Linear-Time Permutation Check

Bünz, Chen & DeStefano (2025)

[name]

Shanghai Jiao Tong University

October 4, 2026

Roadmap

1. **Motivation**: Why do we need better permutation checks?
2. **Key insight**: The arithmetization spectrum of $\widetilde{1}_\sigma$
3. **BiPerm**: Linear-time via 2-way split (sparse PCS only)
4. **MulPerm**: Near-linear-time via ℓ-way split (any PCS)
5. **Extensions**: Prover-provided permutation & lookups
6. **Applications**: Improving HyperPlonk & Spartan

Permutation Check: A Core SNARK Primitive

Given $f, g : \{0,1\}^\mu \to \mathbb{F}$ and a permutation $\sigma : \{0,1\}^\mu \to \{0,1\}^\mu$, prove

$$f(x) = g(\sigma(x)) \quad \forall\, x \in \{0,1\}^\mu, \qquad n = 2^\mu.$$

Where does this appear?

- **Circuit wiring**: Plonk/HyperPlonk copy constraints
- **Memory checking**: RAM-to-sorted-access consistency
- **Lookup arguments**: Table lookups as a special case

$\Rightarrow$ Prover cost of permutation check **dominates** overall SNARK prover time.

Prior Art I: Grand Product (Plonk)

Core idea [GWC19]: $f(x) = g(\sigma(x))$ iff for random $\beta, \gamma \leftarrow \mathbb{F}$,

$$\prod_{x \in \{0,1\}^\mu} \big(f(x) + \beta \cdot x + \gamma\big) = \prod_{x \in \{0,1\}^\mu} \big(g(x) + \beta \cdot \sigma(x) + \gamma\big)$$

Prover builds an **accumulator** $z : \{0,1\}^\mu \to \mathbb{F}$ with $z(0) = 1$ and

$$z(\mathsf{next}(x)) = z(x) \cdot \frac{f(x) + \beta \cdot x + \gamma}{g(x) + \beta \cdot \sigma(x) + \gamma}, \qquad z(\mathsf{last}) \overset{!}{=} 1.$$

Drawbacks:

- **Extra commitment:** Must commit to z — a size-n vector of *random field elements*. Each $z(x)$ is a product of random ratios $\Rightarrow$ not structured (not in $\{0,1\}$, no sparsity). MSM cost is full $\mathcal{O}(n)$ scalar mults with no shortcuts.
- **Soundness $n/|\mathbb{F}|$:** A non-permutation can still satisfy the product check with probability $\leq n/|\mathbb{F}|$ (from Schwartz–Zippel on a degree-n univariate). For $n = 2^{32}$, $|\mathbb{F}| = 2^{64}$, this is only 2^{-32}.
- Runtime commitment to z is often the **bottleneck** of the entire SNARK.

Prior Art II: GKR-Style & HyperPlonk

GKR-style [Spartan/SPARK]: View the grand product as a depth-$\log n$ circuit, use GKR [GKR08] to verify it.

- Prover $\mathcal{O}(n)$ field ops; no extra commitment (removes the accumulator)
- **Drawback 1:** $\mathcal{O}(\log^2 n)$ rounds, proof size, and verifier time
- **Drawback 2:** Still soundness $n/|\mathbb{F}|$ (inherited from the product check)
- **Drawback 3:** Super-constant round GKR has Fiat–Shamir security concerns [KRS25]; significantly increases prover latency and implementation complexity

HyperPlonk [CBBZ23]: Sumcheck-based with degree-μ arithmetization $\widetilde{1}_\sigma(X, Y) = \prod_{i=1}^{\mu} \mathrm{eq}(\widetilde{\sigma}_i(X), Y_i)$.

- Better soundness: $\mathcal{O}(\log^2 n/|\mathbb{F}|)$ via Schwartz–Zippel on the sumcheck
- Prover $\mathcal{O}(n \log^2 n)$: Degree μ in $X \Rightarrow$ computing each eq product costs $\mathcal{O}(\mu^2)$ per point
- Commits μ bit-oracles $\widetilde{\sigma}_1, \ldots, \widetilde{\sigma}_\mu$ (each size n) during preprocessing

This Paper's Contributions

Protocol	Prover	Soundness	PCS requirement
BiPerm	$\mathcal{O}(n)$	$\mathcal{O}(\log n/\lvert\mathbb{F}\rvert)$	sparse PCS
MulPerm	$n \cdot \widetilde{\mathcal{O}}(\sqrt{\log n})$	$\mathrm{polylog}(n)/\lvert\mathbb{F}\rvert$	any PCS

