Linear-time and Logarithmically-sound Permutation and Multiset SNARKs

Bing-Jyue Chen, Lilia Tang, David Heath, and Daniel Kang
{bjchen4,liliat2,daheath,ddkang}@illinois.edu
University of Illinois Urbana-Champaign

Abstract

Permutation and multiset checks underpin many SNARKs, yet existing techniques either incur super-linear prover time or rely on auxiliary commitments with soundness error that grows linearly in the input size. We present new arguments with *linear-time* provers and *logarithmic* soundness, without auxiliary commitments.

Prior work achieving logarithmic soundness error arithmetizes the permutation as a product of several multilinear polynomials, a formulation chosen for compatibility with the classic Sumcheck PIOP. A simpler alternative treats permutations as multilinear extensions of their permutation matrices. While this formulation was previously believed to require *quadratic prover time*, we show that this overhead can be eliminated by taking a linear-algebraic perspective. This viewpoint has a key advantage: partially evaluating the multilinear polynomial of the permutation requires no additional field operations and amounts to applying the inverse permutation to the verifier's challenge vector. This makes the step essentially *free* in terms of algebraic cost, unlike in prior approaches. Compared to concurrent work BiPerm (Bünz et al., ePrint Archive, 2025), our scheme requires no permutation preprocessing and supports prover-supplied permutations.

We show a sparsity-aware PCS like Dory (Lee, TCC, 2021) can compile our PIOP to a SNARK such that the resulting SNARK prover still runs in time $O(n)$. Our construction is the first logarithmically-sound SNARK with an $O(n)$-time prover for both permutation and multiset checks. We further prove a matching optimal prover lower bound, and we identify specific permutations that can be evaluated by the verifier in $O(\text{polylog}(n))$-time. The ability to evaluate these permutations in $O(\text{polylog}(n))$ time allows the verifier to avoid relying on prover-supplied commitments or evaluation proofs. As a result, we obtain the first logarithmically sound, field-agnostic SNARK with an $O(n)$-time prover in this setting.

Keywords: Permutation Proofs, Polynomial IOPs, SNARKs, Sumcheck

Contents

1 Introduction 3

2 Preliminaries 4
- 2.1 Notation . . . 5
- 2.2 Multilinear Extensions . . . 5
- 2.3 Interactive Proofs and Arguments of Knowledge . . . 6
- 2.4 Polynomial Interactive Oracle Proofs . . . 7
- 2.5 Polynomial Commitment Schemes . . . 8
- 2.6 PIOP Compilation . . . 9
- 2.7 The Sumcheck PIOP . . . 9
- 2.8 Permutation as Sumcheck PIOP . . . 11

3 A Linear-Time Lower Bound for Prover 12
- 3.1 Lower Bound for Permutation Check . . . 13
- 3.2 Lower Bound for Multiset Check . . . 14

4 SNARKs for Permutation and Multiset Checks 15
- 4.1 The Core Permutation Check . . . 15
- 4.2 Structural Checks . . . 17
- 4.3 PIOP Compilation . . . 19

5 Extensions to Structured or Non-Bijective Permutations 20
- 5.1 Public and Structured Permutations . . . 20
  - 5.1.1 Definition and evaluation cost . . . 20
  - 5.1.2 Generic circuits as structured permutations . . . 21
  - 5.1.3 Common structured permutations . . . 22
- 5.2 Non-Bijective Permutations . . . 22

6 Related Work 23

7 Conclusion 24

A Computation of Lagrange Basis Polynomials 28

B Simplified Booleanity Check 29

C The Sumcheck PIOP 31

1 Introduction

Succinct non-interactive arguments of knowledge (SNARKs) allow a prover to convince a verifier that an agreed-upon algorithm was executed correctly on a prover-supplied witness. They enable the verifier to validate the claim without re-running the entire computation. A zero-knowledge SNARK (zkSNARK) additionally hides the prover's witness. Thanks to decades of research, SNARKs are now efficient enough for real-world applications including ZKML [1], ZKVM [2], and ZK-Rollups [3].

Permutation and multiset checks are core building blocks in many SNARKs [4, 5]. These primitives ensure that two collections of field elements agree up to reordering (and, for multisets, multiplicities), which is essential for enforcing copy constraints [4], wiring relations [6], and lookup arguments [4, 7]. Despite their importance, existing techniques suffer from notable drawbacks: they either have superlinear prover time [4,8], or they require commitments to large auxiliary data and incur a soundness error that grows linearly with the input size [4]. Concurrent work *BiPerm* [8] presents a permutation check with a logarithmically-scaling (in the size of the two collections) soundness error and a linear-time prover. However, BiPerm requires the permutation to be preprocessed by an indexer (i.e., a party that runs a setup algorithm before the interaction between the prover and the verifier), and it does not support structural checks over the permutation (it provides a Booleanity check but lacks a bijectivity check). Consequently, when preprocessing is disallowed or when the permutation is part of the prover's witness, there is no known way to obtain a linear-time prover in this setting without extra commitments and poor soundness.

In pursuit of a logarithmically-sound permutation check, prior work [4,8] arithmetizes each n by n permutation as the product of multiple multilinear polynomials. This approach is intuitive, because it allows one to invoke an efficient subroutine, Sumcheck [9], which is designed to work on multivariate polynomials. For instance, HyperPlonk [4] proposes to view a permutation as a product of $\log n$ $(\log n + 1)$-variate multilinear polynomials. In [8], BiPerm (resp. MulPerm) treats each permutation as the product of two $3 \log n/2$-variate multilinear polynomials (resp. the product of $\sqrt{\log n}$ $(\log n + \sqrt{\log n})$-variate multilinear polynomials).

It is natural to consider whether it is possible to arithmetize a permutation with *only one* $2 \log n$-variate multilinear polynomial, which is exactly the multilinear extension of the n by n permutation matrix. At first glance, this seems impossible. Sumcheck [9] requires us to partially evaluate the $2 \log n$-variate multilinear polynomial in its first $\log n$ variables, and this will cost $O(n^2)$ time in general.

In this work, we reformulate the above partial polynomial evaluation as a matrix-vector product between the transpose of the permutation matrix M and a verifier-provided challenge vector. Since a permutation matrix is orthogonal, we have $M^\top = M^{-1}$. Therefore, multiplying $M^\top$ by a vector simply permutes its entries according to σ^{-1}. This takes *no* field operations instead of the usual $O(n^2)$ cost.

Based on this insight, we propose new permutation and multiset checks that achieve the best of both worlds: a linear-time prover without committing to auxiliary data, and a soundness error that grows only logarithmically with the input size. Compared to BiPerm, our formulation has several advantages. First, it requires no preprocessing of the permutation. This yields, to our knowledge, the first logarithmically-sound SNARK for the multiset check with a linear-time prover, since the witness permutation linking two multisets is unknown to the verifier. Second, prior approaches typically convert the permutation into the product of multiple multilinear polynomials [4, 8]. This arithmetization introduces both algebraic overhead and commitments to the resulting polynomials. In contrast, our prover never constructs such a polynomial. Instead, it works with the permutation directly in its natural representation (i.e., to permute a length-n vector, a permutation σ can be represented by $\{(i, j) \mid \sigma(i) = j,\ \forall i \in \{0, 1, \ldots, n-1\}\}$). In certain cases, we can even eliminate the permutation commitment and therefore obtain a *field-agnostic* permutation check. Finally, we improve both prover and verifier time concretely for permutation checks: BiPerm invokes a linear-time Sumcheck polynomial interactive oracle proof (PIOP) [4, 9] over a product of three multilinear polynomials, whereas ours uses two; BiPerm requires the verifier to query four oracles, whereas ours uses three. These choices concretely reduce prover/verifier time and proof size.

Technical overview

Let a bijective permutation be σ, which can be applied to any two length-n vectors A, B such that $A_{\sigma(i)} = B_i$, $\forall i \in \{0, 1, \ldots, n-1\}$. We begin by building a PIOP for permutation check. We arithmetize it directly as a matrix-vector product $B = MA$, where M is an n by n permutation matrix defined by $M_{ij} = 1$, if $\sigma(i) = j$ and $M_{ij} = 0$, otherwise. Viewing them as multilinear extensions (i.e., $\log n$-variate multilinear polynomials $\widetilde{A}(y), \widetilde{B}(x)$, and $2 \log n$-variate multilinear polynomial $\widetilde{M}(x, y)$), the classic $O(n)$-time Sumcheck PIOP [4,9] checks if $B = MA$ by checking whether

$$\widetilde{B}(r) = \sum_{y \in \{0,1\}^{\log n}} \widetilde{M}(r, y)\widetilde{A}(y)$$

where $r \in \mathbb{F}^{\log n}$ is the challenge given by the verifier. It then reduces verification to queries to each of the three oracles of these multilinear extensions. Before invoking Sumcheck, we can see that the prover must compute $\widetilde{B}(r)$ and $\widetilde{M}(r, y)$, and computing the latter costs $O(n^2)$ field operations in general. However, given a length-n challenge vector s computed from $r \in \mathbb{F}^{\log n}$ in $O(n)$ time using a known algorithm [10], we show that this computation is equivalent to computing $\widetilde{B}(r) = s^\top B$ and $\widetilde{M}(r, y)$ can be viewed as $s^\top M = (M^\top s)^\top$. Leveraging the property of a permutation matrix (i.e., $M^\top = M^{-1}$), computing $M^\top s = M^{-1} s$ costs *no* field operation because the prover simply permutes s according to σ^{-1}.

To support permutations as part of the prover's witness (e.g., in multiset check), we add structural checks to certify that the matrix is a permutation. Following the linear-algebraic definition [11] and building on structural checks from [12], we obtain $O(n)$ prover time thanks to sparsity-aware Sumcheck techniques [13]. Since there exists sparsity-aware polynomial commitment schemes (PCSs) such as Dory [14], we can compile our PIOPs with them to obtain SNARKs with $O(\log n)$ soundness error and $O(n)$-time provers. Beyond the general case, we show that certain special permutations require no commitment to the permutation matrix because the verifier can evaluate its multilinear extension in $O(\mathsf{polylog}(n))$ time. This makes the permutation SNARK field-agnostic, allowing the use of concretely efficient polynomial commitment schemes such as [15] and enabling small-field optimizations like [16] in our permutation check. To our knowledge, this is the first logarithmically sound, field-agnostic SNARK with an $O(n)$-time prover in this setting.

Our contributions are threefold:

1. We design logarithmically-sound SNARKs with linear-time provers for both permutation and multiset checks, which concretely outperform the concurrent BiPerm construction across prover time, verifier time, and proof size.

2. We prove a matching lower bound showing that the resulting prover complexity is optimal.

3. We show that permutation checks can be made field-agnostic and concretely efficient for certain special permutations.

2 Preliminaries

This section reviews the tools and models used throughout the paper. Section 2.1 sets up notation. Section 2.2 introduces multilinear extensions, which form the algebraic backbone of our constructions and serve as the data carriers in Sumcheck-based proving systems. Sections 2.3–2.6 summarize the proof frameworks we build upon, including interactive proofs, polynomial IOPs, and their compilation with polynomial commitment schemes, which together reflect the standard approach to constructing modern SNARKs. Section 2.7 reviews the classic Sumcheck PIOP, the fundamental building block of our permutation and multiset PIOPs. Section 2.8 shows how permutation checks can be expressed within this framework. These definitions and abstractions will be used directly in our construction and analysis.

2.1 Notation

We use λ to denote the security parameter. For $a, b \in \mathbb{N}$, the notation $[a, b]$ denotes the set $\{a, a+1, \ldots, b\}$. A function $f(\lambda)$ is in $\text{poly}(\lambda)$ if there exists $c \in \mathbb{N}$ such that $f(\lambda) = O(\lambda^c)$. A function $f(\lambda)$ is *negligible*, written $f(\lambda) \in \text{negl}(\lambda)$, if for every $c \in \mathbb{N}$, we have $f(\lambda) = o(\lambda^{-c})$. A probability of the form $1 - \text{negl}(\lambda)$ is said to be *overwhelming*. We use $\mathbb{F}$ to denote a finite field of prime order p, where $\log(p) = \Omega(\lambda)$. Our technique may be used on top of non prime order field such as binary field [17], but this is out of the scope of this paper.

A *multiset* is a set-like collection in which each element has positive multiplicity. Two finite multisets are equal if and only if they contain the same elements with corresponding multiplicities.

An *indexed relation* is a set of triples $(\mathbf{i}; \mathbf{x}; \mathbf{w})$, where $\mathbf{i}$ is an index, $\mathbf{x}$ is the public input, and $\mathbf{w}$ is the prover's witness. The index $\mathbf{i}$ is fixed during setup.

In describing the syntax of our protocols, we follow the convention that public values (known to the prover and the verifier) are distinguished from the private witness (known only to the prover).

Throughout the paper, we use n to denote the size of the input to the permutation. Unless stated otherwise, we assume the permutation is bijective, so the output size is also n. When encoding a length-n vector as a multilinear extension, it has $\log n$ variables (see Sec 2.2 for details) by construction. All logarithms in this paper are base 2.

Any non-negative integer j can be written in binary as $(j_1, j_2, \ldots, j_{\log n})$, where j_1 is the *least significant bit* and $j_{\log n}$ is the *most significant bit*. Hence, we freely move between the equivalent views

$$j \in [0, n-1] \quad \text{and} \quad j = (j_1, j_2, \ldots, j_{\log n}) \in \{0,1\}^{\log n}.$$

For readability, we typically omit the binary tuple when the meaning is clear.

We denote the oracle corresponding to a polynomial f by $[[f]]$, which Sec. 2.4 describes in more detail.

Following [12], any field multiplication and addition is considered to take $O(1)$ time.

2.2 Multilinear Extensions

We introduce multilinear extensions, along with their definitions, properties, and common use cases, as they will be needed throughout the paper.

Definition 1 (Multilinear polynomials). *A μ-variate polynomial $g : \mathbb{F}^\mu \to \mathbb{F}$ is said to be multilinear if g's degree in each variable is at most one.*

Lemma 2.1 (Multilinear extensions). *For any function $f : \{0,1\}^\mu \to \mathbb{F}$, there exists a unique multilinear polynomial $\widetilde{f} : \mathbb{F}^\mu \to \mathbb{F}$ such that $\widetilde{f}(x) = f(x)$ for all $x \in \{0,1\}^\mu$ (i.e., boolean hypercube) [12]. The multilinear $\widetilde{f}$ is called the multilinear extension of f. This implies that for any two multilinear polynomials g and h, if $g(x) = h(x)$, $\forall x \in \{0,1\}^\mu$, then g and h are the same polynomial.*

Lemma 2.2 (Schwartz-Zippel Lemma). *Let $f : \mathbb{F}^\mu \to \mathbb{F}$ be a non-zero polynomial of total degree d. Let S be any finite subset of $\mathbb{F}$, and let $r_1, \ldots, r_\mu$ be μ field elements selected independently and uniformly from S. Then*

$$\Pr\left[f\left(r_1, \ldots, r_\mu\right) = 0\right] \leq \frac{d}{|S|}$$

We next review several useful multilinear extensions.

