the verifier checks that $v = \sum_{i \in [n_2]} F_2[i] \cdot \mathsf{eq}(\mathbf{s}_2, bin(i))$. The proof size is given by $max\{O(n_2), O(\lambda n_1)\}$. By optimizing the choice of n_1 and n_2, the proof size is minimized to $O(\sqrt{\lambda N})$. The verifier's runtime is dominated by the computation of E_1, resulting in complexity $O(\sqrt{N})$.

We show the Brakedown polynomial commitment with proof composition, as primarily outlined in [15] and [16]. The evaluations of the polynomial f are arranged into a matrix of dimensions $B \times \frac{N}{B}$ (typically $B \in [\log N : \sqrt{N}]$), such that $\mathbf{s}_1 \in \mathbb{F}^{\log B}$ and $\mathbf{s}_2 \in \mathbb{F}^{\log \frac{N}{B}}$.

There are two differences from the original Brakedown [7] construction: (1) Instead of sending the full vectors $F_1, F_2 \in \mathbb{F}^{\frac{N}{B}}$ and $E_1, E_2 \in \mathbb{F}^{\frac{cN}{B}}$ directly after linear combination, $\mathcal{P}$ sends to $\mathcal{V}$ the commitment to the $\tilde{E}_1, \tilde{F}_1, \tilde{E}_2, \tilde{F}_2$ and claims that $E_1 = Enc(F_1), E_2 = Enc(F_2)$ (as formally defined by the relation $\mathcal{R}$) in step 9, and $\mathcal{V}$ verifies the claim is valid. (2) To check $v = \sum_{i \in [\frac{N}{B}]} F_2[i] \cdot \mathsf{eq}(\mathbf{s}_2, bin(i))$, the prover opens the commitment to $\tilde{F}_2$ at the point $\mathbf{s}_2$ and the verifier checks the opening proof. The full construction is detailed in Protocol 1.

Protocol 1: Brakedown with Proof Composition

$\mathcal{P}$ claims $f(\mathbf{s}) = v$ to $\mathcal{V}$, where $f \in \mathcal{F}^{(\leq 1)}_{\log N}$, and $\mathbf{s} = \mathbf{s}_1||\mathbf{s}_2$, $\mathbf{s}_1 \in \mathbb{F}^{\log B}$, $\mathbf{s}_2 \in \mathbb{F}^{\log \frac{N}{B}}$. $PC = (Gen, Commit, Open, Eval)$ is the underlying polynomial commitment. $MT = (Commit, Eval)$ is Merkle commitment. $Enc : \mathbb{F}^{\frac{N}{B}} \to \mathbb{F}^{\frac{cN}{B}}$ is the map of the encoding algorithm for the linear

code in Brakedown [7]. Let $H(\cdot) : \mathbb{F}^* \to \mathbb{F}$ be a random oracle.

Commit phase:

Input: the public parameter pp, the multilinear polynomial f.

Output: the commitment C.

Arrange the evaluations of f in a matrix $M_f \in \mathbb{F}^{B \times \frac{N}{B}}$. $\mathcal{P}$ computes $M'_f \in \mathbb{F}^{B \times \frac{cN}{B}}$ such that $M'_f[k,:] \leftarrow Enc(M_f[k,:])$. Let $\mathbf{h} \in \mathbb{F}^{cN/B}$ be a vector such that $\mathbf{h}[i] \leftarrow H(M'_f[:,i])$, for each $i \in [cN/B]$. $\mathcal{P}$ computes $C \leftarrow MT.Commit(\mathbf{h})$.

Eval phase:

Public input: the public parameter pp, the commitment C, evaluation point $\mathbf{s} \in \mathbb{F}^{\log N}$, v.

Private input: the multilinear polynomial f.

1. $\mathcal{V} \to \mathcal{P}$: a random vector $\mathbf{a} \in \mathbb{F}^B$.
2. $\mathcal{P} \to \mathcal{V}$: the commitments $C_{E1}, C_{F1}, C_{E2}, C_{F2}$ to the multilinear extension of E_1, F_1, E_2 and F_2 using $PC.Commit$, where $E_1 = Enc(F_1)$, $F_1 = \sum_{i\in[B]} \mathbf{a}[i] M_f[i,:]$, $E_2 = Enc(F_2)$, $F_2 = \sum_{i\in[B]} \mathsf{eq}(\mathbf{s}_1, bin(i)) M_f[i,:]$.
3. $\mathcal{P} \to \mathcal{V}$: the Merkle-tree roots $rt_1 \leftarrow MT.Commit(E_1)$ and $rt_2 \leftarrow MT.Commit(E_2)$.
4. $\mathcal{V} \to \mathcal{P}$: the column indices $\mathcal{I} = \{idx_1, \ldots, idx_\ell\}$.
5. $\mathcal{P} \to \mathcal{V}$: columns $R \in \mathbb{F}^{\ell \times B}$ such that $R[j,:] = M'_f[:, idx_j]$, for all $idx_j \in \mathcal{I}$, vectors $\hat{\mathbf{h}}$ and proofs $\{\pi_j\}_{j\in[\ell]}$ for each selected column, where $\pi_j \leftarrow MT.Eval.Prove(\mathbf{h}, idx_j, \hat{\mathbf{h}}[j])$ and $\hat{\mathbf{h}}[j] = H(R[j,:])$ for $j \in [\ell]$.
6. $\mathcal{P} \to \mathcal{V}$: vectors $\mathbf{y}_1, \mathbf{y}_2$ such that $\mathbf{y}_1[i] = \sum_{j\in[B]} R[i,j] \cdot \mathbf{a}[j]$, $\mathbf{y}_2[i] = \sum_{j\in[B]} R[i,j] \cdot \mathsf{eq}(\mathbf{s}_1, bin(j))$ for $i \in [\ell]$.
7. $\mathcal{P} \to \mathcal{V}$: the proof of polynomial commitment $\pi_{F2} \leftarrow PC.Eval.Prove(F_2, \mathbf{s}_2, v)$.
8. $\mathcal{P} \to \mathcal{V}$: the Merkle-hashing opening proofs $\{\Pi_{1,j}, \Pi_{2,j}\}_{j\in[\ell]}$, where $\Pi_{1,j} \leftarrow MT.Eval.Prove(E_1, idx_j, e_{1,j})$, $\Pi_{2,j} \leftarrow MT.Eval.Prove(E_2, idx_j, e_{2,j})$, for $j \in [\ell]$, and the claimed values $\{e_{1,j}, e_{2,j}\}_{j\in[\ell]}$.
9. $\mathcal{P} \to \mathcal{V}$: the proof π_r for the relation $\mathcal{R}$, where $\mathcal{R} = \{(C_{E1}, C_{F1}, C_{E2}, C_{F2}; E_1, F_1, E_2, F_2) | E_1 = Enc(F_1) \wedge E_2 = Enc(F_2) \wedge C_{E1} = PC.Commit(\tilde{E}_1) \wedge C_{F1} = PC.Commit(\tilde{F}_1) \wedge C_{E2} = PC.Commit(\tilde{E}_2) \wedge C_{F2} = PC.Commit(\tilde{F}_2)\}$.
10. $\mathcal{V}$ checks:
    (a) $\hat{\mathbf{h}}[j] = H(R[j,:])$ and $MT.Eval.Verify(C, \pi_j, \hat{\mathbf{h}}[j], idx_j) = 1$, for $j \in [\ell]$.
    (b) $\mathbf{y}_1[i] = \sum_{j\in[B]} R[i,j] \cdot \mathbf{a}[j]$, $\mathbf{y}_2[i] = \sum_{j\in[B]} R[i,j] \cdot \mathsf{eq}(\mathbf{s}_1, bin(j))$ for $i \in [\ell]$.
    (c) $PC.Eval.Verify(C_{F2}, \pi_{F2}, v, \mathbf{s}_2) = 1$
    (d) $\mathbf{y}_1[j] = e_{1,j}, \mathbf{y}_2[j] = e_{2,j}$, $MT.Eval.Verify(rt_1, \Pi_{1,j}, e_{1,j}, idx_j) = 1$, $MT.Eval.Verify(rt_2, \Pi_{2,j}, e_{2,j}, idx_j) = 1$, for $j \in [\ell]$.
    (e) the relation $\mathcal{R}$ is valid with the proof π_r.

There are two main challenges when considering the Brakedown protocol in a distributed setting: (1) handling commitments to linearly combined vectors, and (2) proving the relation $\mathcal{R}$. These are discussed in Subsection 3.2 and Subsection 3.3, respectively.

3.2 Distributed Commitments to Combined Vectors in Brakedown

This subsection discusses the distributed generation of commitments, specifically the Merkle hash commitment and the polynomial commitment for the linearly combined vectors $F_1, F_2 \in \mathbb{F}^{\frac{N}{B}}$ and $E_1, E_2 \in \mathbb{F}^{\frac{cN}{B}}$, where $E_1 = Enc(F_1)$, $F_1 = \sum_{i\in[B]} \mathbf{a}[i]M_f[i,:]$, $E_2 = Enc(F_2)$, $F_2 = \sum_{i\in[B]} \mathsf{eq}(\mathbf{s}_1, bin(i))M_f[i,:]$.

Merkle commitment In the Merkle commitment scheme, the prover claims to the verifier that $E_1[idx] = e_1, E_2[idx] = e_2$, where $idx \in [\frac{cN}{B}]$, and E_1, E_2 are linearly combined vectors in Protocol 1, which are committed using Merkle commitments. In the distributed setting, the matrix is partitioned row-wise and evenly distributed among the provers. Each sub-prover $\mathcal{P}_i$ computes the partial vectors $E_1^{(i)} = \sum_{k\in[b]} \mathbf{a}^{(i)}[k]Enc(M_f^{(i)})[k]$, and $E_2^{(i)} = \sum_{k\in[b]} \mathsf{eq}(\mathbf{s}_1, bin(ib + k)) \cdot Enc(M_f^{(i)})[k]$. Then $\mathcal{P}_i$ constructs a local Merkle root for $E_1^{(i)}, E_2^{(i)}$, denoted $rt_{E1}^{(i)}, rt_{E2}^{(i)}$, and sends them to the master prover $\mathcal{P}_0$. $\mathcal{P}_0$ forwards the set of all roots $\{rt_{E1}^{(i)}, rt_{E2}^{(i)}\}_{i\in[M]}$ to the verifier $\mathcal{V}$. During the opening phase, each sub-prover supplies a proof that its Merkle root correctly commits to the corresponding vector $E_1^{(i)}$ and $E_2^{(i)}$. The verifier individually checks each of these proofs, and finally verifies that $e_1 = \sum_{i\in[M]} e_1^{(i)}$ and $e_2 = \sum_{i\in[M]} e_2^{(i)}$.

We show the full protocol in Protocol 2.

