Distributed SNARK via folding schemes

Zesheng Li[1,2], Dongliang Cai[3], Yimeng Tian[4], Yihang Du[5], Xinxuan Zhang[1,2], and Yi Deng[1,2]

[1] Key Laboratory of Cyberspace Security Defense, Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China
[2] School of Cyber Security, University of Chinese Academy of Sciences, Beijing, China
[3] School of Computer Science, Fudan University
[4] School of Telecommunications Engineering, Xidian University
[5] College of Computer Science and Electronic Engineering, Hunan University
lizesheng@iie.ac.cn
22110240060@m.fudan.edu.cn
22009100325@stu.xidian.edu.cn
202208060130@hnu.edu.cn
zhangxinxuan@iie.ac.cn deng@iie.ac.cn

Abstract. Succinct Non-interactive Arguments of Knowledge(SNARKs) have gained widespread application due to their succinct proof size and efficient verification. However, they face significant scalability limitations in proof generation for large-scale circuits. To address this challenge, distributing the prover's computation across multiple nodes has emerged as a promising solution. Existing distributed SNARK constructions rely on distributed polynomial commitments, requiring each prover to perform computationally intensive group operations during the polynomial commitment opening phase.

In this paper, we propose a novel distributed SNARK system constructed by compiling distributed PIOP with additively homomorphic polynomial commitment, rather than distributed polynomial commitment. The core technical component is distributed SumFold, which folds multiple sum-check instances into one. After the folding process, only one prover is required to perform polynomial commitment openings. It facilitates compilation with SamaritanPCS, which is a recently proposed additively homomorphic multilinear polynomial commitment scheme. The resulting SNARK system is specifically optimized for data-parallel circuits. Compared to prior HyperPlonk-based distributed proof systems (e.g., Hyperpianist and Cirrus), our construction achieves improvements in both proof size and prover time.

We implement our protocol and conduct a comprehensive comparison with HyperPianist with 8 machines. Our system achieves shorter proof and 4.1∼4.9× speedup in prover time, while maintaining comparable verification efficiency.

Keywords: SNARK · Distributed zero-knowledge proof · Scalable · Folding scheme

1 Introduction

Succinct non-interactive arguments of knowledge (SNARKs) have been an important cryptographic primitive enabling much shorter proof and fast verification of NP statement. Recent SNARK constructions [8,9,6,3] have been proven practical in blockchain and cryptocurrency. However, as applications of SNARKs expand to large-scale circuits, such as zkRollups, zkEVM and zkML, existing SNARK schemes are unable to meet the growing computational demands of proof generation. A feasible solution is to distribute the computation of proof generation to multiple machines. Although recent schemes like DIZK[23], deVirgo[25], Pianist[18], and HEKATON[21] have advanced this distributed approach, these systems still face inherent bottlenecks in efficiency due to the quasilinear prover complexity.

Recently, Hyperpianist [15] and Cirrus [22] independently proposed distributed SNARK systems and achieve linear complexity for the prover. They eliminate the need for FFT computations based on Hyperplonk system. The proposed distributed SNARK framework comprises two key components: a distributed polynomial interactive oracle proof (PIOP) and a distributed multilinear polynomial commitment scheme, where each prover must perform both commitment and opening phases, resulting in substantial computational overhead. While the computations in commit phase are inherently unavoidable, reducing the computational burden in the opening phase presents a natural optimization opportunity.

1.1 Our Contributions and Techniques

New framework for distributed SNARK system via folding scheme. Existing HyperPlonk-based distributed SNARK systems, including HyperPianist [15] and Cirrus [22], follow the conventional "distributed PIOP+distributed polynomial commitment" framework, which fundamentally ties the opening phase complexity to the distributed nature of the commitment scheme, making computational efficiency challenging to achieve. To overcome this challenge, we develop a novel "distributed PIOP+additively homomorphic polynomial commitment" framework for constructing distributed SNARK systems. In our distributed PIOP, we reduce the proof of multiple sub-circuits to a single circuit via folding scheme, while commitments to subcircuits from different provers are folded into one. Ultimately, it allows us to handle a single folded circuit. Notice that the SumFold protocol introduced in [13] supports efficient folding of multiple sum-check relations into one instance, thus we develop a distributed version of SumFold, which serves as the core component of our distributed PIOP. The complete construction of distributed SumFold is formally presented in Section 3. We compile our distributed PIOP with (non-distributed) additively homomorphic polynomial commitments, as the folding scheme requires linear combination of commitments. Our approach is particularly suited for data-parallel circuits, as folding schemes necessitate structural homogeneity across the circuits being proven.

Crucially, as the folded circuit size becomes comparable to a single subcircuit, our framework enables a *single* prover to perform the final opening operation instead of requiring participation from all provers, thereby significantly reducing the prover's computation time. Additionally, unlike the "distributed PIOP+distributed polynomial commitment" framework, which relies inherently on distributed commitment schemes, our framework is compatiable with non-distributed polynomial commitment. For instance, the recently proposed SamaritanPCS [7] scheme achieves smaller proof sizes and faster verification than both MKZG and Dory, but lacks a distributed version and thus cannot be applied to "distributed PIOP+distributed polynomial commitment" frameworks. In contrast, our framework seamlessly integrates with SamaritanPCS, yielding a substantial reduction in the overall proof size.

Distributed SNARK for Hyperplonk constraint system with optimized prover time and proof size. Building on the above novel framework, we integrate SamaritanPCS [7] with our distributed PIOP. This combination yields a distributed SNARK for data-parallel circuits that achieves optimized prover time and proof size within the HyperPlonk constraint system.

Table 1. Comparison between our work and existing distributed SNARK generation protocols. Comm. denotes the communication cost between provers. $\mathcal{P}_i$ Time denotes the time of each prover. $\mathcal{V}$ Time denotes the verifier time.

Scheme	$\mathcal{P}_i$ **Time**	**Proof size**	$\mathcal{V}$ **Time**	**Comm.**
Ours	$O(T)\mathbb{F}$ $+O(T)\mathbb{G}$	$O(\log N)\lvert\mathbb{F}\rvert$ $+O(1)\lvert\mathbb{G}\rvert$	$O(\log N)\mathbb{F}+O(M)\mathbb{G}$ $+O(1)\mathbb{P}$	$O(N)\lvert\mathbb{F}\rvert$ $+O(M)\lvert\mathbb{G}\rvert$
Cirrus[22]	$O(T)\mathbb{F}$ $+O(T)\mathbb{G}$	$O(\log N)\lvert\mathbb{F}\rvert$ $+O(\log N)\lvert\mathbb{G}\rvert$	$O(\log N)\mathbb{F}$ $+O(\log N)\mathbb{P}$	$O(M\cdot\log N)\lvert\mathbb{F}\rvert$ $+O(M\cdot\log N)\lvert\mathbb{G}\rvert$
HyperPianist [15]	$O(T)\mathbb{F}$ $+O(T)\mathbb{G}$	$O(\log N)\lvert\mathbb{F}\rvert$ $+O(\log N)\lvert\mathbb{G}\rvert$	$O(\log N)\mathbb{F}$ $+O(\log N)\mathbb{P}$	$O(M\cdot\log N)\lvert\mathbb{F}\rvert$ $+O(M\cdot\log N)\lvert\mathbb{G}\rvert$
Hekaton [21]	$O(T\cdot\log T)\mathbb{F}$ $+O(T)\mathbb{G}$	$O(\log M)\lvert\mathbb{G}\rvert$	$O(\log M)\mathbb{G}$	$O(\log M)\lvert\mathbb{G}\rvert$
Pianist[18] Soloist[17]	$O(T\cdot\log T)\mathbb{F}$ $+O(T)\mathbb{G}$	$O(1)\lvert\mathbb{G}\rvert$	$O(1)\mathbb{F}+O(1)\mathbb{G}$	$O(1)\lvert\mathbb{G}\rvert$
deVirgo [25]	$O(T\cdot\log T)\mathbb{F}$ $+O(T)\mathbb{H}$	$O(\log^2 N+M)\lvert\mathbb{H}\rvert$	$O(\log^2 N+M)\mathbb{H}$	$O(N)\lvert\mathbb{F}\rvert$

N: Number of gates (constraints) in the whole circuit
M: Number of workers; $T=N/M$: Size of each sub-circuit
$\mathbb{H}$: Hash operation; $\mathbb{F}$: Field operation; $\mathbb{G}$: Group operation; $\mathbb{P}$: Pairing operation
$|\cdot|$: The size of elements

In Table 1, we provide comparison with prior distributed SNARK systems in terms of asymptotic complexity. Compared to HyperPianist [15] and Cirrus [22], our construction achieves a more compact proof size. While maintaining the same asymptotic linear prover complexity, our scheme significantly reduces the computational overhead in the polynomial commitment opening phase, resulting in optimized practical prover performance.

Implementation and evaluation. We implement our distributed ZKP system using the Rust-based ark-works ecosystem.[6] For comparison, we benchmark against HyperPianist [15], the most recent prior work with publicly available source code. Our implementation demonstrates significant improvements: the prover runs at least 4.1× faster than HyperPianist's on 8 machines, while simultaneously achieving smaller proof sizes. Although the verifier time is slightly longer, it remains highly efficient. Detailed experimental results are presented in Section 5.

1.2 Related Works

Distributed ZKP. Wu et al. [23] pioneered distributed zero-knowledge proof systems with DIZK, a framework that scales Groth16[9]. While DIZK achieves significant parallelism, its communication overhead grows linearly with the circuit size due to cross-machine coordination during FFT operations. Moreover, DIZK inherits Groth16's[9] circuit-specific setup, limiting its flexibility compared to universal setups. DeVirgo[25] is a distributed ZKP system optimized for data-parallel circuits. Building on the multivariate polynomial protocol of [29], deVirgo's PIOP attains linear prover time. However, its reliance on FRI and Merkle-tree-based PCS results in linear per-worker communication costs. Pianist[18] extends Plonk [6] to the distributed setting by decomposing Plonk's global permutation check into local per-worker checks via bivariate polynomials. This yields constant proof size, verifier time, and per-node communication, but trades off quasi-linear prover time and trusted setup scalability. HyperPianist [15] and Cirrus[22] further advance efficiency by eliminating FFT operations using multilinear sum-check-based PIOPs, reducing prover overhead to nearly linear, though their verifier requires logarithmic pairing operations. Soloist[17] is a distributed SNARK optimized for R1CS, achieves constant proof size, communication, and verification—matching Pianist's asymptotics while avoiding its high cost from R1CS-to-Plonk conversion.