Multilinear extension for Lagrange basis of multilinear polynomials. The first special multilinear extension $\widetilde{\text{eq}}$ is the multilinear extension of the function $\text{eq} : \{0,1\}^\mu \times \{0,1\}^\mu \to \mathbb{F}$ defined as follows:

$$\text{eq}(x, y) = \begin{cases} 1 & \text{if } x = y \\ 0 & \text{otherwise} \end{cases}$$

$\widetilde{\text{eq}}$ can be explicitly expressed as :

$$\widetilde{\mathrm{eq}}(x,y)=\prod_{i=1}^{\mu}\left(x_i y_i+(1-x_i)(1-y_i)\right) \tag{1}$$

$\widetilde{\mathrm{eq}}(r,r')$ can be evaluated at any point $(r,r')\in\mathbb{F}^{\mu}\times\mathbb{F}^{\mu}$ in $O(\mu)$ time. Following prior work [12,13], any $O(\log n)$-variate polynomial that can be evaluated in $O(\log n)$ time (instead of $O(n)$) is *structured*, hence $\widetilde{\mathrm{eq}}$ is structured.

Based on $\widetilde{\mathrm{eq}}$, any μ-variate function f and its multilinear extensions $\widetilde{f}$ have the following relations:

$$\widetilde{f}(y)=\sum_{\boldsymbol{b}\in\{0,1\}^{\mu}} f(\boldsymbol{b})\cdot\widetilde{\mathrm{eq}}(\boldsymbol{b},y)$$

The above holds because (1) both $\widetilde{f}(y)$ and $\sum_{\boldsymbol{b}\in\{0,1\}^{\mu}} f(\boldsymbol{b})\cdot\widetilde{\mathrm{eq}}(\boldsymbol{b},y)$ are multilinear polynomials in y, and (2) they agree on all Boolean inputs $y\in\{0,1\}^{\mu}$. Because of the relation, $\widetilde{\mathrm{eq}}$ is often called the Lagrange basis of multilinear polynomials [18].

Multilinear extensions for vectors. Given a vector $v\in\mathbb{F}^n$, one can construct its unique multilinear extension $\widetilde{v}$. Specifically, let v_j denote the j-th element in v, $\forall j\in[0,n-1]$. We can represent the extension naturally by $\widetilde{v}(j_1,j_2,\ldots,j_{\log n})=v_j$, where $(j_1,j_2,\ldots,j_{\log n})$ is the binary representation of j. We can use $\widetilde{\mathrm{eq}}$ to express $\widetilde{v}$ explicitly:

$$\widetilde{v}(y)=\sum_{j\in\{0,1\}^{\log n}} v_j\cdot\widetilde{\mathrm{eq}}(j,y) \tag{2}$$

Given a fixed $y\in\mathbb{F}^{\log n}$, the multilinear polynomial $\widetilde{\mathrm{eq}}(j,y)=\prod_{i=1}^{\log n}\left(j_i y_i+(1-j_i)(1-y_i)\right)$ is the j-th multilinear Lagrange basis polynomial for every $j\in\{0,1\}^{\log n}$. It is well known that given a random point $r\in\mathbb{F}^{\log n}$, we can evaluate all Lagrange basis polynomials (i.e., $\widetilde{\mathrm{eq}}(j,r)$, $\forall j\in\{0,1\}^{\log n}$) with only n field multiplications [10].[1] Therefore, given a vector $v\in\mathbb{F}^n$, it takes $2n$ field multiplications to compute $\widetilde{v}(r)$ (i.e., n multiplications for computing all Lagrange basis polynomials and n multiplications for the inner product between v_j and $\widetilde{\mathrm{eq}}(j,r)$, $\forall j\in\{0,1\}^{\log n}$).

Multilinear extensions for matrices. Given an m by n matrix $M\in\mathbb{F}^{m\times n}$, one can use $\widetilde{M}$ as its unique multilinear extension. Similar to how we encode vectors as multilinear extensions, $\forall i\in[0,m-1], j\in[0,n-1]$ we can encode M by $\widetilde{M}(i_1,\ldots,i_{\log m},j_1,\ldots,j_{\log n})=M_{ij}$. It can also be expressed explicitly using $\widetilde{\mathrm{eq}}$:

$$\widetilde{M}(x,y)=\sum_{i\in\{0,1\}^{\log m},\, j\in\{0,1\}^{\log n}} M_{ij}\widetilde{\mathrm{eq}}(j,y)\widetilde{\mathrm{eq}}(i,x) \tag{3}$$

2.3 Interactive Proofs and Arguments of Knowledge

Definition 2 (Interactive Proof)**.** *A pair of interactive machines $(\mathcal{P},\mathcal{V})$ form an interactive proof (IP) for a relation $\mathcal{R}$ with completeness error ε and soundness error δ ($0\le\varepsilon,\delta<1$) when the following properties hold:*

- ***Completeness****: for every x such that $x\in\mathcal{R}$ holds, then* $\Pr[\mathcal{V}$ *outputs 1*$]\ge 1-\varepsilon$*. An IP has perfect completeness when $\varepsilon=0$.*
- ***Soundness****: for any x such that $x\notin\mathcal{R}$ and for any prover $\mathcal{P}^*$, then* $\Pr[\mathcal{V}$ *outputs 1*$]\le\delta$*.*

Given a relation $\mathcal{R}$, we can define a circuit C such that $C(x)=1$ iff $x\in\mathcal{R}$; and $C(x)=0$, otherwise. In this context, we say an IP is succinct when the verifier time and the proof size are both $O(\mathrm{polylog}(|C|,|x|))$.

We next define interactive proofs of knowledge, which consist of (1) a non-interactive preprocessing phase run by a third-party trusted setup algorithm, often called the indexer [4], and (2) an interactive online phase between the prover and the verifier.

[1] We refer readers to Appendix A for the detailed algorithm.

Definition 3 (Interactive Proof and Argument of Knowledge (adapted from [4]))**.** *An interactive protocol* $\Pi = (\mathit{Setup}, \mathcal{I}, \mathcal{P}, \mathcal{V})$ *is an argument of knowledge for an indexed relation* $\mathcal{R}$ *with knowledge error* $\delta : \mathbb{N} \to [0,1]$ *if the following conditions hold. Given an index* $\mathbf{i}$*, public input* $\mathbf{x}$*, and prover witness* $\mathbf{w}$*, the deterministic indexer computes*

$$(\mathsf{vp}, \mathsf{pp}) \leftarrow \mathcal{I}(\mathbf{i}),$$

and the verifier's output is the random variable

$$\langle \mathcal{P}(\mathsf{pp}, \mathbf{x}, \mathbf{w}),\ \mathcal{V}(\mathsf{vp}, \mathbf{x}) \rangle.$$

- ***Perfect Completeness:*** *for all* $(\mathbf{i}, \mathbf{x}, \mathbf{w}) \in \mathcal{R}$

$$\Pr\left[\langle \mathcal{P}(\mathsf{pp}, \mathbf{x}, \mathbf{w}), \mathcal{V}(\mathsf{vp}, \mathbf{x}) \rangle = 1 \;\middle|\; \begin{array}{l} \mathsf{gp} \leftarrow \mathit{Setup}\left(1^\lambda\right) \\ (\mathsf{vp}, \mathsf{pp}) \leftarrow \mathcal{I}(\mathsf{gp}, \mathbf{i}) \end{array} \right] = 1$$

- δ***-Soundness (adaptive):*** *Let* $\mathcal{L}(\mathcal{R})$ *be the language corresponding to the indexed relation* $\mathcal{R}$ *such that* $(\mathbf{i}, \mathbf{x}) \in \mathcal{L}(\mathcal{R})$ *if and only if there exists* $\mathbf{w}$ *such that* $(\mathbf{i}, \mathbf{x}, \mathbf{w}) \in \mathcal{R}$*.* Π *is* δ*-sound if for every pair of probabilistic polynomial time adversarial prover algorithm* $(\mathcal{A}_1, \mathcal{A}_2)$ *the following holds:*

$$\Pr\left[\begin{array}{c} \langle \mathcal{A}_2(\mathbf{i}, \mathbf{x}, \mathsf{st}), \mathcal{V}(\mathsf{vp}, \mathbf{x}) \rangle = 1 \\ \wedge (\mathbf{i}, \mathbf{x}) \notin \mathcal{L}(\mathcal{R}) \\ \end{array} \;\middle|\; \begin{array}{l} \mathsf{gp} \leftarrow \mathit{Setup}\left(1^\lambda\right) \\ (\mathbf{i}, \mathbf{x}, \mathsf{st}) \leftarrow \mathcal{A}_1(\mathsf{gp}) \\ (\mathsf{vp}, \mathsf{pp}) \leftarrow \mathcal{I}(\mathsf{gp}, \mathbf{i}) \end{array} \right] \le \delta(|\mathbf{i}| + |\mathbf{x}|) \tag{4}$$

- δ***-Knowledge Soundness:*** *There exists a polynomial poly*$(\cdot)$ *and a probabilistic polynomial-time oracle machine* $\mathcal{E}$ *called the extractor such that given oracle access to any pair of probabilistic polynomial time adversarial prover algorithm* $(\mathcal{A}_1, \mathcal{A}_2)$ *the following holds:*

$$\Pr\left[\begin{array}{c} \langle \mathcal{A}_2(\mathbf{i}, \mathbf{x}, \mathsf{st}), \mathcal{V}(\mathsf{vp}, \mathbf{x}) \rangle = 1 \\ \wedge (\mathbf{i}, \mathbf{x}, \mathbf{w}) \notin \mathcal{R} \\ \mathbf{w} \leftarrow \mathcal{E}^{\mathcal{A}_1, \mathcal{A}_2}(\mathsf{gp}, \mathbf{i}, \mathbf{x}) \end{array} \;\middle|\; \begin{array}{l} \mathsf{gp} \leftarrow \mathit{Setup}(1^\lambda) \\ (\mathbf{i}, \mathbf{x}, \mathsf{st}) \leftarrow \mathcal{A}_1(\mathsf{gp}) \\ (\mathsf{vp}, \mathsf{pp}) \leftarrow \mathcal{I}(\mathsf{gp}, \mathbf{i}) \end{array} \right] \le \delta(|\mathbf{i}| + |\mathbf{x}|) \tag{5}$$

A protocol is an argument of knowledge *if* δ *is negligible in* λ*. If the adversary is computationally unbounded (i.e., the adversary is not restricted to polynomial time), the protocol is called an* interactive proof of knowledge.

- **Public coin:** An interactive protocol is public coin if every verifier message (including the final output) is a deterministic function of a publicly sampled random string.
- **Zero knowledge:** An interactive protocol $\langle \mathcal{P}, \mathcal{V} \rangle$ is zero knowledge if there exists a PPT simulator $\mathcal{S}$ such that, for every PPT adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ and every auxiliary input $z \in \{0,1\}^{\mathrm{poly}(\lambda)}$, the simulated and real transcripts are computationally indistinguishable.

We use both soundness and knowledge soundness. Knowledge soundness implies soundness, since the existence of an extractor guarantees $(\mathbf{i}, \mathbf{x}) \in \mathcal{L}(\mathcal{R})$. Moreover, in Lemma 2.3 we recall that, for certain oracle arguments and relations, soundness in fact implies knowledge soundness.

2.4 Polynomial Interactive Oracle Proofs

We now define polynomial interactive oracle proofs, an information-theoretic proof model tailored to polynomial relations.

Definition 4 (Polynomial Interactive Oracle Proof (PIOP) (adapted from [4]))**.** *A PIOP is a public-coin interactive proof for a polynomial oracle relation*

$$\mathcal{R} = \{(\mathbf{i}; \mathbf{x}; \mathbf{w})\}.$$

The relation is an oracle relation *in the sense that* $\mathbf{i}$ *and* $\mathbf{x}$ *may include oracles to* μ*-variate polynomials over a field* $\mathbb{F}$*. For each oracle, the arity* μ *and the degree in each variable are known. The verifier may query such an oracle at any point in* $\mathbb{F}^\mu$ *to obtain the evaluation of the underlying polynomial at that point. The actual polynomials are contained in the prover's parameters and witness. In every message, the prover* $\mathcal{P}$ *sends new multivariate polynomial oracles, and in every round the verifier sends a random challenge.*

Following [4], we evaluate the efficiency of a PIOP in terms of:

- **Prover time:** the total running time of the prover.
- **Verifier time:** the total running time of the verifier.
- **Query complexity:** the number of oracle queries made by the verifier.

Proof of knowledge. A PIOP satisfies perfect completeness and unbounded knowledge soundness with knowledge error δ. The extractor may query polynomials at any desired points; in particular, this allows efficient recovery of the entire polynomial by interpolation when needed.

From interactivity to non-interactivity. Public-coin interactive arguments can be made non-interactive via the Fiat–Shamir transform [19], which replaces the verifier's random challenges with hashes of the transcript so far.

Lemma 2.3 (Sound PIOPs are knowledge sound [4])**.** *Let* $\mathcal{R}$ *be an oracle relation, and consider a* δ*-sound PIOP for* $\mathcal{R}$ *such that for every* $(\mathbf{i};\mathbf{x};\mathbf{w}) \in \mathcal{R}$*, the witness* $\mathbf{w}$ *consists only of polynomials whose oracles appear in the instance* $(\mathbf{i},\mathbf{x})$*. Then the protocol has* δ *knowledge soundness, and there is an extractor running in time* $O(|\mathbf{w}|)$*.*

2.5 Polynomial Commitment Schemes

A polynomial commitment scheme (PCS) [14, 18, 20] allows an untrusted prover to succinctly commit to a polynomial f, and later provide the verifier with an evaluation $f(r)$ at a point r chosen by the verifier, together with a proof that this value is consistent with the committed polynomial [4, 12]. This is precisely the cryptographic primitive needed to transform a polynomial IOP into a succinct argument as described in the next subsection. Instead of sending the full polynomial f to the verifier, as in a PIOP, the prover in the argument system commits to f and later reveals only those evaluations that the verifier needs in order to carry out its checks.