Protocol 2: Distributed Merkle Commitments to E_1, E_2.
$\mathcal{P}_0$ claims to $\mathcal{V}$ that $E_1[idx] = e_1, E_2[idx] = e_2$. $\mathcal{P}_i$ holds sub-matrix $M_f^{(i)} \in \mathbb{F}^{b\times\frac{N}{B}}$ and $\mathbf{a}^{(i)} \in \mathbb{F}^b$, where $M_f^{(i)}[k,:] = M_f[ib + k,:], \mathbf{a}^{(i)}[k] = \mathbf{a}[ib + k]$, and $b = \frac{B}{M}$. Let $\beta^{(i)} \in \mathbb{F}^b$ where $\beta^{(i)}[k] = \mathsf{eq}(\mathbf{s}_1, bin(ib + k))$. $MT = (Commit, Eval)$ is Merkle commitment. $Enc : \mathbb{F}^{\frac{N}{B}} \to \mathbb{F}^{\frac{cN}{B}}$ is the map of the encoding algorithm for the linear code in Brakedown [7].

Commit phase:

1. $\mathcal{P}_i$ computes $M_f'^{(i)} \in \mathbb{F}^{b\times\frac{cN}{B}}$ such that $M_f'^{(i)}[k,:] = Enc(M_f^{(i)}[k,:])$, for $k \in [b]$. Then $\mathcal{P}_i$ computes $E_1^{(i)}, E_2^{(i)}$, where $E_1^{(i)} = \sum_{k\in[b]} \mathbf{a}^{(i)}[k]M_f'^{(i)}[k,:]$, and $E_2^{(i)} = \sum_{k\in[b]} \beta^{(i)}[k]M_f'^{(i)}[k,:]$.
2. $\mathcal{P}_i$ sends the hash roots $rt_{E1}^{(i)} \leftarrow MT.Commit(E_1^{(i)})$ and $rt_{E2}^{(i)} \leftarrow MT.Commit(E_2^{(i)})$ to $\mathcal{P}_0$, and $\mathcal{P}_0$ outputs $\{rt_{E1}^{(i)}, rt_{E2}^{(i)}\}_{i\in[M]}$.

Prove phase:

$\mathcal{P}_i$ sends the Merkle proofs $\Pi_{E1}^{(i)} \leftarrow MT.Eval.Prove(E_1^{(i)}, idx, e_1^{(i)})$

and $\Pi_{E2}^{(i)} \leftarrow MT.Eval.Prove(E_2^{(i)}, idx, e_2^{(i)})$ and the corresponding claimed values $e_1^{(i)}$ and $e_2^{(i)}$ to $\mathcal{P}_0$, and $\mathcal{P}_0$ sends $\{\Pi_{E1}^{(i)}, \Pi_{E2}^{(i)}\}_{i\in[M]}$ and $\{e_1^{(i)}, e_2^{(i)}\}_{i\in[M]}$ to $\mathcal{V}$.
Verify phase:
$\mathcal{V}$ checks:

1. $MT.Eval.Verify(rt_{E1}^{(i)}, \Pi_{E1}^{(i)}, e_1^{(i)}, idx) = 1$, for $i \in [M]$.
2. $MT.Eval.Verify(rt_{E2}^{(i)}, \Pi_{E2}^{(i)}, e_2^{(i)}, idx) = 1$, for $i \in [M]$.
3. $e_1 = \sum_{i\in[M]} e_1^{(i)}$ and $e_2 = \sum_{i\in[M]} e_2^{(i)}$.

Polynomial commitment We now describe the procedure for distributed polynomial commitment to vectors E_1, F_1, E_2, F_2, where the prover claims to the verifier that $\tilde{E}_1(x) = v_{E1}$, $\tilde{F}_1(x) = v_{F1}$, $\tilde{E}_2(x) = v_{E2}$, $\tilde{F}_2(x) = v_{F2}$. The process for $\tilde{E}_1$ and $\tilde{E}_2$ is similar to the Merkle commitment for E_1 and E_2. Each sub-prover $\mathcal{P}_i$ constructs a commitment to $\tilde{E}_1^{(i)}, \tilde{E}_2^{(i)}$, denoted $C_{E1}^{(i)}, C_{E2}^{(i)}$, and sends them to the master prover $\mathcal{P}_0$. Then $\mathcal{P}_0$ forwards the set of all commitments $\{C_{E1}^{(i)}, C_{E2}^{(i)}\}_{i\in[M]}$ to the verifier $\mathcal{V}$. During the evaluation phase, each $\mathcal{P}_i$ supplies the proofs $\pi_{E1}^{(i)}$ and $\pi_{E2}^{(i)}$ to $\mathcal{P}_0$, which forwards the set of all proofs $\{\pi_{E1}^{(i)}, \pi_{E2}^{(i)}\}_{i\in[M]}$ to the verifier $\mathcal{V}$. $\mathcal{V}$ individually checks each of these proofs, and finally verifies that $v_{E1} = \sum_{i\in[M]} v_{E1}^{(i)}$, $v_{E2} = \sum_{i\in[M]} v_{E2}^{(i)}$. The detailed constructions for committing to the polynomials $\tilde{E}_1, \tilde{E}_2$ are presented in Protocol 3. The same approach applies to $\tilde{F}_1, \tilde{F}_2$, with the corresponding details provided in Protocol 4.

Protocol 3: Distributed Polynomial Commitment to $\tilde{E}_1$ and $\tilde{E}_2$. $\mathcal{P}_0$ claims to $\mathcal{V}$ that $\tilde{E}_1(x) = v_{E1}$, $\tilde{E}_2(x) = v_{E2}$. $\mathcal{P}_i$ holds sub-matrix $M_f^{(i)} \in \mathbb{F}^{b\times\frac{N}{B}}$ and $\mathbf{a}^{(i)} \in \mathbb{F}^b$, where $M_f^{(i)}[k,:] = M_f[ib+k,:], \mathbf{a}^{(i)}[k] = \mathbf{a}[ib+k]$, and $b = \frac{B}{M}$. Let $\beta^{(i)} \in \mathbb{F}^b$ where $\beta^{(i)}[k] = \mathsf{eq}(\mathbf{s}_1, bin(ib+k))$. $PC = (Gen, Commit, Open, Eval)$ is the underlying polynomial commitment. $Enc : \mathbb{F}^{\frac{N}{B}} \to \mathbb{F}^{\frac{cN}{B}}$ is the map of the encoding algorithm for the linear code in Brakedown [7].
Commit phase:

1. $\mathcal{P}_i$ computes $M_f'^{(i)} \in \mathbb{F}^{b\times\frac{cN}{B}}$ such that $M_f'^{(i)}[k,:] = Enc(M_f^{(i)}[k,:])$, for $k \in [b]$. Then $\mathcal{P}_i$ computes $E_1^{(i)}, E_2^{(i)}$, where $E_1^{(i)} = \sum_{k\in[b]} \mathbf{a}^{(i)}[k] M_f'^{(i)}[k,:]$, and $E_2^{(i)} = \sum_{k\in[b]} \beta^{(i)}[k] M_f'^{(i)}[k,:]$.
2. $\mathcal{P}_i$ generates the commitments $C_{E1}^{(i)} \leftarrow PC.Commit(\tilde{E}_1^{(i)})$ and $C_{E2}^{(i)} \leftarrow PC.Commit(\tilde{E}_2^{(i)})$. Then $\mathcal{P}_i$ sends $C_{E1}^{(i)}, C_{E2}^{(i)}$ to $\mathcal{P}_0$.
3. $\mathcal{P}_0$ outputs $\{C_{E1}^{(i)}, C_{E2}^{(i)}\}_{i\in[M]}$.

Eval phase:

1. $\mathcal{P}_i$ sends the proofs $\pi_{E1}^{(i)} \leftarrow PC.Eval.Prove(\tilde{E}_1^{(i)}, x, v_{E1}^{(i)})$, $\pi_{E2}^{(i)} \leftarrow PC.Eval.Prove(\tilde{E}_2^{(i)}, x, v_{E2}^{(i)})$ to $\mathcal{P}_0$, and $\mathcal{P}_0$ sends $\{\pi_{E1}^{(i)}, \pi_{E2}^{(i)}\}_{i\in[M]}$ and the claimed values $\{v_{E1}^{(i)}, v_{E2}^{(i)}\}_{i\in[M]}$ to $\mathcal{V}$.
2. $\mathcal{V}$ checks:
   (a) $PC.Eval.Verify(C_{E1}^{(i)}, \pi_{E1}^{(i)}, v_{E1}^{(i)}, x) = 1$, for $i \in [M]$.
   (b) $PC.Eval.Verify(C_{E2}^{(i)}, \pi_{E2}^{(i)}, v_{E2}^{(i)}, x) = 1$, for $i \in [M]$.
   (c) $v_{E1} = \sum_{i\in[M]} v_{E1}^{(i)}$, $v_{E2} = \sum_{i\in[M]} v_{E2}^{(i)}$.

Protocol 4: Distributed Polynomial Commitment to $\tilde{F}_1$ and $\tilde{F}_2$. $\mathcal{P}_0$ claims to $\mathcal{V}$ that $\tilde{F}_1(x) = v_{F1}$, $\tilde{F}_2(x) = v_{F2}$. $\mathcal{P}_i$ holds sub-matrix $M_f^{(i)} \in \mathbb{F}^{b\times\frac{N}{B}}$ and $\mathbf{a}^{(i)} \in \mathbb{F}^b$, where $M_f^{(i)}[k,:] = M_f[ib+k,:], \mathbf{a}^{(i)}[k] = \mathbf{a}[ib+k]$, and $b = \frac{B}{M}$. Let $\beta^{(i)} \in \mathbb{F}^b$ where $\beta^{(i)}[k] = \mathsf{eq}(\mathbf{s}_1, bin(ib+k))$. $PC = (Gen, Commit, Open, Eval)$ is the underlying polynomial commitment. $Enc : \mathbb{F}^{\frac{N}{B}} \to \mathbb{F}^{\frac{cN}{B}}$ is the map of the encoding algorithm for the linear code in Brakedown [7].

Commit phase:

1. $\mathcal{P}_i$ computes $F_1^{(i)}, F_2^{(i)}$, where $F_1^{(i)} = \sum_{k\in[b]} \mathbf{a}^{(i)}[k] M_f^{(i)}[k,:]$, and $F_2^{(i)} = \sum_{k\in[b]} \beta^{(i)}[k] M_f^{(i)}[k,:]$.
2. $\mathcal{P}_i$ generates the commitments $C_{F1}^{(i)} \leftarrow PC.Commit(\tilde{F}_1^{(i)})$ and $C_{F2}^{(i)} \leftarrow PC.Commit(\tilde{F}_2^{(i)})$. Then $\mathcal{P}_i$ sends $C_{F1}^{(i)}, C_{F2}^{(i)}$ to $\mathcal{P}_0$.
3. $\mathcal{P}_0$ outputs $\{C_{F1}^{(i)}, C_{F2}^{(i)}\}_{i\in[M]}$.