Several lines of work have explored code-based polynomial commitments, enabling the construction of transparent and post-quantum secure SNARKs. FRIttata[26] introduces a distributed SNARK framework that utilizes FRI and optimizes communication via a novel fold-and-batch technique. Concurrently, Li et al.[16] propose a distributed version of FRI that reduces opening time through the shred-to-shine technique. Furthermore, HyperFond[27], enhances distributed SNARKs by integrating HyperPlonk's PIOP system with a distributed BaseFold[28] polynomial commitment scheme, achieving polylogarithmic communication costs and linear proving time. All above SNARK follows "distributed PIOP+distributed polynomial commitment" framework.

Folding schemes. Nova[14] introduces a single-round folding protocol for R1CS relations but relies on expensive commitments for cross-term, and subsequent

[6] Our implementation is in https://github.com/skmdianrf111/hp_mercury/tree/test2

works [11,13,4,30] optimized this approach. Hypernova[12] extends folding to more general CCS relations, achieving greater flexibility at the cost of logarithmic rounds and higher communication complexity. Moreover, folding schemes have been utilized to construct efficient recursive proof systems, including IVC and PCD. Mangrove[20] is a folding-based scalable SNARK for Plonk that leverages PCD to natively support chunked arithmetic circuits.

2 Preliminary

Let λ denote the security parameter. A function $f(\lambda)$ is *negligible* (denoted by $\mathsf{negl}(\lambda)$) if $\forall c > 0$, $f(\lambda) = o(\lambda^{-c})$. For $n \in \mathbb{N}$, define $[n] := \{0, 1, \ldots, n-1\}$. Let $\mathbb{F} = \mathbb{F}_p$ be a finite field of prime order p, where $|\mathbb{F}| = \Omega(2^\lambda)$. For an integer b, $\langle b \rangle_\mu$ represents its μ-bit binary expansion. The μ-dimensional Boolean hypercube is denoted by $B_\mu := \{0,1\}^\mu$, and $\mathcal{F}_\mu^{(\leq d)}(X)$ denotes the set of μ-variate polynomials of degree at most d over $\mathbb{F}$.

Definition 1 (Interactive Argument of Knowledge [2]). *An interactive protocol $\Pi = (\mathcal{G}, \mathcal{K}, \mathcal{P}, \mathcal{V})$ is an argument of knowledge for an indexed relation $\mathcal{R}$ with knowledge error $\delta : \mathbb{N} \to [0,1]$ if the following properties hold, where given an index $\mathtt{i}$, common input $\mathtt{x}$ and prover witness $\mathtt{w}$, the deterministic indexer outputs $(\mathsf{vk}, \mathsf{pk}) \leftarrow \mathcal{K}(\mathtt{i})$ and the output of the verifier is denoted by the random variable $\langle \mathcal{P}(\mathsf{pk}, \mathtt{x}, \mathtt{w}), \mathcal{V}(\mathsf{vk}, \mathtt{x}) \rangle$:*

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

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

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

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

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

An interactive argument of knowledge protocol can be made non-interactive by the Fiat-Shamir transformation [5] if it is public coin.

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

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

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

In [10], Kothapalli and Parno introduced the notion of reduction of knowledge (abbreviated as RoK), a generalization of the argument of knowledge. A key property of RoK is its support for sequential and parallel composition, which facilitates the analysis of systems involving multiple such reductions. We provide a formal description of RoK in Appendix A.

Definition 2 (Polynomial Commitment Scheme). *A polynomial commitment scheme is a tuple of PPT algorithms* $(\mathsf{KeyGen}, \mathsf{Commit}, \mathsf{Open}, \mathsf{Verify})$ *where:*

– $\mathsf{Gen}(1^\lambda, \mathcal{F}) \to \mathsf{pp}$ *generates public parameters* pp*.*
– $\mathsf{Commit}(f, \mathsf{pp}) \to \mathsf{com}_f$ *takes a secret polynomial* $f(\boldsymbol{x})$ *and outputs a public commitment* com_f*;*
– $\mathsf{Open}(\mathsf{pp}, \mathsf{com}_f, \boldsymbol{x}) \to (z, \pi_f)$ *evaluates the polynomial* $y = f(\boldsymbol{x})$ *at a point* $\boldsymbol{x}$ *and generates a proof* π_f*;*
– $\mathsf{Verify}(\mathsf{pp}, \mathsf{com}_f, \boldsymbol{x}, z, \pi_f) \to b \in \{0, 1\}$ *is a protocol between the prover* $\mathcal{P}$ *and verifier* $\mathcal{V}$*, convincing the verifier that* $f(\boldsymbol{x}) = z$*.*

A polynomial commitment satisfies the following properties:

– ***Completeness.*** *For any polynomial* $f \in \mathcal{F}$ *and* $\boldsymbol{x} \in \mathbb{F}^\mu$*,*

$$Pr\left[\mathsf{Verify}(\mathsf{com}_f, \boldsymbol{x}, z, \pi_f, \mathsf{pp}) = 1 \;\middle|\; \begin{array}{c} pp \leftarrow \mathsf{Gen}(1^\lambda, \mathcal{F}) \\ \mathsf{com}_f \leftarrow \mathsf{Commit}(f, \mathsf{pp}) \\ (z, \pi_f) \leftarrow \mathsf{Open}(f, \boldsymbol{x}, \mathsf{pp}) \end{array}\right] = 1$$

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

$$Pr\left[\begin{array}{c} \mathsf{Verify}(\mathsf{com}_f, \boldsymbol{x}, z, \pi_f, \mathsf{pp}) = 1 \\ \wedge \mathsf{com}^* = \mathsf{Commit}(f^*, \mathsf{pp}) \\ \wedge f^*(\boldsymbol{x}^*) \neq z^* \end{array} \;\middle|\; \begin{array}{c} \mathsf{pp} \leftarrow \mathsf{KeyGen}(1^\lambda, \mathcal{F}) \\ (z^*, \boldsymbol{x}^*) \leftarrow \mathcal{P}^*(1^\lambda, \mathsf{pp}) \\ (\mathsf{com}^*, \pi^*) \leftarrow \mathcal{P}^*(1^\lambda, \mathsf{pp}) \\ f^* \leftarrow \mathcal{E}^{\mathcal{P}^*(\cdot)}(1^\lambda, \mathsf{pp}) \end{array}\right] \leq \mathsf{negl}(\lambda)$$

Definition 3 (Polynomial Interactive Oracle Proof [2]). *A public-coin polynomial interactive oracle proof (PIOP) is a public-coin interactive proof for a polynomial oracle relation* $\mathcal{R} = (\mathtt{i}; \mathtt{x}; \mathtt{w})$*, where* $\mathtt{i}$ *and* $\mathtt{x}$ *can contain oracles to* n*-variate polynomials over some field* $\mathbb{F}$*. These oracles can be queried at arbitrary points in* $\mathbb{F}^n$ *to evaluate the polynomial at these points. We denote an oracle to a polynomial* f *by* $[[f]]$*.*

The interactive oracle proof can be compiled into an interactive argument of knowledge through the PIOP compilation framework, where the oracle is replaced by a polynomial commitment. For brevity, we will use the oracle notation to describe the protocol's process, with the following correspondences:

– *When the prover* $\mathcal{P}$ *sends* $[[a]]$ *to the verifier* $\mathcal{V}$*, it represents that* $\mathcal{P}$ *computes the polynomial commitment* com_a *for the polynomial* a *and sends it to* $\mathcal{V}$*.*

- *The operation* $[[a]] + [[b]]$ *represents the product of the commitments* com_a *and* com_b *for polynomials* a *and* b*, respectively, i.e.* $\mathsf{com}_a \cdot \mathsf{com}_b$.
- *The operation* $k \cdot [[a]]$ *represents the multiplication of* k *instances of* com_a*, i.e.* com_a^k.
- *When* $\mathcal{V}$ *queries* $[[a]]$ *at a point* z*, it represents that* $\mathcal{P}$ *and* $\mathcal{V}$ *execute the* Open *and* Verify *procedures of the polynomial commitment scheme.*

Definition 4 (Relation Product). *Consider two relations* $\mathcal{R}_1$ *and* $\mathcal{R}_2$ *over public parameters, structure, instance, and witness pairs. We define the relation product* $\mathcal{R}_1 \times \mathcal{R}_2 = \{(\mathsf{pp}, \mathsf{s}, (u_1, u_2), (w_1, w_2)) | (\mathsf{pp}, \mathsf{s}, u_1, w_1) \in \mathcal{R}_1, (\mathsf{pp}, \mathsf{s}, u_2, w_2) \in \mathcal{R}_2\}$. *We denote* $\mathcal{R} \times \ldots \times \mathcal{R}$ *for* n *times as* $\mathcal{R}^n$.

Definition 5 (Folding Scheme). *A folding scheme for* $\mathcal{R}_1^m$ *and* $\mathcal{R}_2^n$ *is a RoK of type* $\mathcal{R}_1^m \times \mathcal{R}_2^n \to \mathcal{R}_1$*, and a folding scheme for* $\mathcal{R}^n$ *is a RoK of type* $\mathcal{R}^n \to \mathcal{R}$.

Definition 6 (Multilinear Extension). *For any* $f : B_\mu \to \mathbb{F}$*, there exists a unique multilinear polynomial* $\tilde{f} : \mathbb{F}^n \to \mathbb{F}$ *such that for all* $\boldsymbol{x} \in B_\mu$*,* $\tilde{f}(\boldsymbol{x}) = f(\boldsymbol{x})$. *The polynomial* $\tilde{f}$ *is called the multilinear extension(MLE) of* f*, and can be expressed as* $\tilde{f}(\boldsymbol{X}) = \sum_{\boldsymbol{x} \in B_\mu} f(\boldsymbol{x}) \cdot \mathsf{eq}(\boldsymbol{x}, \boldsymbol{X})$*, where* $\mathsf{eq}(\boldsymbol{x}, \boldsymbol{X}) := \prod_{i=1}^{\mu} (\boldsymbol{x}_i \boldsymbol{X}_i + (1 - \boldsymbol{x}_i)(1 - \boldsymbol{X}_i))$.

