Soloist: Distributed SNARKs for Rank-One Constraint System

Weihan Li*
leeweihan@buaa.edu.cn
School of Cyber Science and Technology, Beihang University
Beijing, China

Zongyang Zhang†
zongyangzhang@buaa.edu.cn
School of Cyber Science and Technology, Beihang University
Beijing, China

Yun Li
liyun24@antgroup.com
Ant Group
Beijing, China

Pengfei Zhu
zpf21@mails.tsinghua.edu.cn
Tsinghua University
Beijing, China

Cheng Hong
vince.hc@antgroup.com
Ant Group
Beijing, China

Jianwei Liu
liujianwei@buaa.edu.cn
School of Cyber Science and Technology, Beihang University
Beijing, China

ABSTRACT

Distributed SNARKs enable multiple provers to collaboratively generate proofs, enhancing the efficiency and scalability of large-scale computations. The state-of-the-art distributed SNARK for Plonk, Pianist (S&P '24), achieves constant proof size, constant amortized communication complexity, and constant verifier complexity. However, when proving the Rank-One Constraint System (R1CS), a widely used intermediate representation for SNARKs, Pianist must perform the transformation from R1CS into Plonk before proving, which can introduce a start-up cost of 10× due to the expansion of the statement size. Meanwhile, existing distributed SNARKs for R1CS, *e.g.*, DIZK (USENIX Sec. '18) and Hekaton (CCS '24), fail to match the superior asymptotic complexities of Pianist.

We propose Soloist, an optimized distributed SNARK for R1CS. Soloist achieves constant proof size, constant amortized communication complexity, and constant verifier complexity, relative to the R1CS size n. Utilized with ℓ sub-provers, its prover complexity is $O(n/\ell \cdot \log(n/\ell))$. The concrete prover time is $\ell\times$ as fast as the R1CS-targeted Marlin (Eurocrypt '20). For zkRollups, Soloist can prove more transactions, with 2.5× smaller memory costs, 2.8× faster preprocessing, and 1.8× faster proving than Pianist.

Soloist leverages an improved inner product argument and a new batch bivariate polynomial commitment variant of KZG (Asiacrypt '10). To achieve constant verification, we propose a new preprocessing method with a lookup argument for unprescribed tables, which are assumed pre-committed in prior works. Notably, all these schemes are equipped with scalable distributed mechanisms.

1 INTRODUCTION

Zero-knowledge proofs [30] enable privacy-preserving verification of statements without leaking sensitive witnesses. A specific type of zero-knowledge proof, the zero-knowledge succinct non-interactive argument of knowledge (zk-SNARK) [32, 47], provides a small proof size and efficient verification sublinear to the witness size. Benefiting from the practical performance, zk-SNARKs have been deployed in many applications such as blockchain [8, 42, 60].

Prover time, one main efficiency measure of SNARKs, is currently a critical bottleneck, especially for large-scale computations. Many efforts have been made to reduce prover time asymptotically and concretely, mainly at the scheme level [17, 31, 62]. Distributed SNARKs [42, 59, 60] simultaneously employs multiple sub-provers for collaborative proof generation. Each sub-prover handles partial witness, improving both prover efficiency and scalability.

SNARKs typically target some variants of the circuit satisfiability problem, exemplified by Rank-1 constraint system (R1CS) in [9, 18, 53] and Plonk in [17, 26, 27], both widely used in practice. Plonk is more friendly to non-linear functions (although R1CS can describe them as well [56]), while R1CS can handle addition gates for free [56]. Also, there exist efficient compilers to transform various programs into R1CS, such as Circom [21]. Currently, SNARKs targeting R1CS and Plonk cannot be converted to each other without significant overhead. Some SNARKs for R1CS like Groth16 [32] do not support Plonk. Meanwhile, proving R1CS statements using a Plonk-targeted SNARK may be costly due to the statement transformation. For example, to prove zkRollup transactions with 261, 833 constraints encoded as an R1CS, Pianist [42] transforms it into a Plonk with 2, 544, 486 constraints, which incurs at least 10× overhead as the prover time is quasi-linear to the constraint.

There also exist a few distributed SNARKs for R1CS. However, these schemes, *e.g.*, DIZK [59] and Hekaton [52], are less efficient asymptotically than the Plonk-targeted Pianist (Table 1). This inefficiency would impact the performance of numerous R1CS-based applications like blockchain [8, 42], zero-knowledge virtual machine [50], verifiable fully homomorphic encryption [57], and zero-knowledge machine learning [1], where, notably, R1CS may be a more appropriate or even the only feasible choice [3, 8, 50, 57].

Such a state of affairs makes us wonder:

Can we build a more efficient distributed SNARK for R1CS?