Key improvements:

- Soundness: $2^{-96} \to 2^{-120}$ for $n = 2^{32}$, $|\mathbb{F}| = 2^{128}$
- No commitment beyond the witness (no accumulator/GKR)
- $\mathcal{O}(\log n)$ verifier and proof size for both
- Generalizes to **lookups** and **prover-provided** permutations

Notation Refresher

Equality polynomial: $\mathrm{eq}(a, b) = ab + (1-a)(1-b)$ for $a, b \in \mathbb{F}$.
For vectors: $\mathrm{eq}(\mathbf{a}, \mathbf{b}) = \prod_{i=1}^{\mu} \mathrm{eq}(a_i, b_i)$. Satisfies $\mathrm{eq}(\mathbf{a}, \mathbf{b}) = 1$ iff $\mathbf{a} = \mathbf{b}$ over $\{0,1\}^\mu$.

Multilinear extension (MLE): For $f : \{0,1\}^\mu \to \mathbb{F}$, its unique MLE is
$\widetilde{f}(X) = \sum_{x \in \{0,1\}^\mu} f(x) \cdot \mathrm{eq}(x, X)$.

Sumcheck protocol [LFKN'90]: Proves $\sum_{x \in \{0,1\}^\mu} f(x) = v$ for degree-d polynomial f.
Soundness $d\mu/|\mathbb{F}|$; prover sends μ univariate messages of degree $\leq d$.

Permutation as multilinear map: $\sigma : \{0,1\}^\mu \to \{0,1\}^\mu$ encoded as $\widetilde{\sigma}(X) = (\widetilde{\sigma}_1(X), \ldots, \widetilde{\sigma}_\mu(X))$,
each $\widetilde{\sigma}_i$ is an MLE with $\widetilde{\sigma}_i(x) \in \{0,1\}$ over $\{0,1\}^\mu$.

Reduction to Sumcheck (Lemma 4)

Indicator function: $1_\sigma(x, y) = \begin{cases} 1 & \text{if } \sigma(x) = y \\ 0 & \text{otherwise} \end{cases}$

Key reduction: For random $\alpha \leftarrow \mathbb{F}^\mu$, proving $f(x) = g(\sigma(x))$ for all x reduces to

$$\boxed{\sum_{x \in \{0,1\}^\mu} f(x) \cdot \widetilde{1}_\sigma(x, \alpha) = g(\alpha)}$$

via Schwartz–Zippel on α.

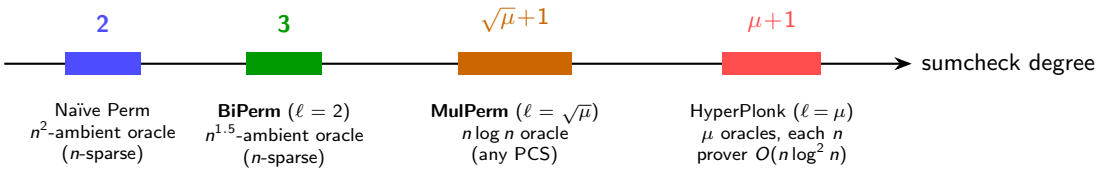
$\Rightarrow$ **The entire protocol design reduces to choosing the arithmetization of $\widetilde{1}_\sigma(X, Y)$.**
Different arithmetizations $\Rightarrow$ different degrees in X $\Rightarrow$ different prover costs.

The Arithmetization Spectrum

Two ways to write the indicator $1_\sigma(X, Y)$ as a polynomial:

Full MLE (2μ variables, degree 1 in each X_i): $\widetilde{1}_\sigma(X, Y) = \sum_{x \in \{0,1\}^\mu} \mathrm{eq}(x \| \sigma(x),\, X \| Y)$. Oracle is n^2-ambient but only n-sparse. After fixing $Y = \alpha$, the slice $T_\alpha(X) = \mathrm{eq}(\sigma(X), \alpha)$ is multilinear $\Rightarrow$ sumcheck with $f \cdot T_\alpha$ has **degree 2**.

Factored (group μ bit-wise relations into ℓ groups, degree ℓ in each X_i): $\widetilde{1}_\sigma(X, Y) = \prod_{j=1}^{\ell} \widetilde{1}_j(X, Y^{(j)})$. Sumcheck degree $\ell + 1$.