Definition 7 (Sum-check Relation for High-degree Polynomials[19]). *The sum-check relation for high-degree polynomials* $\mathcal{R}_{HSUM}$ *is the set of tuples*

$$(\mathsf{s}; \mathbb{x}; \mathbb{w}) = (h; (v, [[w_0]], \ldots, [[w_{t-1}]]); (w_0, \ldots, w_{t-1})).$$

where $\sum_{x \in B_\mu} h(w_0(x), \ldots, w_{t-1}(x)) = v$*. and* $h \in \mathcal{F}_t^{(\leq d)}$*,* $w_0, \ldots, w_{t-1} \in \mathcal{F}_\mu^{(\leq 1)}$.

The completeness of $\mathcal{R}_{HSUM}$ is obvious. By the Theorem 3.1 of [2], the sum-check protocol is proven to be knowledge soundness. Thus, we have the following theorem:

Theorem 1 ([2]). *The sum-check PIOP for* $\mathcal{R}_{HSUM}$ *is perfectly complete and has knowledge error* $dn/|\mathbb{F}|$.

Definition 8 (Constraint System of Hyperplonk[2]). *Fix public parameters* $\mathsf{pp} := (\mathbb{F}, \ell, n, \ell_w, \ell_q, f)$ *where* $\mathbb{F}$ *is the field,* $\ell = 2^{\nu_p}$ *is the public input length,* $n = 2^\mu$ *is the number of constraints,* $\ell_w = 2^{\nu_w}$*,* $\ell_q = 2^{\nu_q}$ *are the number of witnesses and selectors per constraint, and* $f : \mathbb{F}^{\ell_q + \ell_w} \to \mathbb{F}$ *is an algebraic map with degree* d*. The indexed relation* $\mathcal{R}$ *is the set of all tuples*

$$(\mathbb{i}; \mathbb{x}, \mathbb{w}) = ((q, \sigma); (p, [[w]]), w)$$

, where $\sigma : B_{\mu+\nu_w} \to B_{\mu+\nu_w}$ *is a permutation,* $q \in \mathcal{F}_{\mu+\nu_o}^{(\leq 1)}, p \in \mathcal{F}_{\mu+\nu_p}^{(\leq 1)}, w \in \mathcal{F}_{\mu+\nu_w}^{(\leq 1)}$*, such that the wiring identity, the gate identity is satisfied, and the public input is consistent with the witness.*

We present the formal description of Hyperplonk constraint system in the Appendix B.

3 Distributed Sumfold

3.1 Review of SumFold

We first review the SumFold protocol from [13], which folds multiple instances of $\mathcal{R}_{HSUM}$(Definition 7) into one. Let $M = 2^\nu$ be the number of instances. The core idea of SumFold is to propose a new multilinear polynomial f to "interpolate" multiple witnesses. This reduces the verification of multiple sum-check relations to checking whether a multilinear polynomial $H(X)$ is identically zero. Specifically, the verifier can choose a random point $\rho \in \mathbb{F}^\nu$ and check whether $H(\rho) = 0$. The soundness of this method follows from the Schwartz-Zippel Lemma.

The protocol reduces multiple sum-check relations to the condition $H(\rho) = 0$, where $H(X) = \sum_{b\in B_\nu} \mathsf{eq}(X, b) \cdot \left(\sum_{x\in B_\mu} h(f_0(b, x), \ldots, f_{t-1}(b, x))\right)$. We further observe that $H(\rho) = 0$ implies the sum-check relation $\sum_{b\in B_\nu} Q(b) = 0$, where $Q(b) = \mathsf{eq}(\rho, b) \cdot \left(\sum_{x\in B_\mu} h(f_0(b, x), \ldots, f_{t-1}(b, x))\right)$. After ν rounds, the sum-check relation is reduced to the polynomial evaluation $Q(r_b) = c$, which implies a new sum-check relation $\sum_{x\in B_\mu} h(w'_0(x), \ldots, w'_{t-1}(x)) = 0$, where $w'_j = \sum_{i\in[M]} \mathsf{eq}(r_b, \langle i\rangle_\nu) \cdot w_{i,j}$, for $j \in [t]$. The witnesses are committed using a homomorphic polynomial commitment scheme, and these commitments are linearly combined, ultimately producing a commitment to the folded witness. The full construction of SumFold is detailed in Protocol 3.1.

PROTOCOL 3.1 Reduction from $\mathcal{R}^M_{HSUM}$ to $\mathcal{R}_{HSUM}$(SumFold)
Input: $[\mathbb{x}_i = (v_i, [[w_{i,0}]], \ldots, [[w_{i,t-1}]]); \mathbb{w}_i = (w_{i,0}, \ldots, w_{i,t-1})]_{i\in[M]}$;
Output: $[\mathbb{x} = (v', [[w'_0]], \ldots, [[w'_{t-1}]]); \mathbb{w} = (w'_0, \ldots, w'_{t-1})]$;
$\mathcal{K}(\mathsf{pp}, \mathsf{s}_1 = h) \to (\mathsf{pk}, \mathsf{vk}, \mathsf{s}_2)$:
Output $(\mathsf{pk}, \mathsf{vk}) = (h, \perp), \mathsf{s}_2 = h$.
$\langle \mathcal{P}, \mathcal{V}\rangle((\mathsf{pk}, \mathsf{vk}), \{\mathbb{x}_i\}_{i\in[M]}; \{\mathbb{w}_i\}_{i\in[M]})$:

1. $\mathcal{V}$: Sample and send $\rho \xleftarrow{\$} \mathbb{F}^\nu$.
2. $\mathcal{P}, \mathcal{V}$: Run the sum-check protocol $\sum_{b\in B_\nu} Q(b) = T_0$, where

$$T_0 \leftarrow \sum_{i\in[M]} \mathsf{eq}(\rho, \langle i\rangle_\nu) \cdot v_i$$

$$f_j(b, x) \leftarrow \sum_{i\in[M]} \mathsf{eq}(b, \langle i\rangle_\nu) \cdot w_{i,j}(x), \text{for } j \in [t]$$

$$Q(b) \leftarrow \mathsf{eq}(\rho, b) \cdot \left(\sum_{x\in B_\mu} h(f_0(b, x), \ldots, f_{t-1}(b, x))\right).$$

At last round, the sum-check relation is reduced to a polynomial evaluation relation $Q(r_b) = c$, where $r_b \in \mathbb{F}^\nu$ is the randomness generated by the verifier during the sum-check protocol.

3. $\mathcal{P}, \mathcal{V}$: Output the folded instance-witness pair (the verifier only outputs the instance) $((h, v', [[w'_0]], \ldots, [[w'_{t-1}]]); (w'_0, \ldots, w'_{t-1}))$, where

$$v' \leftarrow c \cdot \mathsf{eq}(\rho, r_b)^{-1}$$

$$[[w'_j]] \leftarrow \sum_{i \in [M]} \mathsf{eq}(r_b, \langle i \rangle_\nu) \cdot [[w_{i,j}]], \text{for } j \in [t]$$

$$w'_j \leftarrow \sum_{i \in [M]} \mathsf{eq}(r_b, \langle i \rangle_\nu) \cdot w_{i,j}, \text{for } j \in [t]$$

By Theorem 2 of [13], Protocol 3.1 is proven to be a folding scheme for $\mathcal{R}^n_{HSUM}$, which is also a RoK from $\mathcal{R}^n_{HSUM}$ to $\mathcal{R}_{HSUM}$ (Definition 5). Thus, we have the following theorem:

Theorem 2 ([13]). *Protocol 3.1 is a RoK from $\mathcal{R}^n_{HSUM}$ to $\mathcal{R}_{HSUM}$.*

3.2 Distributed SumFold

We now discuss folding the sum-check protocol in the distributed setting. The SumFold protocol can leverage dynamic programming technique from [24] and achieve linear prover time. We further observe that SumFold prover's computations can be decomposed into M independent tasks, each requiring $O(T)$ work, where $T = 2^\mu$. Specifically, when evaluating Q at a point $b \in B_\nu$, it needs to sum the values of $h(f_0(b, x), \ldots, f_{t-1}(b, x))$ for each $x \in B_\mu$. This summation can be parallelized across provers, as each b corresponds to a distinct sub-computation.

To illustrate the above procedure, we consider the following case. Suppose h multiplies all its inputs, and $\nu = 3$(thus $M = 8$). Let $\mathcal{P}_0, \ldots, \mathcal{P}_7$ are provers where $\mathcal{P}_0$ acts as the coordinator. Each prover $\mathcal{P}_i$ stores locally the evaluation $\mathsf{eq}(\rho, \langle i \rangle_3)$ and the witness slices $f_0(\langle i \rangle_3, x), \ldots, f_{t-1}(\langle i \rangle_3, x)$ for all $x \in B_\mu$.

The SumFold protocol consists of three rounds of interaction. In the first round, $\mathcal{P}_0, \mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3$ send their local data to $\mathcal{P}_4, \mathcal{P}_5, \mathcal{P}_6, \mathcal{P}_7$ respectively, then each $\mathcal{P}_{4+j}$(for $j \in \{0, 1, 2, 3\}$) updates its data and computes a partial polynomial $Q_1^{(j)}(X)$. Then $\mathcal{P}_0$ computes $Q_1(X) = \sum_{j \in [3]} Q_1^{(j)}(X)$ and sends it to $\mathcal{V}$. After the verifier sends the first randomness r_1, $\mathcal{P}_4, \mathcal{P}_5, \mathcal{P}_6, \mathcal{P}_7$ update their data using r_1 and send it to $\mathcal{P}_0, \mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3$ respectively, thus each $\mathcal{P}_i$(for $i \in \{0, 1, 2, 3\}$)now stores $\mathsf{eq}(\rho, r_1 || \langle i \rangle_2)$ and $f_0(r_1 || \langle i \rangle_2, x), \ldots, f_{t-1}(r_1 || \langle i \rangle_2, x)$.