Challenges. Compared with Plonk handling size-n vectors, R1CS is more complex for handling size-$n \times n$ sparse matrices with $O(n)$ non-zero entries. There is a line of SNARKs for R1CS adopting different approaches, but may face challenges in achieving efficient distributed proofs. For example, Ligero-based works [2, 10] arrange the witness into a matrix. The prover operates over all rows, and then columns. Assigning sub-provers with independent rows can be natural for distribution, but sub-provers require to communicate these rows for column-wise operations, leading to a linear communication complexity. SNARKs from univariate sum-check [9, 18, 20] reduce R1CS to proving the sum of some polynomial evaluated over a domain. A possibly feasible method is to split a large sum-check

*Work done during an internship at Ant Group.
†Corresponding author

instance into multiple independent ones, and then run an amortized sum-check [9] to aggregate them. However, the verifier must know all of the claimed sums for each sum-check, leading to logarithmic proof size and verification in the number of sub-provers. Indeed, Hekaton [52] follows a similar "split-then-aggregate" idea but uses aggregated SNARK, leading to such a proof size and verification. Building distributed SNARKs from multilinear sum-check [29] may be feasible following [39, 60], but their proof size and verification are logarithmic due to the multilinear sum-check.

1.1 Results and Techniques

Main contribution. We build Soloist, a Scalable and Optimized Low-overhead SNARK for rank-One constraint system via dISTribution (*cf.*, Table 1). Soloist supports general computations while deVirgo [60] is limited to data-parallel circuits. It has the same complexities as the Plonk-based Pianist [42], and is better than HyperPianist [39] and Cirrus [58]. Compared with other SNARKs for R1CS, Soloist outperforms DIZK [59] in prover and communication complexities, and beats Hekaton [52] in proof size and verifier time.

We implement our scheme. Experiments show good scalability of Soloist, *i.e.*, the prover time decreases linearly as the number of provers increases. Soloist has a prover time of 30s for an R1CS with 2^{25} constraints when using 32 sub-provers, which is 40× as fast as Marlin [18], a non-distributed SNARK for R1CS with the same complexities. When proving a zkRollup circuit of 3,072 transactions written in R1CS, our scheme features a prover time of 400s when utilizing 32 provers, which outperforms Pianist with a 1.8× speedup.

A new (distributed) IOP for R1CS. Like [17, 42, 60], we follow the "distributed polynomial interactive oracle proof (PIOP) + distributed polynomial commitment scheme (PCS)" approach to build Soloist. Leveraging the challenges above, we need to construct a new IOP for R1CS, which is expected to be distributed-friendly. We build such a (distributed) IOP by reducing R1CS into inner products and entry-wise products. Specifically, we split a large inner product into the sum of multiple independent smaller inner products, and transform a large entry-wise product into multiple independent smaller entry-wise products. The latter is then reduced to inner products via random linear combination. Sub-provers can then take partial vectors as inputs to invoke inner product proofs for large vectors, hence improving the efficiency and scalability.

Improved and distributed IPA with constant proof size. Our distributed IOP for R1CS, reduced into inner products, features a linear proof size. For proof size optimization, we need an inner product PIOP with constant proof size. Current inner product PIOPs can not be efficient in all aspects, and we propose an improved PIOP, leading to an improved inner product argument (IPA) when instantiated with the KZG PCS [35]. Table 2 compares such IPAs. Our IPA features the smallest proof size, fewest PCS commitments, and fewest FFTs in prover. It may benefit various IPA-based protocols like range proofs [13, 65], PCSs [64], and SNARKs [10, 66].

To build the PIOP, we observe that IPAs from Laurent polynomials [14, 55] require fewer FFTs as they see vectors as polynomial coefficients instead of evaluations. In contrast, IPAs from univariate sum-check [9, 18] involve fewer PCS commitments and smaller proof size due to fewer low-degree tests. We achieve the best of both worlds, and build our PIOP from univariate sum-check with coefficient-based polynomials. It stems from a newly explored relation between inner product and univariate sum-check.

We extend our IPA to the distributed setting (Table 2), achieving a constant proof size unrelated to the number of sub-provers. This helps to build a distributed SNARK for R1CS with constant proof size. Its core idea is using Lagrange polynomials to transform the univariate sum-check of the sum of multiple univariate polynomial multiplications into a two-time univariate sum-check of bivariate polynomial multiplications. Our approach for the reduction of proof size may be of interest in other SNARK applications.

Distributed preprocessing. Equipped with the distributed IPA, our distributed SNARK features a constant proof size, but with a linear verifier complexity, which stems from computation related to linear-size public statements. We reduce it to constant via preprocessing similar to [18, 53]. As we use a new PIOP from inner products instead of from univariate or multilinear sum-check in these non-distributed schemes, we need a new preprocessing scheme, and further need to equip it with a distributed mechanism.