Naïve Perm introduces a polynomial with 2μ variables (ambient size n^2). The paper argues this leads to "over linear opening cost during compilation, even with the best known PCS."

BiPerm: The Two-Way Split

Split $Y = (Y_L, Y_R)$ with $|Y_L| = |Y_R| = \mu/2$. Define

$$R_{\sigma_L} = \bigcap_{i \le \mu/2} R_{\sigma_i}, \qquad R_{\sigma_R} = \bigcap_{i > \mu/2} R_{\sigma_i}.$$

Arithmetize as product of two sub-indicators:

$$\widetilde{1}_{\sigma}(X, Y) = \widetilde{1}_{\sigma_L}(X, Y_L) \cdot \widetilde{1}_{\sigma_R}(X, Y_R)$$

Substituting into Lemma 4:

$$\boxed{\sum_{x \in \{0,1\}^{\mu}} f(x) \cdot \widetilde{1}_{\sigma_L}(x, \alpha_L) \cdot \widetilde{1}_{\sigma_R}(x, \alpha_R) = g(\alpha)}$$

This is a **single sumcheck** with degree 3 in each variable of x.

BiPerm: Why the Prover Is $\mathcal{O}(n)$

For $X \in \{0,1\}^\mu$:

$$\widetilde{1}_{\sigma_L}(X, Y_L) = \sum_{x \in \{0,1\}^\mu} \mathrm{eq}(x, X) \cdot \mathrm{eq}(\sigma_L(x), Y_L) = \mathrm{eq}(\sigma_L(X), Y_L)$$

Building the evaluation table of $\widetilde{1}_{\sigma_L}(\cdot, \alpha_L)$ over $\{0,1\}^\mu$:

1. Precompute $\mathrm{eq}(y_L, \alpha_L)$ for all $y_L \in \{0,1\}^{\mu/2}$ — $\mathcal{O}(2^{\mu/2}) = \mathcal{O}(\sqrt{n})$
2. For each $x \in \{0,1\}^\mu$: look up $\mathrm{eq}(\sigma_L(x), \alpha_L)$ from the table — $\mathcal{O}(n)$

Same for $\widetilde{1}_{\sigma_R}(\cdot, \alpha_R)$.

The sumcheck itself is constant-degree (≤ 3), so the standard linear-time sumcheck algorithm applies.

$\Rightarrow$ **Total prover: $\mathcal{O}(n)$ field operations.**

BiPerm: The PCS Constraint

The sub-indicators live in 1.5μ variables:

$$\widetilde{1}_{\sigma_L}, \widetilde{1}_{\sigma_R} : \mathbb{F}^{1.5\mu} \to \mathbb{F}, \qquad \text{table size } 2^{1.5\mu} = n^{1.5}.$$

But each has only n non-zero entries ($= n$-**sparse**).

PCS family	Cost driver	BiPerm compatibility
Sparse-friendly (Dory, KZH)	sparsity n	**good fit**
Dense (KZG, FRI-based)	full table $n^{1.5}$	**too costly**

Theorem 4 (BiPerm).

- Prover: $\mathcal{O}(n)$ field ops Verifier: $\mathcal{O}(\log n)$ Soundness: $\mathcal{O}(\log n/|\mathbb{F}|)$
- Index oracles: two $n^{1.5}$-sized sparse polynomials
- **Limitation:** requires sparse PCS for linear prover cost

Remark: Does Splitting Actually Help? ("OnePerm")

BiPerm's linear prover relies on a simple trick: precompute $\mathrm{eq}(y_L, \alpha_L)$ over $\{0,1\}^{\mu/2}$, then look up each $\widetilde{1}_{\sigma_L}(x, \alpha_L) = \mathrm{eq}(\sigma_L(x), \alpha_L)$.