Similarly, in the second round, $\mathcal{P}_0, \mathcal{P}_1$ send their local data to $\mathcal{P}_2, \mathcal{P}_3$ respectively. Then $\mathcal{P}_2, \mathcal{P}_3$ compute the partial polynomials $Q_2^{(0)}(X)$ and $Q_2^{(1)}(X)$ respectively, and $\mathcal{P}_0$ aggregates them into $Q_2(X)$ and sends it to the verifier. Then $\mathcal{P}_2, \mathcal{P}_3$ update the data with r_2 and send it to $\mathcal{P}_0, \mathcal{P}_1$ respectively, thus $\mathcal{P}_i$(for $i = 0, 1$) stores $\mathsf{eq}(\rho, r_1 || r_2 || x)$ and $f_0(r_1 || r_2 || i, x), \ldots, f_{t-1}(r_1 || r_2 || i, x)$ in its memory. In the last round, $\mathcal{P}_0$ holds $\mathsf{eq}(\rho, r_b)$ and $f_0(r_b, x), \ldots, f_{t-1}(r_b, x)$, where $r_b = r_1 || r_2 || r_3$, and $\mathcal{P}_1$ computes $Q_3(X)$. The detailed construction of distributed SumFold is given in Protocol 3.2.

PROTOCOL 3.2 Distributed SumFold

$\mathcal{P}_i$ holds $(\mathbb{x}_i; \mathbb{w}_i) = ((v_i, [[w_{i,0}]], \ldots, [[w_{i,t-1}]]); (w_{i,0}, \ldots, w_{i,t-1}))$, for $i \in [M]$. Let $f_j(\langle i \rangle_\nu, x) = w_{i,j}(x)$ for $x \in B_\mu$, and $Q_0 := v$.

1. $\mathcal{V}$ samples $\rho \xleftarrow{\$} \mathbb{F}^\nu$, and sends it to $\mathcal{P}_0$.
2. $\mathcal{P}, \mathcal{V}$ run the sum-check protocol $\sum_{b \in B_\nu} Q(b) = T$ as follows, where Q and T are defined the same as Step 2 in the Protocol 3.1.
   For round $k = 1, 2, \ldots, \nu$ in the sum-check protocol:
   - for $s \in [2^{\nu-k}]$, $\mathcal{P}_s$ sends following messages to $\mathcal{P}_{2^{\nu-k}+s}$:

     $$\mathsf{eq}(\rho, \{r_1, \ldots, r_{k-1}\} || \langle s \rangle_{\nu-k+1}),$$
     $$f_j(\{r_1, \ldots, r_{k-1}\} || \langle s \rangle_{\nu-k+1}, x) \text{ for } x \in B_\mu, j \in [t]$$

     Define $e_k^{(s)}(X) := (1-X) \cdot \mathsf{eq}(\rho, \{r_1, \ldots, r_{k-1}\} || 0 || \langle s \rangle_{\nu-k}) + X \cdot \mathsf{eq}(\rho, \{r_1, \ldots, r_{k-1}\} || 1 || \langle s \rangle_{\nu-k})$. For $x \in B_\mu$, define $f_{k,x}^{(s,j)}(X) := (1-X) \cdot f_j(\{r_1, \ldots, r_{k-1}\} || 0 || \langle s \rangle_{\nu-k}, x) + X \cdot f_j(\{r_1, \ldots, r_{k-1}\} || 1 || \langle s \rangle_{\nu-k}, x)$. Then $\mathcal{P}_{2^{\nu-k}+s}$ computes $Q_k^{(s)}(X) \leftarrow e_k^{(s)}(X) \cdot \sum_{x \in B_\mu} h(f_{k,x}^{(s,0)}(X), f_{k,x}^{(s,1)}(X), \cdots, f_{k,x}^{(s,t-1)}(X))$ using Algorithm 1 (presented in Appendix C) and sends it to $\mathcal{P}_0$.
   - $\mathcal{P}_0$ computes $Q_k(X) \leftarrow \sum_{s \in [2^{\nu-k}]} Q_k^{(s)}(X)$, and sends it to $\mathcal{V}$.
   - $\mathcal{V}$ checks $Q_{k-1}(r_{k-1}) = Q_k(0) + Q_k(1)$, sends a random challenge $r_k \in \mathbb{F}$ to $\mathcal{P}_0$.
   - for $s \in [2^{\nu-k}]$, $\mathcal{P}_0$ transmits r_k to $\mathcal{P}_{2^{\nu-k}+s}$.
   - for $s \in [2^{\nu-k}]$, $\mathcal{P}_{2^{\nu-k}+s}$ computes $\mathsf{eq}(\rho, \{r_1, \ldots, r_k\} || \langle s \rangle_{\nu-k})$ and $f_j(\{r_1, \ldots, r_k\} || \langle s \rangle_{\nu-k}, x)$, and sends them to $\mathcal{P}_s$.

   Let $r_b = \{r_1, \ldots, r_\nu\}$, $\mathcal{P}_0$ gets $\mathsf{eq}(\rho, r_b)$ and $f_j(r_b, x)$, and computes $c = Q(r_b)$,. $\mathcal{P}_0$ then claims to $\mathcal{V}$ the polynomial evaluation relation $c = Q(r_b)$.
3. For $i \in [M]$, $\mathcal{P}_i$ computes $e_i \leftarrow \mathsf{eq}(r_b, \langle i \rangle_\nu)$, and then computes $([[w'_{i,j}]], w'_{i,j}) \leftarrow (e_i \cdot [[w_{i,j}]], e_i \cdot w_{i,j})$ for $j \in [t]$, and sends $[[w'_{i,j}]]$ to $\mathcal{P}_0$.
4. Set $w'_j = 0$. For $i \in [M-1]$, $\mathcal{P}_i$ updates $w'_j \leftarrow w'_j + w'_{M-i-1,j}$ and sends w'_j to $\mathcal{P}_{M-i-2}$ for $j \in [t]$.
5. $\mathcal{P}_0$ computes $v' \leftarrow c \cdot \mathsf{eq}(\rho, r_b)^{-1}$, and $[[w'_j]] \leftarrow \sum_{i \in [M]} [[w'_{i,j}]]$. $\mathcal{P}_0$ outputs instance $(v', [[w'_0]], \ldots, [[w'_{t-1}]])$ and witness $(w'_0, \ldots, w'_{t-1})$.
6. $\mathcal{V}$ computes $v' \leftarrow c \cdot \mathsf{eq}(\rho, r_b)^{-1}, e_i \leftarrow \mathsf{eq}(r_b, \langle i \rangle_\nu)$ and then $[[w'_j]] \leftarrow \sum_{i \in [M]} e_i \cdot [[w_{i,j}]]$ for $j \in [t]$. $\mathcal{V}$ outputs instance $(v', [[w'_0]], \ldots, [[w'_{t-1}]])$.

Theorem 3. *Protocol 3.2 is a distributed RoK from $\mathcal{R}_{HSUM}^M$ to $\mathcal{R}_{HSUM}$ with M provers. Each prover for $\mathcal{P}_1, \ldots, \mathcal{P}_{M-1}$ performs $O(T)$ field operations and $O(T)$ group operations, and $\mathcal{P}_0$ performs $O(M)$ group operations. The total communi-*

cation is $O(N)$. The proof size is $O(\log M)$ field elements. The verifier requires $O(\log M)$ field operations and $O(M)$-size multi-scalar multiplication. [7]

Proof. We consider the scenario where provers are collaborative and mutually trusted, so the security properties of this protocol align with those of the SumFold protocol under a single-prover setting. We now analyze the computational and communication costs in detail.

For the proving complexity, each $\mathcal{P}_i$ performs the following computations: (a) commit to the witness multilinear polynomials $w_0, \ldots, w_{t-1}$, which costs $O(T)$ group operations. (b) compute $Q_k^{(s)}(X)$, which costs $O(T)$ field operations. (c) generate the next-round message after receiving the verifier's randomness, which costs $O(T)$ field operations. (d) update w'_j when folding the witness, which costs $O(T)$ field operations. $\mathcal{P}_0$ computes $[[w'_j]]$, which performs $O(M)$ group operations.

For the communication, in the k-th round, $\mathcal{P}_s$ sends message of size $O(T)$ to $\mathcal{P}_{2^{\nu-k}+s}$. So the total size of communication is $O(\sum_{k\in[\mu]} 2^{\nu-k} \cdot T) = O(2^\nu \cdot T) = O(N)$.

For the proof size, the prover sends a univariate polynomial $Q_k(X)$, which is of $O(1)$ size, in every round. The total proof size is $O(\nu) = O(\log M)$.

The verifier's computation consists of $O(\log M)$ field operations for the sum-check verification and $O(M)$-size multi-scalar multiplication for folding the commitments from M provers. □

4 Distributed SNARK via Folding Scheme

In subsection 4.1, we clarify the reduction from both the gate identity check relation and the wire identity check relation to the sum-check relation. Subsection 4.2 demonstrates how to reduce the consistency check relations across multiple provers to the sum-check protocol. By combining these reductions with SumFold through the parallel and sequential composition techniques from [10], we construct a distributed SNARK for data-parallel circuits under the Hyperplonk constraint system. The complete protocol is presented in Section 4.3.

To construct SNARK, the oracle is instantiated by the homomorphic polynomial commitment via PIOP compilation. We employ SamaritanPCS[7], a state-of-the-art homomorphic commitment for multilinear polynomials, which achieves linear-time prover complexity and constant proof size.

4.1 The Reduction from the Gate and Wire Identity to Sum-check

In Hyperplonk[2] constraint system, both the gate identity and wire identity checks can be reduced to instances of the sum-check protocol. Specifically, the

[7] In particular, as shown in [2], the prover time scales with the degree d of the polynomial h, contributing a factor of $O(d \log^2 d)$. Similarly, the communication cost and proof size each incur a factor of $O(d)$. For simplicity, we treat d as a constant and omit further analysis regarding its impact.

gate identity corresponds to a permutation check relation (Definition 9), while the wire identity corresponds to a zerocheck relation (Definition 10).