Eval phase:

1. $\mathcal{P}_i$ sends the proofs $\pi_{F1}^{(i)} \leftarrow PC.Eval.Prove(\tilde{F}_1^{(i)}, x, v_{F1}^{(i)})$, $\pi_{F2}^{(i)} \leftarrow PC.Eval.Prove(\tilde{F}_2^{(i)}, x, v_{F2}^{(i)})$ to $\mathcal{P}_0$, and $\mathcal{P}_0$ sends $\{\pi_{F1}^{(i)}, \pi_{F2}^{(i)}\}_{i\in[M]}$ and the claimed values $\{v_{F1}^{(i)}, v_{F2}^{(i)}\}_{i\in[M]}$ to $\mathcal{V}$.
2. For $i \in [M]$, $\mathcal{V}$ checks:
   (a) $PC.Eval.Verify(C_{F1}^{(i)}, \pi_{F1}^{(i)}, v_{F1}^{(i)}, x) = 1$.
   (b) $PC.Eval.Verify(C_{F2}^{(i)}, \pi_{F2}^{(i)}, v_{F2}^{(i)}, x) = 1$.
   (c) $v_{F1} = \sum_{i\in[M]} v_{F1}^{(i)}$, $v_{F2} = \sum_{i\in[M]} v_{F2}^{(i)}$.

3.3 Distributed Proof of Relation $\mathcal{R}$

Since the proof for relation $\mathcal{R}$ comprises the proofs of encoding algorithms $E_1 = Enc(F_1), E_2 = Enc(F_2)$, we consider the claim to the verifier that $E = Enc(F)$ for the vectors $E \in \mathbb{F}^{\frac{cN}{B}}$ and $F \in \mathbb{F}^{\frac{N}{B}}$, while both parties hold the polynomial commitment of $\tilde{E}$ and $\tilde{F}$, denoted C_E and C_F. To prove that $E = Enc(F)$, it

suffices to show that: (1) E is a valid codeword, specifically, there exists a parity check matrix $H \in \mathbb{F}^{\frac{cN}{B} \times \frac{(c-1)N}{B}}$ for the coding, where c is the reciprocal of the rate of the code, such that $E \cdot H = \mathbf{0}$; (2) The first $\frac{N}{B}$ entries of E are exactly equal to the vector F (the encoding is systematic). As shown in [15], the matrix H is sparse and thus has $O(\frac{N}{B})$ non-zero entries. Each sub-prover holds H.

Statement (1) can be reduced to a sumcheck relation $\sum_{x \in \{0,1\}^{\log \frac{cN}{B}}} \tilde{E}(x) \cdot \tilde{H}_{\mathbf{u}}(x) = 0$. Let $Q(x) = \tilde{E}(x) \cdot \tilde{H}_{\mathbf{u}}(x)$. As in the sumcheck procedure, the prover is required to send $Q_k(X_k) = \sum_{x \in \{0,1\}^{\log \frac{cN}{B} - k}} Q(r_1, \ldots, r_{k-1}, X_k, x)$ to the verifier in the k-th round, where $1 \leq k \leq \log \frac{cN}{B}$ and r_k is the randomness sent by the verifier in the k-th round. In our distributed proof, each sub-prover $\mathcal{P}_i$ holds $E^{(i)}, F^{(i)}$ where $\tilde{E} = \sum_{i \in [M]} \tilde{E}^{(i)}, \tilde{F} = \sum_{i \in [M]} \tilde{F}^{(i)}$, and the parity check matrix $H \in \mathbb{F}^{\frac{cN}{B} \times \frac{(c-1)N}{B}}$ for the linear code in Brakedown [7] for the message of length $\frac{N}{B}$. $\mathcal{P}_i$ computes the values of $\tilde{E}_k^{(i)}(X_k, x) = \tilde{E}^{(i)}(r_1, \ldots, r_{k-1}, X_k, x)$ and $\tilde{H}_{\mathbf{u},k}(X_k, x) = \tilde{H}_{\mathbf{u}}(r_1, \ldots, r_{k-1}, X_k, x)$ for all $x \in \{0,1\}^{\log \frac{cN}{B} - k}$ in the k-th round, and then computes $Q_k^{(i)}(X_k) \leftarrow \sum_{x \in \{0,1\}^{\log \frac{cN}{B} - k}} \tilde{E}_k^{(i)}(X_k, x) \cdot \tilde{H}_{\mathbf{u},k}(X_k, x)$ and $\mathcal{P}_0$ gets $Q_k(X_k) = \sum_{i \in [M]} Q_k^{(i)}(X_k)$, as

$$\begin{aligned}
\sum_{i \in [M]} Q_k^{(i)}(X_k) &= \sum_{i \in [M]} \sum_{x \in \{0,1\}^{\log \frac{cN}{B} - k}} \tilde{E}_k^{(i)}(X_k, x) \cdot \tilde{H}_{\mathbf{u},k}(X_k, x) \\
&= \sum_{x \in \{0,1\}^{\log \frac{cN}{B} - k}} \Big(\sum_{i \in [M]} \tilde{E}_k^{(i)}(X_k, x) \Big) \cdot \tilde{H}_{\mathbf{u},k}(X_k, x) \\
&= \sum_{x \in \{0,1\}^{\log \frac{cN}{B} - k}} \tilde{E}_k(X_k, x) \cdot \tilde{H}_{\mathbf{u},k}(X_k, x) = Q_k(X_k)
\end{aligned}$$

Statement (2) is equivalent to $\tilde{E}(0^{\log c}, x) = \tilde{F}(x)$, for any $x \in \{0,1\}^{\log \frac{N}{B}}$. The verifier samples a random $\mathbf{u}' \in \mathbb{F}^{\log \frac{N}{B}}$, and verifies that $\tilde{E}(0^{\log c}, \mathbf{u}') = \tilde{F}(\mathbf{u}')$. The soundness is guaranteed by the Schwartz-Zippel lemma.

At the end of the protocol, the verifier queries the evaluations of $\tilde{E}, \tilde{F}$ and $\tilde{H}_{\mathbf{u}}$. The queries of $\tilde{E}, \tilde{F}$ are instantiated with the distributed PCS construction (Protocol 3 and Protocol 4). The query of $\tilde{H}_{\mathbf{u}}$ is instantiated with the BaseFold protocol. We present the full construction of the distributed proof of the encoding algorithm in Protocol 5.

Protocol 5: Distributed Proof of Encoding Algorithm

$\mathcal{P}_0$ claims to $\mathcal{V}$ that $E = Enc(F)$, where $E \in \mathbb{F}^{\frac{cN}{B}}, F \in \mathbb{F}^{\frac{N}{B}}$ and $Enc : \mathbb{F}^{\frac{N}{B}} \to \mathbb{F}^{\frac{cN}{B}}$ is the map of the encoding algorithm for the linear code in Brakedown [7]. $\mathcal{P}_i$ holds $\tilde{E}^{(i)}(x), \tilde{F}^{(i)}(x)$ and the parity check matrix $H \in \mathbb{F}^{\frac{cN}{B} \times \frac{(c-1)N}{B}}$ for the linear code for the message of length $\frac{N}{B}$, where $\tilde{E} = \sum_{i \in [M]} \tilde{E}^{(i)}, \tilde{F} = \sum_{i \in [M]} \tilde{F}^{(i)}$. Set $q_0 = 0$.

1. $\mathcal{V}$ samples a random vector $\mathbf{u} \in \mathbb{F}^{\log \frac{(c-1)N}{B}}$ and sends $\mathbf{u}$ to $\mathcal{P}_0$, and $\mathcal{P}_0$ transmits $\mathbf{u}$ to $\mathcal{P}_i$.
2. Each $\mathcal{P}_i$ computes $\tilde{H}_{\mathbf{u}}(b) = \sum_{t \in \{0,1\}^{\log \frac{(c-1)N}{B}}} \tilde{H}(b,t) \cdot \mathsf{eq}(t, \mathbf{u})$ for $b \in \{0,1\}^{\log \frac{cN}{B}}$, where $\tilde{H}_{\mathbf{u}}(b) = \tilde{H}(b, \mathbf{u})$.
3. $\mathcal{P}_0, \ldots, \mathcal{P}_{M-1}$ and $\mathcal{V}$ perform the sumcheck protocol for $\sum_{b \in \{0,1\}^{\log \frac{cN}{B}}} \tilde{E}(b) \cdot \tilde{H}_{\mathbf{u}}(b) = 0$. Specifically, in the k-th round, where $1 \le k \le \log \frac{cN}{B}$:
   (a) Each $\mathcal{P}_i$ locally computes the polynomials $\tilde{E}_k^{(i)}(X_k, x)$ and $\tilde{H}_{\mathbf{u},k}(X_k, x)$ for $x \in \{0,1\}^{\log \frac{cN}{B} - k}$, where:
   $\tilde{E}_k^{(i)}(X_k, x) := \tilde{E}^{(i)}(r_1, \ldots, r_{k-1}, X_k, x)$,
   $\tilde{H}_{\mathbf{u},k}(X_k, x) := \tilde{H}_{\mathbf{u}}(r_1, \ldots, r_{k-1}, X_k, x)$.
   (b) $\mathcal{P}_i$ sends $Q_k^{(i)}(X_k) \leftarrow \sum_{x \in \{0,1\}^{\log \frac{cN}{B} - k}} \tilde{E}_k^{(i)}(X_k, x) \cdot \tilde{H}_{\mathbf{u},k}(X_k, x)$ to $\mathcal{P}_0$, and then $\mathcal{P}_0$ sends $Q_k(X_k) \leftarrow \sum_{i \in [M]} Q_k^{(i)}(X_k)$ to $\mathcal{V}$.
   (c) $\mathcal{V}$ checks if $q_{k-1} = Q_k(0) + Q_k(1)$. If the check passes, $\mathcal{V}$ sends a random $r_k \in \mathbb{F}$ to $\mathcal{P}_0$, and sets $q_k \leftarrow Q_k(r_k)$. $\mathcal{P}_0$ sends r_k to $\mathcal{P}_i$.
4. $\mathcal{P}_0, \ldots, \mathcal{P}_{M-1}$ and $\mathcal{V}$ perform the evaluation proof protocol for $\tilde{H}_{\mathbf{u}}(\mathbf{r}) = q_1'$ and $\tilde{E}(\mathbf{r}) = q_2'$, where $\mathbf{r} = (r_1, \ldots, r_{\log \frac{cN}{B}})$, and check $q_{\log \frac{cN}{B}} = q_1' \cdot q_2'$. For the relation $\tilde{H}_{\mathbf{u}}(\mathbf{r}) = q_1'$, $\mathcal{P}_0, \ldots, \mathcal{P}_{M-1}$ and $\mathcal{V}$ perform the distributed BaseFold protocol in [24], where the evaluation table of $\tilde{H}_{\mathbf{u}}$ is evenly partitioned into M blocks, and $\mathcal{P}_i$ only handles the i-th block. For the relation $\tilde{E}(\mathbf{r}) = q_2'$, $\mathcal{P}_0, \ldots, \mathcal{P}_{M-1}$ and $\mathcal{V}$ perform the evaluation phase of the distributed polynomial commitment to $\tilde{E}$ (Protocol 3).
5. $\mathcal{V}$ samples a random vector $\mathbf{u}' \in \mathbb{F}^{\log \frac{N}{B}}$. $\mathcal{P}_0, \ldots, \mathcal{P}_{M-1}$ and $\mathcal{V}$ perform the evaluation phase of the distributed polynomial commitment to $\tilde{E}$ and $\tilde{F}$ (Protocol 3 and Protocol 4), and verify that $\tilde{E}(0^{\log c}, \mathbf{u}') = \tilde{F}(\mathbf{u}')$.