But the same trick works without splitting. Define "OnePerm":

1. Preprocess the full indicator $\widetilde{1}_\sigma(X, Y) = \sum_{x \in \{0,1\}^\mu} \mathrm{eq}(x \| \sigma(x), X \| Y)$
2. After receiving α, compute the slice $T_\alpha(x) = \mathrm{eq}(\sigma(x), \alpha)$ for all $x \in \{0,1\}^\mu$:
   - Precompute $\mathrm{eq}(y, \alpha)$ for all $y \in \{0,1\}^\mu$ $\mathcal{O}(n)$
   - For each x: look up $\mathrm{eq}(\sigma(x), \alpha)$ $\mathcal{O}(n)$
3. Run a **degree-2 sumcheck**: $\sum_x f(x) \cdot T_\alpha(x) = g(\alpha)$ $\mathcal{O}(n)$

Comparison with BiPerm:

	PIOP prover	Oracle ambient size	Soundness
"OnePerm"	$\mathcal{O}(n)$	n^2 (n-sparse)	$\mathcal{O}(\log n/\lvert\mathbb{F}\rvert)$
BiPerm	$\mathcal{O}(n)$	$2 \times n^{1.5}$ (n-sparse each)	$\mathcal{O}(\log n/\lvert\mathbb{F}\rvert)$

Both need sparse PCS. The only difference is oracle ambient size (n^2 vs $n^{1.5}$). With a PCS whose cost depends on **sparsity** rather than ambient size (e.g. Dory, KZH), they are essentially equivalent.

MulPerm: Motivation

BiPerm's problem: The sub-indicators $\widetilde{1}_{\sigma_L}, \widetilde{1}_{\sigma_R}$ are committed in preprocessing. Without sparse PCS, commitment cost is $\Omega(n^{1.5})$.

MulPerm's approach:

- Do **not** preprocess/commit the sub-indicator functions
- Instead, the prover **computes them at runtime** and **proves their correctness** via an additional sumcheck
- Only preprocessed oracle: $\widetilde{\sigma}_{[\mu]}$ (size $n \log n$, maps to $\{0,1\}$)

Result: A "double-sumcheck" protocol compatible with **any** PCS.

MulPerm: ℓ-Way Factorization

Partition the μ output bits into ℓ groups of size μ/ℓ.

Define the **partial product** polynomial $p : \{0,1\}^{\mu+\log \ell} \to \mathbb{F}$:

$$p(x, \langle j \rangle) := \prod_{i=1}^{\mu/\ell} \mathrm{eq}\big(\alpha(\langle j' + i \rangle),\ \widetilde{\sigma}_{[\mu]}(\langle j' + i \rangle, x)\big)$$

where $j' = (j-1)\mu/\ell$ and $\langle j \rangle$ is the $\log \ell$-bit encoding of $j-1$.

Let $\widetilde{p}$ be the MLE of p. The permutation check becomes:

$$\boxed{\sum_{x \in \{0,1\}^{\mu}} f(x) \prod_{j \in [\ell]} \widetilde{p}(x, \langle j \rangle) = g(\alpha)}$$

Since $\widetilde{p}$ is **not** in the index, the prover must additionally prove $\widetilde{p}$ is correct $\Rightarrow$ **second sumcheck**.

Computing $\widetilde{p}$ Efficiently

Naïve cost: $n\ell \cdot (\mu/\ell) = n\mu$ field multiplications. Too expensive.

Key observation: Each $\widetilde{\sigma}_i(x) \in \{0,1\}$ over $\{0,1\}^\mu$, so $\mathrm{eq}(\alpha_i, \widetilde{\sigma}_i(x)) \in \{\alpha_i,\ 1-\alpha_i\}$ (only 2 possible values).
$\Rightarrow$ For fixed j, the product $p(\cdot, \langle j \rangle)$ takes at most $2^{\mu/\ell}$ distinct values across all $x \in \{0,1\}^\mu$.

Algorithm (bucketing):

1. Compute all $\ell \cdot 2^{\mu/\ell}$ distinct evaluations — $(\mu/\ell) \cdot \ell \cdot 2^{\mu/\ell}$ mults
2. For each (x, j): look up the corresponding value — $\mathcal{O}(n\ell)$ lookups

Total Mults: $\mu \cdot 2^{\mu/\ell} = \mu \cdot n^{1/\ell} = o(n)$ for any $\ell > 1$.