Table 1: Distributed SNARKs for size-$n = m\ell$ statements and ℓ sub-provers, each holding a size-$O(m)$ witness

Scheme	Circuit	Prover	Comm.	Proof size	Verifier
deVirgo	Parallel	$O(m \log m)$	$O(n)$	$O(\log^2 n)$	$O(\log^2 n)$
Pianist	Plonk	$O(m \log m)$	$O(\ell)$	$O(1)$	$O(1)$
HyperPianist Cirrus	Plonk	$O(m \frac{\log \lvert\mathbb{F}\rvert}{\log m})$	$O(\ell \log n)$	$O(\log n)$	$O(\log n)$
DIZK	R1CS	$O(m \log^2 m)$	$O(n)$	$O(1)$	$O(1)$
Hekaton	R1CS	$O(m \log m)$	$O(\ell)$	$O(\log \ell)$	$O(\log \ell)$
Soloist	R1CS	$O(m \log m)$	$O(\ell)$	$O(1)$	$O(1)$

[1] Comm. denotes the communication overhead among all sub-provers.
[2] All the schemes, if for general computations, *e.g.*, R1CS or Plonk, require trusted setups and/or trusted preprocessing, otherwise the verifier time is linear due to reading the statement and performing public computation.
[3] The sub-prover time is over field $\mathbb{F}$. The super-linear terms can stem from distributed FFTs [59] - $O(m \log^2 m)$, FFTs - $O(m \log m)$, and group multi-scalar exponentiations - $O(m \frac{\log \lvert\mathbb{F}\rvert}{\log m})$ where $\log \lvert\mathbb{F}\rvert = \omega(\log m)$ [31].

Table 2: (Distributed) Inner product arguments for size-$m\ell$ vectors, all with trusted structured reference string (SRS), and $O(1)$ proof size over group $\mathbb{G}$ and field $\mathbb{F}$

Scheme	SRS size	#Coms	#FFT	Proof size
Marlin [18]	$m\ell$	5	5	5 $\lvert\mathbb{G}\rvert$, 5 $\lvert\mathbb{F}\rvert$
Dark [14]	$m\ell$	5	3	6 $\lvert\mathbb{G}\rvert$, 6 $\lvert\mathbb{F}\rvert$
SZ22 [55]	$2m\ell$	6	3	6 $\lvert\mathbb{G}\rvert$, 6 $\lvert\mathbb{F}\rvert$
Ours	$m\ell$	4	3	4 $\lvert\mathbb{G}\rvert$, 4 $\lvert\mathbb{F}\rvert$
Below is the distributed IPAs, assuming ℓ sub-provers				
Scheme	Prover	Comm.	Proof size	Verifier
Trivial	$O(m \log m)$	$O(\ell)$	$O(\ell)$	$O(\ell)$
Ours	$O(m \log m)$	$O(\ell)$	$O(1)$	$O(1)$

We do not compare IPAs featuring (poly-)log proof size [13, 15, 38, 64]

Table 3: (Distributed) batch PCSs using KZG or mKZG for opening t points on t size-$n = m\ell$ committed polynomials

Scheme	Poly.	Open	Proof size	Verifier
BDF$^+$20 [11]	Uni.	$O(n)\,\mathbb{G}$	$2\,\lvert\mathbb{G}\rvert$	$O(t)\,\mathbb{G}$, $2\,P$
HyperPlonk [17]	Mul.	$O(tn)\,\mathbb{G}$	$O(\log tn)\,\lvert\mathbb{G}\rvert$	$O(\log tn)\,P$
Ours	Biv.	$O(n)\,\mathbb{G}$	$t+4\,\lvert\mathbb{G}\rvert$	$O(t)\,\mathbb{G}$, $4\,P$
Ours-variant	Biv.	$O(n)\,\mathbb{G}$	$4\,\lvert\mathbb{G}\rvert$, $t+1\,\lvert\mathbb{F}\rvert$	$O(t)\,\mathbb{G}$, $5\,P$
Below is the distributed batch bivariate PCS, assuming ℓ provers				
Scheme	Open	Comm.	Proof size	Verifier
Pianist [42]	$O(tm)\,\mathbb{G}$	$O(\ell t)$	$2t\,\lvert\mathbb{G}\rvert$	$O(t)\,P$
Ours	$O(m)\,\mathbb{G}$	$O(\ell t)$	$t+4\,\lvert\mathbb{G}\rvert$	$O(t)\,\mathbb{G}$, $4\,P$

We only count the main costs. Pairing operations P are more expensive than group ones $\mathbb{G}$, which are more expensive than field ones $\mathbb{F}$. Also, group element size $|\mathbb{G}|$ can be 3× larger than field $|\mathbb{F}|$. For a μ-variate polynomial with degree bound d of each variable, its polynomial size is d^μ. The transformation from *Uni*variate polynomials into *Mul*tivariate or *Biv*ariate ones is not straightforward. Typically, t is a small constant when employed in SNARKs [17, 18], while n is linear to the size of statement.