3.4 Our Distributed Code-based PCS

We now integrate the above components to construct our distributed code-based PCS, where the matrix M_f is partitioned row-wise and evenly distributed among the sub-provers, and each sub-prover $\mathcal{P}_i$ holds sub-matrix $M_f^{(i)}$. The protocol incorporates the distributed polynomial commitments to vectors E_1, F_1, E_2, F_2, Merkle hash commitments to vectors E_1, E_2 (Subsection 3.2), and the distributed proof for relation $\mathcal{R}$ (Subsection 3.3). In the commitment phase, $\mathcal{P}_i$ hashes the column vectors of the encoded matrix $M_f'^{(i)}$ and constructs a Merkle tree root $C^{(i)}$, and the commitment is the set $\{C^{(i)}\}_{i \in [M]}$. To open the commitment, each $\mathcal{P}_i$ provides a proof that its root correctly commits to the corresponding encoded sub-matrix. The verifier checks all proofs individually, and the overall verification succeeds only if every proof is valid. Moreover, we note that F_2 is

opened once at point $\mathbf{s}_2$. The opening of $\widetilde{F}_2$ follows Protocol 4 and is therefore omitted here. The full construction is presented in Protocol 6.

Protocol 6: Our Distributed Code-based PCS

$\mathcal{P}_0$ claims $f(\mathbf{s}) = v$ to $\mathcal{V}$, where $f \in \mathcal{F}^{(\leq 1)}_{\log N}$, and $\mathbf{s} = \mathbf{s}_1||\mathbf{s}_2$, $\mathbf{s}_1 \in \mathbb{F}^{\log B}$, $\mathbf{s}_2 \in \mathbb{F}^{\log \frac{N}{B}}$. Let $Enc : \mathbb{F}^{\frac{N}{B}} \to \mathbb{F}^{\frac{cN}{B}}$ be the map of the encoding algorithm for the linear code in Brakedown [7]. $\mathcal{P}_i$ holds sub-matrix $M_f^{(i)} \in \mathbb{F}^{b \times \frac{N}{B}}$, where $M_f^{(i)}[k,:] = M_f[ib+k,:]$. Let $\beta^{(i)} \in \mathbb{F}^b$ where $\beta^{(i)}[k] = \mathsf{eq}(\mathbf{s}_1, bin(ib+k))$. $PC = (Gen, Commit, Open, Eval)$ is the underlying polynomial commitment. $MT = (Commit, Eval)$ is Merkle commitment. Let $H(\cdot) : \mathbb{F}^* \to \mathbb{F}$ be a random oracle.

Commit phase:

$\mathcal{P}_i$ computes $M_f'^{(i)} \in \mathbb{F}^{b \times \frac{cN}{B}}$ such that $M_f'^{(i)}[k,:] = Enc(M_f^{(i)}[k,:])$, for $k \in [b]$. Let $\mathbf{h}^{(i)} \in \mathbb{F}^{\frac{cN}{B}}$ be a vector such that $\mathbf{h}^{(i)}[k] \leftarrow H(M_f'^{(i)}[:,k])$, for each $k \in [\frac{cN}{B}]$. $\mathcal{P}_i$ sends $C^{(i)} \leftarrow MT.Commit(\mathbf{h}^{(i)})$ to $\mathcal{P}_0$. $\mathcal{P}_0$ outputs $\{C^{(i)}\}_{i\in[M]}$.

Eval phase:

1. $\mathcal{V}$ samples a random vector $\mathbf{a} \in \mathbb{F}^B$, and $\mathcal{P}_0$ sends the vector $\mathbf{a}^{(i)}$ to $\mathcal{P}_i$, where $\mathbf{a}^{(i)} = \mathbf{a}[ib : (i+1)b - 1]$.
2. $\mathcal{P}_i$ performs the commit phase of the distributed polynomial commitment to $\tilde{E}_1^{(i)}$, $\tilde{F}_1^{(i)}$, $\tilde{E}_2^{(i)}$ and $\tilde{F}_2^{(i)}$ using $PC.Commit$, where $E_1^{(i)} = Enc(F_1^{(i)})$, $F_1^{(i)} = \sum_{k\in[b]} \mathbf{a}^{(i)}[k] M_f^{(i)}[k,:]$, $E_2^{(i)} = Enc(F_2^{(i)})$, $F_2^{(i)} = \sum_{k\in[b]} \beta^{(i)}[k] M_f^{(i)}[k,:]$ for $i \in [M]$. Then $\mathcal{P}_i$ sends the corresponding commitments $C_{E1}^{(i)}, C_{F1}^{(i)}, C_{E2}^{(i)}, C_{F2}^{(i)}$ to $\mathcal{P}_0$. $\mathcal{P}_0$ sends $\{C_{E1}^{(i)}, C_{F1}^{(i)}, C_{E2}^{(i)}, C_{F2}^{(i)}\}_{i\in[M]}$ to $\mathcal{V}$.
3. $\mathcal{P}_i$ sends the Merkle-tree roots $rt_1^{(i)} \leftarrow MT.Commit(E_1^{(i)})$ and $rt_2^{(i)} \leftarrow MT.Commit(E_2^{(i)})$ to $\mathcal{P}_0$. $\mathcal{P}_0$ sends $\{rt_1^{(i)}, rt_2^{(i)}\}_{i\in[M]}$ to $\mathcal{V}$.
4. $\mathcal{V}$ sends the column indices $\mathcal{I} = \{idx_1, \ldots, idx_\ell\}$ to $\mathcal{P}_0$. $\mathcal{P}_0$ transmits $\mathcal{I}$ to $\mathcal{P}_i$.
5. $\mathcal{P}_i$ chooses the columns $R^{(i)} \in \mathbb{F}^{\ell \times b}$ where $R^{(i)}[j,:] = M_f'^{(i)}[:, idx_j]$ for all $idx_j \in \mathcal{I}$, and generates the proof $\pi_{i,j} \leftarrow MT.Eval.Prove(\mathbf{h}^{(i)}, idx_j, \hat{\mathbf{h}}^{(i)}[j])$ for $j \in [\ell]$, where $\hat{\mathbf{h}}^{(i)} \in \mathbb{F}^\ell$ and $\hat{\mathbf{h}}^{(i)}[j] = H(R^{(i)}[j,:])$ for $i \in [M], j \in [\ell]$. $\mathcal{P}_i$ sends $\{\pi_{i,j}\}_{j\in[\ell]}$, $R^{(i)}$ and $\hat{\mathbf{h}}^{(i)}$ to $\mathcal{P}_0$, and $\mathcal{P}_0$ sends $\{\pi_{i,j}\}_{i\in[M],j\in[\ell]}$, $\{R^{(i)}\}_{i\in[M]}$ and $\{\hat{\mathbf{h}}^{(i)}\}_{i\in[M]}$ to $\mathcal{V}$.
6. $\mathcal{P}_i$ computes $\mathbf{y}_1^{(i)}, \mathbf{y}_2^{(i)} \in \mathbb{F}^\ell$, such that $\mathbf{y}_1^{(i)}[k] = \sum_{j\in[b]} R^{(i)}[k,j] \cdot \mathbf{a}^{(i)}[j], \mathbf{y}_2^{(i)}[k] = \sum_{j\in[b]} R^{(i)}[k,j] \cdot \beta^{(i)}[j]$, for $k \in [\ell]$. $\mathcal{P}_0$ computes $\mathbf{y}_1 \leftarrow \sum_{i\in[M]} \mathbf{y}_1^{(i)}, \mathbf{y}_2 \leftarrow \sum_{i\in[M]} \mathbf{y}_2^{(i)}$ and sends $\mathbf{y}_1, \mathbf{y}_2$ to $\mathcal{V}$.

7. $\mathcal{P}_i$ sends $\pi_{F2}^{(i)}$ and the claimed s $v_{F2}^{(i)}$ to $\mathcal{P}_0$, where $\pi_{F2}^{(i)} \leftarrow PC.Eval.Prove(\tilde{F}_2^{(i)}, \mathbf{s}_2, v_{F2}^{(i)})$. $\mathcal{P}_0$ sends $\{v_{F2}^{(i)}, \pi_{F2}^{(i)}\}_{i\in[M]}$ to $\mathcal{V}$.
8. $\mathcal{P}_i$ generates the proofs $\{\Pi_{1,j}^{(i)}, \Pi_{2,j}^{(i)}\}_{j\in[\ell]}$ and the claimed values $\{e_{1,j}^{(i)}, e_{2,j}^{(i)}\}_{j\in[\ell]}$ and sends them to $\mathcal{P}_0$, where $\Pi_{1,j}^{(i)} = MT.Eval.Prove(E_1^{(i)}, idx_j, e_{1,j}^{(i)})$, $\Pi_{2,j}^{(i)} = MT.Eval.Prove(E_2^{(i)}, idx_j, e_{2,j}^{(i)})$, for $i \in [M], j \in [\ell]$. $\mathcal{P}_0$ sends $\{\Pi_{1,j}^{(i)}, \Pi_{2,j}^{(i)}\}_{i\in[M],j\in[\ell]}$, and $\{e_{1,j}^{(i)}, e_{2,j}^{(i)}\}_{i\in[M],j\in[\ell]}$ to $\mathcal{V}$.
9. $\mathcal{P}_0, \ldots, \mathcal{P}_{M-1}$ and $\mathcal{V}$ perform Protocol 5 twice, once for the relation $E_1 = Enc(F_1)$, and once for $E_2 = Enc(F_2)$, and let π_r denote the pair of resulting proofs for the relation $\mathcal{R}$. $\mathcal{P}_0$ sends π_r to $\mathcal{V}$.
10. $\mathcal{V}$ checks:
    (a) $\hat{\mathbf{h}}^{(i)}[j] = H(R^{(i)}[j,:])$ and $MT.Eval.Verify(C^{(i)}, \pi_{i,j}, \hat{\mathbf{h}}^{(i)}[j], idx_j) = 1$, for $i \in [M], j \in [\ell]$.
    (b) $\mathbf{y}_1[k] = \sum_{i\in[M]} \sum_{j\in[b]} R^{(i)}[k,j]\mathbf{a}^{(i)}[j]$, and $\mathbf{y}_2[k] = \sum_{i\in[M]} \sum_{j\in[b]} R^{(i)}[k,j]\beta^{(i)}[j]$, for $k \in [\ell]$.
    (c) $PC.Eval.Verify(C_{F2}^{(i)}, \pi_{F_2}^{(i)}, v_{F_2}^{(i)}, \mathbf{s}_2) = 1$, for $i \in [M]$, and $\sum_{i\in[M]} v_{F2}^{(i)} = v$.
    (d) $\mathbf{y}_1[j] = \sum_{i\in[M]} e_{1,j}^{(i)}, \mathbf{y}_2[j] = \sum_{i\in[M]} e_{2,j}^{(i)}$, and $MT.Eval.Verify(rt_1^{(i)}, \Pi_{1,j}^{(i)}, e_{1,j}^{(i)}, idx_j) = 1$, $MT.Eval.Verify(rt_2^{(i)}, \Pi_{2,j}^{(i)}, e_{2,j}^{(i)}, idx_j) = 1$, for $i \in [M], j \in [\ell]$.
    (e) the relation $\mathcal{R}$ is valid with the proof π_r.

