BrakingBase - a linear prover, poly-logarithmic verifier, field agnostic polynomial commitment scheme

Vineet Nair
Arithmic Labs
vineet@arithmic.com

Ashish Sharma
Arithmic Labs
ashish@arithmic.com

Bhargav Thankey*
Indian Institute of Science
thankeyd@iisc.ac.in

Abstract

We propose a Polynomial Commitment Scheme (PCS), called BrakingBase, which allows a prover to commit to multilinear (or univariate) polynomials with n coefficients in $O(n)$ time. The evaluation protocol of BrakingBase operates with an $O(n)$ time-complexity for the prover, while the verifier time-complexity and proof-complexity are $O(\lambda \log^2 n)$, where λ is the security parameter. Notably, BrakingBase is field-agnostic, meaning it can be instantiated over any field of sufficiently large size. Additionally, BrakingBase can be combined with the Polynomial Interactive Oracle Proof (PIOP) from Spartan (Crypto 2020) to yield a Succinct Non-interactive ARgument of Knowledge (SNARK) with a linear-time prover, as well as poly-logarithmic complexity for both the verifier runtime and the proof size. We obtain our PCS by combining the Brakedown and Basefold PCS. The commitment protocol of BrakingBase is similar to that of Brakedown. The evaluation protocol of BrakingBase improves upon Brakedown's verifier work by reducing it through multiple instances of the sum-check protocol. Basefold PCS is employed to commit to and later evaluate the multilinear extension (MLE) of the witnesses involved in the sum-check protocol at random points. This includes the MLE corresponding to the parity-check matrix of the linear-time encodable code used in Brakedown. We show that this matrix is sparse and use the Spark compiler from Spartan to evaluate its multilinear extension at a random point. We implement BrakingBase and compare its performance to Brakedown and Basefold over a 128 bit prime field.

*Partially supported by the Prime Minister's Research Fellowship, India. The work was done when the author was a research intern at Arithmic labs.

1 Introduction

A Succinct Non-interactive Argument of Knowledge (SNARK) for an NP relation $\mathcal{R}$ allows a prover to convince a verifier that it possesses a (private) witness w corresponding to a public input x, such that $(x, w) \in \mathcal{R}$. The prover achieves this by producing a proof π, where both the size of π and the verifier's runtime complexity for checking its correctness are sublinear relative to the size of the circuit computing $\mathcal{R}$. SNARKs are crucial in numerous real-world applications where computations must be verified repeatedly by many resource-constrained parties. In such scenarios, a single resource-intensive party acts as the prover, generating a proof π that asserts the validity of a required computation. Resource-constrained parties can then quickly verify this computation by running the verifier algorithm with inputs x and π. If the verifier algorithm accepts, they can be confident with very high probability that the computation is correct. A classic example of this concept in action is validity-rollups, which are used to scale layer one blockchains like Ethereum.

Most SNARKs are constructed by combining information-theoretic protocols known as Polynomial Interactive Oracle Proofs (PIOPs) with a Polynomial Commitment Scheme (PCS). In a PIOP, the verifier is provided with oracle access to certain polynomials, in addition to receiving the proof itself. A PIOP becomes concrete with the integration of a PCS, enabling the prover to commit to a polynomial and subsequently reveal its evaluation at specific points to the verifier in a secure manner. The parameters of both PIOPs and PCS significantly influence the overall quality of a SNARK. Specifically, for a circuit of size T that computes $\mathcal{R}$, PIOPs exist that operate over any field, with a prover runtime of $O(T)$, while the verifier's runtime and proof complexity remain $O(\log T)$ (for example Spartan [Set20]). However, this is not universally true for PCS, as its performance often becomes a critical bottleneck in the efficiency of a SNARK. Furthermore, the security or efficiency requirements of the PCS may restrict the SNARK to operate over certain types of fields, limiting flexibility. Our primary contribution in this work is a novel PCS scheme, denoted as BrakingBase. To provide context for our results, we briefly review the different types of PCS that have been studied, examining the various considerations involved.

Polynomial Commitment Schemes (PCS) with various trade-offs have been extensively studied in recent years and can generally be classified along two main axes: (a) trusted vs. transparent setup, and (b) field-dependent vs. field-agnostic. A PCS with a preprocessed trusted setup involves a secret trapdoor that must be securely destroyed after generation. In contrast, a transparent setup requires no secret trapdoors, which can simplify deployment. Examples of PCS with a trusted setup include KZG [KZG10], Zeromorph [KT24], and KZG-FFT [GNS24]. In comparison, transparent setups are used in schemes such as Bulletproof [BBB+18], Dory [Lee21], FRI [BBHR18], Ligero [AHIV17], Brakedown [GLS+23], and Basefold [ZCF24]. Additionally, the security or efficiency requirements of a PCS can determine whether it, and by extension the resulting SNARK, is field-dependent or field-agnostic. Field-dependent PCS can be instantiated only over fields with specific properties, typically prime fields of large order. For example, KZG and Dory require bilinear pairings, restricting their operation to large fields where pairing-friendly elliptic curves can be defined.

Recently, there has been significant interest in designing PCS that can operate over smaller-sized

fields [DP23, HLP24]. The motivation behind this is rooted in the observation that witness values in practical SNARKs tend to be much smaller than the fields they are embedded in, and the associated constraints on these witnesses can often be expressed over smaller fields. If SNARKs are restricted to construction over large fields, a substantial embedding cost is introduced. For instance, in the Jolt zkVM [AST24], most witness values are around 32 bits, yet the Surge PCS scheme employed in Jolt requires elliptic curves, which necessitate a prime field of 252 bits to ensure 126 bits of security. Thus, a secure and efficient PCS that can operate over smaller fields, when combined with a field-agnostic PIOP, can enable the deployment of SNARKs over relatively smaller-sized fields, yielding significant practical performance improvements.

Brakedown and Basefold introduced the concept of field-agnostic PCS, which can operate over fields of any characteristic, provided they are sufficiently large. This adaptability allows SNARKs to be deployed over fields selected based on application-specific constraints, rather than being restricted by the security and efficiency requirements of the PCS. Field-agnostic PCS are commonly constructed using the following approach: (a) design an Interactive Oracle Proof of Proximity (IOPP) to verify whether a given vector is close to a predetermined linear error-correcting code, and (b) derive a PCS from this IOPP. The second step is well-established, as a PCS can be efficiently constructed from an IOPP by using Merkle commitments to secure the prover's messages. We propose a new field-agnostic PCS, denoted BrakingBase, based on this paradigm. Below, we provide a brief overview of existing PCS schemes built on this paradigm and later compare the performance of BrakingBase with them.

The earliest PCS scheme based on the above approach was derived from the Fast Reed-Solomon Interactive Proof of Proximity (FRI). This scheme achieves a PCS with a prover runtime of $O(n \log n)$ and verifier runtime and proof complexity of $O_\lambda(\log^2 n)$.[1] However, for FRI's prover to be efficient, the underlying field must contain large multiplicative cyclic subgroups, particularly of large powers of 2. This makes FRI field-dependent rather than field-agnostic. Although EC-FFT [BCKL22, BCKL23] reduces this restriction, it raises the prover costs to $O(n \log^2 n)$. Circle Starks extend the EC-FFT approach to enable FRI over prime fields of order p, provided $p+1$ is divisible by a large power of 2. Basefold introduced the notion of foldable linear codes and provided an IOPP for them, producing a field-agnostic PCS with a prover runtime of $O(n \log n)$ and verifier runtime and proof complexity of $O_\lambda(\log^2 n)$.[2] Ligero, implicitly, provides an IOPP-based PCS with prover runtime of $O(n \log n)$ and verifier runtime and proof complexity of $O_\lambda(\sqrt{n})$. While Ligero's prover is faster than those of FRI and Basefold, it has a significantly larger proof size and is field-dependent due to its reliance on an IOPP for RS codes. Ligero++ [BFH+20] further reduces the verifier runtime and proof complexity of Ligero to $O_\lambda(\log^2 n)$ by composing the Ligero PCS with an inner-product argument from [ZXZS20], though it remains field-dependent. Brakedown replaces Ligero's IOPP for RS codes with an IOPP for linear-time encodable codes. It also constructs an expander-graph-based, linear-time encodable code that operates over any sufficiently large field, yielding a PCS with a prover runtime of $O(n)$ and verifier runtime and proof complexity of $O_\lambda(\sqrt{n})$. Orion[XZS22] and Hyperplonk[CBBZ23] later improved Brakedown's verifier

[1] Throughout, we let n be the size of the coefficient vector of the polynomial used in the PCS, and λ be the security parameter.

[2] We remark that the commitment protocols of FRI and Basefold run in $O(n \log n)$, whereas the evaluation prover runs in $O(n)$ time.

PCS	Field Agnostic?	Commit time	Prover time	Verifier time	Proof size
FRI	No	$O(n \log n)$	$O(n)$	$O(\lambda \log^2 n)$	$O(\lambda \log^2 n)$
Ligero	No	$O(n \log n)$	$O(n)$	$O\left(\sqrt{\lambda n}\right)$	$O\left(\sqrt{\lambda n}\right)$
Brakedown	Yes	$O(n)$	$O(n)$	$O\left(\sqrt{\lambda n}\right)$	$O\left(\sqrt{\lambda n}\right)$
Ligero++	No	$O(n \log n)$	$O(n)$	$O\left(\lambda \log^2 n\right)$	$O\left(\lambda \log^2 n\right)$
BaseFold	Yes	$O(n \log n)$	$O(n)$	$O(\lambda \log^2 n)$	$O(\lambda \log^2 n)$
BrakingBase	Yes	$O(n)$	$O(n)$	$O(\lambda \log^2 n)$	$O(\lambda \log^2 n)$

Table 1: Asymptotic costs of different PCSes for security parameter λ and a polynomial in k variables; $n := 2^k$.

complexity to $O_\lambda(\log^2 n)$ using general SNARK and proof composition techniques, though they remain field-dependent.

In summary, Brakedown and Basefold are currently the only two PCS that are field-agnostic. Our proposed PCS, denoted BrakingBase, leverages both Brakedown and Basefold in combination to achieve a more performant PCS compared to either of them. The prover in BrakingBase operates in $O(n)$ time, with the verifier runtime and proof complexity being $O_\lambda(\log^2 n)$. Table 1 provides a performance comparison of BrakingBase with other PCS schemes. Additionally, BrakingBase can be combined with the PIOP from Spartan to construct a SNARK with a linear-time prover, poly-logarithmic verifier runtime and proof complexity over any sufficiently large field. Notably, to our knowledge, this work is the first to achieve a SNARK with both a linear-time prover and poly-logarithmic proof size across all fields of sufficiently large size. Recently, [BCF+24] introduced Blaze, a PCS with a linear-time prover and poly-logarithmic proofs over binary fields. However, this PCS is restricted to fields of characteristic 2; see Section 1.2 for a more detailed comparison.

We organize the remaining paper as follows: In Section 1.1, we present a technical overview of BrakingBase, and in Section 1.2 we provide a more detailed comparison of BrakingBase with all the relevant PCS. In Section 3, we give the protocols corresponding to BrakingBase and prove its correctness. Finally, in Section 4, we compare concretely the performance of BrakingBase with Basefold and Brakedown.

1.1 Technical Overview

This section provides a high-level overview of our PCS, denoted BrakingBase. Throughout, we explain BrakingBase as PCS for multilinear polynomials, but the discussion here can be easily adapted to the setting of univariate polynomials. The prover starts with a multilinear polynomial in ℓ variables, defining $n := 2^\ell$. The goal of the prover is to compute a commitment to this polynomial and subsequently prove that the committed polynomial evaluates to a specified value at a verifier-chosen point. Throughout this work, we focus on polynomials represented in the Lagrange basis over the set $\{0,1\}^\ell$, that is a polynomial is represented as a vector of its evaluations over the boolean hypercube. As stated previously, BrakingBase achieves $O(n)$ commitment and evaluation prover time, and $O(\lambda \log^2 n)$ verifier time and proof-complexity. To construct our PCS, we compose Brakedown's PCS [GLS+23] with the BaseFold PCS [ZCF24]. Before we proceed, we

note that for a vector $\mathbf{p} \in \mathbb{F}^n$, we use $\widetilde{\mathbf{p}}$ to denote the multilinear extension (MLE) corresponding to $\mathbf{p}$ (see Section 2 for a definition of MLE). We begin by outlining Brakedown's PCS and then detail the composition process.

Brakedown Commitment Protocol: The Brakedown PCS leverages a linear error-correcting code that is computable in linear time. For any vector $\mathbf{x}$, denote its encoding under this code as $\mathsf{Enc}(\mathbf{x})$. To commit to an ℓ variate multilinear polynomial f, the Brakedown commitment protocol first represents the evaluations of f over the boolean hypercube $\{0,1\}^\ell$ as a matrix $A \in \mathbb{F}^{n_1 \times n_2}$, where n_1, n_2 satisfy $n = n_1 \cdot n_2$. We assume that n_1 and n_2 are powers of 2, , with rows and columns of A indexed by strings in $\{0,1\}^{\log n_1}$, and $\{0,1\}^{\log n_2}$ respectively. The entry of A at row i and column j, where $i \in \{0,1\}^{\log n_1}$, and $j \in \{0,1\}^{\log n_2}$, corresponds to the evaluation of f at the point (i,j). In the next part of the commitment protocol a matrix C is computed such that each row $C(i,\cdot)$ is the encoding $\mathsf{Enc}(A(i,\cdot))$ for all $i \in \{0,\ldots,n_1-1\}$. The commitment is then made to the Merkle hash of C. Note that $C \in \mathbb{F}^{n_1 \times \rho n_2}$, where ρ is the reciprocal of the rate of the code.

Brakedown Evaluation Protocol: The evaluation proof of f at $(\alpha_0,\ldots,\alpha_{\ell-1})$ consists of two main steps: a proximity test and an evaluation check. A subsequent work [DP24] demonstrates that these two steps can be merged if the evaluation point is chosen uniformly at random; however, for clarity, we retain the two-step approach. In the proximity test, the Brakedown verifier sends a randomly chosen vector $\mathbf{r} \in \mathbb{F}^{n_1}$ to the prover. The prover then responds with a vector $\mathbf{p}$, which should be the sequence $(\langle \mathbf{r}, A(\cdot,0)\rangle, \ldots, \langle \mathbf{r}, A(\cdot,n_2-1)\rangle)$, where each entry represents an inner product of $\mathbf{r}$ with columns of the matrix A. The verifier in turn uniformly at random samples an $I \subseteq \{0,\ldots,n_2-1\}$ of size $\Theta(\lambda)$ and sends I to the prover. In response, the verifier expects the prover to provide the columns of C for the indices in I; specifically $C(\cdot,i)$ for al $i \in I$ along with their corresponding Merkle proofs. Upon receiving them, it checks all the Merkle proofs, and that $\langle \mathbf{r}, C(\cdot,i)\rangle$ is equal to $\mathsf{Enc}(\mathbf{p})(i)$ for all $i \in I$. Here $\mathsf{Enc}(\mathbf{p})(i)$ denotes the i-th component of the vector $\mathsf{Enc}(\mathbf{p})(i)$. If these checks pass, then with high probability all rows of C are close to a codeword. For the actual evaluation, the prover sends $\mathbf{q}$, intended to be $(\langle \boldsymbol{\alpha}_{\text{row}}, A(\cdot,0)\rangle, \ldots, \langle \boldsymbol{\alpha}_{\text{row}}, A(\cdot,n_2-1)\rangle)$, where $\boldsymbol{\alpha}_{\text{row}} := \otimes_{i=0}^{(\log n_1)-1}(1-\alpha_i,\alpha_i)$. The verifier repeats the steps of the proximity check with $\mathbf{p}$ replaced by $\mathbf{q}$ (and with a fresh I). If the checks for $\mathbf{q}$ pass, then with high probability $\mathbf{q} = (\langle \boldsymbol{\alpha}_{\text{row}}, A(\cdot,0)\rangle, \ldots, \langle \boldsymbol{\alpha}_{\text{row}}, A(\cdot,n_2-1)\rangle)$. The verifier outputs $\langle \boldsymbol{\alpha}_{\text{col}}, \mathbf{q}\rangle$, where $\boldsymbol{\alpha}_{\text{col}} := \otimes_{i=\log n_1}^{\ell-1}(1-\alpha_i,\alpha_i)$; it is easy to see that this is indeed $f(\alpha_0,\ldots,\alpha_{\ell-1})$. The proof size is $\max\{O(n_2), O(\lambda n_1)\}$. Thus, n_1, n_2 are chosen so as to minimise this maximum yielding a proof of size $O(\sqrt{\lambda n})$. Also, the verifier's runtime complexity is equal to $O_\lambda(\sqrt{n})$ primarily dominated by the computation of $\mathsf{Enc}(\mathbf{p})$.

BrakingBase: The commitment protocol of BrakingBase is identical to Brakedown except that we set $n_1 = O(\log n)$, and as before $n_2 = \frac{n}{n_1}$. This implies that in the commitment phase the prover encodes vectors of size $\frac{n}{O(\log n)}$. We will observe later that the choice of n_1 guarantees a proof complexity of $O(\lambda \log^2 n)$. Conversely, the evaluation protocol of BrakingBase employs proof composition to achieve poly-logarithmic verifier time and proof-complexity. Although the proof composition technique increases the prover's workload, it remains linear in relation to the size of the input polynomial's coefficient vector. The proof composition process is applied consistently for both the proximity test and the evaluation check. We begin by explaining it for the proximity test.

In BrakingBase, instead of sending $\mathbf{p}$ to the verifier, the prover sends a commitment to $\widetilde{(\mathbf{p}, \mathbf{p}')}$, computed using the BaseFold PCS, where $\mathbf{p}'$ is such that $(\mathbf{p}, \mathbf{p}') = \mathsf{Enc}(\mathbf{p})$.[3] The commitment protocol of Basefold runs in quasi-linear time in the size of the committed polynomial's coefficient vector. Given that the size of the vector $(\mathbf{p}, \mathbf{p}')$ is equal to $\rho \cdot n_2$, the evaluation prover takes time $O(n)$ to commit to $(\mathbf{p}, \mathbf{p}')$. Upon receiving the commitment the verifier needs to validate the following two conditions:

1. For all $i \in I$, $\langle \mathbf{r}, C(\cdot, i) \rangle = (\mathbf{p}, \mathbf{p}')(i)$, and
2. $(\mathbf{p}, \mathbf{p}')$ is a codeword.