Several existing PCSs [18, 20, 21] support committing to a $(\log n)$-variate multilinear polynomial f with *linear* prover time and *sublinear* verifier time in the total number of evaluations of $f(x)$ over $x \in \{0,1\}^{\log n}$. In this work, we focus on PCSs where the prover time depends only on the *sparsity* of f, rather than on the full hypercube size. The concrete design of sparsity-aware PCSs is an active research direction since [12], which is out of the scope of this paper.

Following [8, 12], we use *Dory* [14], a pairing-based, transparent PCS. If m is the number of nonzero evaluations of f over its domain $\{0,1\}^{\log n}$, then:

- committing to f costs $O(m)$ time,
- opening at a point $r \in \mathbb{F}^{\log n}$ costs $O(\sqrt{n})$ prover time,
- both the proof size and verifier time are $O(\log n)$.

To use Dory, the field must be compatible with a pairing-friendly elliptic curve, such as BLS12-381 [22] or BN254 [23].

2.6 PIOP Compilation

PIOP compilation transforms an interactive oracle proof into an interactive argument of knowledge (without oracles) Π. The compilation replaces each oracle with a polynomial commitment, and every verifier query is replaced with an invocation of the Eval protocol at the query point $\mathbf{z}$. The compiled verifier accepts if the original PIOP verifier would accept and if all Eval calls return 1. If Π is public-coin, it can be further compiled into a non-interactive argument of knowledge (NARK) using the Fiat–Shamir transform. If the proof size is sublinear and the verifier runs in time sublinear in the size of the witness, then the resulting system is a SNARK (i.e., a succinct NARK).

Theorem 2.1 (PIOP Compilation (adapted from [4]))**.** *If the polynomial commitment scheme Γ has witness-extended emulation, and if the t-round polynomial IOP for $\mathcal{R}$ has negligible knowledge error, then the compiled protocol Π is a secure (non-oracle) argument of knowledge for $\mathcal{R}$. The compilation preserves zero-knowledge: if Γ is hiding and Eval is honest-verifier zero-knowledge, then Π is also honest-verifier zero-knowledge. The efficiency of Π is determined jointly by the PIOP and Γ:*

- ***Prover time:*** *the sum of (i) the prover time of the PIOP, (ii) the oracle length times the commitment cost, and (iii) the query complexity times the prover cost of Γ.*
- ***Verifier time:*** *the sum of (i) the verifier time of the PIOP and (ii) the query complexity times the verifier cost of Γ.*
- ***Proof size:*** *the sum of (i) the message complexity of the PIOP times the commitment size and (ii) the query complexity times the proof size of Γ. If the proof size is $O(\text{polylog}(|\mathbf{w}|))$, we call it succinct.*

Batching. Prover time, verifier time, and proof size can be further reduced using batch openings of polynomial commitments. After batching, the proof size depends only on the number of oracles plus a single polynomial opening.

2.7 The Sumcheck PIOP

The Sumcheck protocol is the fundamental building block of many efficient PIOPs, e.g. [4, 12]. We start from the description of the protocol, and then we show the existence of several efficient prover algorithms and the corresponding applications.

Definition 5 (Sumcheck Relation)**.** *The relation $\mathcal{R}_{SUM}$ is the set of all tuples $(\mathbf{x}; \mathbf{w}) = ((H, [[f]]); f)$ where $f \in \mathcal{F}_{\mu}^{(\leq d)}$ and $\sum_{\mathbf{b}\in\{0,1\}^{\mu}} f(\mathbf{b}) = H$.*

Given f as a μ-variate polynomial over a finite field $\mathbb{F}$, the goal of the Sumcheck protocol is to prove to the verifier that the following sum is correctly calculated:

$$H := \sum_{x\in\{0,1\}^{\mu}} f(x) \tag{6}$$

The verifier can compute H by evaluating f at all 2^{μ} points in $\{0, 1\}^{\mu}$ and sum the results, but this is expensive. As shown in Protocol C, the Sumcheck PIOP allows the verifier to offload this work to the prover through multiple rounds of interaction. The rounds it takes is μ, which is the number of variables in f. At the end of the protocol, the verifier should additionally check if the evaluation of the original polynomial f at some random point $r = (r_1, r_2, \ldots, r_{\mu})$ equals to the prover's claim (i.e., $f_{\mu}(r_{\mu})$ in Protocol C).

Theorem 2.2 (Sumcheck PIOP [4])**.** *The PIOP for $\mathcal{R}_{SUM}$ is perfectly complete and has knowledge error $\delta_{sum}^{d,\mu} := d\mu/|\mathbb{F}|$.*

Complexity analysis. The complexity of Sumcheck PIOP for $\mathcal{R}_{\text{SUM}}$ with respect to $f \in \mathcal{F}_\mu^{(\leq d)}$ is:

- **Prover time:** $O(2^\mu \cdot d \log^2 d)$ field operations (and $O(2^\mu)$ when d is a constant)
- **Verifier time:** $O(\mu)$
- **Query complexity:** 1

Efficient prover algorithms. Because the generic Sumcheck protocol has superlinear scaling when $d > 1$, it can incur prohibitively high prover computation. The literature has developed asymptotic optimizations to Sumcheck that improve the prover's performance for certain special-case polynomial forms, and we will need such optimizations in this work. Below we briefly describe two types of efficient prover algorithms. They can be used directly as black-box plug-ins in this paper.

1. *Sumcheck for the product of multilinear polynomials.* Let the μ-variate polynomial $f(x)$ in Eq. 6 be a product of $h = O(1)$ multilinear polynomials, which means

$$f(x) = \prod_{k=1}^{h} g_k(x)$$

   where each $g_k(x)$ is a multilinear polynomial. An $O(n)$-time prover algorithm ($n = 2^\mu$) is available in [24, 25]. When $h = 1$, it becomes a Sumcheck for a multilinear polynomial.

2. *Sparse-dense Sumcheck.* Lasso [13] targets on a special case of μ-variate polynomial $f(x) = s(x)t(x)$, where $s(x)$ is m-sparse (i.e., among all $n = 2^\mu$ evaluations over $\{0,1\}^\mu$, only $m \ll n$ elements are nonzero), and updating $t(x)$ can be computed in $O(1)$ time for every x in each round of Sumcheck. The formal properties needed for $t(x)$ are defined by Theorem 9 in Lasso [13]. Then, we can have a $O(m)$-time prover according to [13].

Applications. Following the notation defined in the subsection 2.2, we demonstrate three types of checks for some m by n matrices $M \in \mathbb{F}^{m \times n}$ and the corresponding multilinear extension $\widetilde{M}(x, y)$ here. They will be used as building blocks in the following sections.

1. *A row Hamming weight check* is used to prove whether the sum of all elements in each row of M is 1. Twist and Shout [12] shows $\mathcal{P}$ and $\mathcal{V}$ can use the Sumcheck protocol to prove:

$$1 = \sum_{y \in \{0,1\}^{\log n}} \widetilde{M}(r, y). \tag{7}$$

   where $r \in \mathbb{F}^{\log m}$ is the random challenge given by the verifier. If we ignore the time to compute $\widetilde{M}(r, y)$ from $\widetilde{M}(x, y)$ and r, it is clear that the prover time is $O(n)$ based on *Sumcheck for the product of multilinear polynomials.* We will discuss the prover time to compute $\widetilde{M}(r, y)$ later.

2. *A matrix multiplication check* is used to prove a matrix multiplication is correct computed. Suppose $\mathcal{P}$ aims to show $\mathcal{V}$ that a length-m vector g is computed by $M \cdot f$ given f is a length-n vector f, it is well-known [24] $\mathcal{P}$ and $\mathcal{V}$ can apply the Sumcheck protocol with the corresponding multilinear extensions $\widetilde{g}, \widetilde{f}, \widetilde{M}$ for:

$$\widetilde{g}(r) = \sum_{y \in \{0,1\}^{\log n}} \widetilde{M}(r, y)\widetilde{f}(y) \tag{8}$$

   Similarly, $r \in \mathbb{F}^{\log m}$ is given by the verifier. Ignoring the computation time for $\widetilde{M}(r, y)$, the prover time can be $O(n)$ based on *Sumcheck for the product of multilinear polynomials.*

3. *A Booleanity check* is used to prove that each element in a matrix M is either 0 or 1. According to Twist and Shout [12], $\mathcal{P}$ and $\mathcal{V}$ can perform the following Sumcheck:

$$0 = \sum_{z \in \{0,1\}^{\log m + \log n}} s(z)t(z) \tag{9}$$

where

$$s(z) = s(x,y) = \tilde{M}(x,y)^2 - \tilde{M}(x,y)$$

and

$$t(z) = t(x,y) = \tilde{\text{eq}}(r,x)\tilde{\text{eq}}\,(r',y)\,.$$

$r \in \mathbb{F}^{\log m}$, $r' \in \mathbb{F}^{\log n}$ are random challenges given from $\mathcal{V}$. The authors in [12] have demonstrated $t(z)$ satisfies the properties defined in Theorem 9 in Lasso [13]. Besides, in their scenario, all rows in M are one-hot vector, which means $s(z)$ is m-sparse. Therefore, a prover can invoke *Sparse-dense Sumcheck* algorithm to compute a proof in $O(m)$ time.

2.8 Permutation as Sumcheck PIOP

An instance of permutation or multiset check is a pair of arrays

$$A = (a_0, \dots, a_{n-1}) \in \mathbb{F}^n,\ B = (b_0, \dots, b_{n-1}) \in \mathbb{F}^n,$$

Both permutation and multiset checks imply there exists a bijective permutation $\sigma : [0, n-1] \to [0, n-1]$ such that $b_i = a_{\sigma(i)},\ \forall i \in [0, n-1]$.

Let $\log n$-variate multilinear polynomials $\widetilde{A}(x)$ and $\widetilde{B}(x)$ denote the multilinear extensions of A and B. The relation $\mathcal{P}$ aims to prove is

$$\widetilde{A}(\sigma(x)) = \widetilde{B}(x),\ \forall x \in \{0,1\}^{\log n}$$

which is equivalent to

$$\sum_{y \in \{0,1\}^{\log n}} \widetilde{A}(y)\widetilde{\text{eq}}(\sigma(x), y) = \widetilde{B}(x),\ \forall x \in \{0,1\}^{\log n}.$$

Since σ is bijective, the following holds

$$\sum_{y \in \{0,1\}^{\log n}} \widetilde{A}(y)\widetilde{\text{eq}}(x, \sigma^{-1}(y)) = \widetilde{B}(x),\ \forall x \in \{0,1\}^{\log n}.$$

As described in [4, 8], $\mathcal{V}$ can send a random challenge $r \in \mathbb{F}^{\log n}$, then ask $\mathcal{P}$ to prove the following, via Sumcheck PIOP:

$$\widetilde{B}(r) = \sum_{y \in \{0,1\}^{\log n}} \widetilde{A}(y)\widetilde{\text{eq}}(r, \sigma^{-1}(y)) \tag{10}$$

The corresponding Sumcheck instance can be expressed as $(\mathbf{x}; \mathbf{w}) = (H,\ [[f]];\ f)$, where:

- $H := \widetilde{B}(r)$. The prover can provide the oracle commitment $[[\widetilde{B}(x)]]$, allowing the verifier to obtain H by evaluation.
- f is
$$f(y) := \widetilde{A}(y)\,\widetilde{\text{eq}}\big(r, \sigma^{-1}(y)\big)\,.$$
- The oracle $[[f]]$ is constructed from the two oracles $[[\widetilde{A}(y)]]$ and $[[\widetilde{\text{eq}}(x, \sigma^{-1}(y))]]$.

The computational bottleneck of prior work lies in how to process $\widetilde{\text{eq}}(r, \sigma^{-1}(y))$, which requires either $O(n \log n \log^2(\log n))$ [4] or $O(n\sqrt{\log n})$ [8] prover time in their Sumcheck protocols. In this work, we demonstrate that computing $\widetilde{\text{eq}}(r, \sigma^{-1}(y))$ from $\widetilde{\text{eq}}(x, \sigma^{-1}(y))$ and r requires only $O(n)$ prover time. By instantiating the oracle with some sparsity-aware PCS, we can obtain linear-time SNARK provers for both permutation and multiset checks.

3 A Linear-Time Lower Bound for Prover

Our permutation and multiset checks have $O(n)$ proving time. To show that this is optimal, we show an $\Omega(n)$ lower bound for proving permutation or multiset check. The lower bound shown corresponds to the intuition that the prover must at least read the inputs to construct a proof. While the intuition is clear, this is, to our knowledge, not formally proven anywhere. We begin by defining the protocol model and how we measure the prover costs, which are shared bases for proving lower bounds for both checks. We then prove the lower bounds formally in the subsequent subsections.

Protocol model. We consider an arbitrary interactive protocol $\Pi = (P, V)$ for deciding whether a relation $\mathcal{R}$ holds. The verifier $\mathcal{V}$ has no direct access to A or B. The prover $\mathcal{P}$ has read-only access to the input arrays A and B, may be *randomized* and *fully adaptive* in its strategy to probe input arrays (i.e., reads of entries of A or B), and is computationally unbounded.

By the definition of interactive proof, the following properties hold for fixed constants $0 \le \varepsilon, \delta < 1$:

- **Completeness:** if $(A, B) \in \mathcal{R}$, then $\Pr[V \text{ accepts}] \ge 1 - \varepsilon$.
- **Soundness:** if $(A, B) \notin \mathcal{R}$, then $\Pr[V \text{ accepts}] \le \delta$.

Randomness convention. Let ω denote the verifier's random tape (public or private; revealing it to $\mathcal{P}$ can only help $\mathcal{P}$ and thus makes our lower bound stronger), and let τ denote the prover's internal random tape. For analysis we bundle these as a single sample $\rho := (\omega, \tau)$, and write $\Pr_\rho[\cdot]$ and $\mathbb{E}_\rho[\cdot]$ for probability and expectation over the combined randomness of both parties.

Measurement of prover cost. We measure the prover cost by the number of input-cell probes; we assume local computation is free. This captures the information that must be extracted from the instance to produce any accepting transcript. In particular, on any unit-cost word-RAM with word size $\Omega(\log |\mathcal{U}|)$, a probe costs $O(1)$ time. Therefore, if we can show an $\Omega(n)$ probe lower bound, it implies an $\Omega(n)$ lower bound for the prover time.