3.5 Security and Complexity Analysis

We discuss the security and complexity of Protocol 6.

Security We conduct our security analysis under the same trust assumptions as the single-prover setting, given that the sub-provers in our system are trusted entities and the distribution is intended solely for acceleration. We first analyze the security of two core components: polynomial commitment to $\tilde{E}_1, \tilde{F}_1, \tilde{E}_2, \tilde{F}_2$, and proof for relation $\mathcal{R}$. We then demonstrate the overall security of the distributed code-based PCS.

Security of Polynomial Commitment to $\tilde{E}_1, \tilde{F}_1, \tilde{E}_2, \tilde{F}_2$. We analyze completeness and knowledge soundness of distributed polynomial commitment to $\tilde{E}_1, \tilde{E}_2$ (Protocol 3) and $\tilde{F}_1, \tilde{F}_2$ (Protocol 4). First, we prove the following lemma:

Lemma 1. *Let $m \in \mathbb{N}$ be polynomially bounded in λ. Let $a_0, \ldots, a_{m-1} \in \mathbb{F}$ be fixed public coefficients. Given commitments under a PCS PC to polynomials $f_0, \ldots, f_{m-1}$, we can construct a polynomial commitment scheme PC′, where*

the prover claims $f(\boldsymbol{x}) = v$ *to the verifier, and* $f = \sum_{i\in[m]} a_i f_i$.
Gen phase*: Generate the public parameters for the polynomial commitment to* $f_0, \ldots, f_{m-1}$.
Commit phase*: The prover computes the commitments* $Com_i \leftarrow PC.Commit(\mathsf{pp}, f_i)$ *for* $i \in [m]$. *The output is* $Com := (Com_0, \ldots, Com_{m-1})$.
Open phase*: To open* Com *to a claimed polynomial* f', *the prover provides* $f'_0, \ldots, f'_{m-1}$ *to the verifier. The verifier checks* $Open(pp, Com_i, f'_i) = 1$ *for every* $i \in [m]$, *and verifies that* $f' = \sum_{i\in[m]} a_i f'_i$. *If all checks pass, output 1, else output 0.*
Eval phase*: The prover sends to the verifier the proofs for* $f_i(\boldsymbol{x}) = v_i$, *for all* $i \in [m]$. *Then the verifier checks that all proofs are valid and* $v = \sum_{i\in[m]} a_i v_i$.

Proof. We demonstrate that if the underlying commitments to the polynomials $f_0, \ldots, f_{m-1}$ satisfy completeness, binding, and knowledge soundness, then the resulting commitment to the linear combination $\sum_{i\in[m]} a_i f_i$ also satisfies these properties.

Completeness follows directly from the completeness of the polynomial commitments to $f_0, \ldots, f_{m-1}$, and if $f_i(\mathbf{x}) = v_i$, for $i \in [m]$, then $v = \sum_{i\in[m]} a_i f_i$ $(\mathbf{x}) = \sum_{i\in[m]} a_i v_i$.

To prove the binding property, we open the commitment of f_i twice and get the polynomials $f_{i,1}$ and $f_{i,2}$, for $i \in [m]$. By the binding property of the underlying commitments to $f_0, \ldots, f_{m-1}$, the probability of $f_{i,1} \neq f_{i,2}$ for *any* specified $i \in [m]$ is $\mathsf{negl}(\lambda)$. By the union bound, the probability that $f_{i,1} \neq f_{i,2}$ for *some* $i \in [m]$ is less than $m \cdot \mathsf{negl}(\lambda)$. That is, the probability that $f_{i,1} = f_{i,2}$ for *all* $i \in [m]$ is at least $1 - m \cdot \mathsf{negl}(\lambda)$. Since m is polynomially bounded in λ, the deviation $m \cdot \mathsf{negl}(\lambda)$ remains negligible. Hence, the probability that the two linear combinations differ, i.e.,$\sum_{i\in[m]} a_i f_{i,1} \neq \sum_{i\in[m]} a_i f_{i,2}$, is also negligible in λ, which confirms that the binding property is satisfied.

To prove the knowledge soundness property, we construct an extractor that invokes the extractors of the underlying commitments to extract $f^*_0, \ldots, f^*_{m-1}$ from the underlying extractors and outputs $f^* = \sum_{i\in[m]} a_i f^*_i$. By the knowledge soundness property of the underlying commitments to $f_0, \ldots, f_{m-1}$, the probability of $f^*_i(\mathbf{x}) \neq v_i$ for *any* specified $i \in [m]$ is $\mathsf{negl}(\lambda)$. By the union bound, the probability that $f^*_i(\mathbf{x}) \neq v_i$ for *some* $i \in [m]$ is less than $m \cdot \mathsf{negl}(\lambda)$. That is, the probability that $f^*_i(\mathbf{x}) = v_i$ for *all* $i \in [m]$ is at least $1 - m \cdot \mathsf{negl}(\lambda)$. Notice that $f^* = \sum_{i\in[m]} a_i f^*_i$ and $v = \sum_{i\in[m]} a_i v_i$. Since m is polynomially bounded in λ, the deviation $m \cdot \mathsf{negl}(\lambda)$ remains negligible. Hence, the probability of $f^*(\mathbf{x}) \neq v$ is negligible in λ, which confirms that the knowledge soundness property is satisfied. □

We can achieve linearly combined polynomial commitment to $\tilde{E}_1, \tilde{E}_2$ from the polynomial commitment to $\{\tilde{E}^{(i)}_1\}_{i\in[M]}, \{\tilde{E}^{(i)}_2\}_{i\in[M]}$ respectively. The same procedure applies to $\tilde{F}_1, \tilde{F}_2$. By Lemma 1, we prove that the polynomial commitment to $\tilde{E}_1, \tilde{E}_2, \tilde{F}_1, \tilde{F}_2$ satisfies completeness, binding, and knowledge soundness, provided that the underlying PC already satisfies these properties.

Security of proof for relation $\mathcal{R}$ The proof for relation $\mathcal{R}$ includes the proofs of the two encodings $E_1 = Enc(F_1)$ and $E_2 = Enc(F_2)$. Therefore, it suffices to analyze the security of the proof of the encoding algorithm as specified in Protocol 5.

The completeness of the protocol follows directly from the completeness of the underlying sumcheck protocol, the underlying PCS, and the commitments to $\tilde{E}$ and $\tilde{F}$. The soundness property is established as follows.

Let $\mathcal{A}_{Enc}$ be a PPT adversary that produces an accepting transcript when interacting with the verifier over the protocol proving the relation $\mathcal{R}$. We construct an extractor $\mathcal{E}_{Enc}$ that outputs witnesses E^* and F^* satisfying $E^* = Enc(F^*)$, except with negligible failure probability. We construct our extractor $\mathcal{E}_{Enc}$ as follows: (1) $\mathcal{E}_{Enc}$ uses $\mathcal{A}_{Enc}$ to construct $\mathcal{A}_E$, where $\mathcal{A}_E$ is the PPT adversary that produces an accepting transcript when interacting with the verifier over the commitment to $\tilde{E}$. (2) $\mathcal{E}_{Enc}$ invokes the corresponding extractor $\mathcal{E}_E$ to obtain $\tilde{E}^*$ that is consistent with the commitment to E. (3) $\mathcal{E}_{Enc}$ uses $\mathcal{A}_{Enc}$ to construct $\mathcal{A}'_E$, $\mathcal{A}'_F$, where $\mathcal{A}'_E$, $\mathcal{A}'_F$ are the PPT adversaries that produce accepting transcripts when interacting with the verifier over the commitment to $\tilde{E}$ and the commitment to $\tilde{F}$ in step 5, respectively. (4) $\mathcal{E}_{Enc}$ invokes the corresponding extractor $\mathcal{E}'_E, \mathcal{E}'_F$ to obtain $\tilde{E}'^*, \tilde{F}^*$ consistent with the corresponding commitments. (5) If $\tilde{E}^* \neq \tilde{E}'^*$, abort; else output $\tilde{E}^*, \tilde{F}^*$.

By the knowledge soundness of the distributed BaseFold evaluation proof and the distributed polynomial commitment scheme to $\tilde{E}$, except with negligible probability, $\tilde{H}_{\mathbf{u}}(\mathbf{r}) = q'_1, \tilde{E}^*(\mathbf{r}) = q'_2$. In the final round of the sumcheck protocol, the verifier checks $\tilde{H}_{\mathbf{u}}(\mathbf{r}) = q'_1$ and $\tilde{E}^*(\mathbf{r}) = q'_2$. Hence, the terminal sumcheck claim is consistent with the polynomial $\tilde{H}_{\mathbf{u}}(x) \cdot \tilde{E}^*(x)$. By the soundness of the sumcheck protocol, an accepting transcript therefore implies $\sum_{b \in \{0,1\}^{\log \frac{cN}{B}}} \tilde{E}^*(b) \cdot \tilde{H}_{\mathbf{u}}(b) = 0$ except with negligible probability. Recall that $\tilde{H}_{\mathbf{u}}(b) = \tilde{H}(b, \mathbf{u})$. Define $G(\mathbf{u}) := \sum_{b \in \{0,1\}^{\log \frac{cN}{B}}} \tilde{E}^*(b) \cdot \tilde{H}(b, \mathbf{u})$. If $E^*H \neq 0$, then G is a nonzero polynomial. Therefore, by the Schwartz–Zippel lemma, $\Pr_u[G(u) = 0 \mid E^*H \neq 0] \leq \mathrm{negl}(\lambda)$. Consequently, an accepting transcript implies $E^*H = 0$ except with negligible probability; namely, E^* is a valid codeword.