Item 1 can be efficiently verified using a sum-check protocol, as described below, while the verification of Item 2 requires much more work and is detailed in the next paragraph. The verifier samples uniformly at random $s_i \in \mathbb{F}$, for all $i \in I$ and defines $\mathsf{mask} := \sum_{i \in I} s_i \cdot \mathbf{e}_i$, where $\mathbf{e}_i \in \mathbb{F}^{\rho \cdot n_2}$ is the i-th standard basis vector. Then, the prover convinces the verifier using a sum-check protocol that

$$\sum_{i \in I} s_i \langle \mathbf{r}, C(\cdot, i) \rangle = \sum_{\dot{\mathbf{b}} \in \{0,1\}^{\log(\rho n_2)}} \widetilde{\mathsf{mask}}(\dot{\mathbf{b}}) \widetilde{(\mathbf{p}, \mathbf{p}')}(\dot{\mathbf{b}}).$$

The LHS in the above expression is explicitly computed by the verifier using the columns of C, $C(\cdot, i)$ for all $i \in I$, sent by the prover, whereas the sum-check protocol reduces evaluating RHS to verifying the evaluation of $\widetilde{\mathsf{mask}}$ and $\widetilde{(\mathbf{p}, \mathbf{p}')}$ at a random point. The evaluation of $\widetilde{\mathsf{mask}}$ is computed by the verifier in $O(\lambda \log n)$ time, and the evaluation of $\widetilde{(\mathbf{p}, \mathbf{p}')}$ is verified using the evaluation protocol of Basefold. The prover in the evaluation protocol of Basefold operates linearly in relation to the size of the concerned polynomials coefficient vector, whereas the verifier time and proof-complexity remain poly-logarithmic. This ensures that the claimed evaluation of $\widetilde{(\mathbf{p}, \mathbf{p}')}$ can be validated with prover running in $O(\frac{n}{\log n})$ time, and the verifier time and proof-complexity equal to $O(\lambda \log^2 n)$. We remark here that since the columns of C for all column indices in I are sent to the verifier explicitly, we must set $n_1 = O(\log n)$ in order to achieve poly-logarithmic proof size.

To verify Item 2 we rely on the parity check matrix of the code. If $\mathcal{C}$ is a linear code mapping messages in $\mathbb{F}^{n_2}$ to codewords in $\mathbb{F}^{\rho \cdot n_2}$, then the parity check matrix of $\mathcal{C}$ is an $H \in \mathbb{F}^{\rho n_2 \times (\rho - 1) \cdot n_2}$ such that any $\mathbf{v} \in \mathbb{F}^{\rho \cdot n_2}$ is a codeword if and only if $\mathbf{v}H = \mathbf{0}$. Thus Item 2 can be verified by ensuring that the following polynomial in Equation 1 over $\dot{\mathbf{y}}$ variables is formally zero.[4]

$$\sum_{\dot{\mathbf{b}} \in \{0,1\}^{\log(\rho n_2)}} \widetilde{(\mathbf{p}, \mathbf{p}')}(\dot{\mathbf{b}}) \cdot \widetilde{H}(\dot{\mathbf{b}}, \dot{\mathbf{y}}) \tag{1}$$

It follows from the Schwartz-Zippel lemma that it is sufficient for the verifier to sample a random $\dot{\mathbf{u}}$ and the prover to demonstrate that

$$\sum_{\dot{\mathbf{b}} \in \{0,1\}^{\log(\rho n_2)}} \widetilde{(\mathbf{p}, \mathbf{p}')}(\dot{\mathbf{b}}) \cdot \widetilde{H}(\dot{\mathbf{b}}, \dot{\mathbf{u}}) = 0.$$

[3] Such a $\mathbf{p}'$ exists as the linear code of Brakedown is systematic.

[4] Here we require $\dot{\mathbf{b}} \in \lceil \log(\rho n_2) \rceil$ but avoid ceil-floor notation for clarity.

The above relation can be proved using a sum-check protocol which reduces the problem to checking the evaluation of $\widetilde{(\mathbf{p}, \mathbf{p}')}$ and $\widetilde{H}$ at a random point. As discussed in the above paragraph the claimed evaluation of $\widetilde{(\mathbf{p}, \mathbf{p}')}$ can be done using the evaluation protocol of Basefold. It remains to argue how the verifier can validate the claimed value of $\widetilde{H}$ at a random point. Since the code is generated in the setup phase of the PCS, a commitment to $\widetilde{H}$ can be given to the verifier as a part of its public parameters. $\widetilde{H}$ is a multilinear polynomial in $\log \rho \cdot n_2 + \log((\rho - 1) \cdot n_2)$ variables and opening an evaluation of $\widetilde{H}$ using an existing PCS (for example Basefold) would be costly in terms of prover runtime. Specifically, it would prevent achieving a linear-time evaluation prover for BrakingBase. We get around this problem by showing that H is a sparse matrix, that is, we show that it has $O\left(\frac{n}{\log n}\right)$ non-zero entries. Consequently we use Spark, the sparse polynomial commitment scheme from [Set20] to commit to $\widetilde{H}$. Spark simplifies the evaluation of $\widetilde{H}$ by reducing it to the evaluation of a constant number of polynomials, each with coefficient vectors of size $O\left(\frac{n}{\log n}\right)$, at a random point. These are then committed to and validated using the BaseFold evaluation procedure.

Thus far, we have described proof composition for the proximity testing step. Since the evaluation step closely resembles proximity testing, we can apply a similar composition strategy. Specifically, the prover commits to $\widetilde{(\mathbf{q}, \mathbf{q}')}$, where $\mathbf{q}'$ is such that $(\mathbf{q}, \mathbf{q}') = \mathsf{Enc}(\mathbf{q})$, and then proves the analogs of Items 1 and 2. Additionally, the prover must show that $\widetilde{q}(\alpha_{\log n_1}, \ldots, \alpha_\ell) = y$ where y is the claimed evaluation $f(\alpha_0, \ldots, \alpha_{\ell-1})$. As in the proximity test, each of these requirements reduces to evaluating $\widetilde{(\mathbf{q}, \mathbf{q}')}, \widetilde{H}$ at various random points.

Observe that validating Items 1 and 2 in both the proximity test and the evaluation check ultimately reduces to verifying the evaluations of multiple polynomials, each with coefficient vectors of size $O\left(\frac{n}{\log n}\right)$, at various random points. In the above exposition, we used the BaseFold evaluation protocol to verify each polynomial individually. Instead, we now apply a batched sum-check protocol (similar to [CBBZ23]) to reduce this task to verifying the evaluations of multiple polynomials at a single random point. We then employ the batched evaluation protocol of BaseFold to verify all of them simultaneously. This reduction allows us to concretely reduce the proof size.

1.2 Related Work

Comparison with Orion. [XZS22] introduced Orion, a PCS with $O(n)$ commitment and prover time, $O_\lambda(\log^2 n)$ verifier time and proof-complexity. Orion achieves these metrics by combining the Brakedown PCS with Virgo [ZXZS20], a general-purpose SNARK for arithmetic circuits. We now briefly compare our results with this PCS. First, since Virgo is not field-agnostic, neither is Orion. Second, the proof composition in Orion differs from the approach used in this work; we briefly discuss their composition method here. [XZS22] constructs an arithmetic circuit that takes $\mathbf{p}$ and $C(\cdot, i)$ for all $i \in I$ as inputs, computes $\mathsf{Enc}(\mathbf{p})$, and checks that for all $i \in I$, $\langle \mathbf{r}, C(\cdot, i) \rangle = \mathsf{Enc}(\mathbf{p})(i)$. They show that this circuit has size $O\left(\frac{n}{\log n}\right)$ and depth $O(\log n)$. The prover commits to the inputs of this circuit using FRI and verifies that the circuit outputs 1 using Virgo. Since this circuit must encode $\mathbf{p}$ and the linear code of Brakedown is random, it lacks a uniform wiring pattern. Because Virgo relies on the GKR protocol, [XZS22] also needs to use

Spark to commit to the circuit's wiring predicates. We believe that our approach, which avoids using a general-purpose SNARK and relies on Spark compiler for only opening a single sparse polynomials, is a more lightweight method for proof composition.[5]

Comparison with Blaze [BCF+24]. In a parallel work, [BCF+24] introduced Blaze, a PCS with $O(n)$ commitment and prover time, as well as $O(\lambda \log^2 n)$ verifier time and evaluation proof size, achieved by combining a Brakedown-style PCS with Basefold. Rather than using Brakedown's linear code to commit to the polynomial, Blaze employs a distinct linear-time computable code called the Repeat, Accumulate, Accumulate (RAA) code. RAA codes offer better relative distance and faster encoding than Brakedown's linear code, which allows the verifier to work with fewer columns $C(\cdot, i)$, resulting in smaller proofs. However, RAA codes are currently only defined over binary fields, i.e., fields with characteristic 2, meaning Blaze is not field-agnostic. Additionally, like Brakedown's linear code, RAA codes rely on a randomized construction. While a randomly sampled Brakedown code fails to achieve a "good" distance with a probability at most 2^{-100}, a randomly sampled RAA code has a higher probability of failing to achieve a "good" distance. For example, the specific RAA code instances in [BCF+24] have failure probabilities ranging from $2^{-25.4}$ to $2^{-92.2}$. Note that the soundness of the PCS depends on the code having a good distance.

Comparison with Ligero++ [BFH+20]. Ligero++ achieves a PCS with $O(\lambda \log^2 n)$ verifier time and evaluation proof size by composing the Ligero PCS with an inner product argument from [ZXZS20]. The Ligero PCS is similar to the Brakedown PCS, but it uses the Reed-Solomon (RS) code instead of the linear-time computable code used in Brakedown, simplifying the composition step. To commit to f, [BFH+20] write the evaluations of f in a matrix $A \in \mathbb{F}^{\frac{n}{\log n} \times \log n}$, derive C' from A such that for all i, $C'(i, \cdot) = \mathsf{Enc}(A(i, \cdot))$, and then obtain C from C' such that for all j, $C(\cdot, j) = \mathsf{Enc}(C'(\cdot, j))$. The commitment to f is constructed in two steps: first, build Merkle trees corresponding to each column of C, and then use the Merkle roots of the columns to build a final Merkle tree whose root serves as the commitment to f. Since the size of every row of C is $O(\log n)$, $\mathbf{p}$ will have size $O(\log n)$, allowing the verifier to compute $\mathsf{Enc}(\mathbf{p})$ independently. Here, $\mathbf{p}$ is the vector sent by the prover in the first step (see Section 1.1 for further details). Additionally, the Merkle roots corresponding to the columns of C serve as commitments to the univariate polynomial corresponding to the columns of C'. These commitments, along with the inner product argument from [ZXZS20], are used to check that for all $i \in I$, $\langle \mathbf{r}, C(\cdot, i) \rangle = \mathsf{Enc}(\mathbf{p})(i)$. This approach is only possible because of the use of the RS code. However, due to the reliance on the RS code, the Ligero++ PCS is not field-agnostic, and its commitment time is $O(n \log n)$.

2 Preliminaries

Notations. $\mathbb{N}$ denotes the set of natural numbers. Throughout we assume $n = 2^\ell$ for $\ell \in \mathbb{N}$. $\mathcal{H}$ is used to denote a hash function. Let $i \in [0, n-1]$, and let $(i_0, \ldots, i_{\ell-1})$ be its binary representation. Then given i, we use to-bits(i) to denote its binary representation, and given $(i_0, \ldots, i_{\ell-1})$, $\mathrm{val}(i_0, \ldots, i_{\ell-1}) = i$ to denote the decimal value of its corresponding binary string We use $\mathbb{F}$ to denote a finite field, $\mathbb{F}[X_0, \ldots, X_{n-1}]$ and $\mathbb{F}_{\leq 1}[X_0, \ldots, X_{n-1}]$ to denote the set of polynomials and

[5] A recent work [dHS24] identified a security vulnerability in Orion's proof composition step and proposed a method to secure it.

multilinear polynomials in n variables respectively. A multilinear polynomial f in ℓ variables is represented using its $n = 2^\ell$ Lagrange coefficients over the boolean hypercube. In this we view f as a vector in $\mathbb{F}^n$, whose i-th entry for $i \in [0, n-1]$ is equal to $f(\text{to-bits}(i))$. We use the notation $O_\lambda(\cdot)$ to hide $\text{poly}(\lambda)$ factors.

Definition 2.1. Let $\mathbf{v} \in \mathbb{F}^n$ be a vector for $n = 2^k$. Then the multilinear extension of $\mathbf{v}$, denoted by $\widetilde{\mathbf{v}}$, is the unique multilinear polynomial in $\mathbb{F}[x_0, \ldots, x_{k-1}]$ such that $\widetilde{\mathbf{v}}(b_0, \ldots, b_{k-1}) = v_{\sum_{i=0}^{k-1} 2^i b_i}$ for all $(b_0, \ldots, b_{k-1}) \in \{0,1\}^k$.

Matrices appearing in our PCS scheme are denoted by capital letters A, B, C. Further, for a matrix $A \in \mathbb{F}^{n \times m}$ and $i \in [n], j \in [m]$, $A(i, \cdot)$ denotes its i-th row, $A(\cdot, j)$ denotes its j-th column, and $A(i, j)$ denotes its (i, j)-th entry. We shall use bold lowercase letter to denote vectors. For a vector $\mathbf{v}$, we shall denote its i-th entry by $\mathbf{v}_i$ or v_i. We shall denote vectors of variables as $\dot{\mathbf{x}}, \dot{\mathbf{y}}$, etc. and points in the boolean hypercube as $\dot{\mathbf{b}}$ etc. For any $\ell \in \mathbb{N}$, we define a multilinear polynomial $\widetilde{eq}$ such that $\widetilde{eq}(\dot{\mathbf{a}}, \dot{\mathbf{b}}) = 1$ for any $\dot{\mathbf{a}}, \dot{\mathbf{b}} \in \{0,1\}^\ell$ if and only if $\dot{\mathbf{a}} = \dot{\mathbf{b}}$.

Linear codes. A linear code of size n and dimension k is a k dimensional sub-space $\mathcal{C}$ of $\mathbb{F}^n$. Let $\mathbf{v}_1, \ldots, \mathbf{v}_k \in \mathcal{C}$ be a basis of $\mathcal{C}$ and define $E \in \mathbb{F}^{k \times n}$ to be the matrix with rows $\mathbf{v}_1, \ldots, \mathbf{v}_k$. Then E is said to be the generating matrix of $\mathcal{C}$; it maps messages in $\mathbb{F}^k$ to codewords in $\mathcal{C}$. We shall use $\mathsf{Enc}(\mathbf{u})$ to denote the codeword corresponding to the message $\mathbf{u} \in \mathbb{F}^k$. The rate of $\mathcal{C}$ is the ratio $\frac{k}{n}$; throughout this article we shall use ρ to denote the inverse of the rate. For any two $\mathbf{u}, \mathbf{v} \in \mathbb{F}^n$, let $\Delta(\mathbf{u}, \mathbf{v})$ denote the Hamming distance between $\mathbf{u}$ and $\mathbf{v}$; i.e. the number of coordinates in which $\mathbf{u}$ and $\mathbf{v}$ differ. Then the distance of $\mathcal{C}$ is defined to be $d := \min_{\mathbf{u} \neq \mathbf{v} \in \mathcal{C}} \Delta(\mathbf{u}, \mathbf{v})$. Observe that since $\mathcal{C}$ is a vector space, $d = \min_{\mathbf{v} \neq \mathbf{0} \in \mathcal{C}} \{i : v_i \neq 0\}$. We shall use $\delta := \frac{d}{n}$ to denote the relative distance of $\mathcal{C}$. A code of size n, dimension k, and distance d is generally referred to as an (n, k, d) code. An (n, k, d) code $\mathcal{C}$ is said to be systematic if for all $\mathbf{u} \in \mathbb{F}^k$, there exists a $\mathbf{u}' \in \mathbb{F}^{n-k}$ such that $\mathsf{Enc}(\mathbf{u}) = (\mathbf{u}, \mathbf{u}')$.

Extend $\{\mathbf{v}_1, \ldots, \mathbf{v}_k\}$ to a basis $\{\mathbf{v}_1, \ldots, \mathbf{v}_n\}$ of $\mathbb{F}^n$. Define $H \in \mathbb{F}^{(n-k) \times n}$ to be the matrix whose rows are $\mathbf{v}_{k+1}, \ldots, \mathbf{v}_n$. Observe that a $\mathbf{v} \in \mathbb{F}^n$ is in $\mathcal{C}$ if and only if $\mathbf{v}H = \mathbf{0}$. H is called the parity check matrix of $\mathcal{C}$.

We now mention some of the lemmas that shall be used crucially to prove the knowledge soundness of the PCS.

Lemma 2.2 (Lemma 12.1 of [Tha22])**.** *Let X, Y be jointly distributed random variables and $\mathcal{E}(X, Y)$ be an event such that $\Pr_{X,Y}[\mathcal{E}(X, Y)] \geq \epsilon$. Let $S := \left\{x : \Pr_Y[\mathcal{E}(X, Y) | X = x] \geq \frac{\epsilon}{2}\right\}$. Then $\Pr_X[X \in S] \geq \frac{\epsilon}{2}$.*

Lemma 2.3 (Lemma 9 of [ZCF24])**.** *Let $\Phi : \mathcal{M}^* \to \{0,1\}$ be any predicate such that for any $(m_1, \ldots, m_i) \in \mathcal{M}^i$ where $\Phi(m_1, \ldots, m_i) = 1$,*

$$\Pr_{m_{i+1} \in_R \mathcal{M}}[\Phi(m_1, \ldots, m_{i+1}) = 1] \geq 1 - \mathsf{negl}(\lambda).$$

Let $N = \text{poly}(\lambda)$. For any $\epsilon > 0$, there exists an extractor Ext which runs in time $T \in O\left(\frac{\lambda}{\epsilon}\right)$, and given oracle access to any algorithm A where $\Pr_{m \in_R \mathcal{M}}[A(m) = 1] \geq \epsilon$, the following holds:

$$\Pr\left[\begin{matrix}\Phi(m_1, \ldots, m_N) = 1 \wedge \\ A(m_i) = 1 \; \forall i \in [N]\end{matrix} \;\middle|\; (m_1, \ldots, m_N) \leftarrow \mathsf{Ext}^A\right] \geq 1 - T \cdot \mathsf{negl}(\lambda).$$

Lemma 2.4 ([AHIV17]). *Let $\mathcal{C} \subset \mathbb{F}^n$ be any linear code with relative distance δ and $\mathbf{v}_1, \ldots, \mathbf{v}_m \in \mathbb{F}^m$ be such that their closest codewords in $\mathcal{C}$ are $\mathbf{u}_1, \ldots, \mathbf{u}_m \in \mathbb{F}^m$. Further suppose that for $E := \left\{j \in [n] : \exists i \in n \text{ s.t. } v_{i,j} \neq u_{i,j}\right\}, |E| \leq \frac{\delta}{3}n$. Then*

$$\Pr_{r_1,\ldots,r_m \in_R \mathbb{F}}\left[\sum_{i\in[m]} r_i \mathbf{v}_i \text{ has distance} \geq |E| \text{ from any codeword in } \mathcal{C}\right] \geq 1 - \frac{|E|+1}{|\mathbb{F}|}.$$

2.1 Interactive Oracle Proofs and Arguments of Knowledge

Interactive Proofs for a relation $\mathcal{R}$ enables any prover to convince a verifier that it possesses a (private) witness w corresponding to publicly known x such that $(x, w) \in \mathcal{R}$. Polynomial Interactive Oracle Proofs (IOPs) are special interactive proofs where the prover only sends polynomial oracles to the verifier, that is the messages of the prover constitute of bounded degree polynomials, and the verifier only queries them at desired points. Given a pair of probabilistic interactive algorithms P, V, we denote $\langle P, V\rangle(x; w)$ as the output of the interaction of P and V, where x is publicly known, and w is privately known to P. Let $\mathcal{R} = \{(x, w)\}$ be a relation and $\mathcal{L} = \{x \mid \exists w \text{ such that } (x, w) \in \mathcal{R}\}$ be the language corresponding to it. Next, we formally define Arguments of Knowledge (AoK).