Probe counter. To track the prover's access to the input, we define two probe counters: one counting total reads and one counting distinct reads. In our lower-bound arguments, we focus on the number of distinct reads, since it always lower-bounds the total number of reads and is easier to analyze. For an execution of Π on input (A, B) with joint randomness ρ, we define

- $\text{reads}_\Pi^{\text{total}}(A, B; \rho)$: the total number of cell reads performed by $\mathcal{P}$. If a cell (e.g., a_i) is read twice, it contributes 2.
- $\text{reads}_\Pi^{\text{distinct}}(A, B; \rho)$: the number of distinct cells among all cells $\{a_1, \dots, a_n, b_1, \dots, b_n\}$ that $\mathcal{P}$ reads at least once.

The below inequality always holds for all possible (A, B) and ρ:

$$\text{reads}_\Pi^{\text{total}}(A, B; \rho) \ge \text{reads}_\Pi^{\text{distinct}}(A, B; \rho)$$

For each $i \in [1, n]$, we define the marginal read probabilities over ρ.

$$r_i^A := \Pr_\rho[P \text{ reads } a_i], \; r_i^B := \Pr_\rho[P \text{ reads } b_i]$$

By linearity of expectation,

$$\mathbb{E}_\rho[\text{reads}_\Pi^{\text{distinct}}(A, B; \rho)] = \sum_{i=1}^{n} (r_i^A + r_i^B)$$

3.1 Lower Bound for Permutation Check

We show that an $\Omega(n)$ lower bound is necessary for proving a known permutation σ. Therefore, an $O(n)$ prover for permutation check is theoretically optimal.

Definition 6 (Permutation Check)**.** *Let $\mathcal{U}$ be a finite universe with $|\mathcal{U}| \geq n+1$. An instance consists of two arrays*

$$A = (a_1, \ldots, a_n) \in \mathcal{U}^n \quad \textit{and} \quad B = (b_1, \ldots, b_n) \in \mathcal{U}^n.$$

Given a fixed and publicly known permutation $\sigma \in S_n$ (i.e., a bijection $\sigma : \{1, \ldots, n\} \to \{1, \ldots, n\}$), the goal is to check whether

$$Perm_\sigma = \{(A, B) : \forall i \in [1, n],\ b_i = a_{\sigma(i)}\}.$$

We use Π to denote an arbitrary interactive proof protocol for checking $Perm_\sigma$.

With the above symbols and definitions, we state the theorem that linear probes are necessary.

Theorem 3.1. *To prove $(A, B) \in Perm_\sigma$,*

$$\mathbb{E}_\rho\Big[reads_\Pi^{total}(A, B; \rho)\Big] \ \geq\ 2(1 - \varepsilon - \delta)n.$$

Consequently, the expected prover time on a unit-cost RAM model is $\Omega(n)$.

Remark. The bound holds in expectation over ρ, which includes all randomness of both prover and verifier. Thus, even a randomized and fully adaptive prover cannot achieve $o(n)$ expected probe complexity.

Proof. Fix any instance $(A, B) \in \mathrm{Perm}_\sigma$. For each $i \in [1, n]$, the single equality constraint $b_i = a_{\sigma(i)}$ holds by definition. Define

$$p_i = \min\{r_i^B,\ r_{\sigma(i)}^A\}.$$

An adversary can randomly pick an $i \in [1, n]$ and form an instance $(A', B') \notin \mathrm{Perm}_\sigma$ by changing only the less-read cell of the pair $\{b_i, a_{\sigma(i)}\}$ to some $x \in \mathcal{U}$ such that the cells in the changed pair are not equal. All other pairs in (A, B) are unchanged, so exactly this equality is violated.

$\mathcal{P}$ runs Π on (A, B) and (A', B') with the same joint randomness ρ. The two resulting transcripts can differ only when $\mathcal{P}$ reads the flipped cell, which happens with probability p_i. Hence

$$\Big|\Pr_\rho[V \text{ accepts } (A, B)] - \Pr_\rho[V \text{ accepts } (A', B')]\Big| \leq\ p_i.$$

We next justify why the inequality is true. Let E be the event that $\mathcal{P}$ reads the flipped cell. By construction $\Pr_\rho[E] = p_i$. Let T and T' denote the full transcripts (all messages and probe results) of Π on (A, B) and (A', B') respectively under the same randomness ρ.

Observe that if E does *not* occur, then every probed cell has the same value in both inputs, so $T = T'$ and the verifier's acceptance probabilities are identical:

$$\Pr_\rho[V \text{ accepts } (A, B) \mid \neg E] = \Pr_\rho[V \text{ accepts } (A', B') \mid \neg E].$$

Using the law of total probability,

$$\begin{aligned}
&\Pr_\rho[V \text{ accepts } (A, B)] - \Pr_\rho[V \text{ accepts } (A', B')] \\
&= \Pr_\rho[E] \cdot \big(\Pr_\rho[V \text{ accepts } (A, B) \mid E] - \Pr_\rho[V \text{ accepts } (A', B') \mid E]\big) \\
&+ \Pr_\rho[\neg E] \cdot \underbrace{\big(\Pr_\rho[V \text{ accepts } (A, B) \mid \neg E] - \Pr_\rho[V \text{ accepts } (A', B') \mid \neg E]\big)}_{=0}.
\end{aligned}$$

Taking absolute values and noting that the term in parentheses is bounded by 1 gives

$$\left|\Pr_\rho[V \text{ accepts } (A,B)] - \Pr_\rho[V \text{ accepts } (A',B')]\right| \;\le\; \Pr_\rho[E].$$

According to completeness and soundness of Π,

$$(1-\varepsilon)-\delta \le \Pr_\rho[V \text{ accepts } (A,B)] - \Pr_\rho[V \text{ accepts } (A',B')].$$

Therefore, $p_i \ge (1-\varepsilon)-\delta$.

By summing over i and using $\min(x,y) \le \frac{1}{2}(x+y)$,

$$\frac{1}{2}\sum_{i=1}^{n}(r^A_{\sigma(i)} + r^B_i) \ge \sum_{i=1}^{n} p_i \ge (1-\varepsilon-\delta)n$$

By the definition of probe counter, we have

$$\begin{aligned}\mathbb{E}_\rho\Big[\text{reads}^{\text{total}}_\Pi(A,B;\rho)\Big] &\ge \mathbb{E}_\rho\Big[\text{reads}^{\text{distinct}}_\Pi(A,B;\rho)\Big]\\ &\ge 2(1-\varepsilon-\delta)n\end{aligned}$$

□

3.2 Lower Bound for Multiset Check

We show an $\Omega(n)$ lower bound for performing a multiset check. Therefore, an $O(n)$ prover for multiset check is theoretically optimal.

Definition 7 (Multiset Equality Check)**.** *Let $\mathcal{U}$ be a finite universe with $|\mathcal{U}| \ge n+1$. An instance consists of two arrays*

$$A = (a_1,\ldots,a_n) \in \mathcal{U}^n \quad \textit{and} \quad B = (b_1,\ldots,b_n) \in \mathcal{U}^n.$$

The property to be checked is

$$\textit{MultiSetEq} \iff \{a_1,\ldots,a_n\} \textit{ and } \{b_1,\ldots,b_n\} \textit{ are equal multisets.}$$

We use Π to denote an arbitrary interactive proof protocol that checks multiset equality.

Similar to the previous subsection, we state the theorem that linear probes are necessary.

Theorem 3.2. *To prove $(A,B) \in$ MultiSetEq,*

$$\mathbb{E}_\rho\Big[\textit{reads}^{\textit{total}}_\Pi(A,B;\rho)\Big] \;\ge\; 2(1-\varepsilon-\delta)n.$$

Consequently, the expected prover time on a unit-cost RAM model is $\Omega(n)$.

Remark. The bound holds in expectation over ρ, which includes all randomness of both the prover and verifier. Thus, even a randomized and fully adaptive prover cannot achieve $o(n)$ expected probe complexity.

Proof. Fix any instance $(A,B) \in$ MultiSetEq. For each $i \in [1,n]$, an adversary can choose $x_i^{A^{(i)}} \in \mathcal{U} \setminus \{b_1,\ldots,b_n\}$ (it exists since $|\mathcal{U}| \ge n+1$) and form a new instance $(A^{(i)},B) \notin$ MultiSetEq, where $A^{(i)}$ equals A except we replace the i-th element with $x_i^{A^{(i)}}$.

Similarly, an adversary can choose $x_i^{B^{(i)}} \in \mathcal{U}\backslash\{a_1,\ldots,a_n\}$ and form a new instance $(A,B^{(i)}) \notin$ MultiSetEq, where $B^{(i)}$ equals B except we replace the i-th element with $x_i^{B^{(i)}}$.

$\mathcal{P}$ runs Π on (A, B), $(A^{(i)}, B)$, and $(A, B^{(i)})$ with the same joint randomness ρ. The transcripts can differ only when $\mathcal{P}$ reads the replaced cell. Therefore, based on the similar justification in the previous subsection, the following two inequalities hold

$$\left|\Pr_{\rho}[V \text{ accepts } (A, B)] - \Pr_{\rho}[V \text{ accepts } (A^{(i)}, B)]\right| \leq \ r_i^A.$$

$$\left|\Pr_{\rho}[V \text{ accepts } (A, B)] - \Pr_{\rho}[V \text{ accepts } (A, B^{(i)})]\right| \leq \ r_i^B.$$

According to completeness and soundness of Π, $r_i^A \geq (1-\varepsilon) - \delta$ and $r_i^B \geq (1-\varepsilon) - \delta$.
By summing over i and summing the two inequalities,

$$\sum_{i=1}^{n} (r_i^A + r_i^B) \geq 2(1-\varepsilon-\delta)n$$

By the definition of probe counter, we have

$$\begin{aligned}\mathbb{E}_{\rho}\left[\text{reads}_{\Pi}^{\text{total}}(A, B; \rho)\right] &\geq \mathbb{E}_{\rho}\left[\text{reads}_{\Pi}^{\text{distinct}}(A, B; \rho)\right] \\ &\geq 2(1-\varepsilon-\delta)n\end{aligned}$$

□

4 SNARKs for Permutation and Multiset Checks

The previous section established that any interactive proof for permutation or multiset checks requires $\Omega(n)$ time. Here we present an $O(n)$-time construction that matches this lower bound: a complete protocol consisting of the core permutation check and structural checks to enforce bijectivity. To define them, let an instance of permutation or multiset check be (A, B), where

$$A = (a_0, \ldots, a_{n-1}) \in \mathbb{F}^n, \ B = (b_0, \ldots, b_{n-1}) \in \mathbb{F}^n.$$

The two checks can be defined as follows:

- **Permutation check**: Given a public permutation σ, $\mathcal{R}_{\text{PERM}}$ is to check if $b_i = a_{\sigma(i)}, \ \forall i \in [0, n-1]$.
- **Multiset check**: $\mathcal{R}_{\text{MSET}}$ is to check if there exists a permutation σ such that $b_i = a_{\sigma(i)}, \ \forall i \in [0, n-1]$.

In Permutation check, σ has been processed by the indexer and therefore the prover has no need to provide structural arguments for σ. In Multiset check, the prover must perform additional structural checks to show the verifier that σ is a bijective permutation. We will start from the core permutation check and then show how to perform additional structural checks for σ to ensure bijectivity.

4.1 The Core Permutation Check

The core permutation check validates whether M satisfies $B = MA$. Let M be a n by n permutation matrix, where

$$\begin{aligned}&M_{ij} = 1, \text{ if } \sigma(i) = j; \\ &M_{ij} = 0, \text{ otherwise.}\end{aligned} \tag{11}$$

By standard linear algebra [11], we can represent an arbitrary bijective permutation σ in this way. We denote its unique multilinear extension as $\widetilde{M}(x, y)$, which is encoded as Eq. 3. By definition of $\sigma(i)$,

$$\widetilde{M}(x, y) = \widetilde{\text{eq}}(x, \sigma^{-1}(y)), \forall x \in \{0,1\}^{\log n}, \ y \in \{0,1\}^{\log n}$$

Since $\widetilde{M}(x,y) = \widetilde{\mathrm{eq}}(x, \sigma^{-1}(y))$, $\forall x \in \{0,1\}^{\log n}$, $y \in \{0,1\}^{\log n}$, $\widetilde{M}(x,y)$ is the unique multilinear extension of $\widetilde{\mathrm{eq}}(x, \sigma^{-1}(y))$. Therefore, Eq. 10 can become

$$\widetilde{B}(r) = \sum_{y \in \{0,1\}^{\log n}} \widetilde{A}(y)\widetilde{M}(r,y) \tag{12}$$

The key to obtaining an $O(n)$-time prover is shifting from a Sumcheck perspective to a linear algebra one. Specifically, $\widetilde{B}(r) = \sum_{j \in \{0,1\}^{\log n}} b_j \cdot \widetilde{\mathrm{eq}}(j, r)$ can be viewed as

$$(eq_{0,r}, \ldots, eq_{j,r}, \ldots, eq_{n-1,r}) \cdot (b_0, \ldots, b_j, \ldots, b_{n-1})^\top, \tag{13}$$

where $\cdot$ denotes the inner product operation and $eq_{j,r} = \widetilde{\mathrm{eq}}(j,r)$, $\forall j \in [0, n-1]$ can be computed with n field multiplications according to [10]. To compute the above inner product, we use an additional n field multiplications, which means $\widetilde{B}(r)$ can be computed in $O(n)$ time.

Let $\widetilde{R}_i(y) = \sum_{j \in \{0,1\}^{\log n}} M_{ij}\widetilde{\mathrm{eq}}(j,y)$ be the multilinear extension of i-th row R_i in M. For all $y \in \{0,1\}^{\log n}$,

$$\begin{aligned} \widetilde{M}(r,y) &= \sum_{i \in \{0,1\}^{\log n}} R_i(y)\widetilde{\mathrm{eq}}(i,r) \\ &= (eq_{0,r}, \ldots, eq_{n-1,r}) \cdot (\widetilde{R}_0(y), \ldots, \widetilde{R}_{n-1}(y))^\top \\ &= (eq_{0,r}, \ldots, eq_{n-1,r}) \cdot (M_{0y}, \ldots, M_{(n-1)y})^\top \end{aligned} \tag{14}$$

From Eq. 14, $\widetilde{M}(r,y)$ is random linear combination among rows of M (i.e., the matrix-vector product between $M^\top$ and the challenge vector). If $\widetilde{M}(x,y)$ represents a general n by n dense matrix, computing $\widetilde{M}(r,y)$ requires $O(n^2)$ time in general. Fortunately, $\widetilde{M}(x,y)$ is n-sparse and $(M_{0y}, \ldots, M_{(n-1)y})$ is a one-hot vector (i.e., it only consists of one 1, all other entries are 0s), where $M_{iy} = 1$ if $\sigma(i) = y$. Therefore, Eq. 14 becomes

$$\widetilde{M}(r,y) = eq_{i,r} \tag{15}$$

for all $y \in \{0,1\}^{\log n}$ and $\sigma^{-1}(y) = i$ (i.e., the inverse permutation of the challenge vector).