Definition 9 (Permutation Check Relation). *The indexed relation* $\mathcal{R}_{PERM}$ *is the set of tuples* $(\mathbb{i};\mathbb{x};\mathbb{w}) = (\sigma;([[f]],[[g]]);(f,g))$. *where* $\sigma : B_u \to B_u$ *is a permutation,* $f, g \in \mathcal{F}_\mu^{(\leq d)}$,*and* $g(x) = f(\sigma(x))$ *for all* $x \in B_\mu$.

Definition 10 (Zerocheck Relation). *The relation* $\mathcal{R}_{ZERO}$ *is the set of all tuples* $(\mathbb{x};\mathbb{w}) = ([[f]];f)$ *where* $f \in \mathcal{F}_\mu^{(\leq d)}$ *and* $f(x) = 0$ *and for all* $x \in B_\mu$.

We formally characterize this process as RoK. The detailed demonstration of RoK from Hyperplonk relations to the sum-check relation appears in Appendix D. We formally state this result as the following theorem:

Theorem 4. *Protocol D.1 is a RoK from* $\mathcal{R}_{ZERO}$ *to* $\mathcal{R}_{HSUM}$.

Theorem 5. *Protocol D.2 is a RoK from* $\mathcal{R}_{PERM}$ *to* $\mathcal{R}_{HSUM}$.

Please refer to the protocol and the proof sketch in Appendix D.

4.2 The Distributed Folding of Consistency Check

The consistency check between witness and public input can be reduced to a zero-check relation, which admits a natural reduction to the sum-check protocol. Using the distributed SumFold protocol, we fold input consistency relations from multiple provers into a single sum-check relation.

Definition 11 (Consistency Check Relation). *The relation* $\mathcal{R}_{CON}$ *is the set of tuples* $(\mathbb{x};\mathbb{w}) = ((p,[[w]]);w)$ *where* $p \in \mathcal{F}_{\nu_p}^{(\leq 1)}, w \in \mathcal{F}_\mu^{(\leq 1)}$, *and* $p(X) = w(0^{\mu-\nu_p}, X)$.

The complete construction for reducing multiple input consistency checks to a sum-check relation is formally specified in Protocol 4.2.(The oracle p_i is public, so the verifier can compute its evaluation itself. h_{id} denotes identity function.)

PROTOCOL 4.2 Distributed RoK from $\mathcal{R}_{CON}^M$ to $\mathcal{R}_{HSUM}$
$\mathcal{P}_i$ holds $(\mathbb{x}_i;\mathbb{w}_i) = ((p_i,[[w_i]]);w_i)$, for $i \in [M]$.

1. Let $[[w_i']] = [[w_i(0^{\mu-\nu_p},\cdot)]] - [[p_i]]$. For $i \in [M]$, $\mathcal{P}_i$ and $\mathcal{V}$ run the protocol D.1 with structure h_{id}, input $([[w_i']];w_i')$, and output $((0,[[w_i']],[[e_i']]);w_i')$. The structure is updated to h_1.
2. $\mathcal{P}_0,\ldots,\mathcal{P}_{M-1}$ and $\mathcal{V}$ run the Protocol 3.2 with structure h_1, input $((0,[[w_i']],[[e_i']]);w_i')$ for $\mathcal{P}_i$, and then output $((0,[[w']],[[e']]);w')$.

Theorem 6. *Protocol 4.2 is a distributed RoK from* $\mathcal{R}_{CON}^M$ *to* $\mathcal{R}_{HSUM}$ *with* M *provers. Each* $\mathcal{P}_i$ *performs* $O(T)$ *field operations and* $O(T)$ *group operations. The total communication is* $O(N)$. *The proof size is* $O(\log M)$ *field opeartions. The verifier requires* $O(\log M)$ *field operations and* $O(M)$*-size multi-scalar multiplication.*

Proof. We proof the theorem as follows:
Security. Protocol 4.2 is the sequential composition of two components:

1. A RoK from $\mathcal{R}_{CON}^{M}$ to $\mathcal{R}_{HSUM}^{M}$(Step 1), which itself represents the parallel composition of M individual RoKs from $\mathcal{R}_{CON}$ to $\mathcal{R}_{HSUM}$(as specified in Protocol 3.2).
2. A RoK from $\mathcal{R}_{HSUM}^{M}$ to $\mathcal{R}_{HSUM}$(Protocol 3.2).

Efficiency. The computational and communication complexity matches that of Protocol 3.2, which dominates the overall cost. □

4.3 Putting it Together

By combining these reductions through our folding scheme, we obtain a distributed argument of knowledge, as formally specified in Protocol 4.3.

PROTOCOL 4.3 Distributed Argument of Knowledge via Folding Scheme
For $i \in [M]$,$\mathcal{P}_i$ runs the Hyperplonk indexer and holds the structure f and instance-witness pair $((p_i, [[w_i]]); w_i)$, where

$$(\{[[w_{i,j}]]\}_{j\in[\ell_w]}; \{w_{i,j}\}_{j\in[\ell_w]}) \in \mathcal{R}_{ZERO}$$
$$(([[w_i]], [[w_i]]); (w_i, w_i)) \in \mathcal{R}_{PERM}$$
$$((p_i, [[w_i]]); w_i) \in \mathcal{R}_{CON}$$

and $[[w_{i,j}]] := [[w_i(\langle j \rangle_{\nu_w}, \cdot)]]$.

1. For $i \in [M]$, $\mathcal{P}_i$ and $\mathcal{V}$ run the Protocol D.1 with structure f, input $(\{[[w_{i,j}]]\}_{j\in[\ell_w]}; \{w_{i,j}\}_{j\in[\ell_w]})$ and output $((0, \{[[w_{i,j}]]\}_{j\in[\ell_w]}, [[e_i]]); \{w_{i,j}\}_{j\in[\ell_w]})$. The structure is updated to f'.
2. For $i \in [M]$, $\mathcal{P}_i$ and $\mathcal{V}$ run the Protocol D.2 with structure h_{id}, input $(([[w_i]], [[w_i]]); w_i)$ and then output $((0, [[\hat{g}_{i,1}]], [[\hat{g}_{i,2}]], [[\hat{g}_{i,3}]], [[\hat{e}_i]]); \hat{g}_{i,1}, \hat{g}_{i,2}, \hat{g}_{i,3})$. The structure is updated to h'.
3. $\mathcal{P}_0, \ldots, \mathcal{P}_{M-1}$ and $\mathcal{V}$ run the Protocol 3.2 with structure f', input $((0, \{[[w_{i,j}]]\}_{j\in[\ell_w]}, [[e_i]]); \{w_{i,j}\}_{j\in[\ell_w]})$ for $\mathcal{P}_i$, and then output $((0, \{[[\tilde{w}_j]]\}_{j\in[\ell_w]}, [[\tilde{e}]]); \{\tilde{w}_j\}_{j\in[\ell_w]})$.
4. $\mathcal{P}_0, \ldots, \mathcal{P}_{M-1}$ and $\mathcal{V}$ run the Protocol 3.2 with structure h', input $((0, [[\hat{g}_{i,1}]], [[\hat{g}_{i,2}]], [[\hat{g}_{i,3}]], [[\hat{e}_i]]); \hat{g}_{i,1}, \hat{g}_{i,2}, \hat{g}_{i,3})$ for $\mathcal{P}_i$, and then output $((0, [[\hat{g}_1]], [[\hat{g}_2]], [[\hat{g}_3]], [[\hat{e}]]); \hat{g}_1, \hat{g}_2, \hat{g}_3)$.
5. $\mathcal{P}_0, \ldots, \mathcal{P}_{M-1}$ and $\mathcal{V}$ run the Protocol 4.2 with structure h_{id}, input $((p_i, [[w_i]]); w_i)$, and then output $((0, [[w']], [[e']]); w')$. The structure is updated to h'_c.

6. $\mathcal{P}_0$ and $\mathcal{V}$ run the sum-check protocol to check

$$(f'; (0, \{[[\tilde{w}_j]]\}_{j\in[\ell_w]}, [[\tilde{e}]]); \{\tilde{w}_j\}_{j\in[\ell_w]}) \in \mathcal{R}_{HSUM}.$$
$$(h'; (0, [[\hat{g}_1]], [[\hat{g}_2]], [[\hat{g}_3]], [[\hat{e}]]); \hat{g}_1, \hat{g}_2, \hat{g}_3) \in \mathcal{R}_{HSUM}.$$
$$((0, [[w']], [[e']]); w') \in \mathcal{R}_{HSUM}.$$

Theorem 7. *Protocol 4.3 is a succinct argument of knowledge satisfying completeness, knowledge soundness and succinctness in Definition 1 for the relation $\mathcal{C}(\mathbb{x}, \mathbb{w}) = 1$, where $\mathcal{C}$ consists of M identical copies of $\mathcal{C}_0, \ldots, \mathcal{C}_{M-1}$. Each $\mathcal{P}_i$ computes $O(T)$ field operations and $O(T)$ group operations. The total communication is $O(N)$. The proof size is $O(\log N)$ field elements plus $O(1)$ group elements. The verifier requires $O(\log M)$ field operations, $O(M)$-size multi-scalar multiplication and $O(1)$ pairing operations.*

Proof. We prove the theorem as follows:
Security. We denote the process of step i is Π_i. By Theorem 4, Π_1 is a RoK from $\mathcal{R}_{ZERO}^M$ to $\mathcal{R}_{HSUM}^M$, achieved through parallel composition of M instances of Protocol D.1(each reducing $\mathcal{R}_{ZERO}$ to $\mathcal{R}_{HSUM}$). By Theorem 5, Π_2 is a RoK from $\mathcal{R}_{PERM}^M$ to $\mathcal{R}_{HSUM}^M$, similarly constructed via parallel composition of M RoKs from $\mathcal{R}_{PERM}$ to $\mathcal{R}_{HSUM}$(Protocol D.2). By Theorem 3, both Π_3 and Π_4 are RoKs from $\mathcal{R}_{HSUM}^M$ to $\mathcal{R}_{HSUM}$. By Theorem 6, Π_5 is an RoK from $\mathcal{R}_{CON}$ to $\mathcal{R}_{HSUM}$. Thus, $(\Pi_3 \circ \Pi_1) \times (\Pi_4 \circ \Pi_2) \times \Pi_5$ is an RoK from $\mathcal{R}_{HP}^M$ to $\mathcal{R}_{HSUM}^3$, where $\mathcal{R}_{HP} = \mathcal{R}_{ZERO} \times \mathcal{R}_{PERM} \times \mathcal{R}_{CON}$. By Theorem 1, the completeness and soundness of sumcheck protocol for $\mathcal{R}_{HSUM}$(by Theorem 2) implies the completeness and soundness of Protocol 4.3.
Efficiency. For proving complexity, it requires $O(T)$ group operations and $O(T)$ field operations for $\mathcal{P}_1, \ldots, \mathcal{P}_{M-1}$ to run the SumFold protocol. For $\mathcal{P}_0$, it requires $O(T)$ field operations and $O(T)$ group operations to open the polynomial commitment.