Definition 2.5 (Succinct Argument of Knowledge)**.** An Argument of Knowledge (AoK) for a relation $\mathcal{R}$ constitutes of a probabilistic algorithm $\mathsf{Gen}(1^\lambda)$ that takes as input the security parameter λ and outputs public parameters pp, together with a pair of probabilistic algorithms $\langle P, V\rangle$ satisfying:

1. Completeness: For all $\lambda \in \mathbb{N}$, $\mathsf{pp} \leftarrow \mathsf{Gen}(1^\lambda)$, and for every $(x, w) \in \mathcal{R}$ the following holds:

$$\Pr\left\{\langle P, V\rangle(\mathsf{pp}, x; w) = 1\right\} = 1$$

2. Knowledge-Soundness: For any pair PPT algorithms P_1, P_2 there exists an PPT algorithm Ext such that the following holds:

$$\Pr\left\{\begin{matrix}(x, w) \notin \mathcal{R} \\ \langle P_2, V\rangle(\mathsf{pp}, x; st) = 1\end{matrix} \;\middle|\; \begin{matrix}\mathsf{pp} \leftarrow \mathsf{Gen}(1^\lambda),\ (x, \mathsf{st}) \leftarrow P_1(\mathsf{pp}, 1^\lambda) \\ w \leftarrow \mathrm{Ext}^{P_2}(\mathsf{pp},)\end{matrix}\right\} \quad = \quad \mathsf{negl}(\lambda)$$

3. The AoK is succinct if the communication complexity between the prover and the verifier, and the verifier run-time is poly-logarithmic in the size of the circuit corresponding to $\mathcal{R}$.

Remark 1: The notion of succinctness we consider is stricter than previous works like [AHIV17, GLS+23], where the requirement is only sub-linear. We work with the above definition as the AoK's we propose in this work has poly-logarithmic proof-size and verifier tun-time.

Remark 2: It is well-known (see [BCS16]) how to construct AoK's from polynomial IOPs using Merkle commitments. Our PCS, BrakingBase is in principle an IOP from which an AoK is obtained using these standard techniques.

Remark 3: An AoK is public coin if the random bits sampled by the verifier is public. Succinct Non-interactive Argument of Knowledge (SNARK) is a one-round public-coin AoK, that is, the

prover sends the message to the verifier and consequently verifier outputs either 1 or 0. The protocols we design in this work are public-coin interactive arguments rendered into a SNARK using a Fiat-Shamir transform in the Random Oracle Model (ROM).

Remark 4: We do not consider zero-knowledge in this work, as we believe it can be attained using minor adaptations and standard techniques to our protocols.

2.2 Polynomial Commitment Scheme

A Polynomial Commitment Scheme (PCS) for multilinear polynomials is a tuple of four protocols $\mathsf{pc} = (\mathsf{Gen}, \mathsf{Commit}, \mathsf{Open}, \mathsf{Eval})$ defined as follows:

1. $\mathsf{pp} \leftarrow \mathsf{Gen}(\lambda, \ell)$: Gen takes as input the security parameter and the number of variables in a multilinear polynomial and outputs the public parameters pp.
2. $\mu \leftarrow \mathsf{Commit}(\mathsf{pp}, f)$: Commit takes as input the the public parameters pp and a multilinear polynomial $f \in \mathbb{F}_{\leq 1}[X_0, \ldots, X_{n-1}]$ and outputs a commitment to it, denoted μ.
3. $b \leftarrow \mathsf{Open}(\mathsf{pp}, f, \mu, u)$: Open takes as input the the public parameters pp, a multilinear polynomial $f \in \mathbb{F}_{\leq 1}[X_0, \ldots, X_{n-1}]$, a commitment μ and an opening hint u, and outputs 1 if μ is a commitment to f and 0 otherwise.
4. $b \leftarrow \mathsf{Eval} \langle P, V \rangle (\mathsf{pp}, \mu, \dot{\boldsymbol{\alpha}}, y; f)$: Eval is an interactive protocol between a probabilistic prover P and a verifier V. The public inputs are the public parameters pp, a commitment μ, a point $\dot{\boldsymbol{\alpha}} \in \mathbb{F}^\ell$ and a value $y \in \mathbb{F}$. The prover additionally receives the Lagrange coefficients of the multilinear polynomial $f \in \mathbb{F}_{\leq 1}[X_0, \ldots, X_{n-1}]$. P attempts to convince V that $f(\dot{\boldsymbol{\alpha}}) = y$, and V outputs 1 if it is convinced and 0 otherwise.

$\mathsf{pc} = (\mathsf{Gen}, \mathsf{Commit}, \mathsf{Open}, \mathsf{Eval})$ must additionally satisfy binding, completeness, and knowledge-soundness as given below.

Definition 2.6 (Binding)**.** For any PPT adversary $\mathcal{A}$, and $\ell \geq 1$ the following holds:

$$\Pr\left\{\begin{matrix} \mathsf{pp} \leftarrow \mathsf{Gen}(1^\lambda, \ell),\ (\mu, f_0, f_1) \leftarrow \mathcal{A}(\mathsf{pp}, \ell) \\ \mathsf{Open}(\mathsf{pp}, \mu, f_0) = b_0,\ \mathsf{Open}(\mathsf{pp}, \mu, f_1) = 1, \quad \text{and} \quad f_0 \neq f_1 \end{matrix}\right\} \quad = \quad \mathsf{negl}(\lambda)$$

Definition 2.7 (Completeness)**.** For any $f \in \mathbb{F}_{\leq 1}[X_0, \ldots, X_{n-1}]$ the following holds:

$$\Pr\left\{\begin{matrix} \mathsf{pp} \leftarrow \mathsf{Gen}(1^\lambda, \ell),\ \mu \leftarrow \mathsf{Commit}(\mathsf{pp}, f),\ 1 \leftarrow \mathsf{Open}(\mathsf{pp}, \mu, f), \\ \mathsf{Eval} \langle P, V \rangle (\mathsf{pp}, \mu, \dot{\boldsymbol{\alpha}}, y; f) = 1 \quad \text{and} \quad f(\dot{\boldsymbol{\alpha}}) = y \end{matrix}\right\} = 1$$

Definition 2.8 (Knowledge-Soundness)**.** Eval is an AoK for the following relation given $\mathsf{pp} \leftarrow \mathsf{Gen}(\lambda, \ell)$:

$$\mathcal{R} = \left\{ (\mu, \dot{\boldsymbol{\alpha}}, y; f) \mid f \in \mathbb{F}_{\leq 1}[X_0, \ldots, X_{n-1}], \dot{\boldsymbol{\alpha}} \in \mathbb{F}^\ell, y \in \mathbb{F}, f(\dot{\boldsymbol{\alpha}}) = y \right\}$$

We additionally require the PCS to be succinct as defined below.

Definition 2.9 (Succinctness)**.** In a succinct PCS the size of the commitment μ is $\mathrm{poly}(\lambda) \cdot O(1)$, and Eval is a succinct AoK for the relation in Definition 2.8.

2.3 BaseFold PCS

Recently [ZCF24] proposed a PCS for multilinear polynomials, BaseFold, which can be thought of a generalization of the FRI PCS. They define foldable linear codes which generalize the Reed-Solomon code used by FRI and then show how to obtain a PCS from any foldable linear code with a 'good' distance. We now briefly describe foldable linear codes and their PCS.

Let $\mathcal{C}_0$ be an (n_0, k_0, d_0) linear code. A foldable linear code $\mathcal{C}_\ell$ with $\mathcal{C}_0$ as the base code is defined recursively as follows: Let E_0 be the generator matrix for $\mathcal{C}_0$. For all $i \in [\ell]$, let $n_i = 2n_{i-1}$ and $k_i = 2k_{i-1}$. Let $T_i, T_i' \in \mathbb{F}^{n_{i-1} \times n_{i-1}}$ be some diagonal matrices. Then $\mathcal{C}_i$ is defined to be the code with size n_i and dimension k_i with the generator matrix

$$E_i := \begin{bmatrix} E_{i-1} & E_{i-1} \\ E_{i-1}T_i & -E_{i-1}T_i. \end{bmatrix}.$$

[ZCF24] show that if for all $i \in [\ell]$, T_i, T_i' are picked independently and uniformly at random from $\mathbb{F}$, then $\mathcal{C}_\ell$ has a good distance (provided that $|\mathbb{F}|$ is large). Notice that the Reed-Solomon code is also a foldable code. Further using a recursive encoding procedure similar to that of the Reed-Solomon code, a codeword can be computed in time $O(n_\ell \log n_\ell)$. For ease of exposition, we assume for the rest of this section that $k_0 = 1$.

Let f be a multilinear polynomial with degree ℓ, let $\mathbf{f}$ be its coefficient vector, and let $\mathbf{c}_\ell$ denote codeword in $\mathcal{C}_\ell$ corresponding to $\mathbf{f}$. Then commitment to f is the root of the Merkle tree with entries of $\mathbf{c}_\ell$ as leaves. Let us denote this commitment by c_ℓ. Let $\dot{\boldsymbol{\alpha}} \in \mathbb{F}^\ell$ and suppose that the prover wants to prove that $f(\dot{\boldsymbol{\alpha}}) = y$. The evaluation procedure of BaseFold can be divided into two phases, a commitment phase and a query phase. The commitment phase is an ℓ round protocol with a folding process and a sum-check running in parallel. We first describe the folding process and then the sum-check.

In the first round, the verifier sends a random challenge $r_{\ell-1}$ to the prover. Let $\mathbf{c}_\ell^{(1)}$ and $\mathbf{c}_\ell^{(2)}$ denote the first and second half of $\mathbf{c}_\ell$ respectively. Note that because of the foldable nature of the code, there exist $\mathbf{c}', \mathbf{c}'' \in \mathcal{C}_{\ell-1}$ such that $\mathbf{c}_\ell^{(1)} = \mathbf{c}' + T_\ell \mathbf{c}''$ and $\mathbf{c}_\ell^{(2)} = \mathbf{c}' + T_\ell' \mathbf{c}''$. Also, $\mathbf{c}', \mathbf{c}''$ can be easily computed from $\mathbf{c}_\ell, T_\ell, T_\ell'$ in $O(n_\ell)$ by solving $n_{\ell-1}$ systems of linear equations in two variables. The prover computes $\mathbf{c}', \mathbf{c}''$, defines $\mathbf{c}_{\ell-1} := \mathbf{c}' + r_{\ell-1}$, and sends $\mathsf{c}_{\ell-1}$, the root of a Merkle tree with entries of $\mathbf{c}_{\ell-1}$ as leaves to the verifier. In general, in the i-th round the above process is repeated but with the random challenge $r_{\ell-i}$ and with $\mathbf{c}_{\ell-i+1}$ in place of $\mathbf{c}_\ell$. Observe that if the prover were honest, then $\mathbf{c}_0$ is the codeword in $\mathcal{C}_0$ corresponding to $f(\dot{\mathbf{r}})$ where $\dot{\mathbf{r}} = (r_0, \ldots, r_{\ell-1})$.

Now $f(\dot{\boldsymbol{\alpha}}) = \sum_{\dot{\mathbf{b}} \in \{0,1\}^\ell} \widetilde{eq}(\dot{\boldsymbol{\alpha}}, \dot{\mathbf{b}}) f(\dot{\mathbf{b}})$. For a moment, let us assume that the prover computed $\mathbf{c}_{\ell-1}, \ldots, \mathbf{c}_0$ honestly. Then the prover can prove that $f(\dot{\boldsymbol{\alpha}}) = y$ as follows: it proves that $y = \sum_{\dot{\mathbf{b}} \in \{0,1\}^\ell} \widetilde{eq}(\dot{\boldsymbol{\alpha}}, \dot{\mathbf{b}}) f(\dot{\mathbf{b}})$ using a sum-check protocol. If this sum-check is run in parallel with the folding process described above and with the same randomness, then after the last round of sum-check, the verifier will have $y' := \widetilde{eq}(\dot{\boldsymbol{\alpha}}, \dot{\mathbf{r}}) \cdot f(\dot{\mathbf{r}})$. It can then verify that $\mathbf{c}_0$ is indeed $\frac{y'}{\widetilde{eq}(\dot{\boldsymbol{\alpha}}, \dot{\mathbf{r}})}$. If as assumed $\mathbf{c}_{\ell-1}, \ldots, \mathbf{c}_0$ are computed honestly, then this convinces the verifier that $f(\dot{\boldsymbol{\alpha}}) = y$. It is then verified in the query phase that $\mathbf{c}_{\ell-1}, \ldots, \mathbf{c}_0$ are indeed computed honestly. The query phase

of BaseFold is the same as that of the FRI PCS; for all $i \in [\ell]$, the verifier queries random coordinates in $\mathbf{c}_i$ and verifies that these are computed honestly from the two appropriate coordinates of $\mathbf{c}_{i+1}$.

3 A PCS with Linear-Time Prover and Poly-Logarithmic Verifier

In this section, we present a novel PCS BrakingBase. The commitment and evaluation prover of BrakingBase takes time linear in the size of the coefficient vector of the witness polynomial, whereas the communication complexity and the verifier run-time of BrakingBase is poly-logarithmic in the the size of the coefficient vector of the witness polynomial. Hence, BrakingBase is a succinct PCS. We describe BrakingBase for multilinear polynomials but it can be easily adapted to the case of univariate polynomials. In Section 3.1, we present the generator algorithm that given the security parameter outputs the required public parameters, and in Section 3.2, we present the commitment and open algorithms. Finally, in Section 3.3, we present the evaluation protocol for BrakingBase. In conclusion, in this section we present the BrakingBase succinct PCS scheme, which we formally note in the theorem below, and prove it in Appendix A.

Theorem 3.1. *The tuple of protocols* (BrakingBase.Gen, BrakingBase.Commit, BrakingBase.Open, BrakingBase.Eval) *presented in Protocols 1, 2, 3, 4 together form a succinct PCS. For a multilinear polynomial in k variables and $n = 2^k$, the commitment time and the time complexity of the evaluation prover of the PCS are $O(n)$, while the verifier complexity and proof size are $O(\lambda \log^2 n)$.*

3.1 The Pre-processing Phase

Algorithm 1 generates the public parameters required in BrakingBase. Let $\mathcal{H} \leftarrow \mathcal{G}(\lambda)$ be a hash function with λ bits of security . Algorithm 1 takes as input ℓ the bound on the number of variables in the set of multilinear polynomials. It outputs public parameters $\mathsf{pp} = (\mathsf{pp}_{\mathcal{P}}, \mathsf{pp}_{\mathcal{V}})$. $\mathsf{pp}_{\mathcal{P}}$ is given to the prover and $\mathsf{pp}_{\mathcal{V}}$ is given to the verifier. In Step 1, $E_0 \in \mathbb{F}^{\frac{n}{\ell} \times \rho_1 \frac{n}{\ell}}$ is constructed recursively by sampling adjacency matrices of degree d bipartite expander graphs of progressively smaller sizes. Here, $n = 2^\ell$ and ρ_1 is the rate of the linear code. Further, let δ_1 be the relative distance of this linear code. It is shown in [GLS+23] that δ_1 is a constant as long as the sampled bipartite graphs are 'good' expander graphs (see Section B for details).

In Steps 2-6, the algorithm constructs encoding matrices for the Basefold code, following a recursive approach with rate ρ_2. The smallest encoder matrix $E_1 \in \mathbb{F}^{2 \times \rho_2}$ is generated first, where each entry is sampled uniformly at random for a message length of 2. Then, for each $i \in \{2, \ldots, \log \frac{cn}{\ell}\}$ (with c an absolute constant), the encoding matrix E_i is constructed using $n_i := 2^i$ and $k_i := \rho_2 n_i$ as follows:

1. A random matrix $T_i \in \mathbb{F}^{k_i \times k_i}$ is sampled.
2. The encoding matrix E_i is set by expanding E_{i-1} as follows:

$$E_i := \begin{bmatrix} E_{i-1} & E_{i-1} \\ E_{i-1}T_i & -E_{i-1}T_i. \end{bmatrix}.$$

The Basefold evaluation procedure requires the verifier to access random entries of T_i for $i \in \{2, \ldots, \log \frac{cn}{\ell}\}$. To facilitate this, Step 5 constructs a Merkle tree mt_i using the hash function $\mathcal{H}$. The leaves of mt_i are set to the hashes of the non-zero entries of T_i. The root of mt_i, denoted tc_{T_i} is the commitment to T_i.

In Step 7, the algorithm computes commitments to certain polynomials that enable evaluating the MLE of the parity check matrix corresponding to E_0 at a random point. We denote the parity check matrix as H and elaborate on this next. The Eval protocol of BrakingBase requires the verifier to check a linear relation between two committed vectors specified by H, and this eventually requires the prover to open the MLE $\widetilde{H}$ at a random point. We show in Appendix B that H is sparse; in particular, it has at most $\frac{cn}{\ell}$ non-zero entries, where $c \leq 32$ is a constant. Thus we can use the sparse polynomial commitment scheme from Spartan [Set20] to commit to the parity check matrices. The sparse PCS enables reducing the problem of evaluating a multilinear polynomial in $2 \log \frac{n}{\ell}$ variables to evaluating 9 polynomials in $\log \frac{cn}{\ell}$ variables (see Section C for details of the sparse PCS). Among these 9 polynomials, 7 polynomials are only dependent on the entries of H. Hence, these 7 polynomials, denoted $\widetilde{H}_{\text{row}}$, $\widetilde{H}_{\text{col}}$, $\widetilde{H}_{\text{val}}$, $\widetilde{H}_{\text{read\_ts,row}}$, $\widetilde{H}_{\text{final\_ts,row}}$, $\widetilde{H}_{\text{read\_ts, col}}$, $\widetilde{H}_{\text{final\_ts,col}}$ are committed to using the Basefold.Commit algorithm. We note here that the Basefold.Commit uses $\mathcal{H}$ to execute its hash based commitments (details in Section 2.3).

Algorithm 1 BrakingBase.Pre-processing

Input: ℓ the bound on the number of variables in the set of multilinear polynomials.