We face several challenges. One of these is that the verifier needs evaluations on some polynomials determined by both the public R1CS matrices and the verifier's challenges. This is usually achieved by PCSs. Unfortunately, as the online challenges cannot be known by the indexer in the preprocessing phase, directly seeing the challenge as polynomial variables leads to a super-linear prover complexity for committing and opening the polynomials. Inspired by Marlin [18], we re-describe the matrices using *low-degree* encoded polynomials defined by the non-zero entries. We then build an equivalent relation to compute these polynomial evaluations from encoded polynomials. A subsequent challenge arises as the relation involves non-linear functions. A typical approach for non-linear functions is lookup table arguments, which prove that every element in committed vectors exists in a table. However, most existing lookup arguments [23, 25, 49, 63] assume a correctly pre-constructed table, while our table is determined by online challenges, and directly checking the table requires a linear verifier complexity. We build an efficient online table validity check relying on the table properties. Our approach may inspire other lookup arguments where the tables cannot be pre-determined.

Beyond this, we extend the univariate lookup argument [33] to support bivariate and distributed scenarios, and reduce the master prover of distributed lookup argument from linear to the R1CS size into sub-linear by table decomposition. See Section 4.3 for details.

Distributed batch bivariate KZG. Our PIOP requires opening multiple points on multiple bivariate polynomials, and directly invoking the mKZG [46] incurs a large overhead. We need a batch PCS for bivariate polynomials with constant proof size, but most previous schemes focus on univariate polynomials [11, 18, 27], or multilinear polynomials with logarithmic proof size [17]. The only exception is Boomy [37], but it does not support multiple polynomials. Also, only a special case achieves constant proof size.

We propose such a batch bivariate KZG (*cf.*, Table 3). Opening totally t points on t polynomials requires a proof size of $t+4$ group elements, in contrast to the trivial $2t$ and the logarithmic one in HyperPlonk [17]. If the Y-dimension evaluation points are the same as in Soloist, we propose a PCS variant ("ours-variant" in Table 3) reducing the proof size to 3 group elements plus $t+1$ field elements. Also, we generalize our PCS into distributed, preserving the proof size and verifier complexity like the distributed PCS in Pianist [42]. However, due to the batch method, our distributed PCS has better prover complexity, proof size, and verifier complexity when handling distinct points on multiple polynomials.

We build our PCS following the main idea in the univariate batch KZG [11], which transforms the evaluation validity of each polynomial into a quotient equation with a common multiple of the denominators, and then aggregate these equations into one by random linear combination. However, generalizations from univariate to bivariate are challenging due to the non-uniform denominators in quotient equations and the nonexistence of common quotient factors for linear combination. We resolve these two problems by padding the evaluation points into a Cartesian product and introducing an auxiliary polynomial. The auxiliary polynomial is also important for the distributed PCS, which makes it possible to split the computation held by sub-provers independently.

1.2 Related Work

Inner product arguments. IPAs can be constructed from various techniques, such as discrete-log [12, 13] and Reed-Solomon codes [64]. They can be trustless, but have at least a logarithmic proof size. There is also a line of IPAs from "PIOP + PCS". Such IPAs have modularity by modifying the underlying PCSs with distinct trade-offs. If using KZG, they can achieve a constant proof size. Appendix A recalls two IPAs from univariate sum-check [9] and Laurent polynomials [14, 55].

Distributed SNARKs. There are two kinds of distributed SNARKs. One category [19, 28, 43, 45] delegates the proof generation to multiple servers with privacy via secret sharing. As the size of shared witness are the same as the whole one, achieving proof acceleration can be challenging.

Another category, which we focus on, considers one prover utilizing multiple machines to accelerate proof generation. The original DIZK [59] distributes R1CS-targeted Groth16 [32] with distributed algorithms for basic operations like group multi-scalar exponentiations and FFTs. However, the distributed FFT leads to linear communication costs and has increased total time complexity. Subsequent works [39, 42, 52, 60] share a common idea: splitting a large statement into multiple smaller statements (required to be somewhat independent), assigning each to a sub-prover, and aggregating sub-proofs into a final proof. Most of them follow the "distributed PIOP + distributed PCS" approach. The efficiency can be different according to the underlying PIOP and PCS. DeVirgo [60] uses hash-based PCSs [64] and features a linear communication complexity. HyperPianist [39] and Cirrus [58] reduces it to logarithmic. They also eliminate the FFTs in prover via PIOPs from multilinear sum-check [17, 61]. However, the sub-prover complexity is still super-linear in field operations due to group multi-scalar exponentiations, and the proof size or verifier complexity is logarithmic. Pianist [42] uses bivariate PIOPs and PCSs to achieve constant proof size and constant amortized communication complexity, but it incurs additional overhead when applied to R1CS-based applications. Hekaton [52], using a different proof aggregation approach,

has constant amortized communication, but the proof size and verifier time are linear to the sub-prover number. Further, it requires a circuit-specific setup instead of a universal one [18, 27, 42].