For the communication, there is $O(N)$ communication size in the SumFold protocol.

For verifying complexity, there is $O(\log M)$ field operations plus $O(M)$-size multi-scalar multiplication in the SumFold protocol, $O(\log N)$ field operations in the sum-check protocols in Step 6, and $O(1)$ pairing operations in the polynomial commitment, thus the total verifier time is $O(\log M)$ field operations plus $O(M)$-size multi-scalar multiplication plus $O(1)$ pairing operations.

For the proof size, there is $O(\log M)$ field elements, $O(\log N)$ field elements in the sum-check protocols in Step 6, and $O(1)$ group elements in the polynomial commitment, thus the total proof size is $O(\log N)$ field elements plus $O(1)$ group elements. □

5 Implementation and Evaluation

Setting. We implement our work using ark-work[1] ecosystem based on Rust. All benchmarks were executed on a 2023 Apple MacBook Pro equipped with an

M3 Max processor (14 cores) and 36GB RAM. For fair comparison, we maintain consistent experimental settings with HyperPianist [15], including MSM implementations and vanilla Plonk gates with BN254 elliptic curves for mock circuits.

Comparison with Hyperpianist. Figure 1 compares the total proving time of our scheme against HyperPianist, both using vanilla gates. Although $\mathcal{P}_0$ operates the opening phase of the polynomial commitment—a task distinct from those performed by other sub-provers, its actual runtime is comparable to that of the other sub-provers. For brevity, we report the average values across all provers as the experimental results. Our approach is evaluated to achieve a 4.1∼4.9× speedup in each prover's time with 8 machines. The improvement is primarily because each prover only needs to commit to the multilinear polynomial without performing any commitment openings prior to folding the sum-check.

Table 2 shows the evaluated proof size and verifier time of our work on vanilla gates with 8 machines. Our verifier time is 1.4∼1.8× longer compared to HyperPianist. The overhead is likely to stem from the additional multi-scalar multiplication required for commitment folding.

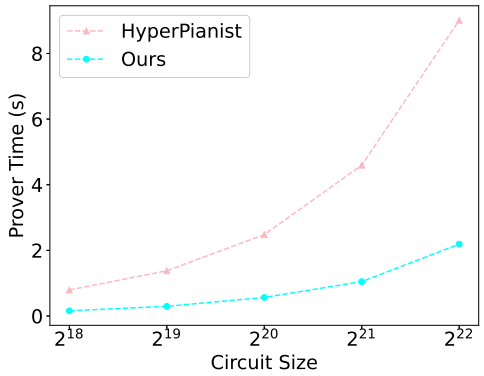

Fig. 1. Comparisons of HyperPianist and our systems on vanilla gates with 8 machines.

Table 2. Proof size and verifier time of Hyperpianist and our work on vanilla gates with 8 machines.

Scheme	Ours					HyperPianist				
Circuit Size	2^{18}	2^{19}	2^{20}	2^{21}	2^{22}	2^{18}	2^{19}	2^{20}	2^{21}	2^{22}
Proof Size (KB)	8.5	8.8	9.2	9.6	9.9	8.9	9.4	9.8	10.2	10.7
$\mathcal{V}$ Time (ms)	4.05	4.23	4.54	4.72	5.08	3.20	3.26	3.34	3.37	3.44

References

1. Arkworks zksnark ecosystem. https://arkworks.rs (2022)
2. Chen, B., Bünz, B., Boneh, D., Zhang, Z.: Hyperplonk: Plonk with linear-time prover and high-degree custom gates. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 499–530. Springer (2023)
3. Chiesa, A., Hu, Y., Maller, M., Mishra, P., Vesely, W., Ward, N.P.: Marlin: Preprocessing zksnarks with universal and updatable srs. In: Advances in Cryptology – EUROCRYPT 2020. p. 738–768. Springer (2020)
4. Dimitriou, N., Garreta, A., Manzur, I., Vlasov, I.: Mova: Nova folding without committing to error terms. Cryptology ePrint Archive (2024)
5. Fiat, A., Shamir, A.: How to prove yourself: Practical solutions to identification and signature problems. In: Conference on the theory and application of cryptographic techniques. pp. 186–194. Springer (1986)
6. Gabizon, A., Williamson, Z.J., Ciobotaru, O.: PLONK: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. Cryptology ePrint Archive (2019)
7. Ganesh, C., Patranabis, S., Singh, N.: Samaritan: Linear-time prover SNARK from new multilinear polynomial commitments. Cryptology ePrint Archive (2025)
8. Gennaro, R., Gentry, C., Parno, B., Raykova, M.: Quadratic span programs and succinct nizks without pcps. In: Advances in Cryptology–EUROCRYPT 2013. pp. 626–645. Springer (2013)
9. Groth, J.: On the size of pairing-based non-interactive arguments. In: Advances in Cryptology - EUROCRYPT 2016. Lecture Notes in Computer Science, vol. 9666, pp. 305–326. Springer (2016)
10. Kothapalli, A., Parno, B.: Algebraic reductions of knowledge. In: Annual International Cryptology Conference. pp. 669–701. Springer (2023)
11. Kothapalli, A., Setty, S.: SuperNova: Proving universal machine executions without universal circuits. Cryptology ePrint Archive (2022)
12. Kothapalli, A., Setty, S.: Hypernova: Recursive arguments for customizable constraint systems. In: Annual International Cryptology Conference. pp. 345–379. Springer (2024)
13. Kothapalli, A., Setty, S.: NeutronNova: Folding everything that reduces to zero-check. Cryptology ePrint Archive (2024)
14. Kothapalli, A., Setty, S., Tzialla, I.: Nova: Recursive zero knowledge arguments from folding schemes. In: Proceedings of the Annual International Cryptology Conference. pp. 359–388. Springer, Springer Cham (2022)
15. Li, C., Zhu, P., Li, Y., Hong, C., Qu, W., Zhang, J.: HyperPianist: Pianist with linear-time prover and logarithmic communication cost. Cryptology ePrint Archive (2024)
16. Li, W., Zhang, Z., Chow, S.S.M., Guo, Y., Gao, B., Song, X., Deng, Y., Liu, J.: Shred-to-shine metamorphosis in polynomial commitment evolution. Cryptology ePrint Archive (2025)
17. Li, W., Zhang, Z., Li, Y., Zhu, P., Hong, C., Liu, J.: Soloist: Distributed SNARKs for rank-one constraint system. Cryptology ePrint Archive (2025)
18. Liu, T., Xie, T., Zhang, J., Song, D., Zhang, Y.: Pianist: Scalable zkrollups via fully distributed zero-knowledge proofs. In: Proceedings of the 45th IEEE Symposium on Security and Privacy (S&P '24). pp. 1777–1793. IEEE (2024)

19. Lund, C., Fortnow, L., Karloff, H., Nisan, N.: Algebraic methods for interactive proof systems. Journal of the ACM (JACM) **39**(4), 859–868 (1992)
20. Nguyen, W., Datta, T., Chen, B., Tyagi, N., Boneh, D.: Mangrove: A scalable framework for folding-based snarks. In: Annual International Cryptology Conference. pp. 308–344. Springer (2024)
21. Rosenberg, M., Mopuri, T., Hafezi, H., Miers, I., Mishra, P.: Hekaton: Horizontally-scalable zkSNARKs via proof aggregation. Cryptology ePrint Archive (2024)
22. Wang, W., Shi, F., Vilardell, D., Zhang, F.: Cirrus: Performant and accountable distributed SNARK. Cryptology ePrint Archive (2024)
23. Wu, H., Zheng, W., Chiesa, A., Popa, R.A., Stoica, I.: Dizk: A distributed zero knowledge proof system. In: Proceedings of the 27th USENIX Security Symposium (USENIX Security '18). pp. 675–692 (2018)
24. Xie, T., Zhang, J., Zhang, Y., Papamanthou, C., Song, D.: Libra: Succinct zero-knowledge proofs with optimal prover computation. In: Advances in Cryptology - CRYPTO 2019. pp. 733–764. Springer (2019)
25. Xie, T., Zhang, J., Cheng, Z., Zhang, F., Zhang, Y., Jia, Y., Boneh, D., Song, D.: zkbridge: Trustless cross-chain bridges made practical. In: Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security (CCS 2022). pp. 3003–3017. ACM (2022)
26. Xu, H., Gama, M., Beni, E.H., Kang, J.: FRIttata: Distributed proof generation of FRI-based SNARKs. Cryptology ePrint Archive (2025)
27. Yu, Y., Liu, M., Zhang, Y., Sun, S.F., Ma, T., Au, M.H., Gu, D.: HyperFond: A transparent and post-quantum distributed SNARK with polylogarithmic communication. Cryptology ePrint Archive (2025)
28. Zeilberger, H., Chen, B., Fisch, B.: Basefold: Efficient field-agnostic polynomial commitment schemes from foldable codes pp. 138–169 (2024)
29. Zhang, J., Xie, T., Zhang, Y., Song, D.: Transparent polynomial delegation and its applications to zero knowledge proof. In: IEEE Symposium on Security and Privacy (SP). p. 859–876. IEEE Computer Society (2020)
30. Zhao, J., Setty, S., Cui, W.: MicroNova: Folding-based arguments with efficient (on-chain) verification. Cryptology ePrint Archive (2024)

A Reduction of Knowledge

We present the formal definition of reduction of knowledge as follows.