1. Let $n = 2^\ell$. Compute $E_0 \in \mathbb{F}^{\frac{n}{\ell} \times \rho_1 \frac{n}{\ell}}$ the encoding matrix of Brakedown's [GLS+23] linear code for messages of length $\frac{n}{\ell}$.
   /* Generate encoding matrices for Basefold's [ZCF24] code for appropriate lengths. */
2. Sample a random $E_1 \in \mathbb{F}^{2 \times 2\rho_2}$.
3. **for** $i \in \{2, \ldots, \log \frac{cn}{\ell}\}$ **do**
4. Let $n_i := 2^i$, $k_i := \rho_2 n_i$. Sample a random diagonal matrix $T_i \in \mathbb{F}^{k_i \times k_i}$. Define

$$E_i := \begin{bmatrix} E_{i-1} & E_{i-1} \\ E_{i-1}T_i & -E_{i-1}T_i. \end{bmatrix}$$

5. Use hash function $\mathcal{H}$ to build a Merkle tree mt_i constituting of the diagonal elements of T_i as its hashed leaves, and denote tc_{T_i} as its root.
6. **end for**
7. Let $H \in \mathbb{F}^{\rho_1 \frac{n}{\ell} \times (\rho_1 - 1)\frac{n}{\ell}}$ be the parity check matrix corresponding to E_0. Use the the sparse polynomial commitment scheme spark from [Set20] to commit to $\widetilde{H}$ (in Section B we show that H has sparsity at most $\frac{cn}{\ell}$). spark uses a mulitlinear PCS as blackbox, and in our case requires preprocessed commitments to 7 multilinear polynomials of length at most $\frac{cn}{\ell}$ (see Section C for details). These 7 polynomials are denoted $\widetilde{H}_{\text{row}}$, $\widetilde{H}_{\text{col}}$, $\widetilde{H}_{\text{val}}$, $\widetilde{H}_{\text{read\_ts,row}}$, $\widetilde{H}_{\text{final\_ts,row}}$, $\widetilde{H}_{\text{read\_ts, col}}$, $\widetilde{H}_{\text{final\_ts,col}}$ and let tc_{row}, tc_{col}, tc_{val}, $\mathsf{tc}_{\text{read\_ts,row}}$, $\mathsf{tc}_{\text{final\_ts,row}}$, $\mathsf{tc}_{\text{read\_ts, col}}$, $\mathsf{tc}_{\text{final\_ts,col}}$ be their commitments computed using Basefold.
8. Let $\mathsf{pp}_{\mathcal{P}} = \{E_0, \ldots, E_{\log \frac{cn}{\ell}}, H\}$, and $\mathsf{pp}_{\mathcal{V}} = \{\mathsf{tc}_{T_1}, \ldots, \mathsf{tc}_{T_{\log \frac{cn}{\ell}}}, \mathsf{tc}_{\text{row}}, \mathsf{tc}_{\text{col}}, \mathsf{tc}_{\text{val}}, \mathsf{tc}_{\text{read\_ts,row}}, \mathsf{tc}_{\text{final\_ts,row}}, \mathsf{tc}_{\text{read\_ts, col}}, \mathsf{tc}_{\text{final\_ts,col}}\}$. Output $\mathsf{pp} = (\mathsf{pp}_{\mathcal{P}}, \mathsf{pp}_{\mathcal{V}})$.

3.2 The Commit and Open Algorithms

Algorithm 2 commits to a polynomial f in ℓ variables. It takes as input the public parameters $\mathsf{pp}_{\mathcal{P}}$, and the Lagrange coefficients of f, and outputs a commitment c to f. We assume that f is itself a vector in $\mathbb{F}^n$ constituting of its Lagrange coefficients ordered as given in Section 2. In Step 1, the algorithm computes a matrix $A \in \mathbb{F}^{\ell \times \frac{n}{\ell}}$ using the Lagrange coefficients of f; the Lagrange coefficients populate A in row-major fashion, that is, the first $\frac{n}{\ell}$ entries constitute the first row, the next $\frac{n}{\ell}$ entries constitute the second row and so on. In Step 2, the matrix $C \in \mathbb{F}^{\frac{n}{\ell} \times \rho_1 \frac{n}{\ell}}$ is computed by setting $C(i, \cdot) = A(i, \cdot) \cdot E_0$ for all $0 \leq i < \ell$. Finally in Step 3, the the columns of C are hashed into the leaves of the Merkle tree mt_C using $\mathcal{H}$, and the root of mt_C, denoted c is the commitment to f that is given as output by the algorithm.

Algorithm 2 BrakingBase.Commit

$\mathcal{P}$'s input: $\mathsf{pp}_{\mathcal{P}}$ the public parameters, and f a multilinear polynomial in ℓ variables.
Output: Commitment tc to f.

1. Compute $A \in \mathbb{F}^{\ell \times \frac{n}{\ell}}$ by arranging the Lagrange coefficients of f in row-major fashion.
2. Compute $C \in \mathbb{F}^{\frac{n}{\ell} \times \rho_1 \frac{n}{\ell}}$ such that $\forall i \in \frac{n}{\ell}, C(i, \cdot) := A(i, \cdot) \cdot E_0$. Here E_0 is part of $\mathsf{pp}_{\mathcal{P}}$, and is as described in Section 3.1.
3. Compute the Merkle tree whose leaves are hashes of the columns of C, and output its root c as the commitment f.

Algorithm 3 is the opening protocol for BrakingBase. It takes as input $\mathsf{pp}_{\mathcal{P}}$ the public parameters, f a multilinear polynomial in ℓ variables, c the commitment to f, and $\mathfrak{p}$ a set of distinct Merkle paths against c, and outputs 1 if it accepts, and 0 otherwise. Steps 1-2 of the algorithm are the same as BrakingBase.Commit. In Step 3, the algorithm checks that paths in $\mathfrak{p}$ are consistent with c. The sets J and J' are as defined in Step 4. Let C' be the matrix such that for $j \in J$, $C'(\cdot, j) = p_j$, and for $j \notin J$, $C'(\cdot, j) = \mathbf{0}$. It is easily seen that the check in Step 4 ensures that $\Delta(C, C') \leq \frac{\delta_1}{2}$, where δ_1 is the distance of the linear code corresponding to E_0.

Algorithm 3 BrakingBase.Open

$\mathcal{P}$'s input: $\mathsf{pp}_{\mathcal{P}}$ the public parameters, f a multilinear polynomial in ℓ variables, c the commitment to f, and $\mathfrak{p}$ a set of distinct Merkle paths against c.
Output: 1 if accept and 0 if reject.

1. Compute $A \in \mathbb{F}^{\ell \times \frac{n}{\ell}}$ by arranging the Lagrange coefficients of f in row-major fashion.
2. Compute $C \in \mathbb{F}^{\frac{n}{\ell} \times \rho_1 \frac{n}{\ell}}$ such that $\forall i \in \frac{n}{\ell}, C(i, \cdot) := A(i, \cdot) \cdot E_0$.
3. Check that the Merkle paths in $\mathfrak{p}$ are consistent with tc.
4. Let $J \subset \{0, \ldots, \rho_1 \cdot n - 1\}$ be the set of of indices corresponding to which we have paths in $\mathfrak{p}$. Hence, for every $j \in J$, we have $p_j \in \mathfrak{p}$ such that $p_{j,\text{col-data}}$ is the data hashed to obtain the leaf in p_j. Now, let $J' = \left\{ j \mid j \in J, p_{j,\text{col-data}} = C(\cdot, j) \right\}$. If $|J'| \geq \frac{\delta_1}{2}$ then output 1 and 0 otherwise.

3.3 The evaluation phase

Protocol 4 states the evaluation protocol of BrakingBase. The public input of the protocol is $(\mathsf{pp}, \mathsf{c}, \dot{\boldsymbol{\alpha}}, y)$, where $(\mathsf{pp}_{\mathcal{P}}, \mathsf{pp}_{\mathcal{V}})$ are the public parameters of the protocol generated by BrakingBase.Gen, c is the commitment to a polynomial $f \in \mathbb{F}_{\leq 1}[X_0, \ldots, X_{\ell-1}]$, $\dot{\boldsymbol{\alpha}} \in \mathbb{F}^{\ell}$ and $y \in \mathbb{F}$. The public input is given as input to both the prover $\mathcal{P}$ and the verifier $\mathcal{V}$; $\mathcal{P}$ and $\mathcal{V}$ receive $\mathsf{pp}_{\mathcal{P}}$ and $\mathsf{pp}_{\mathcal{V}}$ parts of the pp respectively. The prover additionally receives the Lagrange coefficients of f. As part of Theorem 3.1, we show that Protocol 4 is a succinct AoK for the following relation:

$$\mathcal{R} = \{(\mathsf{c}, \dot{\boldsymbol{\alpha}}, y; f) \mid f \in \mathbb{F}_{\leq 1}[X_0, \ldots, X_{n-1}], \mathsf{pp} \leftarrow \mathsf{BrakingBase.Gen}(\lambda),$$
$$\mathsf{c} \leftarrow \mathsf{BrakingBase.Commit}(\mathsf{pp}, f), \dot{\boldsymbol{\alpha}} \in \mathbb{F}^{\ell}, y \in \mathbb{F}, f(\dot{\boldsymbol{\alpha}}) = y\}.$$

The evaluation protocol conceptually has two parts: proximity test, and evaluation test. Both of them are very similar and are executed in parallel. The goal of proximity check is to ensure that all the rows of C (see Algorithm 2, Step 2) are close to codewords, and the evaluation test is the actual evaluation protocol. The proximity test can be avoided if $\dot{\boldsymbol{\alpha}}$ is sampled uniformly at random from $\mathbb{F}^{\ell}$ [DP24]. Next, we explain Protocol 4. We note that Protocol 4 and its explanation below use objects defined in Sections 3.1, and 3.2.

Steps 1-3: In Step 1, $\mathcal{V}$ samples an $\mathbf{r} \in \mathbb{F}^{\ell}$ uniformly at random and sends it $\mathcal{P}$. In Step 2 $\mathcal{P}$ computes $\boldsymbol{\alpha} = \otimes_{i=0}^{\log \ell - 1}(1 - \dot{\alpha}_{\ell-i},\ \dot{\alpha}_{\ell-i})$. At this step, $\mathcal{P}$ also computes $\mathbf{p} := \mathbf{r} \cdot A \in \mathbb{F}^{\frac{n}{\ell}}$, $\mathbf{q} := \boldsymbol{\alpha} \cdot A \in \mathbb{F}^{\frac{n}{\ell}}$ and $\mathbf{p}', \mathbf{q}' \in \mathbb{F}^{\rho_1 \cdot (\frac{n}{\ell} - 1)}$ such that $(\mathbf{p}, \mathbf{p}') = \mathbf{p} \cdot E_0$, $(\mathbf{q}, \mathbf{q}') = \mathbf{q} \cdot E_0$. Since the linear code corresponding to E_0 is systematic, such $\mathbf{p}'$ and $\mathbf{q}'$ exist. In Step 3 the polynomials $\widetilde{\mathbf{p}}, \widetilde{\mathbf{p}'}, \widetilde{\mathbf{q}}, \widetilde{\mathbf{q}'}$ are committed to using the Basefold.Commit procedure.[6] The commitments to $\widetilde{\mathbf{p}}, \widetilde{\mathbf{p}'}, \widetilde{\mathbf{q}}, \widetilde{\mathbf{q}'}$ are denoted $\mathsf{c}_{\mathbf{p}}$, $\mathsf{c}_{\mathbf{p}'}$, $\mathsf{c}_{\mathbf{q}}$, $\mathsf{c}_{\mathbf{q}'}$ respectively. At this point in the protocol, $\mathcal{V}$ has two objectives: a) check $(\mathbf{p}, \mathbf{p}')$ and $(\mathbf{q}, \mathbf{q}')$ are indeed the linear combinations of the rows of C computed using the elements of $\mathbf{r}$ and $\boldsymbol{\alpha}$, respectively, b) check $(\mathbf{p}, \mathbf{p}')$ and $(\mathbf{q}, \mathbf{q}')$ are indeed encodings of $\mathbf{p}$ and $\mathbf{q}$ respectively. Steps 4-8 and Steps 9-13 correspond to checking (a) and (b) respectively.

Steps 4-7: Checking (a) is accomplished by sampling an $I \subseteq \{0, \ldots, \rho_1 \cdot \frac{n}{\ell} - 1\}$ uniformly at random of size $\Theta(\lambda)$ (see Step 4), and $\forall i \in I$, verifying that $(\mathbf{p}, \mathbf{p}')_i = \langle \mathbf{r}, C(\cdot, i) \rangle$, $(\mathbf{q}, \mathbf{q}')_i = \langle \boldsymbol{\alpha}, C(\cdot, i) \rangle$. The latter part is checked in Step 6 using two sum-check protocols running in parallel and with the same randomness as explained next. Before that, we note that in Step 5, the i-th column of C and its associated Merkle proof is sent by $\mathcal{P}$ to $\mathcal{V}$, and $\mathcal{V}$ in turn checks that $C(\cdot, i)$ is consistent with the commitment c known to it. In Step 6, $\mathcal{V}$ samples $s_i \in_r \mathbb{F}\ \forall i \in I$ and sets $\mathsf{mask} := \sum_{i \in I} s_i \cdot \mathbf{e}_i$. Here $\mathbf{e}_i \in \mathbb{F}^{\rho_1 \cdot \frac{n}{\ell}}$ is the i-th standard basis vector. Then, $\mathcal{P}$ and $\mathcal{V}$ engage in sum-check protocols to verify that

$$\sum_{i \in I} s_i \langle \mathbf{r}, C(\cdot, i) \rangle = \sum_{\dot{\mathbf{b}} \in \{0,1\}^{\log(\rho_1 \cdot \frac{n}{\ell})}} \widetilde{\mathsf{mask}}(\dot{\mathbf{b}}) \widetilde{(\mathbf{p}, \mathbf{p}')}(\dot{\mathbf{b}})$$

and

$$\sum_{i \in I} s_i \langle \boldsymbol{\alpha}, C(\cdot, i) \rangle = \sum_{\dot{\mathbf{b}} \in \{0,1\}^{\log(\rho_1 \cdot \frac{n}{\ell})}} \widetilde{\mathsf{mask}}(\dot{\mathbf{b}}) \widetilde{(\mathbf{q}, \mathbf{q}')}(\dot{\mathbf{b}}).$$

[6] The polynomials are supposed to be in $\frac{cn}{\ell}$ variables and the commitment is computed by encoding their evaluation vectors using $E_{\frac{cn}{\ell}}$.

The verifier can compute the left-hand side (LHS) of the above equations succinctly because it has knowledge of $\mathbf{r}$, $\boldsymbol{\alpha}$, s_i, and $C(\cdot, i)$ for every $i \in I$. On the other hand, the right-hand side (RHS) involves evaluating the polynomials $\widetilde{\mathsf{mask}}(\dot{\mathbf{x}}) \cdot \widetilde{(\mathbf{p}, \mathbf{p}')}(\dot{\mathbf{x}})$, and $\widetilde{\mathsf{mask}}(\dot{\mathbf{x}})\widetilde{(\mathbf{q}, \mathbf{q}')}(\dot{\mathbf{x}})$ over the boolean hypercube. The verifier validates the RHS in the above equations by engaging in a sum-check protocol with the prover $\mathcal{P}$. The sum-check protocol eventually reduces the task of verifying these equations to checking the evaluations of $\widetilde{\mathsf{mask}}(\dot{\mathbf{x}})$, $\widetilde{(\mathbf{p}, \mathbf{p}')}(\dot{\mathbf{x}})$, and $\widetilde{(\mathbf{q}, \mathbf{q}')}(\dot{\mathbf{x}})$ at a random point $\dot{\boldsymbol{\beta}}$. In Step 7, $\mathcal{P}$ provides the evaluations of these polynomials at $\dot{\boldsymbol{\beta}}$, which the verifier will use to complete the sum-check protocol. However, the final verification of these evaluations is deferred to Step 14.

Steps 8-9: In order to check $(\mathbf{p}, \mathbf{p}')$ and $(\mathbf{q}, \mathbf{q}')$ are indeed encodings of $\mathbf{p}$ and $\mathbf{q}$ respectively, it is sufficient to check

$$(\mathbf{p}, \mathbf{p}') \cdot H = 0 \tag{2}$$
$$(\mathbf{q}, \mathbf{q}') \cdot H = 0 \tag{3}$$

where $H \in \mathbb{F}^{\frac{\rho_1 n}{\ell} \times (\rho_1 - 1)\frac{n}{\ell}}$ is the parity check matrix of the code corresponding to E_0. The MLE of H is expressed with $\log(\rho_1 \cdot \frac{n}{\ell})$ variables in $\dot{\mathbf{x}}$, and $\log(\frac{n}{\ell})$ variables in $\dot{\mathbf{y}}$. The Lagrange basis in $\dot{\mathbf{x}}$ and $\dot{\mathbf{y}}$ variables are used to index the rows and columns of H respectively. Hence, Equations (4) and (5) hold if and only if the following equations hold respectively:

$$\sum_{\dot{\mathbf{b}} \in \{0,1\}^{\log(\rho_1 \cdot \frac{n}{\ell})}} \widetilde{(\mathbf{p}, \mathbf{p}')}(\dot{\mathbf{b}}) \cdot \widetilde{H}(\dot{\mathbf{b}}, \dot{\mathbf{y}}) = 0 \tag{4}$$

$$\sum_{\dot{\mathbf{b}} \in \{0,1\}^{\log(\rho_1 \cdot \frac{n}{\ell})}} \widetilde{(\mathbf{q}, \mathbf{q}')}(\dot{\mathbf{b}}) \cdot \widetilde{H}(\dot{\mathbf{b}}, \dot{\mathbf{y}}) = 0 \tag{5}$$

$\mathcal{V}$ checks this by sampling a $\dot{\mathbf{u}} \in \mathbb{F}^{\frac{n}{\ell}}$ uniformly at random in Step 8, and checking

$$\sum_{\dot{\mathbf{b}} \in \{0,1\}^{\log(\rho_1 \cdot \frac{n}{\ell})}} \widetilde{(\mathbf{p}, \mathbf{p}')}(\dot{\mathbf{b}}) \cdot \widetilde{H}(\dot{\mathbf{b}}, \dot{\mathbf{u}}) = 0 \tag{6}$$

$$\sum_{\dot{\mathbf{b}} \in \{0,1\}^{\log(\rho_1 \cdot \frac{n}{\ell})}} \widetilde{(\mathbf{q}, \mathbf{q}')}(\dot{\mathbf{b}}) \cdot \widetilde{H}(\dot{\mathbf{b}}, \dot{\mathbf{u}}) = 0 \tag{7}$$