By the binding property of the polynomial commitment to $\tilde{E}$, $\tilde{E}'^* = \tilde{E}^*$ except with negligible probability. The verifier further checks at a uniformly random $\mathbf{u}' \in \mathbb{F}^{\log \frac{N}{B}}$ that $\widetilde{E}^*(0^{\log c}, u') = \widetilde{F}^*(u')$. By the Schwartz–Zippel lemma, except with negligible probability, this implies $\widetilde{E}^*(0^{\log c}, x) = \widetilde{F}^*(x)$. Thus, F^* is exactly the systematic part of E^*. Combining codeword validity and systematic consistency gives $E^* = Enc(F^*)$. Since all invoked extractors run in polynomial time, so does $\mathcal{E}_{Enc}$. Hence, Protocol 5 satisfies knowledge soundness.

Security of our PCS We now have all the components required to establish the completeness, binding, and knowledge soundness of our distributed polynomial commitment scheme. The completeness property follows directly from the completeness of the underlying Brakedown construction, as well as the completeness of the commitments to $\tilde{E}_1, \tilde{F}_1, \tilde{E}_2, \tilde{F}_2$ and the proof for relation $\mathcal{R}$. The binding

property follows from the binding of each local Merkle commitment and the collision resistance of H (modeled as a random oracle). Since the partition of M'_f is fixed, two distinct global encoded matrices must differ in at least one local submatrix. Opening both matrices consistently would therefore violate the binding property of the corresponding local Merkle commitment.

What remains is to show how our PCS achieves the knowledge soundness property. We construct our extractor $\mathcal{E}$ in a way similar to the extractor proposed by Brakedown PCS [7]. Specifically, let $\mathcal{A}$ be a PPT adversary that produces an accepting transcript when interacting with the verifier over our PCS protocol. $\mathcal{E}$ rewinds $\mathcal{A}$ multiple times. Each time $\mathcal{A}$ produces an accepting transcript, $\mathcal{E}$ first constructs $\mathcal{A}_R, \mathcal{A}_{F2}$, where $\mathcal{A}_R, \mathcal{A}_{F2}$ are PPT adversaries that produce an accepting transcript when interacting with the verifier over the proof for relation $\mathcal{R}$, the polynomial commitment to $\tilde{F}_2$, respectively. Then $\mathcal{E}$ invokes $\mathcal{E}_R$ to construct $\tilde{E}_1^*, \tilde{F}_1^*, \tilde{E}_2^*, \tilde{F}_2^*$, and invokes $\mathcal{E}_{F2}$ to construct $\tilde{F}_2'^*$, where $\mathcal{E}_R, \mathcal{E}_{F2}$ are the extractors corresponding to $\mathcal{A}_R, \mathcal{A}_{F2}$. By the knowledge soundness of the proof for $\mathcal{R}$ and the polynomial commitment to $\tilde{F}_2$, $\mathcal{E}$ runs in polynomial time and successfully extracts these vectors with overwhelming probability. Let $R := R^{(0)}||R^{(1)}||\ldots||R^{(M-1)}$ and $a := a^{(0)}||a^{(1)}||\ldots||a^{(M-1)}$. It remains to show that, given these extracted vectors, the following conditions hold with overwhelming probability.

1. For all $j \in [\ell]$, $E_1^*[idx_j] = \mathbf{y}_1[j]$, $E_2^*[idx_j] = \mathbf{y}_2[j]$, and $E_1^* = Enc(F_1^*)$, $E_2^* = Enc(F_2^*)$ where $\mathbf{y}_1[i] = \sum_{j\in[B]} R[i,j]\mathbf{a}[j]$, $\mathbf{y}_2[i] = \sum_{j\in[B]} R[i,j]\mathsf{eq}(\mathbf{s}_1, bin(j))$.
2. $\tilde{F}_2^* = \tilde{F}_2'^*$ and $v = \tilde{F}_2^*(\mathbf{s}_2)$.

To demonstrate this, we observe that: (1) By the binding property of Merkle commitment and the collision resistance of H (modeled as a random oracle), the probability that an adversary can find some $R' \neq R$ that produces verifier-accepted consistency values while passing verification is negligible. (2) By the binding property of the polynomial commitment to $\tilde{F}_2$, the probability that $\tilde{F}_2^* \neq \tilde{F}_2'^*$ is negligible. Applying the union bound, we conclude that given $E_1^*, F_1^*, E_2^*, F_2^*, F_2'^*$, both conditions above hold with probability $1 - M\ell \cdot negl(\lambda)$. Since M is polynomially bounded in λ and $\ell = \Theta(\lambda)$, the probability is overwhelming.

Having established that, we follow the soundness proof in Brakedown [7]. Specifically, the extractor $\mathcal{E}$ continues to rewind the adversary until it obtains B linearly independent vectors $\mathbf{a}_1, \ldots, \mathbf{a}_B$ along with the corresponding reply vectors $F_{1,1}, \ldots, F_{1,B}$ that pass the verification checks of the Brakedown verifier. The extractor then applies Gaussian elimination to recover the vector $F_f \in \mathbb{F}^N$, which contains the evaluations of the polynomial f. In terms of performance, our extractor incurs an additional multiplicative factor of $poly(\lambda, \frac{N}{B})$ in running time compared to [7], as it executes the sub-extractor $\mathcal{E}_{Enc}$ (as explained above) on each rewinding step. Since the original Brakedown extractor runs in expected polynomial time with respect to λ and N, as demonstrated in [7], we conclude that $\mathcal{E}$ runs in (expected) polynomial time.

Complexity The complexity analysis of our code-based distributed PCS (Protocol 6) is summarized as follows.

Prover time The main computational tasks of each sub-prover $\mathcal{P}_i$ are as follows: (1) Computing the vectors $E_1^{(i)}, F_1^{(i)}, E_2^{(i)}, F_2^{(i)}$, which requires $O(\frac{N}{B} \cdot b) = O(\frac{N}{M})$ field and hash operations. (2) Committing to and opening $\tilde{E}_1, \tilde{F}_1, \tilde{E}_2, \tilde{F}_2$ using the underlying polynomial commitment scheme PC, where each $\mathcal{P}_i$ incurs a cost of $t_P(\frac{N}{B})$, with t_P denoting the prover time complexity of PC. (3) Committing to and opening vectors using the Merkle commitment scheme MT, where each $\mathcal{P}_i$ performs $O(\frac{N}{B})$ hash operations. (4) Computing $\tilde{H}_{\mathbf{u}}$ and $\tilde{H}_{\mathbf{u},k}^{(i)}, \tilde{E}_k^{(i)}$ during the sumcheck protocol when proving the relation $\mathcal{R}$, where each $\mathcal{P}_i$ performs $O(\frac{cN}{B})$ field operations. (5) Distributed BaseFold evaluation proof in the distributed proof for relation $\mathcal{R}$, where $\mathcal{P}_i$ performs $O(\frac{cN}{BM} \log \frac{cN}{BM})$ field operations. Overall, each sub-prover $\mathcal{P}_i$ performs $O(\frac{cN}{B} + \frac{cN}{BM} \log \frac{cN}{BM})$ field operations and $O(\frac{N}{B} + \frac{cN}{BM} \log \frac{cN}{BM})$ hash operations, in addition to the cost $t_P(\frac{N}{B})$ incurred by the underlying polynomial commitment scheme.

Verifier time The main computational tasks of $\mathcal{V}$ are as follows: (1) Verifying the proof for commitments to $\tilde{E}_1, \tilde{F}_1, \tilde{E}_2, \tilde{F}_2$, where the running time of $\mathcal{V}$ is $Mt_V(\frac{N}{B})$, with t_V denoting the verifier time complexity of PC. (2) Verifying the proof for the Merkle commitments, where $\mathcal{V}$ performs $O(M\ell \log \frac{cN}{B})$ hash operations. (3) Computing $\mathbf{y}_1, \mathbf{y}_2$ in step 10b and 10d, where $\mathcal{V}$ performs $O(B\ell)$ field operations. (4) Verifying during the sumcheck protocol when proving the relation $\mathcal{R}$, where $\mathcal{V}$ performs $O(M \log \frac{cN}{B})$ field operations. (5) Distributed BaseFold evaluation proof in the distributed proof for relation $\mathcal{R}$, where $\mathcal{V}$ performs $O(M \log \frac{cN}{BM})$ field operations and $O(M \log^2 \frac{cN}{BM})$ hash operations. Overall, the verifier $\mathcal{V}$ performs $O(B\ell + M \log \frac{cN}{B})$ field operations and $O(M\ell \log \frac{cN}{B} + M \log^2 \frac{cN}{BM})$ hash operations, in addition to the cost $Mt_V(\frac{N}{B})$ incurred by the underlying PCS.

Communication and proof size The communication in the whole protocol mainly includes: (1) The columns $R^{(i)} \in \mathbb{F}^{\ell \times b}$ from $\mathcal{P}_i$ to $\mathcal{P}_0$, and then from $\mathcal{P}_0$ to $\mathcal{V}$. The message size is $O(B\ell)$. (2) The proof of Merkle hash commitments from $\mathcal{P}_i$ to $\mathcal{P}_0$, and then from $\mathcal{P}_0$ to $\mathcal{V}$. The message size is $O(M\ell \log \frac{cN}{B})$. (3) The proof of commitments to $\tilde{E}_1, \tilde{F}_1, \tilde{E}_2, \tilde{F}_2$ from $\mathcal{P}_i$ to $\mathcal{P}_0$, and then from $\mathcal{P}_0$ to $\mathcal{V}$. The message size is $Ms_P(\frac{N}{B})$, with s_P denoting the proof size of PC. (4) The local sumcheck messages $Q_k^{(i)}(X_k)$ from $\mathcal{P}_i$ to $\mathcal{P}_0$ when proving the relation $\mathcal{R}$, with total size $O(M \log \frac{cN}{B})$; the aggregated messages $Q_k(X_k)$ from $\mathcal{P}_0$ to $\mathcal{V}$, with total size $O(\log \frac{cN}{B})$. (5) Distributed BaseFold evaluation proof in the distributed proof for relation $\mathcal{R}$, where the communication size between sub-provers and the proof size is $O(M \log^2 \frac{N}{BM})$. Overall, the communication size between sub-provers (from $\mathcal{P}_i$ to $\mathcal{P}_0$) is $O(B\ell + M\ell \log \frac{cN}{B} + M \log^2 \frac{N}{BM}) + Ms_P(\frac{N}{B})$, and the proof size (from $\mathcal{P}_0$ to $\mathcal{V}$) is the same.