Definition 12 (Reduction of Knowledge[10]). *Consider relation $\mathcal{R}_1$ and $\mathcal{R}_2$ over public parameters, structure, instance and witness tuples. A reduction of knowledge from $\mathcal{R}_1$ to $\mathcal{R}_2$ is defined by PPT algorithms $(\mathcal{G}, \mathcal{P}, \mathcal{V})$ and deterministic algorithm $\mathcal{K}$, called the generator, the prover, the verifier and the encoder respectively with the following interface.*

- $\mathcal{G}(\lambda, n) \to \mathsf{pp}$: *Takes as input security parameter λ and size parameters n. Outputs public parameters* pp.
- $\mathcal{K}(\mathsf{pp}, \mathsf{s}_1) \to (\mathsf{pk}, \mathsf{vk}, \mathsf{s}_2)$: *Takes as input public parameters* pp *and the structure* s_1. *Output prover key* pk *,verifier key* vk*, and updated structure* s_2.
- $\mathcal{P}(\mathsf{pk}, u_1, w_1) \to (u_2, w_2)$: *Takes as input public parameters* pp*, and an instance-witness pair (u_1, w_1). Interactively reduces the task of checking $(\mathsf{pp}, \mathsf{s}, u_1, w_1) \in \mathcal{R}_1$ to $(\mathsf{pp}, \mathsf{s}, u_2, w_2) \in \mathcal{R}_2$.*

– $\mathcal{V}(\mathsf{vk}, u_1) \to u_2$: *Takes as input public parameters* pp*, and an instance* u_1 *in* $\mathcal{R}_1$*. Interactively reduces th task of checking instance* u_1 *to the task of checking a new instance* u_2 *in* $\mathcal{R}_2$*.*

Let $\langle \mathcal{P}, \mathcal{V} \rangle$ *denote the interaction between* $\mathcal{P}$ *and* $\mathcal{V}$*. We treat* $\langle \mathcal{P}, \mathcal{V} \rangle$ *as a function that takes as input* $((\mathsf{pk}, \mathsf{vk}), u_1, w_1)$ *and runs the interaction on the prover's input* (pk, u_1, w_1) *and the verifier's input* (vk, u_1)*. At the end of the interaction,* $\langle \mathcal{P}, \mathcal{V} \rangle$ *outputs the verifier's instance* u_2 *and the prover's witness* w_2*. A reduction of knowledge* $(\mathcal{G}, \mathcal{K}, \mathcal{P}, \mathcal{V})$ *satisfies the following conditions.*

1. ***Perfect Completeness****: For any PPT adversary* $\mathcal{A}$*, given* $\mathsf{pp} \leftarrow \mathcal{G}(1^\lambda)$ *and* $(\mathsf{s}_1, u_1, w_1) \leftarrow \mathcal{A}(\mathsf{pp})$ *such that* $(\mathsf{s}_1, \mathsf{pp}, u_1, w_1) \in \mathcal{R}_1$ *and* $(\mathsf{pk}, \mathsf{vk}, \mathsf{s}_2) \leftarrow \mathcal{K}(\mathsf{pp}, \mathsf{s}_1)$*, we have that the prover's output statement is equal to the verifier's output statement and that*

$$(\mathsf{pp}, \mathsf{s}_2, \langle \mathcal{P}, \mathcal{V} \rangle((\mathsf{pk}, \mathsf{vk}), u_1, w_1)) \in \mathcal{R}_2.$$

2. ***Knowledge Soundness****: For any expected polynomial-time adversaries* $\mathcal{A}$ *and* $\mathcal{P}^*$*, there exists an expected polynomial-time extractor* $\mathcal{E}$ *such that given* $\mathsf{pp} \leftarrow \mathcal{G}(\lambda, n)$ *and* $(u_1, w_1) \leftarrow \mathcal{A}(\mathsf{pp})$*, we have that*

$$\mathsf{Pr}[(\mathsf{pp}, \mathsf{s}_1, u_1, \mathcal{E}(\mathsf{pp}, \mathsf{s}, u_1, \mathsf{st})) \in \mathcal{R}_1] \approx \mathsf{Pr}[(\mathsf{pp}, \mathsf{s}_2, \langle \mathcal{P}^*, \mathcal{V} \rangle((\mathsf{pk}, \mathsf{vk}), u_1, \mathsf{st})) \in \mathcal{R}_2].$$

3. ***Public reducibility****: There exists a deterministic polynomial-time function* φ *such that for any PPT adversary* $\mathcal{A}$ *and expected polynomial-time adversary* $\mathcal{P}^*$*, given* $\mathsf{pp} \leftarrow \mathcal{G}(\lambda, n)$*,* $(\mathsf{s}_1, u_1, \mathsf{st}) \leftarrow \mathcal{A}(\mathsf{pp})$*,* $(\mathsf{pk}, \mathsf{vk}, \mathsf{s}_2) \leftarrow \mathcal{K}(\mathsf{pp}, \mathsf{s}_1)$ *and* $(u_2, w_2) \leftarrow \langle \mathcal{P}^*, \mathcal{V} \rangle((\mathsf{pk}, \mathsf{vk}), u_1, \mathsf{st})$ *with the interaction transcript* tr*, we have that* $\varphi(\mathsf{pp}, \mathsf{s}_1, u_1, \mathsf{tr}) = u_2$*.*

Lemma 1 (Sequential composition [10]). *For reductions* $\Pi_1 = (\mathcal{G}, \mathcal{K}_1, \mathcal{P}_1, \mathcal{V}_1) : \mathcal{R}_1 \to \mathcal{R}_2$ *and* $\Pi_2 = (\mathcal{G}, \mathcal{K}_2, \mathcal{P}_2, \mathcal{V}_2) : \mathcal{R}_2 \to \mathcal{R}_3$*, we have that* $\Pi_2 \circ \Pi_1 = (\mathcal{G}, \mathcal{K}, \mathcal{P}, \mathcal{V}) : \mathcal{R}_1 \to \mathcal{R}_3$ *where* $\mathcal{K}(\mathsf{pp}, \mathsf{s}_1)$ *computes* $(\mathsf{pk}_1, \mathsf{vk}_1, \mathsf{s}_2) \leftarrow \mathcal{K}_1(\mathsf{pp}, \mathsf{s}_1)$*,* $(\mathsf{pk}_2, \mathsf{vk}_2, \mathsf{s}_3) \leftarrow \mathcal{K}_2(\mathsf{pp}, \mathsf{s}_2)$ *and outputs* $((\mathsf{pk}_1, \mathsf{pk}_2), (\mathsf{vk}_1, \mathsf{vk}_2), \mathsf{s}_3)$ *and where*

$$\mathcal{P}((\mathsf{pk}_1, \mathsf{pk}_2), u_1, w_1) = \mathcal{P}_2(\mathsf{pk}_2, \mathcal{P}_1(\mathsf{pk}_1, u_1, w_1))$$
$$\mathcal{V}((\mathsf{vk}_1, \mathsf{vk}_2), u_1) = \mathcal{V}_2(\mathsf{vk}_2, \mathcal{V}_1(\mathsf{vk}_1, u_1, w_1))$$

Lemma 2 (Parallel composition [10]). *Consider relations* $\mathcal{R}_1$*,* $\mathcal{R}_2$*,* $\mathcal{R}_3$*, and* $\mathcal{R}_4$*. For reductions of knowledge* $\Pi_1 = (\mathcal{G}, \mathcal{K}, \mathcal{P}_1, \mathcal{V}_1) : \mathcal{R}_1 \to \mathcal{R}_2$ *and* $\Pi_2 = (\mathcal{G}, \mathcal{K}, \mathcal{P}_2, \mathcal{V}_2) : \mathcal{R}_3 \to \mathcal{R}_4$ *we have that* $\Pi_1 \times \Pi_2 = (\mathcal{G}, \mathcal{K}, \mathcal{P}, \mathcal{V}) : \mathcal{R}_1 \times \mathcal{R}_3 \to \mathcal{R}_2 \times \mathcal{R}_4$ *where*

$$\mathcal{P}(\mathsf{pk}, (u_1, u_3), (w_1, w_3)) = (\mathcal{P}_1(\mathsf{pk}, u_1, w_1), \mathcal{P}_2(\mathsf{pk}, u_3, w_3))$$
$$\mathcal{V}(\mathsf{vk}, (u_1, u_3)) = (\mathcal{V}_1(\mathsf{vk}, u_1), \mathcal{V}_2(\mathsf{vk}, u_3))$$

B Hyperplonk

The HyperPlonk relation [2] consists of three components: gate identities, wire identities, and consistency checks. These checks can be formally defined as follows:

- the wiring identity is satisfied, that is, $(\sigma; ([[w]], [[w]]); (w, w)) \in \mathcal{R}_{PERM}$ (Definition 9);
- the gate identity is satisfied, that is, $([[\tilde{f}]]; \tilde{f}) \in \mathcal{R}_{ZERO}$(Definition 10), where the virtual polynomial $\tilde{f} \in \mathcal{F}_{\mu}^{(\leq d)}$ is defined as

$$\tilde{f}(\mathbf{X}) := f(q(\langle 0 \rangle_{\nu_q}, \boldsymbol{X}), \ldots, q(\langle \ell_q - 1 \rangle_{\nu_q}, \boldsymbol{X}), w(\langle 0 \rangle_{\nu_w}, \boldsymbol{X}), \ldots, w(\langle \ell_w - 1 \rangle_{\nu_w}, \boldsymbol{X})); \tag{1}$$

- the public input is consistent with the witness, that is, the public input polynomial $p \in \mathcal{F}_{\nu_p}^{(\leq 1)}$ is identical to $w(0^{\mu+\nu_w-\nu_p}, \boldsymbol{X}) \in \mathcal{F}_{\nu_p}^{(\leq 1)}$, that is, $((p, [[w]]); w) \in \mathcal{R}_{CON}$(Definition 11).

The protocol B describes the PIOP for HyperPlonk. This PIOP can be compiled with a multilinear polynomial commitment scheme to produce a SNARK, which checks $(\mathtt{i}; \mathtt{x}, \mathtt{w}) \in \mathcal{R}$, where $(\mathtt{i}; \mathtt{x}, \mathtt{w}) = ((q, \sigma); (p, [[w]]), w)$.

PROTOCOL B PIOP for Hyperplonk.