2 PRELIMINARIES

We use bold lowercase letters like $\boldsymbol{a} \in \mathbb{F}^n$ for size-n vectors, and a_i is the i-th element of $\boldsymbol{a}$. For vectors $\boldsymbol{a}, \boldsymbol{b}$, $\langle \boldsymbol{a}, \boldsymbol{b} \rangle$ and $\boldsymbol{a} \circ \boldsymbol{b}$ denote the inner product and entry-wise product of $\boldsymbol{a}$ and $\boldsymbol{b}$, respectively. Capitalized letters like A represent matrices on $\mathbb{F}$, and $A[i]$ is its i-th column. Denote the set $\{1, 2, \ldots, n\}$ by $[n]$. $f \in \mathbb{F}_m[X]$ means univariate polynomials $f(X)$ with a degree bound m, and $g \in \mathbb{F}_{m,\ell}[X, Y]$ represents bivariate polynomial $g(X, Y)$ with degree bounds m over X and ℓ over Y. we sometimes use f, g for simplicity if X, Y are explicit. We use $\mathrm{FFT}(d, m)$ for an FFT for polynomial evaluations (or IFFT for polynomial interpolations) for a degree-d polynomial on a size-m multiplicative subgroup, which costs $O(m \log d)$ field operations. Given a subgroup $\mathbb{L} = (\eta^0, \ldots, \eta^{\ell-1})$, denote $L_i(Y) = \frac{\eta^{i-1}}{\ell} \cdot \frac{Y^\ell - 1}{Y - \eta^{i-1}}$ as the i-th Lagrange polynomial s.t. $L_i(\eta^{i-1}) = 1$ and for $k \in [\ell] \wedge k \neq i - 1$, $L_i(\eta^k) = 0$. All bivariate polynomials are defined using $\{L_i(Y)\}$, *e.g.*, $f(X, Y) = \sum_{i \in [\ell]} f_i(X) L_i(Y)$.

For bilinear groups with pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$, denote $x \cdot g_i$ by $[x]_i$ for $x \in \mathbb{F}$ and $\mathbb{G}_i$ generated by $g_{i \in \{1,2\}}$.

R1CS. Given public $P_a, P_b, P_c \in \mathbb{F}^{m\ell \times m\ell}$, an R1CS [9] instance $\mathrm{R1CS}(\mathbb{F}, m\ell, n, P_a, P_b, P_c; \boldsymbol{w}, \boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c})$ is satisfied if there exists vectors $\boldsymbol{w}, \boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c} \in \mathbb{F}^{m\ell}$ such that $P_a \boldsymbol{w} = \boldsymbol{a}, P_b \boldsymbol{w} = \boldsymbol{b}, P_c \boldsymbol{w} = \boldsymbol{c}, \boldsymbol{a} \circ \boldsymbol{b} = \boldsymbol{c}$. We denote its size by $n = O(m\ell)$, where $n = \max\{||P_a||, ||P_b||, ||P_c||\}$ and $||A||$ means the number of non-zero entries in matrix A.

Interactive oracle proof (IOP). IOP is a multi-round interactive proof where the verifier sends a challenge and the prover replies with an oracle in each round. The verifier can query entries on it. PIOP [18] is an IOP variant where all $\mathcal{P}$'s oracles are polynomials.

2.1 Interactive Argument

An interactive argument for an NP relation $\mathcal{R}$ is a tuple of algorithms $(\mathcal{G}, \mathcal{P}, \mathcal{V})$. $\mathcal{G}$ represents a public parameter generation algorithm. $\mathcal{P}$ and $\mathcal{V}$ represent a PPT prover and verifier, respectively. $\mathcal{P}$ tries to convince $\mathcal{V}$ the existence of $\mathtt{w}$ s.t. $(\mathtt{x}, \mathtt{w}) \in \mathcal{R}$ for a public statement $\mathtt{x}$ through multiple rounds of interaction. An interactive argument of knowledge (AoK) further allows $\mathtt{w}$ to be efficiently extractable by an extractor.