It is easy to see that Equations (4), and (5) imply Equations (6) and (7) respectively, whereas a simple argument using the Schwartz-Zippel lemma shows that the other direction holds with high probability over the random choice of $\dot{\mathbf{u}}$ (see Proof of Lemma A.2 in Appendix A). Equations (6) and (7) are checked using sum-check protocols running in parallel, where the sum-check polynomials are $\widetilde{(\mathbf{p}, \mathbf{p}')}(\dot{\mathbf{x}}) \cdot \widetilde{H}(\dot{\mathbf{x}}, \dot{\mathbf{u}})$ and $\widetilde{(\mathbf{q}, \mathbf{q}')}(\dot{\mathbf{x}}) \cdot \widetilde{H}(\dot{\mathbf{x}}, \dot{\mathbf{u}})$, respectively. At the end of the sum-check protocols, $\mathcal{V}$ needs to check the evaluations of $\widetilde{(\mathbf{p}, \mathbf{p}')}(\dot{\mathbf{x}})$, $\widetilde{(\mathbf{q}, \mathbf{q}')}(\dot{\mathbf{x}})$, and $\widetilde{H}(\dot{\mathbf{x}}, \dot{\mathbf{u}})$ at a random point say $\dot{\boldsymbol{\gamma}} \in \mathbb{F}^{\log(\rho_1 \cdot \frac{n}{\ell})}$. Since $\rho_1 < 2$, we have that $\dot{\boldsymbol{\gamma}} \in \mathbb{F}^{\log(\frac{n}{\ell})+1}$. Letting $\dot{\boldsymbol{\gamma}} = (\gamma_0, \ldots, \gamma_{\log(\frac{n}{\ell})})$, it is easily seen that $\left(\widetilde{\mathbf{p}, \mathbf{p}'}\right)(\dot{\boldsymbol{\gamma}})$ and $\left(\widetilde{\mathbf{q}, \mathbf{q}'}\right)(\dot{\boldsymbol{\gamma}})$ can be evaluated using the evaluations of $\widetilde{\mathbf{p}}$, $\widetilde{\mathbf{p}'}$, $\widetilde{\mathbf{q}}$, and $\widetilde{\mathbf{q}'}$ at $(\gamma_0, \ldots, \gamma_{\log(\frac{n}{\ell})-1})$ as follows:

$$\left(\widetilde{\mathbf{p}, \mathbf{p}'}\right)(\dot{\boldsymbol{\gamma}}) = (1 - \gamma_{\log(\frac{n}{\ell})}) \cdot (\widetilde{\mathbf{p}}(\gamma_0, \ldots, \gamma_{\log(\frac{n}{\ell})-1})) + \gamma_{\log(\frac{n}{\ell})} \cdot (\widetilde{\mathbf{p}'}(\gamma_0, \ldots, \gamma_{\log(\frac{n}{\ell})-1}))$$

$$\left(\widetilde{\mathbf{q},\mathbf{q}'}\right)(\dot{\gamma}) = (1-\gamma_{\log(\frac{n}{\ell})}) \cdot (\widetilde{\mathbf{q}}(\gamma_0,\ldots,\gamma_{\log(\frac{n}{\ell})-1})) + \gamma_{\log(\frac{n}{\ell})} \cdot (\widetilde{\mathbf{q}}'(\gamma_0,\ldots,\gamma_{\log(\frac{n}{\ell})-1}))$$

Hence, checking evaluations $\left(\widetilde{\mathbf{p},\mathbf{p}'}\right)$, and $\left(\widetilde{\mathbf{q},\mathbf{q}'}\right)$ at $\dot{\gamma}$ is reduced to checking the evaluations of $\widetilde{\mathbf{p}}$, $\widetilde{\mathbf{p}}'$, $\widetilde{\mathbf{q}}$, and $\widetilde{\mathbf{q}}'$ at $(\gamma_0,\ldots,\gamma_{\log(\frac{n}{\ell})-1})$. To validate the evaluation of $\widetilde{H}$ at $(\dot{\gamma},\dot{\mathbf{u}})$ the protocol leverages the sparsity of H, which contains at most $c \cdot \frac{n}{\ell}$ non-zero entries (where c is a constant). In Step 10, $\mathcal{P}$ and $\mathcal{V}$ employ the Spark commitment scheme from [Set20]. This approach reduces the evaluation problem to validating the values of the polynomials $\widetilde{H}_{\text{erow}}$, $\widetilde{H}_{\text{ecol}}$, $\widetilde{H}_{\text{val}}$ at $\dot{\boldsymbol{\theta}}_1$, as well as checking the evaluations of $\widetilde{H}_{\text{row}}$, $\widetilde{H}_{\text{col}}$, $\widetilde{H}_{\text{read\_ts,row}}$, $\widetilde{H}_{\text{final\_ts,row}}$, $\widetilde{H}_{\text{read\_ts, col}}$, $\widetilde{H}_{\text{final\_ts,col}}$, $\widetilde{H}_{\text{erow}}$, $\widetilde{H}_{\text{ecol}}$ at $\dot{\boldsymbol{\theta}}_2$. Among the nine polynomials stated above, seven of them have been committed to by BrakingBase.Gen protocol. These are namely, $\widetilde{H}_{\text{row}}$, $\widetilde{H}_{\text{col}}$, $\widetilde{H}_{\text{val}}$, $\widetilde{H}_{\text{read\_ts,row}}$, $\widetilde{H}_{\text{final\_ts,row}}$, $\widetilde{H}_{\text{read\_ts, col}}$, $\widetilde{H}_{\text{final\_ts,col}}$. The remaining two polynomials $\widetilde{H}_{\text{erow}}$ and $\widetilde{H}_{\text{erow}}$ committed to by $\mathcal{P}$ in Step 8. The details of this reduction using the Spark PCS is given in Appendix C.

Steps 14-15: In Step 14, the protocol uses the Batch-Evaluate protocol (as outlined in Section 5) to optimise the evaluation checks. This approach reduces the verification of evaluations from Steps 8, 12, and 13, which involve multiple polynomials evaluated at various points, into a simpler task: checking the evaluations of these polynomials at a single, common point. Among these evaluation checks, the protocol also checks that the evaluation of $\widetilde{\mathbf{q}}$ at $(\alpha_{\ell-1},\ldots,\alpha_{\log\frac{n}{\ell}-1}) \in \mathbb{F}^{\log\frac{n}{\ell}-\ell}$ is equal to y (see Step 14). It is easily seen that $f(\dot{\boldsymbol{\alpha}}) = y$ if and only if $\mathbf{q} = \boldsymbol{\alpha} \cdot A$, and $\widetilde{\mathbf{q}}$ evaluates to y at $(\alpha_{\ell-1},\ldots,\alpha_{\log\frac{n}{\ell}-1})$. Note that in its last step, Batch-Evaluate uses the evaluation protocol of Basefold to check the evaluation of all the required polynomials at a single point $\dot{\boldsymbol{\zeta}}$.

Algorithm 4 BrakingBase.Evaluate

Public input: pp, c, $\dot{\boldsymbol{\alpha}} \in \mathbb{F}^{\ell}$, $y \in \mathbb{F}$.
$\mathcal{P}$'s private input: f a multilinear polynomial in ℓ variables
Output: 1 if $\mathcal{V}$ accepts, and 0 otherwise.

1. $\mathcal{V}$ picks $\mathbf{r} \in_r \mathbb{F}^{\ell}$ and sends it to $\mathcal{P}$.
2. $\mathcal{P}$ defines $\boldsymbol{\alpha} = \otimes_{i=0}^{\log \ell - 1}(1-\dot{\alpha}_{\ell-i},\ \dot{\alpha}_{\ell-i})$. $\mathcal{P}$ computes $\mathbf{p} := \mathbf{r}A$, $\mathbf{q} := \boldsymbol{\alpha}A$ and $\mathbf{p}', \mathbf{q}'$ s.t. $(\mathbf{p},\mathbf{p}') = \mathbf{p}E_0$, $(\mathbf{q},\mathbf{q}') = \mathbf{q}E_0$.
3. $\mathcal{P}$ computes $\mathsf{c}_{\mathbf{p}}, \mathsf{c}_{\mathbf{p}'}, \mathsf{c}_{\mathbf{q}}, \mathsf{c}_{\mathbf{q}'}$, the commitments to $\widetilde{\mathbf{p}}, \widetilde{\mathbf{p}}', \widetilde{\mathbf{q}}, \widetilde{\mathbf{q}}'$, respectively using BaseFold PCS and sends them to $\mathcal{V}$.
   /* Verifying that $(\mathbf{p},\mathbf{p}')$ and $(\mathbf{q},\mathbf{q}')$ are indeed the linear combinations of the rows of C. */
4. $\mathcal{V}$ samples a set $I \subseteq \{0,\ldots,\rho_1\frac{n}{\ell}-1\}$ uniformly at random of size $\Theta(\lambda)$ and sends it to $\mathcal{P}$.
5. $\mathcal{P}$ sends $C(\cdot,i)$ and its associated Merkle proof (with respect to c) to $\mathcal{V}$, and $\mathcal{V}$ checks $C(\cdot,i)$ is consistent with the commitment c for every $i \in I$.
6. $\mathcal{V}$ samples $s_i \in_r \mathbb{F}\ \forall i \in I$ and sets $\mathsf{mask} := \sum_{i\in I} s_i \mathbf{e}_i$, where $\mathbf{e}_i \in \mathbb{F}^{\rho\frac{n}{\ell}}$ is the i-th standard basis vector.
7. $\mathcal{P}$ and $\mathcal{V}$ engage in sum-check protocols (in parallel, with the same randomness) to verify that $\sum_{i\in I} s_i \langle \mathbf{r}, C(\cdot,i)\rangle = \sum_{\dot{\mathbf{b}}\in\{0,1\}^{\log(\rho_1\cdot\frac{n}{\ell})}} \widetilde{\mathsf{mask}}(\dot{\mathbf{b}})\widetilde{(\mathbf{p},\mathbf{p}')}(\dot{\mathbf{b}})$ and $\sum_{i\in I} s_i \langle \boldsymbol{\alpha}, C(\cdot,i)\rangle = \sum_{\dot{\mathbf{b}}\in\{0,1\}^{\log(\rho_1\cdot\frac{n}{\ell})}} \widetilde{\mathsf{mask}}(\dot{\mathbf{b}})\widetilde{(\mathbf{q},\mathbf{q}')}(\dot{\mathbf{b}})$. At the end of the sum-checks, $\mathcal{V}$ needs to evaluate $\widetilde{\mathbf{p}}, \widetilde{\mathbf{p}}', \widetilde{\mathbf{q}}, \widetilde{\mathbf{q}}'$ at a point, say $\dot{\boldsymbol{\beta}}$, whereas $\mathcal{V}$ evaluates $\widetilde{\mathsf{mask}}$ on its own.

8. $\mathcal{P}$ sends the evaluations of $\widetilde{\mathbf{p}}, \widetilde{\mathbf{p}'}, \widetilde{\mathbf{q}}, \widetilde{\mathbf{q}'}$ at $\dot{\boldsymbol{\beta}}$ to $\mathcal{V}$. $\mathcal{V}$ checks that these are consistent with the claims made during the sum-checks. If not, then it outputs 0 and halts. The check to ensure the values sent are the evaluations of the above polynomials at $\dot{\boldsymbol{\beta}}$ is postponed to Steps 14-15.

/* Verifing that $(\mathbf{p}, \mathbf{p}')H = 0$ and $(\mathbf{q}, \mathbf{q}')H = 0$. */

9. $\mathcal{V}$ picks $\dot{\mathbf{u}} \in_R \mathbb{F}^{\lceil \log \frac{n}{\ell} \rceil}$ and sends it to $\mathcal{P}$.
10. $\mathcal{P}$ sends the commitments to two multilinear polynomials $\widetilde{H}_{\text{erow}}$ and $\widetilde{H}_{\text{ecol}}$ in $\log \frac{cn}{\ell}$ variables. The commitments are computed using Basefold.Commit algorithm. The polynomials $\widetilde{H}_{\text{erow}}$, $\widetilde{H}_{\text{ecol}}$ are used to evaluate $\widetilde{H}$ using spark [Set20].
11. $\mathcal{P}$ and $\mathcal{V}$ engage in sum-checks to verify that $\sum_{\dot{\mathbf{b}} \in \{0,1\}^{\lceil \log \rho_1 \frac{n}{\ell} \rceil}} \widetilde{(\mathbf{p}, \mathbf{p}')}(\dot{\mathbf{b}}) \widetilde{H}(\dot{\mathbf{b}}, \dot{\mathbf{u}}) = 0$ and $\sum_{\dot{\mathbf{b}} \in \{0,1\}^{\lceil \log \rho_1 \frac{n}{\ell} \rceil}} \widetilde{(\mathbf{q}, \mathbf{q}')}(\dot{\mathbf{b}}) \widetilde{H}(\dot{\mathbf{b}}, \dot{\mathbf{u}}) = 0$. At the end of the sum-checks, $\mathcal{V}$ needs to evaluate $\widetilde{(\mathbf{p}, \mathbf{p}')}, \widetilde{(\mathbf{q}, \mathbf{q}')}$ at a point say $\dot{\gamma}$, and $\widetilde{H}$ at $(\dot{\gamma}, \dot{\mathbf{u}})$.
12. $\mathcal{P}$ sends the evaluations of $\widetilde{\mathbf{p}}, \widetilde{\mathbf{p}'}, \widetilde{\mathbf{q}}, \widetilde{\mathbf{q}'}$ at $\dot{\gamma}$ to $\mathcal{V}$. $\mathcal{P}$ also sends the evaluation of $\widetilde{H}$ at $(\dot{\gamma}, \dot{\mathbf{u}})$. $\mathcal{V}$ checks that these are consistent with the claims made during the sum-checks. If not, then it outputs REJECT and halts. The check to ensure the values sent are the evaluations of $\widetilde{\mathbf{p}}, \widetilde{\mathbf{p}'}, \widetilde{\mathbf{q}}, \widetilde{\mathbf{q}'}$ at $\dot{\gamma}$ is postponed to Steps 14-15. The check to ensure the correctness of the value sent as the evaluation of $\widetilde{H}$ at $(\dot{\gamma}, \dot{\mathbf{u}})$ is done at the next step.
13. Use spark [Set20] (see Appendix C), to reduce checking the correctness of claimed evaluation of $\widetilde{H}$ at $(\dot{\gamma}, \dot{\mathbf{u}})$ to validating the evaluation of $\widetilde{H}_{\text{erow}}$, $\widetilde{H}_{\text{ecol}}$, $\widetilde{H}_{\text{val}}$ at $\dot{\boldsymbol{\theta}}_1$, and the evaluations of $\widetilde{H}_{\text{row}}$, $\widetilde{H}_{\text{col}}$, $\widetilde{H}_{\text{read\_ts,row}}$, $\widetilde{H}_{\text{final\_ts,row}}$, $\widetilde{H}_{\text{read\_ts, col}}$, $\widetilde{H}_{\text{final\_ts,col}}$, $\widetilde{H}_{\text{erow}}$, $\widetilde{H}_{\text{ecol}}$ at $\dot{\boldsymbol{\theta}}_2$.

/* Evaluating all the polynomials and checking that $y = f(\dot{\boldsymbol{\alpha}})$. */

14. Run the Batch-Evaluate protocol (Protocol 5). The Batch-Evaluate protocol first reduces a) checking the evaluations of $\widetilde{\mathbf{p}}$, $\widetilde{\mathbf{p}'}$, $\widetilde{\mathbf{q}}$, $\widetilde{\mathbf{q}'}$ in Steps 8 and 12, b) checking the evaluation of $\widetilde{\mathbf{q}}$ at $(\alpha_{\ell-1}, \ldots, \alpha_{\log \frac{n}{\ell}-1}) \in \mathbb{F}^{\log \frac{n}{\ell} - \ell}$, and c) checking the evaluations of $\widetilde{H}_{\text{row}}$, $\widetilde{H}_{\text{col}}$, $\widetilde{H}_{\text{val}}$, $\widetilde{H}_{\text{read\_ts,row}}$, $\widetilde{H}_{\text{final\_ts,row}}$, $\widetilde{H}_{\text{read\_ts, col}}$, $\widetilde{H}_{\text{final\_ts,col}}$, $\widetilde{H}_{\text{erow}}$, $\widetilde{H}_{\text{ecol}}$ in Step 13 to checking the evaluation of all the polynomials involved above at a single random point $\dot{\boldsymbol{\zeta}} \in \mathbb{F}^{\log \frac{cn}{\ell}}$.[7] Then it uses Basefold's evaluation protocol to check the evaluation of all these polynomials at $\dot{\boldsymbol{\zeta}}$. Output 1 if and only if Batch-evaluate outputs 1.

3.4 The Batch-Evaluate protocol

The Batch-Evaluate protocol, as outlined in Protocol 5 optimises the problem of validating the evaluations of a set of polynomials each evaluated at a possibly different point. Instead of checking each polynomial at a distinct point (relation $\mathcal{R}_1$ stated below), Batch-Evaluate reduces it to a single verification: checking all polynomials at a common point (relation $\mathcal{R}_2$ stated below).

$$\mathcal{R}_1 = \left\{ \left(\{\mathsf{c}_t\}_{t\in[k]}, \{\omega_t\}_{t\in[k]}, \{y_t\}_{t\in[k]}; \{f_t\}_{t\in[k]} \right) \mid f_t(\omega_t) = y_t, \omega_t \in \mathbb{F}^{\ell_t} \ \ \forall t \in [k] \right\} \tag{8}$$

$$\mathcal{R}_2 = \left\{ \left(\{\mathsf{c}_t\}_{t\in[k]}, \omega, \{y'_t\}_{t\in[k]}; \{f_t\}_{t\in[k]} \right) \mid \ \forall t \in [k] \ \ f_t(\omega) = y'_t, \omega \in \mathbb{F}^{\ell} \right\} \tag{9}$$

[7] Since all polynomials are in $\log \frac{cn}{\ell}$ or fewer variables, here we treat them as polynomials in exactly $\log \frac{cn}{\ell}$ variables.

In the above equations, c_t is the commitment to a multilinear polynomial f_t in ℓ_t variables, and ω_t is a point in $\mathbb{F}^{\ell_t}$, and $\ell = \max(\ell_t \mid t \in [k])$. We remark that Batch-Evaluate is crucially used in Step 14 of the BrakingBase.Eval protocol to achieve such a reduction, and reduce the proof complexity of the BrakingBase.Eval protocol.

Algorithm 5 Batch-Evaluate

Public input: $\{\mathsf{c}_t\}_{t\in[k]}, \{\omega_t\}_{t\in[k]}, \{y_t\}_{t\in[k]}$.
$\mathcal{P}$'s private input: The Lagrange coefficients of multilinear polynomial f_t for $t \in [k]$.
Output: 1 if the $\mathcal{V}$ accepts and 0 otherwise.

1. $\mathcal{V}$ sends $r_1, \ldots, r_k \in_R \mathbb{F}$ to $\mathcal{P}$.
2. $\mathcal{P}$ and $\mathcal{V}$ engage in a sum-check protocol to verify that

$$\sum_{t\in[k]} r_t\omega_t = \sum_{\dot{\mathbf{x}}\in\{0,1\}^{\ell'}} \left(\sum_{t\in[k]} r_t \, \widetilde{eq}(\dot{\boldsymbol{\alpha}}_t, \dot{\mathbf{x}}) f_t(\dot{\mathbf{x}}) \right).$$

   At the end of the sum-check protocol, $\mathcal{V}$ needs to evaluate $f_1, \ldots, f_k$ at a certain point, say $\dot{\boldsymbol{\zeta}} \in \mathbb{F}^{\ell'}$.
3. $\forall t \in [k]$, $\mathcal{P}$ sends $\theta_t := f_t(\dot{\boldsymbol{\zeta}})$ to $\mathcal{V}$. $\mathcal{V}$ checks that these evaluations are consistent with the claims made by $\mathcal{P}$ in the sum-check protocol. Then it sends $s_1, \ldots, s_k \in_r \mathbb{F}$ to $\mathcal{P}$.
4. $\mathcal{P}$ proves to $\mathcal{V}$ that $\sum_{t\in[k]} s_t f_t(\dot{\boldsymbol{\zeta}}) = \sum_{t\in[k]} s_t\theta_t$ using the evaluation algorithm of BaseFold PCS. $\mathcal{V}$ simulates a commitment to this polynomial using $\mathsf{c}_1, \ldots, \mathsf{c}_t$.
5. $\mathcal{V}$ returns 1 if and only if BaseFold's verifier accepts.