We assume the reciprocal of the code rate c is a constant, $\ell = \Theta(\lambda)$, $M = \Theta(1)$, and set $B = \Theta(M \log \frac{N}{M})$. If the underlying polynomial commitment

scheme PC has complexity $t_P(n) = O(n \log n), t_V(n) = s_P(n) = O(\lambda \log^2 n)$, our distributed polynomial commitment achieves the following performance:

- Each prover $\mathcal{P}_i$ runs in $O(\frac{N}{M})$ time.
- The verifier $\mathcal{V}$ runs in $O(\lambda M \log^2 \frac{N}{M})$ time.
- The communication size between sub-provers is $O(\lambda M \log^2 \frac{N}{M})$.
- The proof size is $O(\lambda M \log^2 \frac{N}{M})$.

We can select DeepFold [8] as the underlying PC, because it is transparent, post-quantum, and satisfies the required complexity properties. As a result, we obtain a transparent distributed polynomial commitment with linear prover time, polylogarithmic verifier time, communication, and proof size.

4 Evaluation

We evaluate the concrete performance of our distributed PCS and compare it with prior distributed PCS implementations. Our implementation instantiates our code-based distributed PCS construction with DeepFold as the underlying PCS, denoted by *Ours*. We compare against the released implementations of LigeSIS [9], FRIttata's Fold-and-Batch PCS [23], denoted by *dFRIttata*, the non-accountable variant of $\text{dePIP}_{\text{FRI}}$ [11], denoted by *dPIP-FRI*, and HyperFond [24]. We do not include Veloz [25] or UltraFold [19] in the empirical comparison because runnable public implementations were not available to us when the benchmark suite was finalized; both are nevertheless included in the asymptotic comparison in Table 1. We report prover time, verifier time, proof size, communication between parties, and parallel scalability.

4.1 Settings and Parameters

All experiments are conducted on a single GCE `n2-highmem-96` instance with 96 vCPUs, corresponding to 48 physical Intel Ice Lake cores across two NUMA nodes, and approximately 768 GiB of memory. We evaluate multilinear polynomials of length $N \in \{2^{22}, \ldots, 2^{28}\}$. For the parallel-scaling experiment, Ours is evaluated with $M \in \{1, 2, 4, 8, 16\}$ logical parties, where $M = 1$ serves as the single-party baseline. Each party is allocated three CPU threads. Parties are instantiated as localhost processes, and configurations are executed sequentially to avoid interference between schemes. Thus, M represents the logical degree of distribution rather than the number of physical servers.

For every scheme, prover time is the sum of commitment and opening time; separately exposed setup and random-input generation are excluded. Verifier time is measured by executing the corresponding verifier, and all reported proofs verify successfully. Proof size is the serialized verifier-facing proof, and communication denotes the total protocol-level data exchanged among the logical parties.

Table 2 summarizes the main cryptographic parameters used in the evaluation. Ours, LigeSIS, dPIP-FRI, and HyperFond are evaluated at a 100-bit

Table 2. Security targets and main implementation parameters used in the evaluation.

Scheme	Security	Field	Rate	PCS	Hash
Ours	100-bit	Mersenne61	1/2	DeepFold	BLAKE3-256
LigeSIS	100-bit	Goldilocks	1/4	DeepFold	Subset-sum/SIS
dFRIttata	128-bit	128-bit prime	1/4	FRI	BLAKE3-256
dPIP-FRI	100-bit	Goldilocks	1/4	PIPFRI	BLAKE3-256
HyperFond	100-bit	Goldilocks	1/8	BaseFold	BLAKE2s-256

security target, while dFRIttata retains its released 128-bit configuration. For consistency, the table reports the underlying base field; extension-field arithmetic, where used, follows the corresponding implementation configuration.

For reproducibility, Ours uses the quadratic extension $\mathbb{F}_{p^2}$ with $p = 2^{61} - 1$, while dFRIttata uses the quadratic extension with $p = 2^{128} - 45 \cdot 2^{40} + 1$. For LigeSIS, the listed 1/4 rate refers to the main Reed–Solomon code in the evaluated configuration, with DeepFold used as its auxiliary PCS.

Because the released implementations do not expose identical cryptographic configurations, the experiments should be interpreted as comparisons of concrete implementation operating points rather than a strictly identical-parameter cryptographic ranking. Since all parties run as localhost processes on a single host, the experiments isolate computational scalability and protocol-level communication volume; they do not model multi-host network latency or bandwidth contention.

4.2 Parallel Scalability

We first evaluate the parallel scalability of the prover by varying the number of logical parties from 1 to 16. As shown in Figure 1, doubling the number of parties approximately halves the prover time across all tested polynomial sizes.

Relative to the single-party baseline, eight parties achieve a 7.17×–7.89× speedup, corresponding to 90%–99% parallel efficiency, defined as the speedup divided by the number of parties. With 16 parties, the speedup reaches 13.95×–15.58×, corresponding to 87%–97% parallel efficiency. For the largest instance, $N = 2^{28}$, prover time decreases from 176.0 s with one party to 87.2 s, 43.7 s, 22.7 s, and 11.5 s with 2, 4, 8, and 16 parties, respectively.

At $M = 16$, the parties use 48 CPU threads in total, matching the number of physical cores of the host. These results show that the dominant computation of our construction can be effectively partitioned across parties and scales close to linearly throughout the available physical-core range.

4.3 Comparison with Prior Work

We next compare Ours with prior implementations. The comparison fixes $M = 8$, corresponding to 24 CPU threads in total. Figure 2 reports prover time, verifier time, and communication between parties over polynomial sizes from 2^{22} to 2^{28}.

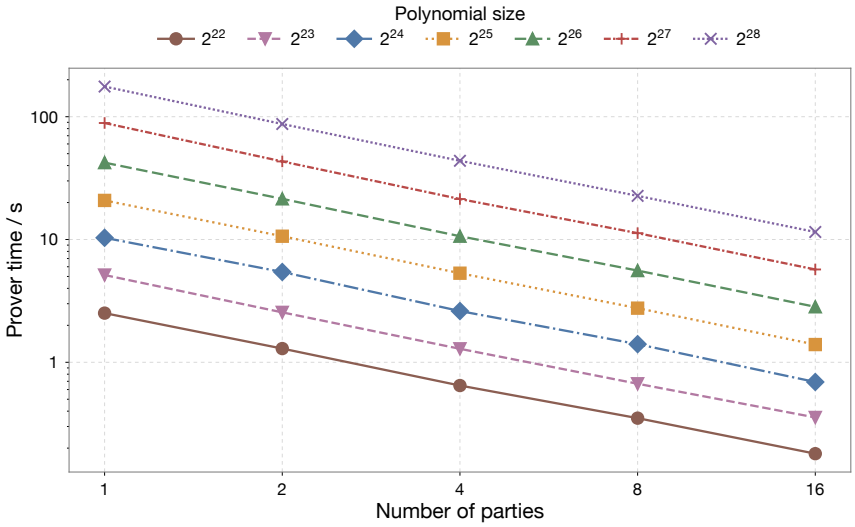


Fig. 1. Parallel scalability of the prover for polynomial lengths $N = 2^{22}$ to 2^{28}. $M = 1$ is the single-party baseline, and each party is allocated three CPU threads. The vertical axis is logarithmic.

Prover time. Figure 2(a) shows that Ours and dPIP-FRI form the fastest group over most of the tested range. Ours is consistently 4.1×–4.4× faster than dFRIttata and 3.6×–3.9× faster than HyperFond. Compared with LigeSIS, Ours is 1.2×–3.5× faster for $N \leq 2^{26}$; the two become comparable at $N = 2^{27}$, while LigeSIS is faster at $N = 2^{28}$. This crossover is consistent with LigeSIS's reported behavior that the overhead of its secondary PCS is amortized more effectively at larger polynomial sizes. Ours also closely tracks dPIP-FRI throughout the tested range and remains within 1.31× of the fastest implementation at every polynomial size. At $N = 2^{28}$, Ours proves in 22.7 s, compared with 23.2 s for dPIP-FRI, 19.0 s for LigeSIS, 87.1 s for HyperFond, and 92.9 s for dFRIttata. Thus, the substantial reduction in communication achieved by our construction does not require a large prover-time penalty.

Verifier time. Figure 2(b) shows a clear separation of Ours and dPIP-FRI from the remaining implementations. The verifier time of Ours increases only from 2.81 ms to 4.18 ms as the polynomial length grows by a factor of 64, while dPIP-FRI ranges from 2.73 ms to 4.77 ms. The other three implementations require at least an order of magnitude more verification time across the tested range. At $N = 2^{28}$, Ours verifies in 4.18 ms, compared with 4.77 ms for dPIP-FRI, 52.5 ms for dFRIttata, 556.5 ms for HyperFond, and 661.3 ms for LigeSIS. These results show that distributing the proof-composition layer preserves a lightweight verifier even at the largest tested polynomial size.

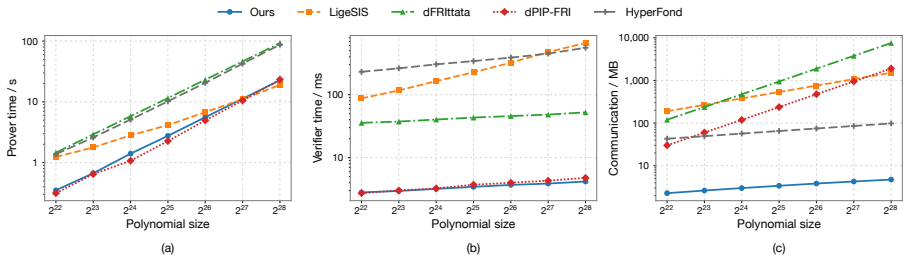


Fig. 2. Comparison at $M = 8$ logical parties: (a) prover time, measured as commitment time plus opening time; (b) verifier time; and (c) communication between parties. "Polynomial size" denotes the committed multilinear-polynomial length N. All vertical axes are logarithmic.

Communication. Communication shows a clear separation among the evaluated implementations in Figure 2(c). Ours communicates 2.28–4.75 MB as the polynomial size increases by a factor of 64, remaining at least 13× below every evaluated baseline throughout the tested range. HyperFond is the closest baseline in communication, but still communicates approximately 19×–21× more data than Ours across the tested range, while the gaps relative to the other schemes are larger still. At $N = 2^{28}$, Ours communicates 4.75 MB, compared with 98.1 MB for HyperFond, 1,515 MB for LigeSIS, 1,896 MB for dPIP-FRI, and 7,520 MB for dFRIttata. These correspond to reductions of approximately 20.6×, 319×, 399×, and 1,583×, respectively. The low communication is consistent with the structure of our construction, where cross-party traffic is dominated by compact commitments, sampled openings, and sumcheck messages.