Definition 1 (Interactive argument of knowledge). *$(\mathcal{G}, \mathcal{P}, \mathcal{V})$ is an interactive argument of knowledge for $\mathcal{R}$ if it satisfies:*

- ***Completeness.*** *For every* pp *and all* $(\mathtt{x}, \mathtt{w}) \in \mathcal{R}$,

$$\Pr[\langle \mathcal{P}(\mathtt{w}), \mathcal{V} \rangle(\mathsf{pp}, \mathtt{x}) = 1] = 1.$$

- ***Knowledge soundness.*** *For any PPT $\mathcal{P}^*$, there exists an expected polynomial time extractor $\mathcal{E}^{\mathcal{P}^*}$ such that for all* pp*, the following probability is* $\mathsf{negl}(n)(\lambda)$*:*

$$\Pr[\langle \mathcal{P}^*(), \mathcal{V} \rangle(\mathsf{pp}, \mathtt{x}) = 1, (\mathtt{x}, \mathtt{w}) \notin \mathcal{R} \mid \mathtt{w} \leftarrow \mathcal{E}^{\mathcal{P}^*}(\mathsf{pp}, \mathtt{x})].$$

$\mathcal{E}^{\mathcal{P}^*}$ means $\mathcal{E}$ has access to the randomness of $\mathcal{P}^*$.

A public-coin interactive AoK can be transformed into non-interactive via the standard Fiat-Shamir transformation. A SNARK for R1CS is a non-interactive AoK with succinctness such that the proof size is $\mathsf{poly}(\lambda, \log n)$ *and the verifier complexity is* $\mathsf{poly}(\lambda, |\mathtt{x}|, \log n)$.

2.2 Polynomial Commitment Scheme

Definition 2 (PCS [31]). *A PCS for polynomial f with security parameter λ, variate μ, and variate-degree bound d is defined by a tuple of algorithms or protocols.*

- $\mathsf{pp} \leftarrow \mathsf{Gen}(1^\lambda, d, \mu)$*: takes the security parameter λ, μ, d; generates public parameter* pp.
- $C \leftarrow \mathsf{Com}(\mathsf{pp}, f)$*: takes f and outputs commitment C.*
- $b \leftarrow \mathsf{Eval}(\mathsf{pp}, C, \boldsymbol{x}, y; f)$ *is an interactive argument. Both $\mathcal{P}$ and $\mathcal{V}$ hold the commitment C, the scalar y, and the evaluation point $\boldsymbol{x}$. $\mathcal{P}$ attempts to convince $\mathcal{V}$ that there exists f with bounded size d^μ corresponding to C and $f(\boldsymbol{x}) = y$. $\mathcal{V}$ outputs $b \in \{0, 1\}$ at the end.*

In addition, a PCS requires:

- ***Completeness.*** *For any f with* $\mathsf{pp} \leftarrow \mathsf{Gen}(1^\lambda, d, \mu)$*, commitment* $C \leftarrow \mathsf{Com}(\mathsf{pp}, f)$*, and $f(\boldsymbol{x}) = y$,* $\Pr[\mathsf{Eval}(\mathsf{pp}, C, \boldsymbol{x}, y; f) = 1] = 1$.
- ***Knowledge soundness.*** Eval *is an AoK for the following NP relation* $\mathcal{R}_{\mathsf{Eval}}(\mathsf{pp})$ *given* $\mathsf{pp} \leftarrow \mathsf{Gen}(1^\lambda, d, \mu)$*:*

$$(C, \boldsymbol{x}, y; f) : \mathsf{Eval}(\mathsf{pp}, C, \boldsymbol{x}, y; f) = 1 \wedge C = \mathsf{Com}(\mathsf{pp}, f) \wedge f(\boldsymbol{x}) = y.$$

It has been proven if the PCS satisfies knowledge soundness, then the compiled interactive argument from "PIOP + PCS" also inherits this knowledge property [14, 40, 42].

2.3 Low-Degree Tests

Low-Degree Tests (LDTs) enable a prover to prove that the degree d of some polynomial f satisfies $d \leq D$ for public D, given size-D system parameters. LDTs are important in several recent zk-SNARKs [9, 16, 18, 20, 51]. Some schemes, like fast Reed-Solomon interactive oracle proofs of proximity [7], are born to be LDTs (with proximity). A general LDT can be achieved by "PIOP + PCS", and we recall the PIOP in [16]. Given oracle f, the prover sends a polynomial oracle $f'(X) = f(X) \cdot X^{D-d}$. The verifier then queries f, f' at random α, and accepts iff $f'(\alpha) = f(\alpha) \cdot \alpha^{D-d}$. The soundness error is $D/|\mathbb{F}|$ by the Schwartz-Zippel lemma.