3.5 Completeness and succinctness

The completeness of BrakingBase follows from the descriptions of the Commit and Open in Section 3.2, and Eval in Section 3.3. We now give a running time analysis for the verifier and the prover of the compiled PCS.

Verifier running time. The verifier running time is dominated by the following:

1. the time spent executing the sum-checks in Steps 7, 11 of BrakingBase.Evalaute,
2. the time spent verifying the Merkle-tree opening proofs for $C(\cdot, i)$ for all $i \in I$,
3. the time spent executing the Spartan verifier in Step 13 of BrakingBase.Evaluate,
4. the verifier time for the sum-check in Step 3 of BrakingBase.Batch-Evaluate, and
5. the time spent evaluating $\sum_{t\in[k]} s_t f_t$ using BaseFold PCS in Step 4 of Batch-Evaluate.

Items 1 and 4 requires $O(\log n)$ time. Item 2, requires $O(\lambda \log n)$ time assuming $\mathcal{P}$ commits to C as follows: hashes every column of C, and then use the hashes as the leaves of the Merkle tree whose root is the commitment c. Using the techniques from [SL20], Item 3 can also be made to require $O(\log n)$ time. Finally, Item 5 requires $O(\lambda \log^2 n)$ time. Thus, the verifier time is $O(\lambda \log^2 n)$.

Prover running time. The dominant factor in $\mathcal{P}$'s running time are:

1. time required to commit to $\widetilde{\mathbf{p}}, \widetilde{\mathbf{p}'}, \widetilde{\mathbf{q}}, \widetilde{\mathbf{q}'}$ in Step 3.
2. the run-time of the sum-check provers called on Steps 7, 11 BrakingBase.Evaluate,
3. the run-time of Spartan's prover called on Step 13, and
4. the run-time of BaseFold's prover called on Step 4 of Batch-Evaluate.

All four require $O(\frac{n}{\ell} \log \frac{n}{\ell}) = O(n)$ time.

Proof Size: As the proof size is upper bounded by the verifier's running time, the proof size is $O\left(\lambda \log^2 n\right)$.

4 Implementation

In this section, we describe the concrete costs of BrakingBase and compare these costs with those of the BaseFold and Brakedown PCS. All experiments are conducted on a QCT Rack Mount server with 46 cores and 256 GB of RAM, running Ubuntu 22.04.5 LTS. For comparison, we use the BaseFold and Brakedown implementations available at [Bas]. Experiments are performed on polynomials with 20 to 28 variables over a 128-bit field, with parameters for all three PCS chosen to ensure at least 100 bits of security. We evaluate four metrics: commitment time, evaluation prover time, proof size, and verifier time. For each $k \in \{20, \ldots, 28\}$, we run the commitment algorithm and the evaluation protocol 10 times on polynomials with k variables, then report the average values for all four metrics in Figure 1. We also apply an optimization suggested by [DP24] to reduce the proof size: specifically, the proximity test and the evaluation check are combined under the assumption that the point at which the polynomial is evaluated is sampled uniformly at random. Below, we briefly discuss how BrakingBase compares with the BaseFold and Brakedown PCS.

Comparison with BaseFold. For polynomials with more than 25 variables, the commitment time of BaseFold is approximately $12\times$ that of BrakingBase. This difference arises because the commitment time for the BaseFold PCS is $O(n \log n)$, whereas for BrakingBase, it is $O(n)$. However, the evaluation prover and verifier of BrakingBase are slower than those of BaseFold by factors of approximately $3.5\times$ and $3\times$, respectively. Importantly, due to the batch evaluation protocol described in Section 3, the evaluation protocol needs to be executed only once in a SNARK setting. As a SNARK may require commitments to multiple polynomials, a PCS with a slower evaluation prover but faster commitments is advantageous over one with a faster evaluation prover but slower commitments. Additionally, while the proof-size of BrakingBase exceeds that of BaseFold up to 24 variables, beyond this, BrakingBase produces smaller proofs. We note that the proof sizes of BaseFold could be further optimized since the current implementation at [Bas] writes both the left and right nodes along a Merkle path.

Comparison with Brakedown. For polynomials with more than 25 variables, the commitment and evaluation prover times of BrakingBase are approximately $1.9\times$ and $155\times$ those of Brakedown, respectively. However, BrakingBase offers a significantly faster verifier and smaller proof-sizes, as both the proof-size and verifier time for BrakingBase are $O_\lambda(\log^2 n)$, compared to $O_\lambda(\sqrt{n})$ for Brakedown. Specifically, for polynomials with 28 variables, the proofs produced by BrakingBase are $\approx 2.7\times$ smaller, and the verifier time is reduced by $\approx 2.8\times$.

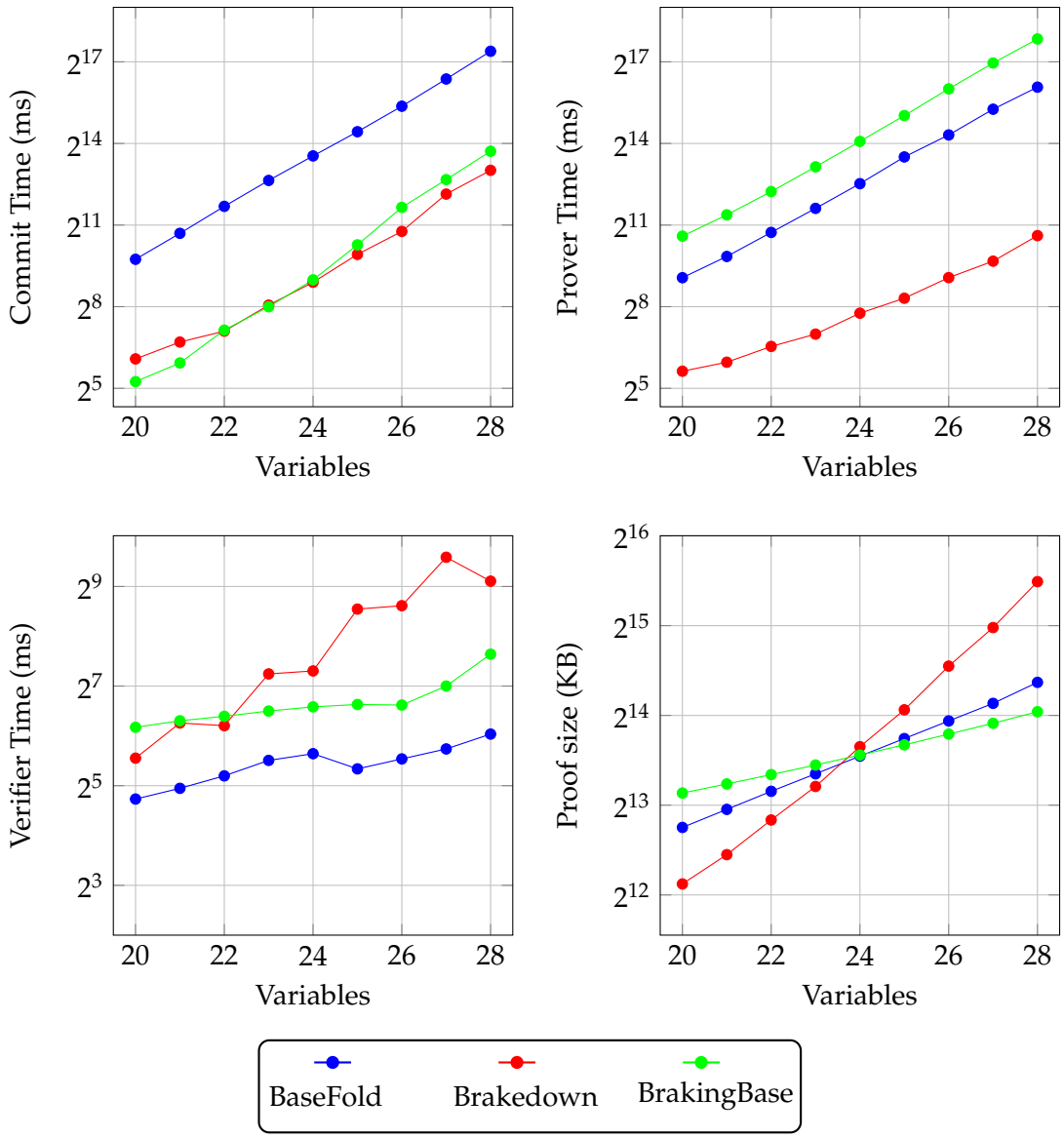

Figure 1: Performance of BaseFold, Brakedown, and BrakingBase PCS on 128 bit fields.

References

[AHIV17] Scott Ames, Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. Ligero: Lightweight sublinear arguments without a trusted setup. In Bhavani Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, pages 2087–2104. ACM, 2017. 2, 10

[AST24] Arasu Arun, Srinath T. V. Setty, and Justin Thaler. Jolt: Snarks for virtual machines via lookups. In *Advances in Cryptology - EUROCRYPT 2024 - 43rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zurich, Switzerland,*

May 26-30, 2024, Proceedings, Part VI, volume 14656 of *Lecture Notes in Computer Science*, pages 3–33. Springer, 2024. 3

[Bas] plonkish_basefold. https://github.com/hadasz/plonkish_basefold. 21

[BBB+18] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Gregory Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *2018 IEEE Symposium on Security and Privacy, SP 2018, Proceedings, 21-23 May 2018, San Francisco, California, USA*, pages 315–334. IEEE Computer Society, 2018. 2

[BBHR18] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Fast reed-solomon interactive oracle proofs of proximity. In Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella, editors, *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*, volume 107 of *LIPIcs*, pages 14:1–14:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. 2

[BCF+24] Martijn Brehm, Binyi Chen, Ben Fisch, Nicolas Resch, Ron D. Rothblum, and Hadas Zeilberger. Blaze: Fast SNARKs from interleaved RAA codes. Cryptology ePrint Archive, Paper 2024/1609, 2024. 4, 8

[BCKL22] Eli Ben-Sasson, Dan Carmon, Swastik Kopparty, and David Levit. Scalable and transparent proofs over all large fields, via elliptic curves - (ECFFT part II). In Eike Kiltz and Vinod Vaikuntanathan, editors, *Theory of Cryptography - 20th International Conference, TCC 2022, Chicago, IL, USA, November 7-10, 2022, Proceedings, Part I*, volume 13747 of *Lecture Notes in Computer Science*, pages 467–496. Springer, 2022. 3

[BCKL23] Eli Ben-Sasson, Dan Carmon, Swastik Kopparty, and David Levit. Elliptic curve fast fourier transform (ECFFT) part I: low-degree extension in time $O(n \log n)$ over all finite fields. In Nikhil Bansal and Viswanath Nagarajan, editors, *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023, Florence, Italy, January 22-25, 2023*, pages 700–737. SIAM, 2023. 3

[BCS16] Eli Ben-Sasson, Alessandro Chiesa, and Nicholas Spooner. Interactive oracle proofs. In Martin Hirt and Adam D. Smith, editors, *Theory of Cryptography - 14th International Conference, TCC 2016-B, Beijing, China, October 31 - November 3, 2016, Proceedings, Part II*, volume 9986 of *Lecture Notes in Computer Science*, pages 31–60, 2016. 10, 25

[BFH+20] Rishabh Bhadauria, Zhiyong Fang, Carmit Hazay, Muthuramakrishnan Venkitasubramaniam, Tiancheng Xie, and Yupeng Zhang. Ligero++: A new optimized sublinear IOP. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *CCS '20: 2020 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, USA, November 9-13, 2020*, pages 2025–2038. ACM, 2020. 3, 8

[CBBZ23] Binyi Chen, Benedikt Bünz, Dan Boneh, and Zhenfei Zhang. Hyperplonk: Plonk with linear-time prover and high-degree custom gates. In Carmit Hazay and Martijn Stam, editors, *Advances in Cryptology - EUROCRYPT 2023 - 42nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Lyon, France, April 23-27,*

2023, Proceedings, Part II, volume 14005 of *Lecture Notes in Computer Science*, pages 499–530. Springer, 2023. 3, 7

[dHS24] Thomas den Hollander and Daniel Slamanig. A crack in the firmament: Restoring soundness of the orion proof system and more. Cryptology ePrint Archive, Paper 2024/1164, 2024. 8

[DP23] Benjamin E. Diamond and Jim Posen. Succinct arguments over towers of binary fields. *IACR Cryptol. ePrint Arch.*, page 1784, 2023. 3

[DP24] Benjamin E. Diamond and Jim Posen. Proximity testing with logarithmic randomness. *IACR Commun. Cryptol.*, 1(1):2, 2024. 5, 16, 21

[GLS+23] Alexander Golovnev, Jonathan Lee, Srinath T. V. Setty, Justin Thaler, and Riad S. Wahby. Brakedown: Linear-time and field-agnostic snarks for R1CS. In Helena Handschuh and Anna Lysyanskaya, editors, *Advances in Cryptology - CRYPTO 2023 - 43rd Annual International Cryptology Conference, CRYPTO 2023, Santa Barbara, CA, USA, August 20-24, 2023, Proceedings, Part II*, volume 14082 of *Lecture Notes in Computer Science*, pages 193–226. Springer, 2023. 2, 4, 10, 13, 14, 29

[GNS24] Chaya Ganesh, Vineet Nair, and Ashish Sharma. Dual polynomial commitment schemes and applications to commit-and-prove snarks. In *Proceedings of the 2024 ACM SIGSAC Conference on Computer and Communications Security, CCS 2024*. ACM, 2024. 2

[HLP24] Ulrich Haböck, David Levit, and Shahar Papini. Circle starks. *IACR Cryptol. ePrint Arch.*, page 278, 2024. 3

[KT24] Tohru Kohrita and Patrick Towa. Zeromorph: Zero-knowledge multilinear-evaluation proofs from homomorphic univariate commitments. *J. Cryptol.*, 37(4):38, 2024. 2

[KZG10] Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. Constant-size commitments to polynomials and their applications. In Masayuki Abe, editor, *Advances in Cryptology - ASIACRYPT 2010 - 16th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 5-9, 2010. Proceedings*, volume 6477 of *Lecture Notes in Computer Science*, pages 177–194. Springer, 2010. 2

[Lee21] Jonathan Lee. Dory: Efficient, transparent arguments for generalised inner products and polynomial commitments. In Kobbi Nissim and Brent Waters, editors, *Theory of Cryptography - 19th International Conference, TCC 2021, Raleigh, NC, USA, November 8-11, 2021, Proceedings, Part II*, volume 13043 of *Lecture Notes in Computer Science*, pages 1–34. Springer, 2021. 2

[Set20] Srinath T. V. Setty. Spartan: Efficient and general-purpose zksnarks without trusted setup. In Daniele Micciancio and Thomas Ristenpart, editors, *Advances in Cryptology - CRYPTO 2020 - 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17-21, 2020, Proceedings, Part III*, volume 12172 of *Lecture Notes in Computer Science*, pages 704–737. Springer, 2020. 2, 7, 14, 18, 19, 30, 31

[SL20] Srinath T. V. Setty and Jonathan Lee. Quarks: Quadruple-efficient transparent zksnarks. *IACR Cryptol. ePrint Arch.*, page 1275, 2020. 20

[Tha22] Justin Thaler. Proofs, arguments, and zero-knowledge. *Foundations and Trends® in Privacy and Security*, 4(2–4):117–660, 2022. 9

[XZS22] Tiancheng Xie, Yupeng Zhang, and Dawn Song. Orion: Zero knowledge proof with linear prover time. In Yevgeniy Dodis and Thomas Shrimpton, editors, *Advances in Cryptology - CRYPTO 2022 - 42nd Annual International Cryptology Conference, CRYPTO 2022, Santa Barbara, CA, USA, August 15-18, 2022, Proceedings, Part IV*, volume 13510 of *Lecture Notes in Computer Science*, pages 299–328. Springer, 2022. 3, 7

[ZCF24] Hadas Zeilberger, Binyi Chen, and Ben Fisch. Basefold: Efficient field-agnostic polynomial commitment schemes from foldable codes. In Leonid Reyzin and Douglas Stebila, editors, *Advances in Cryptology - CRYPTO 2024 - 44th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2024, Proceedings, Part X*, volume 14929 of *Lecture Notes in Computer Science*, pages 138–169. Springer, 2024. 2, 4, 9, 12, 14, 26, 27

[ZXZS20] Jiaheng Zhang, Tiancheng Xie, Yupeng Zhang, and Dawn Song. Transparent polynomial delegation and its applications to zero knowledge proof. In *2020 IEEE Symposium on Security and Privacy, SP 2020, San Francisco, CA, USA, May 18-21, 2020*, pages 859–876. IEEE, 2020. 3, 7, 8

A Knowledge Soundness of BrakingBase

In this section we prove that BrakingBase is knowledge sound. In particular, we prove that the IOPP version of BrakingBase is knowledge sound. In the IOP version, instead of commitments to the polynomials, i.e. roots of some Merkle trees, the verifier is given oracle access to the vector from which the Merkle tree was constructed. [BCS16] prove that any such IOPP can be compiled into a succinct argument using Merkle hashes. Our BrakingBase PCS can be thought of as having been obtained from its IOP version by using this compiler. Because of the knowledge soundness of this compiler, the knowledge soundness of the IOP version of BrakingBase implies the soundness of BrakingBase as a PCS.

We first prove that the Batch-Evaluate protocol is knowledge sound. Since we are proving knowledge soundness for the IOP version of Batch-Evaluate, the public input now contains oracles $\{\mathcal{O}_t\}_k$ instead of commitments $\{\mathsf{c}_t\}_k$.