MulPerm: The Double Sumcheck

First sumcheck (degree $\ell + 1$ in x):

$$\sum_{x \in \{0,1\}^\mu} f(x) \prod_{j \in [\ell]} \widetilde{p}(x, \langle j \rangle) = g(\alpha) \quad \xrightarrow{\text{reduces to}} \quad \widetilde{p}(\beta, \langle j \rangle) = P_j \ \ \forall j \in [\ell]$$

Prover cost: $n \cdot \widetilde{\mathcal{O}}(\ell)$ Soundness: $(\ell + 1)(\mu + \log \ell)/|\mathbb{F}|$

Second sumcheck (degree $\mu/\ell + 1$, proving $\widetilde{p}$ is correct):

$$\sum_{\substack{x \in \{0,1\}^\mu \\ j \in [\ell]}} \mathrm{eq}(\beta', x \| \langle j \rangle) \underbrace{\prod_{i=1}^{\mu/\ell} \mathrm{eq}\big(\alpha(\langle j' + i \rangle), \widetilde{\sigma}_{[\mu]}(\langle j' + i \rangle, x)\big)}_{= p(x, \langle j \rangle)} = S_{\widetilde{p}}$$

Prover cost: $n \cdot \widetilde{\mathcal{O}}(\mu/\ell) + \ell \cdot 2^\ell$

The Bucketing Trick in the Second Sumcheck

Problem: Degree $\mu/\ell + 1$ is high $\Rightarrow$ naïve cost $n \cdot \widetilde{\mathcal{O}}(\mu/\ell)$ per round.

Observation: In round k, each univariate $\widetilde{\sigma}_{[\mu]}(\langle i \rangle, (\gamma, X, x'))$ has at most 2^{2^k} polynomial identities (from $\{0, 1, X, 1 - X\}$ possibilities).

$\Rightarrow$ The partial product p has at most $2^{2^k \mu/\ell} \cdot \ell$ distinct polynomial identities.

Strategy:

- **Rounds** $k < \log \ell$: Use bucketing — group terms by polynomial identity, compute eq-sums per bucket. Cost: $\mathcal{O}(\mu^2/\ell) \cdot 2^{2^k \mu/\ell}$ mults.
- **Round** $k' = \log \ell$: # identities exceeds direct computation cost. **Switch** to standard sumcheck algorithm.

The switch requires computing collapsed evaluation tables: cost $\ell \cdot 2^\ell$ field ops (Algorithm 11).

The Collapse Algorithm (Algorithm 11)

Goal: At round $k' = \log \ell$, switch to direct computation. Need evaluation tables of $\widetilde{\sigma}_{[\mu]}(\langle i \rangle, (\gamma, x))$ for all $i \in [\mu]$, where $\gamma \in \mathbb{F}^{k'}$ are prior challenges, $x \in \{0,1\}^{\mu - k'}$.

Key insight: $\widetilde{\sigma}_{[\mu]}(\langle i \rangle, \cdot)$ is $\{0,1\}$-valued on $\{0,1\}^{\mu}$. Each block of $2^{k'} = \ell$ entries (over the first k' Boolean variables) forms a bit string $s \in \{0,1\}^{\ell}$. Only 2^{ℓ} possible patterns $\Rightarrow$ precompute all of them.

Step 1 — Build lookup table T (size 2^{ℓ}): For each $s \in \{0,1\}^{\ell}$, compute $\mathrm{Fold}(s, \gamma)$ — standard MLE evaluation that iteratively collapses Boolean variables to challenges:

$$s'_j \leftarrow s_j (1 - \gamma_i) + s_{|s|/2+j}\, \gamma_i, \qquad i = 1, \ldots, \log \ell$$

After $\log \ell$ rounds: $\ell \to \ell/2 \to \cdots \to 1$ field element. Cost per call: $\mathcal{O}(\ell)$.
Total table construction: $2^{\ell} \times \mathcal{O}(\ell) = \mathcal{O}(\ell \cdot 2^{\ell})$ field mults.

Step 2 — Lookup: For each (i, j) with $i \in [\mu]$, $j \in [n/\ell]$: read the ℓ-bit block from the original evaluation table of $\widetilde{\sigma}_{[\mu]}(\langle i \rangle, \cdot)$, look up $T[\text{block}]$. **Zero** additional field operations.

Total collapse cost: $\ell \cdot 2^{\ell} = \sqrt{\log n} \cdot 2^{\sqrt{\log n}} = o(n)$. Sublinear!