2.4 Univariate Sum-check

Given a polynomial $f \in \mathbb{F}_d[X]$, an order-m multiplicative subgroup $\mathbb{H}$, and a claimed sum y, the univariate sum-check PIOP [9] allows to prove $\sum_{x \in \mathbb{H}} f(x) = y$. WLOG, assume $d > m$. To complete this, the prover $\mathcal{P}$ computes polynomials $g \in \mathbb{F}_{m-1}[X]$ and $h(X)$ s.t. $f(X) = X \cdot g(X) + y/m + Z_{\mathbb{H}}(X) \cdot h(X)$, where $Z_{\mathbb{H}}$ is a degree-m vanishing polynomial s.t. $Z_{\mathbb{H}}(x) = 0$ for all $x \in \mathbb{H}$. $\mathcal{P}$ sends polynomials oracles g, h to the verifier $\mathcal{V}$, who then queries f, g, h at random α to check the validity. The prover and verifier also run LDT to prove the low-degree property of g.

3 NEW PIOPS FOR INNER PRODUCTS

An inner product PIOP proves $\langle \boldsymbol{f}, \boldsymbol{s} \rangle = y$ for vectors $\boldsymbol{f}, \boldsymbol{s} \in \mathbb{F}^m$ and public $y \in \mathbb{F}$. We assume $\boldsymbol{f}$ and $\boldsymbol{s}$ are secret, while extending one to be public is natural. Appendix A recalls two typical inner product

Protocol 1 (Improved inner product PIOP). *A prover $\mathcal{P}$ has witness $\boldsymbol{f}, \boldsymbol{s} \in \mathbb{F}^m$. $\mathcal{P}$ and the verifier $\mathcal{V}$ hold $y \in \mathbb{F}$, and a multiplicative coset $\mathbb{H}$ with order m. $\mathcal{P}$ proves to $\mathcal{V}$ it holds that $\langle \boldsymbol{f}, \boldsymbol{s} \rangle = y$.*

1. *$\mathcal{P} \to \mathcal{V}$: polynomial oracles $f(X), s'(X)$ by Equation (2).*
2. *$\mathcal{V} \to \mathcal{P}$: a random $u \in \mathbb{F}/\{0\}$.*
3. *$\mathcal{P} \to \mathcal{V}$: polynomial oracles $g'(X), h(X)$ by Equation (3).*
4. *$\mathcal{V}$: queries $f(\alpha), s'(\alpha), g'(\alpha), h(\alpha)$ at random $\alpha \in \mathbb{F}$. Compute $Z_{\mathbb{H}}(\alpha)$, and accept iff Equation (3) holds when $X = \alpha$.*

PIOPs from univariate sum-check and Laurent polynomials. The former features fewer oracles and queries due to fewer LDTs, while the latter uses coefficient-based polynomials to achieve fewer FFTs.

We combine the advantages and present a new univariate sum-check for inner products using coefficient-based polynomials. Specifically, given size-m vectors $\boldsymbol{f}, \boldsymbol{s}$, we find the following result

$$\langle \boldsymbol{f}, \boldsymbol{s} \rangle = y \Leftrightarrow \sum_{x \in \mathbb{H}} f(x) s(x^{-1}) = m \cdot y, \tag{1}$$

where f, s are coefficient-based polynomials defined by coefficients $\boldsymbol{f}, \boldsymbol{s}$, respectively. We prove Equation (1) in Appendix **??**.

A direct univariate sum-check for Equation (1) fails as s is evaluated on fractions x^{-1}. To transform it into a standard polynomial, our first attempt is to rewrite Equation (1) as $\sum_{x \in \mathbb{H}} f(x) \cdot x^m \cdot s(x^{-1}) = my$. This holds because for any $x \in \mathbb{H}$, $x^m = 1$. Unfortunately, the degree of $X^m s(X^{-1})$ is m, which introduces larger-size system parameters and new LDTs for f. Instead, we use a degree-$(m-1)$ polynomial $s'(X)$ equal to $s(X^{-1})$ anywhere on $\mathbb{H}$. Given $s(X^{-1}) = s_0 + s_1 X^{-1} + \cdots + s_{m-1} X^{1-m}$, we let

$$s'(X) = X^m \cdot \sum_{i \in [m]} (s_{i-1} \cdot X^{-i+1}) - s_0 X^m + s_0. \tag{2}$$

Now Equation (1) can be rewritten as $\sum_{x \in \mathbb{H}} f(x) \cdot s'(x) = m \cdot y$, which can be converted into a standard univariate sum-check. $f(X) \cdot s'(X) = X \cdot g(X) + y + Z_{\mathbb{H}}(X) h(X)$.