As a result of Eq. 15, we can compute $\widetilde{M}(r,y)$ for all $y \in [0, n-1]$ with n element accesses from the vector $(eq_{0,r}, \ldots, eq_{i,r}, \ldots, eq_{n-1,r})$ that was computed when evaluating $\widetilde{B}(r)$.

With the target sum $\widetilde{B}(r)$ and the product of two $(\log n)$-variate multilinear polynomials $\widetilde{A}(y)\widetilde{M}(r,y)$, a prover can use *Sumcheck for the product of multilinear polynomials* to complete the Sumcheck proof in $O(n)$ prover time.

Based on the above, we can formalize the permutation relation to construct the Permutation PIOP.

Definition 8 (Permutation Relation). *Given a public permutation σ, the relation $\mathcal{R}^{\sigma}_{PERM}$ is the set of all tuples $(\mathbf{i}; \mathbf{x}; \mathbf{w}) =$*

$$\{(\widetilde{M}(x,y), [[\widetilde{M}(x,y)]]; [[\widetilde{A}(x)]], [[\widetilde{B}(x)]]; \widetilde{A}(x), \widetilde{B}(x))\}$$

where $\widetilde{M}(x,y)$ is the multilinear extension of permutation matrix defined in Eq. 11 and $\widetilde{A}(\sigma(x)) = \widetilde{B}(x)$, $\forall x \in \{0,1\}^{\log n}$.

Theorem 4.1 (Permutation PIOP). *The PIOP for $\mathcal{R}^{\sigma}_{PERM}$ is perfectly complete and has knowledge error $O(\log n/|\mathbb{F}|)$.*

Proof. **Perfect completeness.** Recall that the Sumcheck PIOP is perfectly complete. Because for every $(\widetilde{M}(x,y), \widetilde{A}(x), \widetilde{B}(y))$ that belongs to $\mathcal{R}^{\sigma}_{\mathrm{PERM}}$, Eq. 12 always holds $\forall r \in \mathbb{F}^{\log n}$, the resulting protocol is also perfectly complete.

Knowledge soundness. The soundness error in the Sumcheck PIOP is $\delta^{d,\mu}_{\mathrm{sum}} := d\mu/|\mathbb{F}| = 2\log n/|\mathbb{F}|$. Evaluating $\widetilde{B}(x)$ at r results in a $\log n/|\mathbb{F}|$ soundness error because of the Schwartz-Zippel Lemma. The total soundness is $O(\log n/|\mathbb{F}|)$. By Lemma 2.3, the knowledge soundness is also $O(\log n/|\mathbb{F}|)$. □

Complexity analysis.

- *Prover complexity* is $O(n)$ field operations because it takes $2n$ field multiplications to compute step 3 in Protocol 1 and $O(n)$ field multiplications for standard linear Sumcheck PIOP for step 4.
- *Verifier complexity* is $O(\log n)$ field operations because there are $\log n$ rounds in the Sumcheck PIOP for step 4, and it takes $O(1)$ time for the verifier to validate the message in each round.
- *Query complexity* is 3 because the verifier needs to query $[[\widetilde{M}(x,y)]], [[\widetilde{A}(x)]]$ and $[[\widetilde{B}(x)]]$ once at the end of the PIOP.

Protocol 1: Permutation PIOP (PermPIOP) for $\mathcal{R}^{\sigma}_{\text{PERM}}$
Goal: Prove that $\widetilde{A}(\sigma(x)) = \widetilde{B}(x),\ \forall x \in \{0,1\}^{\log n}$ given some public bijective permutation σ.
The PermPIOP proceeds as follows.

1. $\mathcal{P}$ sends the oracles $[[\widetilde{A}(x)]], [[\widetilde{B}(x)]]$ to $\mathcal{V}$.
2. $\mathcal{V}$ sends a random challenge $r \in \mathbb{F}^{\log n}$ to $\mathcal{P}$.
3. $\mathcal{P}$ computes $B_r = \widetilde{B}(r)$ and $\widetilde{M}(r,y)$ by Eq. 13 and Eq. 15.
4. $\mathcal{P}$ and $\mathcal{V}$ use Sumcheck PIOP to check Eq. 12. Specifically, $\mathcal{P}$ makes $f(y) = \widetilde{A}(y)\widetilde{M}(r,y)$ and $\mathcal{V}$ can query $[[f(y)]]$ at a random point $r' \in \mathbb{F}^{\log n}$ by querying $[[\widetilde{M}(x,y)]]$ at (r,r'), $[[\widetilde{A}(x)]]$ at r' and computing their product. Then, the instance in $\mathcal{R}_{\text{SUM}}$ becomes $(B_r,\ [[f(y)]];f(y))$.
5. $\mathcal{V}$ queries $[[\widetilde{B}(x)]]$ at r to check if the evaluation is equal to B_r.
6. $\mathcal{V}$ outputs 1 if both verifiers in step 4 and 5 output 1.

4.2 Structural Checks

In addition to proving the correctness of the permutation, we must also prove that the arithmetized permutation was constructed correctly. The structural check confirms that a prover permutation is a bijective permutation. We use it to complete the PIOP for multiset check. It is unclear how to prove the permutation is bijective for BiPerm [8]. To our best knowledge, this is the first paper that proposes a $O(n)$-time prover even when the permutation is unknown to the verifier.

We rely on the definition of a permutation matrix: a permutation matrix is a square matrix with binary entries such that each row and each column contains exactly one 1 and all remaining entries are 0. Therefore, we can perform the following three checks to complete the structural check.

1. *Booleanity*: It checks that all entries in M are either 0 or 1. $\mathcal{P}$ and $\mathcal{V}$ use the Sumcheck PIOP to validate if Eq. 9 is true.
2. *Hamming weight of rows*: $\mathcal{P}$ and $\mathcal{V}$ use the Sumcheck PIOP to validate if the Hamming weight of rows in M is 1.
3. *Hamming weight of columns*: $\mathcal{P}$ and $\mathcal{V}$ use the Sumcheck PIOP to validate if the Hamming weight of columns in M is 1.

The Shout lookup protocol [12] includes descriptions for 1. and 2. but not 3. We show that it suffices if $\mathcal{P}$ and $\mathcal{V}$ check if $\sum_{x\in\{0,1\}^{\log n}} M(x,r) = 1$ given the verifier challenge $r \in \mathbb{F}^{\log n}$. The reasoning is similar to

Protocol 2: Multiset PIOP (MsetPIOP) for $\mathcal{R}_{\text{MSET}}$
Goal: Prove that there exists a bijective permutation σ such that $\widetilde{A}(\sigma(x)) = \widetilde{B}(x)$, $\forall x \in \{0,1\}^{\log n}$. The MsetPIOP proceeds as follows.

1. $\mathcal{P}$ sends the oracles $[[\widetilde{M}(x,y)]], [[\widetilde{A}(x)]]$ and $[[\widetilde{B}(x)]]$ to $\mathcal{V}$.

2. *Core permutation check.* $\mathcal{P}$ and $\mathcal{V}$ follow steps (2)-(6) in PermPIOP to validate Eq. 12.

3. *Booleanity.*
(3.1) $\mathcal{V}$ sends random challenges $r_{\text{bool}}, r'_{\text{bool}} \in \mathbb{F}^{\log n}$ to $\mathcal{P}$.
(3.2) $\mathcal{P}$ computes $\widetilde{\text{eq}}(r_{\text{bool}}, x)\widetilde{\text{eq}}(r'_{\text{bool}}, y)$ and makes $f(x,y) = [\widetilde{M}(x,y)^2 - \widetilde{M}(x,y)]\widetilde{\text{eq}}(r_{\text{bool}}, x)\widetilde{\text{eq}}(r'_{\text{bool}}, y)$
(3.3) $\mathcal{P}$ and $\mathcal{V}$ use Sumcheck PIOP with the sparse-dense Sumcheck prover [12] (see the simplified version in Appendix B). Specifically, $\mathcal{V}$ can query $[[f(x,y)]]$ at random points $r, r' \in \mathbb{F}^{\log n}$ by querying $[[\widetilde{M}(x,y)]]$ at (r, r') and computing
$$[\widetilde{M}(r,r')^2 - \widetilde{M}(r,r')]\widetilde{\text{eq}}(r_{\text{bool}}, r)\widetilde{\text{eq}}(r'_{\text{bool}}, r')$$
which is feasible in $O(\log n)$ time for $\mathcal{V}$ because $\widetilde{\text{eq}}$ is structured. Then, the instance to check becomes (0, $[[f(x,y)]]$;$f(x,y)$).

4. *Hamming weight of rows.*
(4.1) $\mathcal{V}$ sends a random challenge $r_{\text{row}} \in \mathbb{F}^{\log n}$ to $\mathcal{P}$.
(4.2) $\mathcal{P}$ computes $f(y) = \widetilde{M}(r_{\text{row}}, y)$
(4.3) $\mathcal{P}$ and $\mathcal{V}$ use Sumcheck PIOP with the sparse-dense Sumcheck prover [12]. Specifically, $\mathcal{V}$ can query $[[f(y)]]$ at random points $r \in \mathbb{F}^{\log n}$ by querying $[[\widetilde{M}(x,y)]]$ at (r_{row}, r). Then, the instance to check becomes (1, $[[f(y)]]$;$f(y)$).

5. *Hamming weight of columns.*
(5.1) $\mathcal{V}$ sends a random challenge $r_{\text{col}} \in \mathbb{F}^{\log n}$ to $\mathcal{P}$.
(5.2) $\mathcal{P}$ computes $f(x) = \widetilde{M}(x, r_{\text{col}})$
(5.3) $\mathcal{P}$ and $\mathcal{V}$ use Sumcheck PIOP with the sparse-dense Sumcheck prover [12]. Specifically, $\mathcal{V}$ can query $[[f(x)]]$ at random points $r \in \mathbb{F}^{\log n}$ by querying $[[\widetilde{M}(x,y)]]$ at (r, r_{col}). Then, the instance to check becomes (1, $[[f(x)]]$;$f(x)$).

6. $\mathcal{V}$ outputs 1 if all four verifiers in the above checks output 1.

Eq. 14. Let $\widetilde{C}_j(x) = \sum_{i\in\{0,1\}^{\log n}} M_{ij}\widetilde{\text{eq}}(i,x)$ be the multilinear extension of j-th column C_j in M. For all $x \in \{0,1\}^{\log n}$,
$$\begin{aligned}\widetilde{M}(x,r) &= \sum_{j\in\{0,1\}^{\log n}} C_j(x)\widetilde{\text{eq}}(j,r) \\ &= (eq_{0,r}, \dots, eq_{n-1,r}) \cdot (\widetilde{C}_0(x), \dots, \widetilde{C}_{n-1}(x))^\top \\ &= (eq_{0,r}, \dots, eq_{n-1,r}) \cdot (M_{x0}, \dots, M_{x(n-1)})^\top \\ &= eq_{j,r} \text{ (if } \sigma(x) = j)\end{aligned} \tag{16}$$

Therefore, $\sum_{x\in\{0,1\}^{\log n}} \widetilde{M}(x,r) = \sum_{j\in[0,n-1]} eq_{j,r} = 1$. Moreover, Eq. 16 can be computed in $O(n)$ prover time since it is simply the permutation of the challenge vector $(eq_{0,r}, \dots, eq_{n-1,r})$, which is computable in $O(n)$ from r [10].

Based on the above structural check, we can formalize the multiset relation to construct the Multiset PIOP in Protocol 2.

Definition 9 (Multiset Relation)**.** *The relation $\mathcal{R}_{MSET}$ is the set of all tuples* $(\mathbf{i};\mathbf{x};\mathbf{w}) =$

$$\{(\emptyset;[[\widetilde{M}(x,y)]],[[\widetilde{A}(x)]],[[\widetilde{B}(x)]];\widetilde{M}(x,y),\widetilde{A}(x),\widetilde{B}(x))\}$$

where the bijective permutation σ is only known to the prover (either computed during witness generation or known ahead of time) and $\widetilde{M}(x,y)$ is the multilinear extension of permutation matrix defined in Eq. 11 and $\widetilde{A}(\sigma(x)) = \widetilde{B}(x),\ \forall x \in \{0,1\}^{\log n}$.

Theorem 4.2 (Multiset PIOP)**.** *The PIOP for $\mathcal{R}_{MSET}$ is perfectly complete and has knowledge error $O(\log n/|\mathbb{F}|)$.*

Proof. **Perfect completeness.** Recall PermPIOP and Sumcheck PIOP are perfectly complete. The checks for booleanity and Hamming weight of rows are perfectly complete following [12]. The check for Hamming weight of columns is perfectly complete by the analysis done in Eq. 16.

Knowledge soundness. The soundness errors in Sumcheck PIOP for two Hamming weight checks are both $2\log n/|\mathbb{F}|$, and for booleanity check, it is $3\log n^2/|\mathbb{F}| = 6\log n/|\mathbb{F}|$. Evaluating all oracles results in a $O(\log n/|\mathbb{F}|)$ soundness error because of the Schwartz-Zippel Lemma. Therefore, the total soundness is still $O(\log n/|\mathbb{F}|)$. By Lemma 2.3, the knowledge soundness is $O(\log n/|\mathbb{F}|)$. □

Complexity analysis.

- *Prover complexity* is $O(n)$ field operations. Recall it takes $O(n)$ field operations for the core permutation check. For booleanity check, it costs $O(n)$ field multiplications to compute $\widetilde{\text{eq}}(r_{\text{bool}},x)\widetilde{\text{eq}}(r'_{\text{bool}},y)$ [10]; it takes $O(n)$ prover time to run the sparse-dense Sumcheck prover for the PIOP [12, 13]. In total, it takes $2n$ field multiplications to compute $\widetilde{M}(x,r_{\text{col}})$ and $\widetilde{M}(r_{\text{row}},y)$ for the two Hamming weight checks. It costs $O(n)$ for both Sumcheck PIOPs, so the overall prover complexity is $O(n)$.

- *Verifier complexity* is $O(\log n)$ field operations. It takes $\log n$ rounds for the core check and the two Hamming weight checks, and $2\log n$ rounds for the booleanity check. In each round of Sumcheck PIOP, the verifier performs $O(1)$ field operations. Therefore, the overall verifier complexity is $O(\log n)$.