MulPerm: Prover Complexity Analysis

Three components:

1. **Pre-computation:** evaluate $\widetilde{p}$ over $\{0,1\}^{\mu+\log \ell}$ via bucketing (Lemma 5): $\mu \cdot n^{1/\ell} = o(n)$.
2. **First sumcheck** (degree $\ell + 1$, μ rounds): $n \cdot \widetilde{\mathcal{O}}(\ell)$ field ops.
3. **Second sumcheck** (degree $\mu/\ell + 1$, $\mu + \log \ell$ rounds):
   - *Bucketing phase* ($k < \log \ell$): round-k cost $\mathcal{O}(\mu^2/\ell) \cdot 2^{2^k \mu/\ell}$; geometric $\Rightarrow$ total $\ll n$.
   - *Collapse* ($k = \log \ell$): $\ell \cdot 2^\ell$ field ops (Algorithm 11).
   - *Direct phase* ($k > \log \ell$): $n \cdot \mathcal{O}(\mu^2/\ell^2) \xrightarrow{\text{FFT}} n \cdot \widetilde{\mathcal{O}}(\mu/\ell)$.

Total:

$$n \cdot \widetilde{\mathcal{O}}\Big(\ell + \frac{\mu}{\ell}\Big) + \ell \cdot 2^\ell \quad \xrightarrow{\ell = \sqrt{\mu}} \quad n \cdot \widetilde{\mathcal{O}}(\sqrt{\log n})$$

Theorem 5 (MulPerm).

- Prover: $n \cdot \widetilde{\mathcal{O}}(\sqrt{\log n})$ Verifier/Proof: $\mathcal{O}(\log n)$
- Soundness: $\mathcal{O}(\mu^{1.5}/|\mathbb{F}|) = \mathrm{polylog}(n)/|\mathbb{F}|$ **Any PCS**

Prover-Provided Permutation

Setting: σ is not preprocessed — e.g. memory-checking, where the mapping depends on the witness.

Prover must additionally show σ is a permutation:

1. Provide $\widetilde{\tau}_{[\mu]}$ (claimed σ^{-1})
2. Prove $\tau(\sigma(y)) = y$ for all $y \in \{0,1\}^\mu$:

$$\sum_{x \in \{0,1\}^\mu} x \cdot \widetilde{1}_\sigma(x, \alpha) = \widetilde{\tau}(\alpha) \qquad \text{(batched with main permcheck via random } r\text{)}$$

3. Prove σ maps to binaries: $\widetilde{\sigma}_{[\mu]}(i, x) \in \{0,1\}$ for all $x \in \{0,1\}^\mu, i \in [\mu]$:

$$\sum_{x \in \{0,1\}^\mu, i \in [\mu]} \mathrm{eq}((x, \langle i \rangle), s) \cdot \widetilde{\sigma}_{[\mu]}(i, x)(1 - \widetilde{\sigma}_{[\mu]}(i, x)) = 0$$

Result (Thm 6): Triple-sumcheck PIOP, prover $n \cdot \widetilde{\mathcal{O}}(\sqrt{\log n})$, soundness $\mathrm{polylog}(n)/|\mathbb{F}|$.

Binary Check: Bucketing Revisited

Goal: Prove $\widetilde{\sigma}_{[\mu]}(i,x) \in \{0,1\}$ for all $x \in \{0,1\}^{\mu}$, $i \in [\mu]$.

Define $h(x,i) := \widetilde{\sigma}_{[\mu]}(i,x)\big(1 - \widetilde{\sigma}_{[\mu]}(i,x)\big)$. Let $\mu' = \mu + \log \mu$. Sumcheck:

$$\sum_{x' \in \{0,1\}^{\mu'}} \mathrm{eq}(x', s) \cdot h(x') = 0, \qquad s \leftarrow \mathbb{F}^{\mu'}$$

Degree 3 in each variable $\Rightarrow$ naïve cost $\mathcal{O}(n\mu)$ multiplications.

Trick — "Prover buckets in their head": Fold the last b variables of h using partial sums with components of s:

$$h^{(b)}(x', s'') = \sum_{z \in \{0,1\}^{b}} h(x' \| z) \cdot \mathrm{eq}(z, s'')$$

Replace sumcheck target with $\sum_{x' \in \{0,1\}^{\mu'-b}} \mathrm{eq}(x', s_{[:\mu'-b]}) \cdot h^{(b)}(x', s_{[-b:]}) = 0$.

Why it helps: Same as MulPerm — $\widetilde{\sigma}_{[\mu]}$ is binary-valued, so in round k, $h^{(b)}$ has $\leq 2^{2^{b+k}}$ polynomial identities. Set $b = \log \mu' - k - 2$:

- Multiplications: $8 \cdot 2^{2^{\log \mu' - 2}} = 8(n\mu)^{1/4}$ per round $\Rightarrow o(n)$ total.
- Additions: $2^{\mu' - \log \mu' + 2}$ per round $\Rightarrow \mathcal{O}(n \log \log n)$ over $\log \mu'$ rounds.