$\mathcal{K}(\mathsf{pp}, \mathtt{i} = (q, \sigma)) \to (\mathsf{pk}, \mathsf{vk})$:

Output $\mathsf{pk} = \mathsf{vk} = ([[q]], [[s_{id}]], [[s_\sigma]])$.

$\langle \mathcal{P}, \mathcal{V} \rangle((\mathsf{pk}, \mathsf{vk}), \mathtt{x}; \mathtt{w})$:

1. $\mathcal{P}$ sends $\mathcal{V}$ the witness oracle $[[w]]$ where $w \in \mathcal{F}_{\mu+\nu_m}^{(\leq 1)}$.
2. $\mathcal{P}$ and $\mathcal{V}$ run a PIOP for the gate identity, which is $(f; \{[[q_j]]\}_{j\in[\ell_q]}, \{[[w_j]]\}_{j\in[\nu_w]}; \{[[w_j]]\}_{j\in[\nu_w]}) \in \mathcal{R}_{ZERO}$, where f is as defined in Equation 1.
3. $\mathcal{P}$ and $\mathcal{V}$ run a PIOP for the wire identity, which is $(h_{id}; ([[w]], [[w]]); (w, w)) \in \mathcal{R}_{PERM}$
4. $\mathcal{V}$ checks the consistency between witness and public input, which is $((p, [[w]]); w) \in \mathcal{R}_{CON}$

C The Algorithm Computing $Q_k^{(s)}(X)$

We present the detailed algorithm(shown in Algorithm 1) computing $Q_k^{(s)}(X)$ in Protocol 3.2.

D The RoK from Hyperplonk Relations to Sum-check

D.1 Reduction from Zerocheck to Sum-check

In the Hyperplonk framework [2], the gate identity relation naturally corresponds to a zero-check relation(denoted as $\mathcal{R}_{ZERO}$), which is reduced to sum-check relation via RoK. The complete reduction procedure is formally specified below.

Algorithm 1 Evaluating $h(X) := \Pi_{i=1}^{d} g_i(X)$

Input: $g_1(X), \ldots, g_d(X)$ are linear univariate functions.
Output: $h(X)$

```
t_{1,j} ← g_j for all j ∈ [d]
for i = 0 to log d do
    for j = 0 to d/2^i − 1 do
        t_{i+1,j}(X) ← t_{i,2j−1}(X) · t_{i,2j}(X) {Using FFT}
    end for
end for
return h(X) = t_{log d,1} =0
```

PROTOCOL D.1 Reduction from $\mathcal{R}_{ZERO}$ to $\mathcal{R}_{HSUM}$.
Input: $\mathbb{x} = (\{[[\tilde{w}_j]]\}_{j\in[t]})$; $\mathbb{w} = \{\tilde{w}_j\}_{j\in[t]}$.
Output: $\mathbb{x} = (0, \{[[\tilde{w}_j]]\}_{j\in[t]}, [[e_{\mathbf{r}}]])$; $\mathbb{w} = (\{\tilde{w}_j\}_{j\in[t]})$.
$\mathcal{K}(\mathsf{pp}, \mathsf{s}_1 = h) \to (\mathsf{pk}, \mathsf{vk}, \mathsf{s}_2)$:
Output $(\mathsf{pk}, \mathsf{vk}) = (h, \perp), \mathsf{s}_2 = h'$, where $h'(\{\tilde{w}_j\}_{j\in[t]}, g) = h(\{\tilde{w}_j\}_{j\in[t]}) \cdot g$.
$\langle \mathcal{P}, \mathcal{V} \rangle((\mathsf{pk}, \mathsf{vk}), \mathbb{x}; \mathbb{w})$:
1. $\mathcal{V}$ sends $\mathcal{P}$ a random vector $\mathbf{r} \leftarrow \mathbb{F}^{\mu}$.
2. $\mathcal{V}$ outputs $\mathbb{x} = (0, \{[[\tilde{w}_j]]\}_{j\in[t]}, [[e_{\mathbf{r}}]])$, and $\mathcal{P}$ outputs $\mathbb{w} = (\{\tilde{w}_j\}_{j\in[t]})$, where $[[e_{\mathbf{r}}]] = \mathsf{eq}(\cdot, \mathbf{r})$.

Theorem 8. *Protocol D.1 is a RoK from $\mathcal{R}_{ZERO}$ to $\mathcal{R}_{HSUM}$.*

Proof. We refer to [2] for the proof of completeness and knowledge soundness. We proof the property of public reducibility as follows.
Given the input instance $(\{[[\tilde{w}_j]]\}_{j\in[t]})$ and the transcript that includes a random vector $\mathbf{r}$, one can output $\mathbb{x} = (0, \{[[\tilde{w}_j]]\}_{j\in[t]}, [[e_{\mathbf{r}}]])$ where $[[e_{\mathbf{r}}]] = \mathsf{eq}(\cdot, \mathbf{r})$ if the verification passes and $\perp$ otherwise. □

D.2 Reduction from Permutation Check to Sum-check

Similarly, in the Hyperplonk framework [2], the wire identity relation naturally corresponds to a permutation check relation(denoted as $\mathcal{R}_{PERM}$), which is reduced to sum-check relation via RoK. The complete reduction procedure is formally specified below.

PROTOCOL D.2 Reduction from $\mathcal{R}_{PERM}$ to $\mathcal{R}_{HSUM}$.
Input: $\mathbb{x} = ([[w]], [[w]])$; $\mathbb{w} = w$.
Output: $\mathbb{x} = ((0, [[\hat{f}]], [[\hat{e}]])$; $\mathbb{w} = (\hat{f}, \hat{e})$.
$\mathcal{K}(\mathsf{pp}, \mathsf{s}_1 = h_{id}) \to (\mathsf{pk}, \mathsf{vk}, \mathsf{s}_2)$:
Output $(\mathsf{pk}, \mathsf{vk}) = (h_{id}, \perp), \mathsf{s}_2 = h_0$, where $h_0(g_1, g_2, g_3, g_4) = (g_1 \cdot g_2 +$

$g_3) \cdot g_4$.

$\langle \mathcal{P}, \mathcal{V} \rangle((\mathsf{pk}, \mathsf{vk}), \mathbb{x}; \mathbb{w})$:

1. $\mathcal{V}$ samples $\alpha, \beta \in \mathbb{F}$, and sends them to $\mathcal{P}$.
2. $\mathcal{P}$ computes f_{id}, f_σ where $f_{id} = s_{id} + \alpha \cdot w + \beta$, $f_\sigma = s_\sigma + \alpha \cdot w + \beta$, and $\mathcal{P}, \mathcal{V}$ get the oracle $[[f_{id}]]$, $[[f_\sigma]]$ using $[[s_{id}]], [[s_\sigma]], [[w]]$.
3. $\mathcal{P}$ computes $v \in \mathcal{F}_{\mu+1}^{(\leq 1)}$ and send oracle $[[v]]$ to $\mathcal{V}$, where for all $\mathbf{x} \in B_\mu$, $v(0, \mathbf{x}) = f_{id}(\mathbf{x}) / f_\sigma(\mathbf{x})$, $v(1, \mathbf{x}) = v(\mathbf{x}, 0) \cdot v(\mathbf{x}, 1)$. $\mathcal{V}$ queries $[[v]]$ at point $(1, \ldots, 1, 0) \in \mathbb{F}^{\mu+1}$, and check that the evaluation is 1.
4. $\mathcal{P}$ computes $\hat{g} \in \mathcal{F}_{\mu+1}^{(\leq 2)}$ where for all $x_0 \in B, \mathbf{x} \in B_\mu$, $\hat{g}(x_0, \mathbf{x}) = (1 - x_0) \cdot (v(1, \mathbf{x}) - v(\mathbf{x}, 0) \cdot v(\mathbf{x}, 1)) + x_0 \cdot (f_\sigma(\mathbf{x}) \cdot v(0, \mathbf{x}) - f_{id}(\mathbf{x}))$, and $\mathcal{P}, \mathcal{V}$ get the oracle $[[\hat{g}]]$ using $[[v]], [[f_{id}]], [[f_\sigma]]$.
5. $\mathcal{P}$ and $\mathcal{V}$ run the reduction from $\mathcal{R}_{ZERO}$ to $\mathcal{R}_{HSUM}$ with structure $\mathsf{s}' = h'$ and input $\mathbb{x} = (h_0, [[\hat{g}]])$ and $\mathbb{w} = \hat{g}$, where $h'(a, b, c) = a \cdot b + c$ [a]. $\mathcal{V}$ outputs $\mathbb{x} = (0, [[\hat{f}]], [[\hat{e}]])$ and $\mathbb{w} = \hat{f}$, and $\mathcal{P}$ outputs $\mathbb{w} = (\hat{f}, \hat{e})$.

[a] When we query the oracle $[[\hat{g}]]$, we choose to query one of $[[\hat{g}(0, \cdot)]]$ and $[[\hat{g}(1, \cdot)]]$, where $[[\hat{g}(0, \cdot)]] = \{[[-v(\cdot, 0)]], [[v(\cdot, 1)]], [[v(1, \cdot)]]\}$, $[[\hat{g}(1, \cdot)]] = \{[[f_\sigma(\cdot)]], [[v(0, \cdot)]], [[-f_{id}(\cdot))]]\}$.

Theorem 9. *Protocol D.2 is a RoK from $\mathcal{R}_{PERM}$ to $\mathcal{R}_{HSUM}$.*

Proof. We refer to [2] for the proof of completeness and knowledge soundness. We proof the property of public reducibility as follows.
Given the instance including the oracle $[[w]]$ and the transcript including the randomness α, β and the oracle $[[v]]$, one can obtain the output $\mathbb{x} = ((0, [[\hat{f}]], [[\hat{e}]])$ through the following steps:

1. Compute $[[f_{id}]] = [[s_{id}]] + \alpha \cdot [[w]] + \beta$, and $[[f_\sigma]] = [[s_\sigma]] + \alpha \cdot [[w]] + \beta$.
2. Compute $[[\hat{g}]]$ where $\hat{g}$ is defined as the Step 4 in the protocol D.2.
3. Compute $[[\hat{f}]]$ and $[[\hat{e}]]$ using $[[\hat{g}]]$ as the Step 5 in the protocol D.2, and output $\mathbb{x} = ((0, [[\hat{f}]], [[\hat{e}]])$. □