- *Query complexity* is 6 because the verifier needs to query $[[\widetilde{A}(x)]]$ and $[[\widetilde{B}(x)]]$ once, and $[[\widetilde{M}(x,y)]]$ three times at the end of the PIOP. This may be improved if batch query is supported.

4.3 PIOP Compilation

By standard practice [26–28], $\mathcal{P}$ can use a compatible PCS to instantiate oracles such that PermPIOP and MsetPIOP can be transformed into interactive arguments of knowledge. Specifically, the oracles are replaced with the commitments of $\widetilde{B}(x)$, $\widetilde{A}(x)$, and $\widetilde{M}(x,y)$. Each time $\mathcal{V}$ queries the oracles, $\mathcal{P}$ and $\mathcal{V}$ invoke the Eval protocol defined in the PCS. Moreover, since they are public-coin PIOPs, they can be turned into NARKs by Fiat-Shamir transformation [19].

The question then becomes *which PCSs allow the NARKs to become SNARKs with a linear-time prover.* To achieve this for both PIOPs, the Commit time and Eval time for $\mathcal{P}$ should be $O(n)$, the Eval time and proof size for $\mathcal{V}$ should be $O(\text{polylog}(n))$. While there are n^2 evaluations for $\widetilde{M}(x,y)$ over the Boolean hypercube, there are only n nonzero terms. This means that we can use a sparsity-aware PCS such as Dory [14], Zeromorph [18], and HyperKZG [20] to achieve a $O(n)$ Commit time. Moreover, for a $\log N$-variate multilinear, Dory [14] enables $\mathcal{P}$ to have $O(\sqrt{N})$ Eval time and $\mathcal{V}$ to have $O(\log N)$ Eval time and $O(\log N)$ proof size. By instantiating the PIOPs with Dory, the resulting NARKs for permutation and multiset checks are both SNARKs with an $O(n)$-time prover.

5 Extensions to Structured or Non-Bijective Permutations

In this section, we further demonstrate that our permutation SNARK can (1) be *field-agnostic* for certain classes of permutations, and (2) support non-bijective permutations.

First, if a permutation σ can be described by a specific family of circuits, then the verifier can evaluate it in $O(\text{polylog}(n))$ time rather than requesting an evaluation proof from the prover. This eliminates the need for a sparsity-aware PCS and yields the first logarithmically-sound *field-agnostic* $O(n)$-time prover.

Second, we show that when the permutation is non-bijective, our scheme naturally extends to handle it in linear time. Finally, we discuss how our approach relates to the state-of-the-art lookup protocol [12].

5.1 Public and Structured Permutations

When the permutation σ is *verifier-computable* (public and efficiently evaluable on indices), our protocol becomes both more efficient and more flexible. In this case, the verifier $\mathcal{V}$ can evaluate $\widetilde{\text{eq}}(r, \sigma^{-1}(r')) = \widetilde{\text{eq}}(\sigma(r), r')$ at the end of the core permutation check, so the prover $\mathcal{P}$ never commits to the permutation. This removes the need for a sparsity-aware PCS (e.g., Dory [14]), makes the argument field-agnostic, and lets us skip the three structural checks because $\mathcal{V}$ knows σ.

5.1.1 Definition and evaluation cost

We now characterize which permutations admit $O(\text{polylog}(n))$-time verifier-side evaluation. In prior work [12, 13], a polynomial is called structured if it can be evaluated in $O(\log n)$ time. We generalize this notion to mean $O(\text{polylog}(n))$-time evaluability. We also relax the requirement in Eq. 12 that the permutation polynomial must be a multilinear polynomial $\widetilde{M}(x, y)$. Indeed, it only needs to be multilinear in the x-variables. Let $M(x, y)$ be a $2\log n$-variate multivariate polynomial and multilinear in x. Let $M(x, y)$ be equal to $\widetilde{M}(x, y)$ for all $x, y \in \{0,1\}^{\log n}$. Given $x \in \mathbb{F}^{\log n}$, the following equation still suffices for permutation checks

$$\widetilde{B}(x) = \sum_{y \in \{0,1\}^{\log n}} \widetilde{A}(y) M(x, y) \tag{17}$$

To verify it, see that Eq. 17 represents a permutation because of the definition of $\widetilde{M}(x, y)$ and σ. Next, the left and the right terms are both multilinear in x and agree with each other over all $x \in \{0,1\}^{\log n}$. This means they are the same and unique multilinear extension of the permutation output B.

Now we show one can always construct an $M(x, y)$ from the definition of σ by setting

$$M(x, y) = \widetilde{\text{eq}}(x, \sigma^{-1}(y)) = \prod_{i=1}^{\log n} \Big((1 - x_i)(1 - \sigma_i^{-1}(y)) + x_i \sigma_i^{-1}(y) \Big) \tag{18}$$

where $\sigma_i^{-1}(y)$ denotes the function that takes $\log n$ bits (the binary representation of y) and output a single bit (the i-th output bit of σ^{-1}). $M(x, y)$ satisfies the above description and is a $2\log n$-variate multivariate polynomial and a multilinear polynomial in x by construction.

To allow the verifier to evaluate $M(x, y) = \widetilde{\text{eq}}(x, \sigma^{-1}(y))$ without any commitment or evaluation proof from the prover, σ^{-1} must be computable in $O(\text{polylog}(n))$ time. We now relate the circuit complexity of σ^{-1} to this evaluation cost, and the theorem below formalizes the precise condition under which this holds.

Theorem 5.1 (Circuit–Evaluation Equivalence). *Let $\sigma : \{0,1\}^{\log n} \to \{0,1\}^{\log n}$ be a permutation and $\sigma^{-1} : \{0,1\}^{\log n} \to \{0,1\}^{\log n}$ be the corresponding inverse permutation.*

Assume Boolean (resp. arithmetic) circuits over $\{0,1\}$ (resp. any field $\mathbb{F}$) use gates with bounded fan-in, and let size *denote the circuit size, measured as the number of gates (equivalently, the number of gates plus wires up to constant factors).*

Then the following are equivalent:

1. *σ^{-1} is computable by a DLOGTIME-uniform Boolean circuit of size $O(\text{polylog}(n))$;*

2. *There is a deterministic evaluator (e.g., a verifier) that, on any input $y \in \{0,1\}^{\log n}$, computes every i-th output bit $\sigma_i^{-1}(y)$ in time $O(\mathrm{polylog}(n))$ by on-the-fly uniform decoding and gate simulation for all $i \in [1, \log n]$. That is, given (n, g), a DLOGTIME-uniformity procedure returns the type and fan-in of gate g and the indices of its input wires, and the evaluator traverses the circuit in topological order to produce the output.*

Proof. (*1 ⇒ 2*) If σ^{-1} can be computed by a DLOGTIME-uniform Boolean circuit of size $O(\mathrm{polylog}(n))$, an evaluator can simulate it gate-by-gate. Bounded fan-in ensures $O(1)$ Boolean operations per gate, and uniformity guarantees that the description of each gate can be recovered in $O(\log n)$ time. Since there are $O(\mathrm{polylog}(n))$ gates, the total evaluation time is $O(\mathrm{polylog}(n))$. Computing each σ_i^{-1} to extract the i-th evaluation from σ^{-1} for all $i \in [1, \log n]$ takes $O(\mathrm{polylog}(n))$ in total.

(*2 ⇒ 1*) Suppose a deterministic procedure (e.g., a verifier) evaluates every output bit of $\sigma^{-1}(y)$ in time $O(\mathrm{polylog}(n))$ with uniform access to the gate structure. By the result of Fischer and Pippenger (see [29]), any function computable by a Turing machine in time $T(m)$ on m input bits can be simulated by a DLOGTIME-uniform Boolean circuit of size $O\big(T(m) \log T(m)\big)$. Specializing to $m = \log n$ and $T(m) = O(\mathrm{polylog}(n))$ yields a Boolean circuit of size $O(\mathrm{polylog}(n))$ for each function $\sigma_i^{-1}(y)$. This implies that $\sigma^{-1}(y)$ can be computed by a multi-output DLOGTIME-uniform Boolean circuit of size $O(\mathrm{polylog}(n))$ because exposing all output bits adds only constant-factor overhead. □

Given any DLOGTIME-uniform Boolean circuit of σ^{-1}, we can always convert each Boolean gate to an arithmetic gate over $\mathbb{F}$.[2]

Since $M(x, y)$ is defined as $\widetilde{\mathrm{eq}}(x, \sigma^{-1}(y))$, evaluating $M(x, y)$ reduces to evaluating $\sigma^{-1}(y)$ and then computing $\log n$ second-degree factors of the form $(1 - x_i)(1 - \sigma_i^{-1}(y)) + x_i \sigma_i^{-1}(y)$ for all $i \in [1, \log n]$. Hence, if σ^{-1} is computable by any DLOGTIME-uniform Boolean circuit of size $O(\mathrm{polylog}(n))$, then $M(x, y)$ can also be evaluated in $O(\mathrm{polylog}(n))$ time for all $(x, y) \in \mathbb{F}^{\log n} \times \mathbb{F}^{\log n}$. In particular, M is structured, which implies that the verifier does not need a proof of correct evaluation for $M(x, y)$ from the prover. This removes the need for a sparsity-aware polynomial commitment, yielding a field-agnostic SNARK. However, to retain a linear-time Sumcheck prover in Eq. 17, we still require $M(x, y)$ to be a multilinear polynomial in x and y, because we are not aware of any $O(n)$-time Sumcheck protocol for general polynomials in that relation.

5.1.2 Generic circuits as structured permutations

To see why structured permutations can be useful for modern SNARK frameworks [4–6], we briefly outline how they can be used to construct a circuit C (for the relation to be checked) in the SNARKs. Let C be a Boolean or arithmetic circuit with M wires and N gates. List all intermediate wire values in topological order as

$$w \in \mathbb{F}^M.$$

Each gate specifies which positions in w serve as its inputs and output. For multiplication gates, this induces three index mappings:

$$L_{\mathrm{mul}} = w[\pi_L], \quad R_{\mathrm{mul}} = w[\pi_R], \quad O_{\mathrm{mul}} = w[\pi_O],$$

and analogously for addition gates. These permutations $\pi_\star$ are entirely determined by the public description of C.

Our permutation check can be used directly to confirm that each routed column (e.g., L_{mul}) is consistent with w under $\pi_\star$.

In many structured circuits (e.g., FFT/NTT layers and Merkle hashing), the wiring permutations $\pi_\star$ are structured. By Theorem 5.1, the verifier can then evaluate $\widetilde{\mathrm{eq}}(\pi_\star(x), y)$ in $O(\mathrm{polylog}(n))$ time, so the prover does not commit to it, and no structural checks are needed. This means the prover can take advantage of any field-agnostic optimizations (e.g., small field Sumcheck [16]) and the NARK retains succinctness.

[2]It is well-known in algebraic complexity theory that Boolean circuits can be simulated by arithmetic circuits over any field with only constant-factor overhead. In particular, the standard embeddings $a \wedge b \mapsto a \cdot b$, $a \vee b \mapsto a + b - ab$, $a \oplus b \mapsto a + b - 2ab$, and $\neg a \mapsto 1 - a$ preserve correctness on $\{0,1\}$-inputs and increase size by at most a constant factor.

The other remaining gate constraints such as $L_{\text{mul}} \cdot R_{\text{mul}} = O_{\text{mul}}$ can be enforced by standard arithmetic checks [4] or lookups [7, 12].

5.1.3 Common structured permutations

Here we discuss three common structured permutations that can be evaluated in $O(\log n)$. This is not an exhaustive list.

1. *Full reversal* flips an entire array (i.e., $a_j = b_{n-1-j}, \ \forall j \in [0, n-1]$). It can be done by $\sigma(x)_i = NOT(x_i) = 1 - x_i$. Therefore, the multilinear extension of the permutation matrix becomes

$$\prod_{i=1}^{\log n} x_i + y_i - 2x_i y_i$$

   In signal processing, it is useful to compute cross-correlation between signals f and g since it is equivalent to 1D convolution with one input reversed. In ZKML, it can be applied on the inputs of a bidirectional recurrent neural network (RNN) [30].

2. *Bit reversal* reorders the array by flipping the index bits, so index $b_1 b_2 \dots b_{\log n}$ is sent to $b_{\log n} \dots b_2 b_1$. It means $\sigma(x)_i = x_{\log n + 1 - i}$. Therefore, the multilinear extension of the permutation matrix is

$$\prod_{i=1}^{\log n} 1 - x_{\log n+1-i} - y_i + 2x_{\log n+1-i} y_i$$

   It is a fundamental building block in FFT algorithms such as radix-2 Cooley-Tukey FFT [31].

3. *2D transpose* calculates the transpose of a 2D matrix. Let $n = 2^{p+q}$ and write the $\log n = \mu = p + q$ index bits as a row/column concatenation $x = (r_1, \dots, r_p \mid c_1, \dots, c_q)$. The 2D transpose swaps these groups:

$$\sigma(x) = (c_1, \dots, c_q \mid r_1, \dots, r_p).$$

   The multilinear extension of the permutation matrix can be explicitly expressed by

$$\prod_{i=1}^{q} \big(1 - y_i - c_i + 2y_i c_i\big) \prod_{j=1}^{p} \big(1 - y_{q+j} - r_j + 2y_{q+j} r_j\big).$$

   This technique can further be applied to do ND transpose, which is frequently used to run a convolutional neural network (CNN) with a GPU [32].

5.2 Non-Bijective Permutations

In previous sections, we assume a permutation is a bijective function (i.e., a one-to-one mapping between $\{0,1\}^{\log n} \to \{0,1\}^{\log n}$). Here we show our linear-time prover can support a non-bijective permutation π, where the instance of permutation check becomes

$$A = (a_0, \dots, a_{n-1}) \in \mathbb{F}^n, \ B = (b_0, \dots, b_{m-1}) \in \mathbb{F}^m, \ m \neq n.$$

Let a non-bijective permutation be a set $\pi \subset \{(i, j) \mid i \in [0, m-1], j \in [0, n-1]\}$, where $|\pi| = m$ and each i is unique in π. The relation to prove becomes

$$b_i = a_j, \text{ if } (i, j) \in \pi$$

This description allows us to form the permutation matrix M by simply iterating over the elements in π and setting $M_{ij} = 1$ if $(i, j) \in \pi$; $M_{ij} = 0$, otherwise.