After $\log \mu'$ rounds, switch to direct (Collapse): remaining $o(n)$.

Generalization to Lookups

Replace permutation $\sigma : \{0,1\}^\mu \to \{0,1\}^\mu$ with a **map** $\rho : \{0,1\}^\mu \to \{0,1\}^\kappa$ ($T = 2^\kappa$).
Prove: $f(\rho(x)) = g(x)$ for all $x \in \{0,1\}^\mu$, where f is a κ-variate table.

Extra challenge: ρ is not injective $\Rightarrow$ no inverse, cannot apply Schwartz–Zippel directly on α.

Solution: Add an outer sum with random $s \in \mathbb{F}^\mu$:

$$\underbrace{\sum_{y \in \{0,1\}^\kappa} f(y) \sum_{x \in \{0,1\}^\mu} \mathrm{eq}(x,s) \cdot \widetilde{1}_\rho(x,y)}_{\text{outer sumcheck (exploits sparsity of } 1_\rho)} = \sum_{x \in \{0,1\}^\mu} \mathrm{eq}(x,s) \cdot g(x)$$

After outer sumcheck reduces to a point, apply **MulPerm** on the inner sum.

Theorem 7 (MulLookup): Prover $n \cdot \widetilde{\mathcal{O}}(\sqrt{\log T})$ if $T \leq n$; soundness $\mathrm{polylog}(n+T)/|\mathbb{F}|$.

Application: Improving HyperPlonk & Spartan

HyperPlonk (copy constraints in Plonk-like systems):

Variant	Extra commitments	Prover field ops	PCS
Original HyperPlonk	μ bit-oracles	$\mathcal{O}(n \log^2 n)$	any
+ BiPerm	2 sparse oracles	$\mathcal{O}(n)$	sparse
+ MulPerm	1 oracle ($n \log n$)	$n \cdot \widetilde{\mathcal{O}}(\sqrt{\log n})$	any

Spartan / SPARK (R1CS with sparse matrices):

- Sparse matrix $\widetilde{M}(x, y) = \sum_j \widetilde{\mathrm{val}}(j) \cdot \widetilde{1}_{\mathrm{row}}(j, x) \cdot \widetilde{1}_{\mathrm{col}}(j, y)$
- $(\mathrm{row}, \mathrm{col})$ is a preprocessed lookup/permutation object
- $\Rightarrow$ Replace memory-checking + GKR by MulPerm-style lookup
- Result: avoid GKR entirely, better verifier cost ($\log n$ vs $\log^2 n$)

Summary Comparison

Protocol	Prover	Verifier	Soundness	PCS
Grand product	$\mathcal{O}(n)$+comm	$\mathcal{O}(\log n)$	$n/\lvert\mathbb{F}\rvert$	any
GKR-style	$\mathcal{O}(n)$	$\mathcal{O}(\log^2 n)$	$n/\lvert\mathbb{F}\rvert$	any
HyperPlonk	$\mathcal{O}(n \log^2 n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log^2 n/\lvert\mathbb{F}\rvert)$	any
BiPerm	$\mathcal{O}(n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n/\lvert\mathbb{F}\rvert)$	sparse
MulPerm	$n\widetilde{\mathcal{O}}(\sqrt{\log n})$	$\mathcal{O}(\log n)$	$\mathrm{polylog}(n)/\lvert\mathbb{F}\rvert$	any

- **BiPerm**: asymptotically optimal prover, but restricted to sparse PCS
- **MulPerm**: universal construction, near-linear, works with any PCS
- Both generalize to lookups and prover-provided permutations

Open Problems & Discussion

Open questions from the paper:

1. **Fully linear** permutation/lookup argument with no PCS dependency?
2. Reduce soundness error further to **constant**/$|\mathbb{F}|$?

Discussion points:

- Is the bucket→direct-computation switch at $k' = \log \ell$ optimal in practice?
- For concrete backends (IPA/FRI), where is the HyperPlonk $\leftrightarrow$ MulPerm crossover?
- Can the "small image space $\Rightarrow$ bucketing" idea be reused in other contexts (e.g. GKR layers, memory-checking)?
- If sparse PCS is available, does BiPerm dominate once preprocessing is amortized?

Reference: Bünz, Chen & DeStefano, *Linear-Time Permutation Check*, 2025.