Lemma A.1 (Knowledge Soundness of Batch Evaluate)**.** *Let $k = \mathrm{poly}(\lambda)$ and $m = 2^\ell$. Suppose that Batch-Evaluate accepts with probability $\frac{1}{\mathrm{poly}(\lambda)}$ on public input $\{\mathcal{O}_t\}_{t\in[k]}, \{\dot{\omega}_t\}_{t\in[k]}, \{y_t\}_{t\in[k]}$. Then there exists an extractor* Ext *running in time $\mathrm{poly}(n,\lambda)$ which outputs multilinear polynomials $\{f_t\}_{t\in[k]} \in \mathbb{F}[x_0,\ldots,x_{\ell-1}]$, such that $\Delta(\mathsf{Enc}_2(f_t)^8, \mathcal{O}_t) \leq \frac{\delta_2}{3}\cdot\rho_2 m$ and $f_t(\dot{\omega}_t) = y_t$ for all $t \in [k]$ with probability $1 - \mathsf{negl}(\lambda)$.*

Proof. Let $\mathbf{r} = (r_1,\ldots,r_k)$, $\mathbf{s} = (s_1,\ldots,s_k)$. Then, from the hypothesis of the lemma, we have that,

$$\Pr_{\mathbf{r},\check{\xi},\mathbf{s},\boldsymbol{\rho}}[\text{Batch-evaluate accepts}] \geq \frac{1}{\mathrm{poly}(\lambda)},$$

[8] Define Enc_1 and Enc_2

where $\boldsymbol{\rho}$ is the randomness used by BaseFold's evaluation procedure. Thus, from Lemma 2.2, for at least $\frac{1}{2\cdot\text{poly}(\lambda)} = \frac{1}{\text{poly}(\lambda)}$ fraction of $\mathbf{r}, \dot{\boldsymbol{\zeta}}$,

$$\Pr_{\mathbf{s},\boldsymbol{\rho}}\left[\text{Batch evaluate accepts}|\mathbf{r}, \dot{\boldsymbol{\zeta}}\right] \geq \frac{1}{2\cdot\text{poly}(\lambda)} = \frac{1}{\text{poly}(\lambda)}.$$

Again by applying Lemma 2.2, for at least $\frac{1}{2\cdot\text{poly}(\lambda)} = \frac{1}{\text{poly}(\lambda)}$ fraction of $\mathbf{s}$,

$$\Pr_{\boldsymbol{\rho}}\left[\text{Batch evaluate accepts}|\mathbf{r}, \dot{\boldsymbol{\zeta}}, \mathbf{s}\right] \geq \frac{1}{2\cdot\text{poly}(\lambda)} = \frac{1}{\text{poly}(\lambda)}.$$

The above is equivalent to

$$\Pr_{\boldsymbol{\rho}}\left[\text{BaseFold's evaluation procedure accepts on public input } \sum_{t\in[k]} s_t\mathcal{O}_t, \dot{\boldsymbol{\zeta}}, \text{ and } \sum_{t\in[k]} s_t\theta_t|\mathbf{r}, \dot{\boldsymbol{\zeta}}, \mathbf{s}\right] \geq \frac{1}{\text{poly}(\lambda)}.$$

Then from Theorem 7 of [ZCF24], there exists a multilinear polynomial $f_{\mathbf{s}} \in \mathbb{F}[x_0, \ldots, x_{\ell-1}]$ such that $f_{\mathbf{s}}(\dot{\boldsymbol{\zeta}}) = \sum_{t\in[k]} s_t\theta_t$ and $\Delta(\mathsf{Enc}_2(f_{\mathbf{s}}), \sum_{t\in[k]} s_t\mathcal{O}_t) < \frac{\delta_2}{3}\cdot\rho_2 m$. Since this happens for $\frac{1}{\text{poly}(\lambda)}$ fraction of $\mathbf{s}$, Lemma 2.4 implies that there must exist multilinear polynomials $\{f_t\}_{t\in[k]} \in \mathbb{F}[x_0, \ldots, x_{\ell-1}]$ such that $\Delta(\mathsf{Enc}_2(f_t), \mathcal{O}_t) < \frac{\delta_2}{3}\cdot\rho_2 m$ for all $t \in [k]$. As $\Delta(\mathsf{Enc}_2(f_{\mathbf{s}}), \sum_{t\in[k]} s_t\mathcal{O}_t) < \frac{\delta_2}{3}\cdot\rho_2 m$ and $\Delta(\mathsf{Enc}_2(\sum_{t\in[k]} s_t f_t), \sum_{t\in[k]} s_t\mathcal{O}_t) < \frac{\delta_2}{3}\cdot\rho_2 m$, by triangle inequality, $\Delta(\mathsf{Enc}_2(\sum_{t\in[k]} s_t f_t), f_{\mathbf{s}}) < \frac{2\cdot\delta_2}{3}\cdot\rho_2 m$. Thus $f_{\mathbf{s}} = \sum_{t\in[k]} s_t f_t$. Moreover as for $\frac{1}{\text{poly}(\lambda)} > \frac{1}{|\mathbb{F}|}$ fraction of $\mathbf{s}$, $\sum_{t\in[k]} s_t f_t(\dot{\boldsymbol{\zeta}}) = \sum_{t\in[k]} s_t\theta_t$, $\theta_t = f_t(\dot{\boldsymbol{\zeta}})$ for all $t \in [k]$.

So far we have proved that for at least $\frac{1}{\text{poly}(\lambda)}$ fraction of $\mathbf{r}, \dot{\boldsymbol{\zeta}}$, $f_t(\dot{\boldsymbol{\zeta}}) = \theta_t$ and $\Delta(\mathsf{Enc}_2(f_t), \mathcal{O}_t) < \frac{\delta_2}{3}\cdot\rho_2 m$ for all $t \in [k]$. Once again Lemma 2.2 implies that for at least $\frac{1}{2\cdot\text{poly}(\lambda)} = \frac{1}{\text{poly}(\lambda)}$ fraction of $\mathbf{r}$, this happens with probability at least $\frac{1}{2\cdot\text{poly}(\lambda)} = \frac{1}{\text{poly}(\lambda)}$ over the random choice of $\dot{\boldsymbol{\zeta}}$; let us call all such $\mathbf{r}$ 'good'. Then for any good $\mathbf{r}$ the check on Step 3 and soundness of the sum-check protocol together imply that

$$\sum_{t\in[k]} r_t y_t = \sum_{\dot{\mathbf{x}}\in\{0,1\}^\ell}\left(\sum_{t\in[k]} r_t\,\widetilde{eq}(\dot{\boldsymbol{\omega}}_t, \dot{\mathbf{x}})f_t(\dot{\mathbf{x}})\right) = \sum_{t\in[k]} r_t \sum_{\dot{\mathbf{x}}\in\{0,1\}^\ell}\widetilde{eq}(\dot{\boldsymbol{\omega}}_t, \dot{\mathbf{x}})f_t(\dot{\mathbf{x}}) = \sum_{t\in[k]} r_t f_t(\dot{\boldsymbol{\omega}}_t).$$

As the above holds for at least $\frac{1}{\text{poly}(\lambda)} > \frac{1}{|\mathbb{F}|}$ fraction of $\mathbf{r}$, $y_t = \sum_{\dot{\mathbf{x}}\in\{0,1\}^\ell}\widetilde{eq}(\dot{\boldsymbol{\omega}}_t, \dot{\mathbf{x}})f_t(\dot{\mathbf{x}}) = f(\dot{\boldsymbol{\omega}}_t)$ for all $t \in [k]$. Hence, we have shown the existence of multilinear polynomials $\{f_t\}_{t\in[k]} \in \mathbb{F}[x_0, \ldots, x_{\ell-1}]$ satisfying $\Delta(\mathsf{Enc}_2(f_t), \mathcal{O}_t) < \frac{\delta_2}{3}\cdot\rho_2 m$ for all $t \in [k]$. We now show how these polynomials can be computed in time $\text{poly}(n, \lambda)$.

Observe that to compute $\{f_t\}_{t\in[k]}$ it is sufficient to find $\mathbf{r}, \dot{\boldsymbol{\zeta}}$ such that for at least $\frac{1}{\text{poly}(\lambda)}$ fraction of $\mathbf{s}$,

$$\Pr_{\boldsymbol{\rho}}\left[\text{BaseFold's evaluation procedure accepts on public input } \sum_{t\in[k]} s_t\mathcal{O}_t, \dot{\boldsymbol{\zeta}}, \text{ and } \sum_{t\in[k]} s_t\theta_t|\mathbf{r}, \dot{\boldsymbol{\zeta}}, \mathbf{s}\right] \geq \frac{1}{\text{poly}(\lambda)}.$$

Call such $\mathbf{r}, \dot{\boldsymbol{\zeta}}$ 'good'. Then we can find linearly independent $\mathbf{s}^{(1)}, \ldots, \mathbf{s}^{(k)}$ such that the above holds for all of them, use BaseFold's extraction procedure to obtain $f_{\mathbf{s}^{(t)}}$, and solve a system of linear equations to obtain $\{f_t\}_{t\in[k]}$.

Suppose that we have good $\mathbf{r}, \dot{\boldsymbol{\zeta}}$. Let $\mathcal{M} := \mathbb{F}^k$ and $\Phi : \mathcal{M}^* \to \{0,1\}$ be a predicate such that $\Phi(\mathbf{s}^{(1)}, \ldots, \mathbf{s}^{(m)}) = 1$ if and only if $\mathbf{s}^{(1)}, \ldots, \mathbf{s}^{(m)}$ are linearly independent. Observe that

$$\Pr_{\mathbf{s}^{(m)}}[\Phi(\mathbf{s}^{(1)}, \ldots, \mathbf{s}^{(m)}) = 1 \mid \mathbf{s}^{(1)}, \ldots, \mathbf{s}^{(m-1)}] \geq 1 - \frac{k}{|\mathbb{F}|} = 1 - \mathsf{negl}(\lambda).$$

Further, let A be an algorithm such that $A(\mathbf{s}) = 1$ if and only if BaseFold's extractor Ext_1 outputs a polynomial $f_{\mathbf{s}}$ satisfying $\Delta(\sum_{t\in[k]} s_t \mathcal{O}_t, \mathsf{Enc}_2(f_{\mathbf{s}})) < \frac{\delta_2}{3} \cdot \rho_2 m$ and $f_{\mathbf{s}}(\dot{\boldsymbol{\zeta}}) = \sum_{t\in[k]} s_t \theta_t$. Then from Theorem 4 of [ZCF24], $\Pr_{\mathbf{s}\in_R \mathcal{M}}[A(\mathbf{s}) = 1] \geq 1 - \mathsf{negl}(\lambda)$ and its running time is $\mathrm{poly}(n, \lambda)$. Since $k = \mathrm{poly}(\lambda)$, 2.3 implies that there exists an extractor Ext_2 which given oracle access to A outputs linearly independent $\mathbf{s}^{(1)}, \ldots, \mathbf{s}^{(m)}$ (i.e. $\Phi(\mathbf{s}^{(1)}, \ldots, \mathbf{s}^{(m)}) = 1$) such that $\forall u \in [k]$, Ext_1 outputs a polynomial $f_{\mathbf{s}^{(u)}}$ satisfying $\Delta(\sum_{t\in[k]} s_{u,t}\mathcal{O}_t, f_{\mathbf{s}^{(t)}}) < \frac{\delta_2}{3} \cdot \rho_2 m$ and $f_{\mathbf{s}^{(t)}}(\dot{\boldsymbol{\zeta}}) = \sum_{t\in[k]} s_{u,t}\omega_t$ (i.e. $A(\mathbf{s}^{(t)})) = 1$) with probability $1 - \mathsf{negl}(\lambda)$. Furthermore, the running time of Ext_2 is polynomial in λ and the running time A. Thus it runs in time $\mathrm{poly}(\lambda, n)$.

Consider an Ext' that runs Ext_2 to obtain $\mathbf{s}^{(1)}, \ldots, \mathbf{s}^{(m)}$, runs Ext_1 for each $\mathbf{s}^{(u)}$ to find $f_{\mathbf{s}^{(m)}}$, and finally computes $\{f_t\}_{t\in[k]}$ in time $\mathrm{poly}(\lambda, n)$ by solving a system of linear equations. If $\Delta(\mathsf{Enc}_2(f_t), \mathcal{O}_t) < \frac{\delta_2}{3} \cdot \rho_2 m$ and $f_t(\dot{\boldsymbol{\omega}}_t) = y_t$ for all $t \in [k]$, Ext' outputs $\{f_t\}_{t\in[k]}$, else it reports failure. Note that the running time of Ext' is $\mathrm{poly}(n, \lambda)$ and its success probability is $1 - \mathsf{negl}(\lambda)$. To finish off the proof, observe that since at least $\frac{1}{\mathrm{poly}(\lambda)}$ fraction of $\mathbf{r}, \dot{\boldsymbol{\zeta}}$ are good, it is not difficult to see that if we run Ext' $\mathrm{poly}(\lambda)$ times, then we are bound to have run it for good $\mathbf{r}, \dot{\boldsymbol{\zeta}}$ at least once with probability $1 - \mathsf{negl}(\lambda)$. This yields the desired extractor Ext.

□

We now prove that BrakingBase satisfies binding. Since we are proving the knowledge soundness of the IOP version of the PCS, the public input will contain (oracle access to) the matrix C rather than a commitment c to the root of a Merkle tree of C.

Lemma A.2 (Binding). *Suppose that a PPT adversary $\mathcal{A}$ causes $\mathcal{V}$ to accept an evaluation proof with probability $\frac{1}{\mathrm{poly}(\lambda)}$ on public input C, $\dot{\boldsymbol{\alpha}}$, y. Then there exists a multilinear $f' \in \mathbb{F}[x_1, \ldots, x_\ell]$ such that $f'(\dot{\boldsymbol{\alpha}}) = y$ and if $A' :=$ the coefficients of f' written as an $\ell \times \frac{n}{\ell}$ matrix, then for all $i \in \{0, \ldots, \ell - 1\}$, $\Delta(\mathsf{Enc}_1(A'(i, \cdot)), C(i, \cdot)) \leq \frac{\delta_1}{3}$.*

Proof. Let $\mathbf{s} := \{s_i\}_{i\in I}$. We have that

$$\Pr_{\mathbf{r}, I, \mathbf{s}, \dot{\boldsymbol{\beta}}, \dot{\mathbf{u}}, \dot{\gamma}, \dot{\boldsymbol{\theta}}_1, \dot{\boldsymbol{\theta}}_2, \boldsymbol{\rho}}[\mathcal{V} \text{ accepts}] \geq \frac{1}{\mathrm{poly}(\lambda)},$$

where $\boldsymbol{\rho}$ is the randomness used in Batch-Evaluate. An application of Lemma 2.2 implies that for at least $\frac{1}{\mathrm{poly}(\lambda)}$ fraction of $\mathbf{r}, I, \mathbf{s}, \dot{\boldsymbol{\beta}}, \dot{\mathbf{u}}, \dot{\gamma}, \dot{\boldsymbol{\theta}}_1, \dot{\boldsymbol{\theta}}_2$,

$$\Pr_{\boldsymbol{\rho}}[\text{Batch-Evaluate accepts} | \mathbf{r}, I, \mathbf{s}, \dot{\boldsymbol{\beta}}, \dot{\mathbf{u}}, \dot{\gamma}, \dot{\boldsymbol{\theta}}_1, \dot{\boldsymbol{\theta}}_2] \geq \frac{1}{\mathrm{poly}(\lambda)}.$$

Let us call all $\mathbf{r}, I, \mathbf{s}, \dot{\boldsymbol{\beta}}, \dot{\mathbf{u}}, \dot{\gamma}, \dot{\theta}_1, \dot{\theta}_2$ for which the above holds good. Lemma A.1 implies that with probability $1 - \mathsf{negl}(\lambda)$, Ext outputs $\widetilde{\mathbf{p}}, \widetilde{\mathbf{p}'}, \widetilde{\mathbf{q}}, \widetilde{\mathbf{q}'}$, such that $\mathsf{tc}_{\mathbf{p}}, \mathsf{tc}_{\mathbf{p}'}, \mathsf{tc}_{\mathbf{q}}, \mathsf{tc}_{\mathbf{q}'}$ are their commitments, respectively, using BaseFold's PCS. Again applying Lemma 2.2 to the set of good $\mathbf{r}, I, \mathbf{s}, \dot{\boldsymbol{\beta}}, \dot{\mathbf{u}}, \dot{\gamma}, \dot{\theta}_1, \dot{\theta}_2$ implies that for at least $\frac{1}{\mathrm{poly}(\lambda)}$ fraction of $\mathbf{r}, I, \mathbf{s}, \dot{\boldsymbol{\beta}}$, Steps 9-14 of Evaluate succeed with probability at least $\frac{1}{\mathrm{poly}(\lambda)}$. Thus the soundness of Spark and the sum-checks in Step 11 implies that $(\mathbf{p}, \mathbf{p}')H = 0$ and $(\mathbf{q}, \mathbf{q}')H = 0$. As H is the parity check matrix corresponding to the code with encoding matrix E_0, this means that $(\mathbf{p}, \mathbf{p}') = \mathbf{p}E_0$ and $(\mathbf{q}, \mathbf{q}') = \mathbf{q}E_0$.

We again apply Lemma 2.2 to get a $\frac{1}{\mathrm{poly}(\lambda)}$ fraction of $\mathbf{r}, I$ such that the sum-checks on Step 7 succeed with probability at least $\frac{1}{\mathrm{poly}(\lambda)}$; here the probability is over $\mathbf{s}, \dot{\boldsymbol{\beta}}$. Thus the soundness of sum-check and the fact that $\mathbf{s}$ is randomly chosen implies that $\langle \mathbf{r}, C_i \rangle = (\mathbf{p}, \mathbf{p}')_i$ and $\langle \boldsymbol{\alpha}, C_i \rangle = (\mathbf{q}, \mathbf{q}')_i$ for all $i \in I$. Because of the former, Lemma 2.4 implies that $J := |j : \exists i \text{ s.t. } C(i,j) \neq C'(i,\cdot)| \leq \frac{\delta_1}{3} \cdot \frac{\rho_1 n}{\ell}$, where $C' \in \mathbb{F}^{\ell \times \frac{n}{\ell}}$ is such that $C'(i,\cdot)$ is the unique codeword closest to $C(i,\cdot)$.