To prove the above relation, $\mathcal{P}$ and $\mathcal{V}$ can check if their multilinear extensions follow the equation $\widetilde{B}(x) = \sum_{y \in \{0,1\}^{\log n}} \widetilde{A}(y)\widetilde{M}(x, y)$. Similar to Section 4, $\mathcal{V}$ sends a random challenge $r \in \mathbb{F}^{\log m}$, and uses the Sumcheck protocol with $\mathcal{P}$ to verify

$$\widetilde{B}(r) = \sum_{y \in \{0,1\}^{\log n}} \widetilde{A}(y)\widetilde{M}(r, y) \tag{19}$$

where $\widetilde{B}(r)$ can be computed by $2m$ field multiplications.

The difference between this and Section 4 is that the columns of M are no longer one-hot vectors (only the rows of M are still one-hot). This fact means $\widetilde{M}(r, y)$ should be computed in a slightly different way, which is

$$\begin{aligned}
\widetilde{M}(r, y) &= \sum_{i \in \{0,1\}^{\log m}} R_i(y)\widetilde{\text{eq}}(i, r) \\
&= (eq_{0,r}, \dots, eq_{m-1,r}) \cdot (\widetilde{R}_0(y), \dots, \widetilde{R}_{m-1}(y))^\top \\
&= (eq_{0,r}, \dots, eq_{m-1,r}) \cdot (M_{0y}, \dots, M_{(m-1)y})^\top \\
&= \sum_{(i,y) \in \pi} eq_{i,r}
\end{aligned} \tag{20}$$

Since $|\pi| = m$, we need at most m field additions to compute $\widetilde{M}(r, y)$. Then, $\mathcal{P}$ and $\mathcal{V}$ can use Sumcheck to prove Eq. 19 in $O(n)$ time. Since the time it takes to compute $\widetilde{B}(r)$ and $\widetilde{M}(r, y)$ is $O(m)$, the overall prover time is $O(m + n)$.

The relation with lookups. When $m \ll n$ and A is a public array for both $\mathcal{P}$ and $\mathcal{V}$, Eq. 19 becomes a lookup check, where $\mathcal{P}$ proves to $\mathcal{V}$ that all elements in B lie in a public or committed table A. Indeed, our PermPIOP becomes the simplest variant of Shout lookup [12]. Since an $O(m + n)$ prover is not practical, they leverage the fact that A is public and usually structured to achieve a much better asymptotic result (e.g., $O(m \log n)$). Their techniques cannot be adopted for permutation and multiset checks since A is only known to $\mathcal{P}$ and not structured in general.

6 Related Work

There are two lines of work on Sumcheck-based permutation checks. The first one is known as grand product check. Given two arrays $A = (a_1, a_2, \dots, a_n)$ and $B = (b_1, b_2, \dots, b_n)$ and a bijective permutation σ such that $a_{\sigma(i)} = b_i$, the grand product check can be used to prove the following equation holds

$$\prod_{i=1}^{n}(b_i + \beta \cdot i + \gamma) = \prod_{i=1}^{n}(a_i + \beta \cdot \sigma^{-1}(i) + \gamma) \tag{21}$$

where β and γ are challenges sent by the verifier. The arrays, index function i and the permutation σ^{-1} will be replaced by multilinear polynomials in Sumcheck-based protocols. In the multiset check, the β term can be discarded because there is no prescribed permutation.

Despite its linear-time prover, the soundness error increases linearly with the input size n because Eq. 21 is a degree-n univariate polynomial in terms of γ. Moreover, parts of the proof size and verifier time in the grand product check are used for checking auxiliary data beyond the witness arrays and the permutation. This is because the protocol [4] requires the prover to commit to the fraction of the partial grand product, which is

$$f_k = \frac{\prod_{i=1}^{k}(b_i + \beta \cdot i + \gamma)}{\prod_{i=1}^{k}(a_i + \beta \cdot \sigma^{-1}(i) + \gamma)}.$$

so that the prover can prove

$$f_k = f_{k-1}\frac{b_k + \beta \cdot k + \gamma}{a_k + \beta \cdot \sigma^{-1}(k) + \gamma}.$$

and $f_n = 1$ eventually.

To remove the $O(n)$ commitment costs, one may use a GKR circuit [6] to recursively verify that every f_k is well constructed layer by layer. However, this turns the proof size and the verifier time from $O(\log n)$ into $O(\log^2 n)$. Other variants of grand product check exist (e.g., logarithmic derivatives [33]), but they share similar properties: an $O(n)$ prover with $O(n)$-soundness error and some overheads for handling the auxiliary in the protocols.

The second line of permutation check is also proposed by HyperPlonk [4]. The main goal is to reduce the $O(n)$-soundness error to $O(\log n)$ so that a permutation check can be secure even in small fields. However, its $O(n \log n \log^2(\log n))$-time prover [4] prohibits the widespread use of this protocol. A concurrent work [8] proposes BiPerm and MulPerm, which improves the prover time to $O(n)$ and $O(n\sqrt{\log n})$, respectively. Instead of proving permutation as a single matrix-vector product, BiPerm arithmetizes the linear-time permutation check as

$$\begin{aligned}\widetilde{B}(r_L, r_R) &= \sum_{y \in \{0,1\}^{\log n}} \widetilde{A}(y) M((r_L, r_R), y) \\ &= \sum_{y \in \{0,1\}^{\log n}} \widetilde{A}(y) \widetilde{M}_L(r_L, y) \widetilde{M}_R(r_R, y)\end{aligned}$$

where $r_L, r_R \in \mathbb{F}^{0.5 \log n}$ and $M((x_L, x_R), y) = \widetilde{M}_L(x_L, y) \cdot \widetilde{M}_R(x_R, y), \forall x, y \in \{0,1\}^{\log n}$, requiring preprocessing of the permutation σ into two $1.5 \log n$-variate multilinear polynomials. This formulation requires BiPerm to preprocess σ and makes it difficult to verify its structure. Moreover, since the Sumcheck PIOP becomes degree-3 instead of 2 and at the end of the protocol the verifier needs to query four oracles $[[\widetilde{M}_L(x_L, y)]], [[\widetilde{M}_R(x_R, y)]], [[\widetilde{A}(x)]]$ and $[[\widetilde{B}(x)]]$ rather than three, our linear-time permutation PIOP has a better prover complexity, verifier complexity, and query complexity, resulting in better prover time, verifier time, and the proof size with the same PCS (e.g., Dory [14]).

Furthermore, since BiPerm provides no bijectivity check for the permutation, the permutation cannot be unknown to the verifier; this means whether an $O(n)$-time multiset prover exists with $O(\log n)$ soundness error remains an open problem [4]. We close this knowledge gap in this paper. To learn the exact formulation of the permutation check, we refer readers to Section 2.8 for more details.

7 Conclusion

We have introduced new permutation and multiset arguments that pair *linear-time* provers with *logarithmic* soundness, while avoiding commitments to auxiliary data. Our design arithmetizes the relation as a matrix-vector product and leverages the observation that the needed partial evaluation of the permutation multilinear is just the inverse permutation applied to the challenge vector. This leads to concrete efficiency gains and supports prover-provided permutations without preprocessing via $O(n)$-time structural checks. Instantiated with a sparsity-aware PCS (e.g., Dory), the resulting SNARKs remain $O(n)$-time for the prover. We further identified special permutations that eliminate permutation commitments entirely, yielding a logarithmically sound, field-agnostic SNARK with $O(n)$ proving, and we proved a matching lower bound showing the prover complexity is optimal.

Looking ahead, it is natural to (1) explore alternative PCS instantiations (e.g., optimizing constants, amortization, or universality), (2) study the compatibility for recursion and folding, and (3) optimize for memory- or GPU-bound provers. We view these directions as promising steps toward making permutation- and multiset-heavy SNARKs both theoretically tight and practically scalable.

Acknowledgements

We would like to acknowledge the Open Philanthropy project and the VAIL project for funding this research in part.

References

[1] B.-J. Chen, S. Waiwitlikhit, I. Stoica, and D. Kang, “ZKML: An optimizing system for ML inference in zero-knowledge proofs,” in *Proceedings of the Nineteenth European Conference on Computer Systems*, ser. EuroSys ’24. New York, NY, USA: Association for Computing Machinery, 2024, p. 560–574.

[2] T. Liu, Z. Zhang, Y. Zhang, W. Hu, and Y. Zhang, “Ceno: Non-uniform, segment and parallel zero-knowledge virtual machine,” Cryptology ePrint Archive, Paper 2024/387, 2024.

[3] S. Chaliasos, I. Reif, A. Torralba-Agell, J. Ernstberger, A. Kattis, and B. Livshits, “Analyzing and benchmarking ZK-rollups,” Cryptology ePrint Archive, Paper 2024/889, 2024.

[4] B. Chen, B. Bünz, D. Boneh, and Z. Zhang, “Hyperplonk: Plonk with linear-time prover and high-degree custom gates.” Berlin, Heidelberg: Springer-Verlag, 2023, p. 499–530.

[5] A. Gabizon, Z. J. Williamson, and O. Ciobotaru, “PLONK: Permutations over Lagrange-bases for oecumenical noninteractive arguments of knowledge,” Cryptology ePrint Archive, Paper 2019/953, 2019.

[6] J. Zhang, T. Liu, W. Wang, Y. Zhang, D. Song, X. Xie, and Y. Zhang, “Doubly efficient interactive proofs for general arithmetic circuits with linear prover time,” in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS ’21. New York, NY, USA: Association for Computing Machinery, 2021, p. 159–177.

[7] A. Gabizon and Z. J. Williamson, “plookup: A simplified polynomial protocol for lookup tables,” Cryptology ePrint Archive, Paper 2020/315, 2020.

[8] B. Bünz, J. Chen, and Z. DeStefano, “Linear*-time permutation check,” Cryptology ePrint Archive, Paper 2025/1850, 2025.

[9] C. Lund, L. Fortnow, H. Karloff, and N. Nisan, “Algebraic methods for interactive proof systems,” in *Proceedings [1990] 31st Annual Symposium on Foundations of Computer Science*, 1990, pp. 2–10 vol.1.

[10] V. Vu, S. Setty, A. J. Blumberg, and M. Walfish, “A hybrid architecture for interactive verifiable computation,” in *Proceedings of the 2013 IEEE Symposium on Security and Privacy*, ser. SP ’13. USA: IEEE Computer Society, 2013, p. 223–237.

[11] G. Strang, *Linear Algebra and Its Applications*, 4th ed. Belmont, CA: Thomson, Brooks/Cole, 2006.

[12] S. Setty and J. Thaler, “Twist and shout: Faster memory checking arguments via one-hot addressing and increments,” Cryptology ePrint Archive, Paper 2025/105, 2025.

[13] S. Setty, J. Thaler, and R. Wahby, “Unlocking the lookup singularity with lasso,” 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*. Berlin, Heidelberg: Springer-Verlag, 2024, p. 180–209.

[14] J. Lee, “Dory: Efficient, transparent arguments for generalised inner products and polynomial commitments,” in *Theory of Cryptography: 19th International Conference, TCC 2021, Raleigh, NC, USA, November 8–11, 2021, Proceedings, Part II*. Berlin, Heidelberg: Springer-Verlag, 2021, p. 1–34.

[15] H. Zeilberger, B. Chen, and B. Fisch, “Basefold: Efficient field-agnostic polynomial commitment schemes from foldable codes,” in *Advances in Cryptology – CRYPTO 2024*, L. Reyzin and D. Stebila, Eds. Cham: Springer Nature Switzerland, 2024, pp. 138–169.

[16] S. Bagad, Y. Domb, and J. Thaler, “The sum-check protocol over fields of small characteristic,” Cryptology ePrint Archive, Paper 2024/1046, 2024.

[17] B. E. Diamond and J. Posen, "Succinct arguments over towers of binary fields," Cryptology ePrint Archive, Paper 2023/1784, 2023. [Online]. Available: https://eprint.iacr.org/2023/1784

[18] T. Kohrita and P. Towa, "Zeromorph: Zero-knowledge multilinear-evaluation proofs from homomorphic univariate commitments," Cryptology ePrint Archive, Paper 2023/917, 2023.

[19] A. Fiat and A. Shamir, "How to prove yourself: Practical solutions to identification and signature problems," in *Advances in Cryptology — CRYPTO' 86*, A. M. Odlyzko, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1987, pp. 186–194.

[20] J. Zhao, S. Setty, W. Cui, and G. Zaverucha, "Micronova: Folding-based arguments with efficient (on-chain) verification," in *2025 IEEE Symposium on Security and Privacy (SP)*, 2025, pp. 1964–1982.

[21] T. Xie, Y. Zhang, and D. Song, "Orion: Zero knowledge proof with linear prover time," in *Advances in Cryptology – CRYPTO 2022: 42nd Annual International Cryptology Conference, CRYPTO 2022, Santa Barbara, CA, USA, August 15–18, 2022, Proceedings, Part IV.* Berlin, Heidelberg: Springer-Verlag, 2022, p. 299–328.

[22] S. Bowe, "Bls12-381: New zk-snark elliptic curve construction," 2017. [Online]. Available: https://electriccoin.co/blog/new-snark-curve/

[23] P. S. L. M. Barreto and M. Naehrig, "Pairing-friendly elliptic curves of prime order," in *Proceedings of the 12th International Conference on Selected Areas in Cryptography*, ser. SAC'05. Berlin, Heidelberg: Springer-Verlag, 2005, p. 319–331.

[24] J. Thaler, "Time-optimal interactive proofs for circuit evaluation," in *Advances in Cryptology – CRYPTO 2013*, R. Canetti and J. A. Garay, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 71–89.

[25] T. Xie, J. Zhang, Y. Zhang, C. Papamanthou, and D. Song, "Libra: Succinct zero-knowledge proofs with optimal prover computation," in *Advances in Cryptology – CRYPTO 2019: 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18–22, 2019, Proceedings, Part III.* Berlin, Heidelberg: Springer-Verlag, 2019, p. 733–764.

[26] E. Ben-Sasson, A. Chiesa, and N. Spooner, "Interactive oracle proofs," in *Proceedings, Part II, of the 14th International Conference on Theory of Cryptography - Volume 9986.* Berlin, Heidelberg: Springer-Verlag, 2016, p. 31–60.

[27] E. Ben-Sasson, A. Chiesa, E. Tromer, and M. Virza, "Scalable zero knowledge via cycles of elliptic curves," *Algorithmica*, vol. 79, no. 4, p. 1102–1160, Dec. 2017.

[28] J. Zhang, T. Xie, Y. Zhang, and D. X. Song, "Transparent polynomial delegation and its applications to zero knowledge proof," *2020 IEEE Symposium on Security and Privacy (SP)*, pp. 859–876, 2020.