Proof size. Although proof size is not shown in Figure 2, we report it for completeness. At $M = 8$, Ours produces verifier-facing proofs of 2.26–4.73 MB over the tested range. These proofs are 12%–31% smaller than those of dFRIttata and more than 20× smaller than those of HyperFond, but larger than those of LigeSIS and dPIP-FRI. Thus, proof size is not the primary empirical advantage of our construction.

References

1. Arnon, G., Chiesa, A., Fenzi, G., Yogev, E.: Stir: Reed-Solomon proximity testing with fewer queries. In: Advances in Cryptology - EUROCRYPT 2024. pp. 380–413. Springer (2024)
2. Arnon, G., Chiesa, A., Fenzi, G., Yogev, E.: WHIR: Reed-Solomon proximity testing with super-fast verification. In: Advances in Cryptology - EUROCRYPT 2025. pp. 214–243. Springer (2025)
3. Bowe, S., Hornby, T.G., Wilcox, N.: Zcash protocol specification (2016)

4. Chen, B., Bünz, B., Boneh, D., Zhang, Z.: HyperPlonk: Plonk with linear-time prover and high-degree custom gates. In: Advances in Cryptology - EUROCRYPT 2023. pp. 499–530. Springer (2023)
5. Ethereum Foundation: ZK-Rollups: Scaling solutions for ethereum (2024)
6. Fiat, A., Shamir, A.: How to prove yourself: Practical solutions to identification and signature problems. In: Conference on the theory and application of cryptographic techniques. pp. 186–194. Springer (1986)
7. Golovnev, A., Lee, J., Setty, S.T.V., Thaler, J., Wahby, R.S.: Brakedown: Linear-time and field-agnostic SNARKs for R1CS. In: Advances in Cryptology - CRYPTO 2023. pp. 193–226. Springer (2023)
8. Guo, Y., Liu, X., Huang, K., Qu, W., Tao, T., Zhang, J.: DeepFold: Efficient multilinear polynomial commitment from Reed-Solomon code and its application to zero-knowledge proofs. In: 34th USENIX Security Symposium (USENIX Security '25). pp. 3497–3516 (2025)
9. Guo, Y., Lou, H., Qu, W., Xu, Z., Zhang, J.: LigeSIS: Distribution-friendly polynomial commitment based on error-correcting code. Cryptology ePrint Archive (2026)
10. Li, C., Zhu, P., Li, Y., Hong, C., Qu, W., Zhang, J.: HyperPianist: Pianist with linear-time prover and logarithmic communication cost. In: Proceedings of the 46th IEEE Symposium on Security and Privacy (S&P '25). pp. 3383–3401. IEEE (2025)
11. Li, W., Zhang, Z., Chow, S.S.M., Guo, Y., Gao, B., Song, X., Deng, Y., Liu, J.: Shred-to-shine metamorphosis of (distributed) polynomial commitments. In: Proceedings of the 35th USENIX Security Symposium (USENIX Security '26) (2026), https://github.com/leeweihanwickham/PIP_FRI-and-DePIP_FRI
12. Li, W., Zhang, Z., Li, Y., Zhu, P., Hong, C., Liu, J.: Soloist: Distributed SNARK for rank-one constraint system. In: Advances in Cryptology - EUROCRYPT 2026. Springer (2026), https://github.com/AntCPLab/soloist
13. Liu, T., Xie, T., Zhang, J., Song, D., Zhang, Y.: Pianist: Scalable zkRollups via fully distributed zero-knowledge proofs. In: Proceedings of the 45th IEEE Symposium on Security and Privacy (S&P '24). pp. 1777–1793. IEEE (2024)
14. Lund, C., Fortnow, L., Karloff, H., Nisan, N.: Algebraic methods for interactive proof systems. Journal of the ACM (JACM) **39**(4), 859–868 (1992)
15. Nair, V., Sharma, A., Thankey, B.: BrakingBase - a linear prover, poly-logarithmic verifier, field agnostic polynomial commitment scheme. In: Advances in Cryptology - ASIACRYPT 2025. pp. 427–455. Springer (2025)
16. Pappas, C., Papadopoulos, D.: Hobbit: Space-efficient zkSNARK with optimal prover time. In: 34th USENIX Security Symposium (USENIX Security '25). pp. 4917–4936 (2025)
17. Rosenberg, M., Mopuri, T., Hafezi, H., Miers, I., Mishra, P.: Hekaton: Horizontally-scalable zkSNARKs via proof aggregation. In: Proceedings of the 2024 ACM SIGSAC Conference on Computer and Communications Security (CCS 2024). pp. 929–940. Association for Computing Machinery (2024)
18. Wang, W., Shi, F., Vilardell, D., Zhang, F.: Cirrus: Performant and accountable distributed SNARK. Cryptology ePrint Archive, to appear at NDSS 2026 (2024)
19. Wang, W., Zhang, F.: UltraFold: Efficient distributed BaseFold from packed interleaved merkle trees. Cryptology ePrint Archive (2026)
20. Wu, H., Zheng, W., Chiesa, A., Popa, R.A., Stoica, I.: DIZK: A distributed zero knowledge proof system. In: Proceedings of the 27th USENIX Security Symposium (USENIX Security '18). pp. 675–692 (2018)

21. Xie, T., Zhang, J., Zhang, Y., Papamanthou, C., Song, D.: Libra: Succinct zero-knowledge proofs with optimal prover computation. In: Advances in Cryptology - CRYPTO 2019. pp. 733–764. Springer (2019)
22. Xie, T., Zhang, J., Cheng, Z., Zhang, F., Zhang, Y., Jia, Y., Boneh, D., Song, D.: zkbridge: Trustless cross-chain bridges made practical. In: Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security (CCS 2022). pp. 3003–3017. Association for Computing Machinery (2022)
23. Xu, H., Gama, M., Beni, E.H., Kang, J.: FRIttata: A FRI-based polynomial commitment scheme for distributed proof generation. Cryptology ePrint Archive (2025)
24. Yu, Y., Liu, M., Zhang, Y., Sun, S.F., Ma, T., Au, M.H., Gu, D.: HyperFond: A transparent and post-quantum distributed SNARK with polylogarithmic communication. Cryptology ePrint Archive (2025)
25. Yu, Y., Sun, S.F., Zhang, Y., Fan, C., Ma, T., Gu, D.: Veloz: Efficient and flexible distribution framework for code-based polynomial commitment scheme. Cryptology ePrint Archive (2026)
26. Zeilberger, H., Chen, B., Fisch, B.: BaseFold: Efficient field-agnostic polynomial commitment schemes from foldable codes. In: Advances in Cryptology–CRYPTO 2024. pp. 138–169. Springer (2024)
27. Zhang, Z., Li, W., Guo, Y., Shi, K., Chow, S.S., Liu, X., Dong, J.: Fast RS-IOP multivariate polynomial commitments and verifiable secret sharing. In: 33rd USENIX Security Symposium (USENIX Security '24). pp. 3187–3204 (2024)

A Sumcheck Protocol

We present the sumcheck PIOP for the relation $\mathcal{R}_{SUM}$ for high-degree polynomials in [4], where the prover claims to the verifier that $((v, [[f]]); f) \in \mathcal{R}_{SUM}$. In the k-th round, the prover can compute the univariate polynomial f_k in linear time by adapting the optimal algorithm from [21]. The construction is shown in Protocol A.1.

Protocol A.1 Sumcheck PIOP for high-degree polynomials.
$\mathcal{P}$ claims $((v, [[f]]); f) \in \mathcal{R}_{SUM}$ to $\mathcal{V}$ for a μ-variate degree-d polynomial f such that $\sum_{b\in\{0,1\}^\mu} f(b) = v$.

1. For $k = 1, 2, \ldots, \mu$,
   (a) The prover computes $f_k(X) = \sum_{\mathbf{b}\in\{0,1\}^{\mu-k}} f(r_1, \ldots, r_{k-1}, X, \mathbf{b})$ and sends the oracle $[[f_k]]$ to the verifier. f_k is univariate and of degree at most d.
   (b) The verifier checks that $v = f_k(0) + f_k(1)$, if the check fails, abort; else the verifier samples $r_k \leftarrow \mathbb{F}$, sends r_k to the prover, and sets $v \leftarrow f_k(r_k)$.
2. Finally, the verifier accepts if $f(r_1, \ldots, r_\mu) = v$.

The completeness of $\mathcal{R}_{SUM}$ is obvious. By Theorem 3.1 of [4], the sumcheck protocol is proven to be knowledge sound.

HyperPianist [10] provides a distributed variant of sumcheck PIOP. The construction is shown in Protocol A.2.

Protocol A.2 Distributed sumcheck PIOP.
$\mathcal{P}_0$ claims $((v, [[f]]); f) \in \mathcal{R}_{SUM}$ to $\mathcal{V}$, where $f \in \mathcal{F}_n^{(\leq 1)}$. $\mathcal{P}_i$ holds $f^{(i)}(x) = f(x, bin(i))$, $i \in [M]$. Let $v_0 := v, m = \log M$.

1. In the k-th round where $1 \leq k \leq n - m$:
   (a) Each $\mathcal{P}_i$ sends a local univariate polynomial $f_k^{(i)}(X_k) := \sum_{x \in \{0,1\}^{n-m-k}} f(r_1, \ldots, r_{k-1}, X_k, x, bin(i))$ to $\mathcal{P}_0$.
   (b) $\mathcal{P}_0$ sums up all the univariate polynomials to get $f_k(X_k) = \sum_{i \in [M]} f_k^{(i)}(X_k)$,
   and sends it to $\mathcal{V}$.
   (c) $\mathcal{V}$ checks if $v_{k-1} = f_k(0) + f_k(1)$. If the check passes, $\mathcal{V}$ sends a random $r_k \in \mathbb{F}$ to $\mathcal{P}_0$, and sets $v_k := f_k(r_k)$.
   (d) $\mathcal{P}_0$ transmits r_k to the other $\mathcal{P}_i$. Each $\mathcal{P}_i$ then updates the local evaluations of $f^{(i)}(x)$.
2. Each $\mathcal{P}_i$ sends $f^{(i)}(r_1, \cdots, r_{n-m})$ to $\mathcal{P}_0$. $\mathcal{P}_0$ constructs the polynomial $f'(x) = f(r_1, \cdots, r_{n-m}, x)$.
3. $\mathcal{P}_0$ and $\mathcal{V}$ run the sumcheck PIOP to check $v_{n-m} = \sum_{x \in \{0,1\}^m} f'(x)$.