Now suppose that $(\mathbf{q}, \mathbf{q}') \neq \boldsymbol{\alpha}C'$. Then since $\Delta((\mathbf{q}, \mathbf{q}'), \boldsymbol{\alpha}C') \geq \delta_1 \cdot \frac{\rho_1 n}{\ell}$ and $\Delta(\boldsymbol{\alpha}C', \boldsymbol{\alpha}C) \leq \frac{\delta_1}{3} \cdot \frac{\rho_1 n}{\ell}$, $\Delta((\mathbf{q}, \mathbf{q}'), \boldsymbol{\alpha}C) \geq \frac{2\delta_1}{3} \cdot \frac{\rho_1 n}{\ell}$. Hence the probability that $(\mathbf{q}, \mathbf{q}')_i = \langle \alpha C(\cdot, i) \rangle$ for all $i \in I$ is at most $(1 - \frac{2\delta_1}{3})^{\Theta(\lambda)} = \mathsf{negl}(\lambda)$, a contradiction. Hence, $(\mathbf{q}, \mathbf{q}') = \boldsymbol{\alpha}C$. Define A' to be the matrix such that $A'(i,\cdot)E_0 = C'(i,\cdot)$ for all $i \in I$ and f' to be the polynomial whose coefficients are the entries of A. Then $f'(\dot{\boldsymbol{\alpha}}) = \alpha^T C' \alpha' = \langle \mathbf{q}, \boldsymbol{\alpha}' \rangle = \widetilde{\mathbf{q}}(\dot{\boldsymbol{\alpha}}')$. □

Lemma A.3. *Suppose that a PPT adversary $\mathcal{A}$ causes $\mathcal{V}$ to accept an evaliuation proof with probability $\frac{1}{\mathrm{poly}(\lambda)}$ on public input C, $\dot{\boldsymbol{\alpha}}$, y. Then there exists an extractor* Ext *running in time* $\mathrm{poly}(n, \lambda)$ *which outputs a multilinear polynomial $f \in \mathbb{F}[x_0, \ldots, x_{\ell-1}]$ such that C is the commitment to f computed using BaseFold PCS and $f(\dot{\boldsymbol{\alpha}}) = val$ with probability $1 - \mathsf{negl}(\lambda)$.*

Proof. From the hypothesis of the lemma, we have that,

$$\Pr_{\mathbf{r}, \boldsymbol{\rho}}[\mathcal{V} \text{ accepts}] \geq \frac{1}{\mathrm{poly}(\lambda)}$$

where $\boldsymbol{\rho}$ is the randomness used by rest of the evaluation procedure. Let $\mathcal{M} := \mathbb{F}^\ell$ and $\Phi : \mathcal{M}^* \to \{0,1\}$ be a predicate which such that $\Phi(\mathbf{r}_1, \ldots, \mathbf{r}_m) = 1$ if and only if $\mathbf{r}_1, \ldots, \mathbf{r}_m$ are linearly independent. Observe that $\Pr_{\mathbf{r}_m}[\Phi(\mathbf{r}_1, \ldots, \mathbf{r}_m) = 1 | \Phi(\mathbf{r}_1, \ldots, \mathbf{r}_{m-1}) = 1] \geq 1 - \frac{m}{|\mathbb{F}|} = 1 - \mathsf{negl}(\lambda)$. Further, let A be an algorithm such that $A(\mathbf{r}) = 1$ if and only if the verifier accepts. Note that $\Pr[A(\mathbf{r}) = 1] \geq \frac{1}{\mathrm{poly}(\lambda)}$ and from the proof of the above lemma, $(\mathbf{p}, \mathbf{p}') = \langle \mathbf{r}, C' \rangle$. Hence, Lemma 2.3 implies that there exists an extractor Ext_3 which given oracle access to A outputs linearly independent $\mathbf{r}_1, \ldots, \mathbf{r}_{\frac{n}{\ell}}$ such that $\forall u \in \frac{n}{\ell}$, the verifier accepts and therefore $(\mathbf{p}, \mathbf{p}') = \langle \mathbf{r}_u, C' \rangle$. Ext then extracts C' (defined in Lemma A.2) by solving a system of linear equations. The first $\frac{n}{\ell}$ columns of C' give A' (defined in the previous lemma). □

Proof of Theorem 3.1. We proved that BrakingBase is complete and succinct in Section 3.5. Lemma A.3 proves that it is also knowledge sound. □

B Parity check matrix for Brakedown's linear code

In this section we show that the parity check matrix for Brakedown's linear code is sparse. We first recall how their linear code is computed.

Let $0 < \alpha < 1$, $0 < \beta < \frac{\alpha}{1.28}$, $r > \frac{1+2\beta}{1-\alpha}$, $c_n, d_n > 3$ be some parameters. Let $\mathcal{G}_{n,m,c}$ denote the set of all bipartite graphs $G = (L, R, E)$ where $|L| = n, |R| = m$, and every vertex in L has exactly c neighbours in R. To encode a message of length n, we first define $A_n, A_{\alpha n}, \ldots$ as the bi-adjacency matrices of graphs sampled uniformly at random from $\mathcal{G}_{n,\alpha n,c_n}, \mathcal{G}_{\alpha n,\alpha^2 n,c_{\alpha n}}, \ldots$ and also define $B_n, B_{\alpha n}, \ldots$, as the bi-adjacency matrices of graphs sampled uniformly at random from $\mathcal{G}_{r\alpha n,(r-1-r\alpha)n,d_n}, \mathcal{G}_{r\alpha^2 n,(r-1-r\alpha)\alpha n,d_{\alpha n}}, \ldots$. Then matrices $M_n, M_{\alpha n}, \ldots$, $N_n, N_{\alpha n}, \ldots$ are created by replacing every non-zero entry in $A_n, A_{\alpha n}, \ldots$ and $B_n, B_{\alpha n}, \ldots$, respectively, by a random field element.

The code is defined recursively. For any $m \in \{n, \alpha n, \alpha^2 n, \ldots\}$, let $\mathsf{Enc}_m(\mathbf{x})$ denote the codeword corresponding to an $\mathbf{x} \in \mathbb{F}^m$. Then for an $\mathbf{x} \in \mathbb{F}^m$, $\mathsf{Enc}_m(\mathbf{x}) := (\mathbf{x}, \mathbf{z}, \mathbf{w}) \in \mathbb{F}^{rm}$ where $\mathbf{z} := \mathsf{Enc}_{\alpha m}(\mathbf{z}')$, $\mathbf{z}' := \mathbf{x}A_m$, $\mathbf{w} := \mathbf{z}B_m$. It is shown in [GLS+23] that there exist values of $c_n, c_{\alpha_n}, \ldots\ d_n, d_{\alpha n}, \ldots$ such that this code has rate $\frac{1}{r}$ and relative distance $\delta := \frac{\beta}{r}$. Moreover, they show that $c := \lim_{n\to\infty} c_n$ and $d := \lim_{n\to\infty} d_n$ are both $\Theta(1)$. Thus the time required to compute $\mathsf{Enc}_n(\mathbf{x}) = c_n n + d_n r\alpha n +$ time required to compute $\mathsf{Enc}_{\alpha n}(\mathbf{z}')$. Hence time required to compute $\mathsf{Enc}_n(\mathbf{x})$ is

$$\begin{aligned}
&(c_n \cdot n + c_{\alpha n} \cdot \alpha n + c_{\alpha^2 n} \cdot \alpha^2 n + \ldots) + (d_n \cdot r\alpha n + d_{\alpha n} \cdot r\alpha^2 n + d_{\alpha^2 n} \cdot r\alpha^3 n + \ldots) \\
&= (c_n + r\alpha \cdot d_n)n + (c_{\alpha n} + r\alpha \cdot d_{\alpha n})\alpha n + (c_{\alpha^2 n} + r\alpha \cdot d_{\alpha^2 n})\alpha^2 n + \cdots \\
&\approx (c + r\alpha \cdot d)(n + \alpha n + \alpha^2 n + \cdots) \\
&= \Theta(n).
\end{aligned}$$

Let $H_m \in \mathbb{F}^{rm \times (r-1)m}$ be the parity check matrix for the code with message length m.

Claim B.1. *For all $m \in \{n, \alpha n, \alpha^2 n, \ldots\}$, H_m looks like:*

$$\begin{pmatrix} A_m & 0_{m\times(r-1)\alpha m} & 0_{m\times(r-1-r\alpha)m} \\ -I'_{\alpha rm\times\alpha m} & H_{\alpha m} & B_m \\ 0_{(r-1-r\alpha)m\times\alpha m} & 0_{(r-1-r\alpha)m\times(r-1)\alpha m} & -I_{(r-1-r\alpha)m\times(r-1-r\alpha)m} \end{pmatrix},$$

where $0_{a\times b} \in \mathbb{F}^{a\times b}$ is the all zeros matrix, $I_{(r-1-r\alpha)m\times(r-1-r\alpha)m} \in \mathbb{F}^{(r-1-r\alpha)m\times(r-1-r\alpha)m}$ is the identity matrix, and $I'_{\alpha rm\times\alpha m} \in \mathbb{F}^{\alpha rm\times\alpha m}$ is obtained by appending $(r-1)\alpha m$ many all zero rows to the $\alpha m \times \alpha m$ identity matrix .

Proof. For an $\mathbf{x} \in \mathbb{F}^m$, let $\mathsf{Enc}_m(\mathbf{x}) = (\mathbf{x}, \mathbf{z}, \mathbf{w})$ and $\mathbf{z}' := \mathbf{x}A_m$. Observe that because of the way the code is defined, the first αm many entries of $\mathbf{z}$ are $\mathbf{z}'$. Then, it is easy to see that

$$\begin{pmatrix} \mathbf{x} & \mathbf{z} & \mathbf{w} \end{pmatrix} \begin{pmatrix} A_m & 0_{m\times(r-1)\alpha m} & 0_{m\times(r-1-r\alpha)m} \\ -I'_{\alpha rm\times\alpha m} & H_{\alpha m} & B_m \\ 0_{(r-1-r\alpha)m\times\alpha m} & 0_{(r-1-r\alpha)m\times(r-1)\alpha m} & -I_{(r-1-r\alpha)m\times(r-1-r\alpha)m} \end{pmatrix} = \begin{pmatrix} \mathbf{0} & \mathbf{z}H_{\alpha m} & \mathbf{0} \end{pmatrix} = \mathbf{0}$$

where the last equality follows from the fact that $H_{\alpha m}$ is the parity check matrix for a code with message length αm.

Conversely, suppose that for some $(\mathbf{x}, \mathbf{z}, \mathbf{w})$, where $\mathbf{x} \in \mathbb{F}^m$, $\mathbf{z} \in \mathbb{F}^{r\alpha m}$, $\mathbf{w} \in \mathbb{F}^{(r-1-r\alpha)m}$,

$$\begin{pmatrix} \mathbf{x} & \mathbf{z} & \mathbf{w} \end{pmatrix} \begin{pmatrix} A_m & 0_{m\times(r-1)\alpha m} & 0_{m\times(r-1-r\alpha)m} \\ -I'_{\alpha r m\times \alpha m} & H_{\alpha m} & B_m \\ 0_{(r-1-r\alpha)m\times\alpha m} & 0_{(r-1-r\alpha)m\times(r-1)\alpha m} & -I_{(r-1-r\alpha)m\times(r-1-r\alpha)m} \end{pmatrix} = \mathbf{0}.$$

Since $H_{\alpha m}$ is the parity check matrix for the code with message length αm, $\mathbf{z}$ is a valid codeword. Then $\mathbf{x}A_m - \mathbf{z}I'_{\alpha r m\times\alpha m} = \mathbf{0}$ implies that the first αm entries of $\mathbf{z}$ are $\mathbf{z}' := \mathbf{x}A_m$. Thus $\mathbf{z} = \mathsf{Enc}_{\alpha m}(\mathbf{z}')$. Also, $\mathbf{z}B_m - \mathbf{w}I_{(r-1-r\alpha)m\times(r-1-r\alpha)m} = \mathbf{0}$ means that $\mathbf{w} = \mathbf{z}B_m$. Hence $(\mathbf{x}, \mathbf{z}, \mathbf{w})$ is a valid codeword. □

For all $m \in \{n, \alpha n, \alpha^2 n, \dots\}$ let $s(m)$ denote the number of non-zero entries in H_m. Then,

$$\begin{aligned} s(n) &= c_n \cdot n + \alpha n + s(\alpha n) + d_n \cdot r\alpha n + (r-1-r\alpha)n \\ &= (c_n + r\alpha \cdot d_n)n + (r+\alpha-1-r\alpha)n + s(\alpha n) \\ &= (c_n + r\alpha \cdot d_n)n + (c_{\alpha n} + r\alpha \cdot d_{\alpha n})\alpha n + (c_{\alpha^2 n} + r\alpha \cdot d_{\alpha^2 n})\alpha^2 n + \cdots \\ &\quad + (r+\alpha-1-r\alpha)(n + \alpha n + \alpha^2 n + \cdots) \\ &\approx (c + r\alpha \cdot d + r + \alpha - 1 - r\alpha)(n + \alpha n + \alpha^2 n + \cdots) \\ &= \Theta(n). \end{aligned}$$

C The Spark sparse polynomial commitment scheme

In this section we give a brief overview of the Spark sparse polynomial commitment scheme proposed in [Set20]. Since we use Spark to commit to the multilinear extension of a sparse matrix we describe Spark in this case.

Let $M \in \mathbb{F}^{n\times n}$ be a matrix with only m non-zero entries. Let $\dot{\mathbf{x}}, \dot{\mathbf{y}} \in \mathbb{F}^n$ and $\widetilde{M} \in \mathbb{F}_{\leq 1}[\dot{\mathbf{x}}, \dot{\mathbf{y}}]$ be the multilinear extension of M. Fix any PCS and for any $k \in \mathbb{N}$, $n = 2^k$ let its commitment time and evaluation prover time for a polynomial in k variables be $T_c(n)$ and $T_e(n)$ respectively. Notice that committing to $\widetilde{M}$ directly using this PCS will require $T_c(n^2)$ time. Spark is a PCS with commit time $O(T_c(m) + T_c(n))$ and evaluation prover time $O(T_c(m) + T_e(m) + T_e(n))$.[9] Let $\mathrm{val}, \mathrm{row}, \mathrm{col} \in \mathbb{F}^m$ be such that the i-th coordinate of val is the i-th non-zero entry of M and the i-th coordinates of row and col are row index and column index of this entry, respectively. Let to-bits : $\mathbb{F} \to \{0,1\}^{\log|\mathbb{F}|}$ be a function mapping field elements to their binary representation. Then it is not difficult to see that

$$\widetilde{M}(\dot{\mathbf{x}}, \dot{\mathbf{y}}) = \sum_{\dot{\mathbf{k}} \in \{0,1\}^{\log m}} \widetilde{\mathrm{val}}(\dot{\mathbf{k}}) \cdot \widetilde{eq}(\dot{\mathbf{x}}, \text{to-bits}(\widetilde{\mathrm{row}}(\dot{\mathbf{k}}))) \cdot \widetilde{eq}(\dot{\mathbf{y}}, \text{to-bits}(\widetilde{\mathrm{col}}(\dot{\mathbf{k}}))).$$

Thus, to commit to $\widetilde{M}$, the prover can simply commit to $\widetilde{\mathrm{val}}, \widetilde{\mathrm{row}}, \widetilde{\mathrm{col}}$; this requires $3 \cdot T_c(m)$ time.

For any $\dot{\boldsymbol{\alpha}}, \dot{\boldsymbol{\beta}} \in \mathbb{F}^{\log n}$, the prover can prove that $\widetilde{M}(\dot{\boldsymbol{\alpha}}, \dot{\boldsymbol{\beta}}) = y$, the prover can

[9] For ease of exposition, we assume that m is a power of 2.

1. Commit to two multilinear polynomials $\widetilde{\text{erow}}$, $\widetilde{\text{ecol}}$ in $\log m$ variable each,
2. Prove to the verifier that $y = \sum_{\dot{\mathbf{k}} \in \{0,1\}^{\log m}} \widetilde{\text{val}}(\dot{\mathbf{k}}) \cdot \widetilde{\text{erow}}(\dot{\mathbf{k}}) \cdot \widetilde{\text{ecol}}(\dot{\mathbf{k}})$, and
3. Prove that for all $\dot{\mathbf{k}} \in \{0,1\}^{\log m}$, $\widetilde{\text{erow}}(\dot{\mathbf{k}}) = \widetilde{eq}(\dot{\boldsymbol{\alpha}}, \text{to-bits}(\widetilde{\text{row}}(\dot{\mathbf{k}})))$ and $\widetilde{\text{ecol}}(\dot{\mathbf{k}}) = \widetilde{eq}(\dot{\boldsymbol{\beta}}, \text{to-bits}(\widetilde{\text{col}}(\dot{\mathbf{k}})))$.

Item 2 can be proved using a sum-check protocol while Item 3 is proved using offline memory checking that we now describe. We only focus on proving $\widetilde{\text{erow}}(\dot{\mathbf{k}}) = \widetilde{eq}(\dot{\boldsymbol{\alpha}}, \text{to-bits}(\widetilde{\text{row}}(\dot{\mathbf{k}})))$ since the other expression can be proved in a similar manner.

Consider a memory will n cells which are indexed by elements in $\mathbb{F}$ and whose i-th cell contains $\widetilde{eq}\left(\dot{\boldsymbol{\alpha}}, \text{to-bits}(i)\right)$. The prover wants to prove that for all $\dot{\mathbf{k}} \in \{0,1\}^{\log m}$, $\widetilde{\text{erow}}(\dot{\mathbf{k}})$ was obtained by reading from the $\widetilde{\text{row}}(\dot{\mathbf{k}})$-th location of this memory. [Set20] considers a process in which there is a read time-stamp associated with every cell in the memory and whenever a cell is read, its read time-stamp is incremented by 1. Let read-ts $\in \mathbb{F}^m$ be the vector whose $\dot{\mathbf{k}}$-th entry is the read time-stamp of to-bits$(\widetilde{\text{row}}(\dot{\mathbf{k}}))$ at the time of the $\dot{\mathbf{k}}$-th read. Let final-ts $\in \mathbb{F}^n$ be the vector containing the final values of all the read time-stamps. Then [Set20] shows that it is enough to prove that $WS = RS \cup S$, where

$$WS := \left\{\left(i, \widetilde{eq}\left(\dot{\boldsymbol{\alpha}}, \text{to-bits}(i)\right), 0\right) : i \in [n]\right\} \cup \left\{\left(\text{row}(\dot{\mathbf{k}}), \widetilde{\text{erow}}(k), \widetilde{\text{read-ts}}(\dot{\mathbf{k}}) + 1\right) : \dot{\mathbf{k}} \in \{0,1\}^{\log m}\right\},$$

$$RS := \left\{\left(\text{row}(\dot{\mathbf{k}}), \widetilde{\text{erow}}(k), \widetilde{\text{read-ts}}(\dot{\mathbf{k}})\right) : \dot{\mathbf{k}} \in \{0,1\}^{\log m}\right\}$$

$$S := \left\{\left(i, \widetilde{eq}\left(\dot{\boldsymbol{\alpha}}, \text{to-bits}(i)\right), \widetilde{\text{final-ts}}(\text{to-bits}(i))\right) : i \in [n]\right\}.$$

$WS = RS \cup S$ can be proved by showing that for variables x, y, $\prod_{(a,b,c) \in WS}(a + bx + cx^2 - y)$ and $\prod_{(a,b,c) \in RS}(a + bx + cx^2 - y) \cdot \prod_{(a,b,c) \in S}(a + bx + cx^2 - y)$ are the same polynomials. This is established by picking $\gamma, \tau \in_R \mathbb{F}$ and proving using a grand-product check that $\prod_{(a,b,c) \in WS}(a + b\gamma + c\gamma^2 - \tau) = \prod_{(a,b,c) \in RS}(a + b\gamma + c\gamma^2 - \tau) \cdot \prod_{(a,b,c) \in S}(a + b\gamma + c\gamma^2 - \tau)$. Notice that the verifier can evaluate all the three polynomials involved in the above expression if it is provided with commitments to read-ts and final-ts. Thus the prover also commits to $\widetilde{\text{read-ts}_{\text{row}}}$, $\widetilde{\text{final-ts}_{\text{row}}}$, $\widetilde{\text{read-ts}_{\text{col}}}$, $\widetilde{\text{final-ts}_{\text{col}}}$ along with $\widetilde{\text{val}}, \widetilde{\text{row}}, \widetilde{\text{col}}$ in the commitment phase.

Observe that since the read time-stamps are incremented by 1 after every read, we must have that the characteristic of $\mathbb{F}$ is at least the number of m. So the PCS described above is not field agnostic. However there is an easy way to fix this: let g be the generator of the multiplicative group of $\mathbb{F}$. Then initialize all the read time-stamps to g and after every read, multiply the current read time-stamp by g. Now we only require that $|\mathbb{F}| \geq m$.