[29] A. Borodin, "On relating time and space to size and depth," *SIAM J. Comput.*, vol. 6, no. 4, p. 733–744, Dec. 1977.

[30] M. Schuster and K. Paliwal, "Bidirectional recurrent neural networks," *IEEE Transactions on Signal Processing*, vol. 45, no. 11, pp. 2673–2681, 1997.

[31] J. W. Cooley and J. W. Tukey, "An algorithm for the machine calculation of complex fourier series," *Mathematics of Computation*, vol. 19, pp. 297–301, 1965.

[32] C. Li, Y. Yang, M. Feng, S. Chakradhar, and H. Zhou, "Optimizing memory efficiency for deep convolutional neural networks on gpus," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '16. IEEE Press, 2016.

[33] U. Haböck, “Multivariate lookups based on logarithmic derivatives,” Cryptology ePrint Archive, Paper 2022/1530, 2022.

[34] A. Gruen, “Some improvements for the PIOP for ZeroCheck,” Cryptology ePrint Archive, Paper 2024/108, 2024. [Online]. Available: https://eprint.iacr.org/2024/108

[35] E. Savas and C. Koc, “The montgomery modular inverse-revisited,” *IEEE Transactions on Computers*, vol. 49, no. 7, pp. 763–766, 2000.

A Computation of Lagrange Basis Polynomials

Lemma A.1 ($O(n)$-time Lagrange basis evaluation [10], adapted from [12])**.** *Given a point $\gamma \in \mathbb{F}^{\log n}$, evaluations of all Lagrange basis polynomials at γ can be computed using n field multiplications (i.e., to compute $\widetilde{\mathrm{eq}}(x, \gamma), \forall x \in \{0,1\}^{\log n}$ given the random point γ). Moreover, for any vector $B \in \mathbb{F}^n$, the multilinear extension $\widetilde{B}(\gamma)$ can be computed using $2n$ field multiplications.*

Proof. For $i = 1, \dots, \log n$, define an array R_i of length 2^i indexed by $x \in \{0,1\}^i$, where

$$R_i[x] = \widetilde{\mathrm{eq}}(x, \gamma_1, \dots, \gamma_i).$$

Then each R_{i+1} can be computed from R_i via

$$R_{i+1}[x, 1] = \gamma_{i+1} \cdot R_i[x]$$

and

$$R_{i+1}[x, 0] = (1 - \gamma_{i+1}) \cdot R_i[x] = R_i[x] - R_{i+1}[x, 1].$$

Thus, constructing $R_{\log n}$ takes

$$1 + 2 + \cdots + 2^{\log n - 1} = n$$

multiplications in total. The final array $R_{\log n}$ gives the evaluations of all Lagrange basis polynomials at γ.

For any $B \in \mathbb{F}^n$, the value of its multilinear extension at γ is

$$\widetilde{B}(\gamma) = \sum_{x \in \{0,1\}^{\log n}} B[x] \cdot R_{\log n}[x],$$

which is an inner product requiring n additional multiplications. Together with the n multiplications to compute $R_{\log n}$, this totals $2n$ field multiplications. □

B Simplified Booleanity Check

Recall in Eq. 9, $\mathcal{P}$ and $\mathcal{V}$ perform the Sumcheck PIOP to verify if the following is zero:

$$\sum_{(x,y)\in\{0,1\}^{\log m+\log n}} [\widetilde{M}(x,y)^2 - \widetilde{M}(x,y)]\widetilde{\mathrm{eq}}(r_{\mathrm{bool}},x)\widetilde{\mathrm{eq}}\,(r'_{\mathrm{bool}},y) \tag{22}$$

where $r_{\mathrm{bool}} \in \mathbb{F}^{\log m}$, $r'_{\mathrm{bool}} \in \mathbb{F}^{\log n}$ are random challenges given by $\mathcal{V}$. When $\widetilde{M}(x,y)$ is the multilinear extension of the permutation matrix in our protocol, $m = n$. We show that the linear-time Booleanity prover introduced in Sec 7.2 of [12] can be simplified here.

Specifically, we propose two new ideas. To begin with, our construction only needs *one* field inversion in the entire protocol; however, [12] needs $O(\log n)$ field inversions. Secondly, we do not need the $O(n)$-time tree recomputation procedure described in [12], we show that it takes only $O(1)$ time to switch to standard linear-time Sumcheck prover after the first $\log n$ rounds.

Let $f(x,y) = s(x,y)t(x,y)$, where $s(x,y) = \widetilde{M}(x,y)^2 - \widetilde{M}(x,y)$, $t(x,y) = \widetilde{\mathrm{eq}}(r_{\mathrm{bool}},x)\widetilde{\mathrm{eq}}\,(r'_{\mathrm{bool}},y)$, and $\gamma = r_{\mathrm{bool}}$. Recall that in round i of Sumcheck PIOP (Protocol C), $\mathcal{P}$ sends a univariate polynomial

$$\begin{aligned} f_i(x_i) &:= \sum_{\mathbf{b}\in\{0,1\}^{2\log n-i}} f(r_1,\ldots,r_{i-1},x_i,\mathbf{b}) \\ &= \sum_{\mathbf{b}\in\{0,1\}^{2\log n-i}} s(r_1,\ldots,r_{i-1},x_i,\mathbf{b})t(r_1,\ldots,r_{i-1},x_i,\mathbf{b}) \end{aligned} \tag{23}$$

given the $i-1$ challenges $r_1,\ldots,r_{i-1} \in \mathbb{F}$ that $\mathcal{V}$ sent in previous $i-1$ rounds (one challenge per round). Since $f(x,y)$ is a degree-3 multivariate polynomial, $\mathcal{V}$ can reconstruct $f_i(x_i)$ if $\mathcal{P}$ sends four points $f_i(0), f_i(1), f_i(2)$ and $f_i(3)$ to $\mathcal{V}$. Moreover, [12] and [34] show that it suffices for $\mathcal{P}$ to send only two points $f_i(0)$ and $f_i(2)$ and $\mathcal{V}$ can recover $f_i(1)$ and $f_i(3)$ themselves. Below we focus on how to compute $f_i(c)$ for $c \in \{0,2\}$ in the first $\log n$ rounds.

Based on the definition of $\widetilde{M}$ and $\widetilde{\mathrm{eq}}$, Twist and Shout [12] shows the following two equations are true.

$$\begin{aligned} &s(r_1,\ldots,r_{i-1},c,\mathbf{b}) = \\ &\sum_{\mathbf{k}\in\{0,1\}^i} s(\mathbf{k},\mathbf{b})[\widetilde{\mathrm{eq}}(r_1,\ldots,r_{i-1},c,\mathbf{k})^2 - \widetilde{\mathrm{eq}}(r_1,\ldots,r_{i-1},c,\mathbf{k})] \end{aligned} \tag{24}$$

$$\begin{aligned} &t(r_1,\ldots,r_{i-1},c,\mathbf{b}) = \\ &\sum_{\mathbf{k}\in\{0,1\}^i} t(\mathbf{k},\mathbf{b})\frac{\widetilde{\mathrm{eq}}(r_1,\ldots,r_{i-1},c,\gamma_1,\ldots,\gamma_i)}{\widetilde{\mathrm{eq}}(\mathbf{k},\gamma_1,\ldots,\gamma_i)} \end{aligned} \tag{25}$$

Let $\mathbf{k} = (\mathbf{k}_1,\ldots,\mathbf{k}_i)$ be the binary representation of $k \in \mathbb{N}$ starting from the least significant bit. For simplicity, set $m_{i,k}(c) = \widetilde{\mathrm{eq}}(r_1,\ldots,r_{i-1},c,\mathbf{k})$, $v_i^{\mathrm{num}}(c) = \widetilde{\mathrm{eq}}(r_1,\ldots,r_{i-1},c,\gamma_1,\ldots,\gamma_i)$, and $v_{i,k}^{\mathrm{den}}(c) = \widetilde{\mathrm{eq}}(\mathbf{k},\gamma_1,\ldots,\gamma_i)$, and they all equal to 1 when $i = 0$. Let $g_{i,k}(c)$ be

$$[m_{i,k}(c)^2 - m_{i,k}(c)]\frac{v_i^{\mathrm{num}}(c)}{v_{i,k}^{\mathrm{den}}(c)},$$

then $f_i(c)$ becomes

$$\begin{aligned} &\sum_{\mathbf{b}\in\{0,1\}^{2\log n-i}} \sum_{\mathbf{k}\in\{0,1\}^i} s(\mathbf{k},\mathbf{b})t(\mathbf{k},\mathbf{b})g_{i,k}(c) \\ &= \sum_{\mathbf{k}\in\{0,1\}^i} L_i(\mathbf{k})g_{i,k}(c) \end{aligned} \tag{26}$$

where $L_i(\mathbf{k}) = \sum_{\mathbf{b}\in\{0,1\}^{2\log n - i}} s(\mathbf{k},\mathbf{b})t(\mathbf{k},\mathbf{b})\ \forall i \in [1, \log n], \mathbf{k} \in \{0,1\}^i$ and can be computed in $O(n)$ time in the beginning of the Sumcheck PIOP as demonstrated by [12]. They also show it is possible to compute $g_{i,k}(c)$ for each i and k in $O(1)$ time in the i-th round. Specifically, they show

$$m_{i,k}(c) = m_{i-1,k_{\text{rem}}}(r_{i-1})\widetilde{\text{eq}}(c, k_{\text{quo}})$$

where $k = 2^{i-1}k_{\text{quo}} + k_{\text{rem}};\ k_{\text{quo}} \in \{0,1\}, 0 \le k_{\text{rem}} < 2^{i-1}$ and

$$\frac{v_i^{\text{num}}(c)}{v_{i,k}^{\text{den}}(c)} = \frac{v_{i-1}^{\text{num}}(r_{i-1})\widetilde{\text{eq}}(c,\gamma_i)}{v_{i-1,k_{\text{rem}}}^{\text{den}}(c)\widetilde{\text{eq}}(k_{\text{quo}},\gamma_i)},$$

a batch inversion [35] can be used here so that only 1 field inversion is needed in i-th round. Therefore, the total number of field inversions is $O(\log n)$ in [12].

Our key observation is $v_{i,k}^{\text{den}}(c)$ is independent of c and any r_i; therefore, it can be computed in the beginning of the Sumcheck prover as well. Indeed, $v_{i,k}^{\text{den}}$ is equal to $R_i[\mathbf{k}]^{-1}$ introduced in Appendix A and all R_i arrays $\forall i \in [1, \log n]$ were already computed within n field multiplications when computing $\widetilde{\text{eq}}(r_{\text{bool}}, x)$ (recall we set $\gamma = r_{\text{bool}}$). Once we have all R_i arrays, we can use only *one* batch inversion instead of $O(\log n)$ in the entire prover time for each $v_{i,k}^{\text{den}}$.

In the i-th round, the prover performs 2^i field multiplications to compute Eq. 26 given $L_i(\mathbf{k})$ and $g_{i,k}(c)$, so the total time complexity for the first $\log n$ rounds is $O(n)$. To proceed to the next $\log n$ rounds, one can continue to use the protocol described in [12]; however, this requires an additional $O(n)$ tree recomputation procedure to update the map $L_i(\mathbf{k})$.

The second key observation for the Booleanity check is one can switch to the classic linear-time Sumcheck prover *freely* for checking

$$\sum_{y\in\{0,1\}^{\log n}} [\widetilde{M}(r,y)^2 - \widetilde{M}(r,y)]\widetilde{\text{eq}}(r_{\text{bool}}, r)\widetilde{\text{eq}}\left(r'_{\text{bool}}, y\right) \tag{27}$$

where $r \in \mathbb{F}^{\log n}$ is the challenge $\mathcal{V}$ gave in the first $\log n$ rounds. To achieve this goal, we need to obtain $\widetilde{\text{eq}}(r_{\text{bool}}, r)\widetilde{\text{eq}}\left(r'_{\text{bool}}, y\right)$ and $\widetilde{M}(r,y)$ in constant time. First, recall $\widetilde{\text{eq}}\left(r'_{\text{bool}}, y\right)$ was already computed in the beginning of the first $\log n$ rounds and $\widetilde{\text{eq}}(r_{\text{bool}}, r) = \widetilde{\text{eq}}(\gamma, r) = v_{\log n}^{\text{num}}(r_{\log n})$. Next, since $\widetilde{M}(x,y)$ is the multilinear extension of a permutation matrix, from Eq. 15, we can compute $\widetilde{M}(r,y)$ without any field operations given $\widetilde{\text{eq}}(r,y), \forall y \in \{0,1\}^{\log n}$, which is exactly $m_{i,k}(c)$ when $i = \log n, c = r_{\log n}$ and $\mathbf{k} = y$.

C The Sumcheck PIOP

Protocol C: Sumcheck PIOP for $\mathcal{R}_{\mathrm{SUM}}$ [4, 9]

Goal: Given a tuple $(\mathbf{x}; \mathbf{w}) = (H, [[f]]; f)$ for μ-variate degree d polynomial $f \in \mathcal{F}_\mu^{(\leq d)}$, prove that $H = \sum_{x \in \{0,1\}^\mu} f(x)$.

The protocol proceeds in μ rounds:

Round 1: $\mathcal{P}$ sends univariate polynomial

$$f_1(x_1) := \sum_{b_2, \ldots, b_\mu \in \{0,1\}} f(x_1, b_2, \ldots, b_\mu)$$

$\mathcal{V}$ checks $H = f_1(0) + f_1(1)$ and sends random challenge $r_1 \in \mathbb{F}$.

Round i (for $2 \leq i \leq \mu - 1$): $\mathcal{P}$ sends univariate polynomial

$$f_i(x_i) := \sum_{b_{i+1}, \ldots, b_\mu \in \{0,1\}} f(r_1, \ldots, r_{i-1}, x_i, b_{i+1}, \ldots, b_\mu)$$

$\mathcal{V}$ checks $f_{i-1}(r_{i-1}) = f_i(0) + f_i(1)$ and sends random challenge $r_i \in \mathbb{F}$.

Round μ: $\mathcal{P}$ sends univariate polynomial

$$f_\mu(x_\mu) := f(r_1, r_2, \ldots, r_{\mu-1}, x_\mu)$$

$\mathcal{V}$ checks $f_{\mu-1}(r_{\mu-1}) = f_\mu(0) + f_\mu(1)$, sends random challenge $r_\mu \in \mathbb{F}$, and outputs 1 if $f_\mu(r_\mu) = f(r_1, r_2, \ldots, r_\mu)$ by querying the oracle $[[f]]$.