We next eliminate the LDT inspired by [51, §E]. Given a challenge u, by multiplying $(X - u)$. The sum-check can be written as

$$f(X) \cdot s'(X) \cdot (X - u) = X \cdot g'(X) + (y + Z_{\mathbb{H}}(X) h(X)) \cdot (X - u), \tag{3}$$

where $\deg(g') < m$. Given a size-m parameter, the LDT is free.

We propose our inner product PIOP in Protocol 1.

Theorem 1. *Protocol 1 is an inner product PIOP. The oracle number and proof size are 4. The soundness error is $2m/|\mathbb{F}|$. The prover complexity is $6\mathit{FFT}(m, m) + O(m)$. The verifier complexity is $O(\log m)$.*

Proof. **Completeness.** Completeness holds by Equation (1) and univariate sum-check. Below we prove Equation (1).

By definition, the possible terms in $\hat{f}(X)\hat{s}(X^{-1})$ include $\{X^{-m+1}, X^{-m+2}, \ldots, X^{m-1}\}$. We then prove

$$\text{for any } i \in [-m+1, m-1]/\{0\}, \quad \sum_{x \in \mathbb{H}} x^i = 0. \tag{4}$$

As $\mathbb{H}$ is of order m, for any i, it holds $(\omega^i)^m = 1$. Hence, $(\omega^i)^m - 1 = (\omega^i - 1)((\omega^i)^{m-1} + (\omega^i)^{m-2} + \cdots + (\omega^i)^0) = 0$. For any $i \in [-m+1, m-1]/\{0\}$, as $\omega^i \neq 1$, we have $(\omega^i)^{m-1} + (\omega^i)^{m-2} + \cdots + (\omega^i)^0 = \sum_{j \in [0, m-1]} (\omega^j)^i = 0$, which exactly proves Equation (4).

Soundness. To prove soundness, we first introduce an auxiliary equation.

$$f(X) \cdot s'(X) = X \cdot g(X) + y + Z_{\mathbb{H}}(X) h(X). \tag{5}$$

Equation (1) $\Leftrightarrow$ Equation (5) is due to the fact that $\forall x \in \mathbb{H}$, $s'(x) = s(x^{-1})$ and the validity of Equation (2). Equation (1) is then equivalent to Equation (5) due to univariate sum-check. Equation (5) $\Rightarrow$ Equation (3) directly holds by setting $g'(X) = (X - u) g(X)$. Equation (5) $\Leftarrow$ Equation (3): Since all the sum terms in Equation (3), except for $Xg'(X)$, are divisible by $X - u$ and $u \neq 0$, then $(X - u)$ divides $g'(X)$. Dividing Equation (3) by $(X - u)$, we have $g'(X)/(X - u) = g(X)$ satisfies Equation (5) and is of size at most $m - 1$. Hence, the soundness error is only related to random α, and is $2m/|\mathbb{F}|$ due to Schwartz-Zippel lemma.

Complexity. The oracle number, query size, and verifier complexity hold directly. The prover requires $4\,\mathrm{FFT}(m, m)$ for obtaining size-$2m$ evaluations of f, s' and $2\,\mathrm{FFT}(m, m)$ for computing polynomial $f \cdot s'$. The prover computation other than this is $O(m)$. □

4 DISTRIBUTED PIOPS FOR R1CS

4.1 Reducing R1CS into Inner Product PIOPs

We transform an R1CS described in Section 2 into three linear constraints and one quadratic constraint:

- Witness $\boldsymbol{w}, \boldsymbol{a} \in \mathbb{F}^{m\ell}$ satisfy $P\boldsymbol{w} = \boldsymbol{a}$ for public $P \in \mathbb{F}^{m\ell \times m\ell}$.
- Witness vectors $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c} \in \mathbb{F}^{m\ell}$ satisfy $\boldsymbol{a} \circ \boldsymbol{b} - \boldsymbol{c} = \boldsymbol{0}$.

Testing linear constraints. We use the classic Freivalds' algorithm to transform the linear constraints into inner product constraints. Given a random challenge vector $\boldsymbol{r} \in \mathbb{F}^{m\ell}$, this constraint can be transformed into $(\boldsymbol{r}^\top P)\boldsymbol{w} = \boldsymbol{r}^\top \boldsymbol{a}$. For distribution, we split a size-$m\ell$ inner product into the sum of ℓ size-m inner products. Taking the left hand as an example, we sequentially split the size-$m\ell$ vector $\boldsymbol{p'} = \boldsymbol{r}^\top P$ and $\boldsymbol{w}$ into ℓ sub-vectors, leading to matrices P' and W of size $m \times \ell$ such that the i-th column (*i.e.*, $P'[i]$ and $W[i]$) corresponds to the i-th sub-vectors. The linear constraint is then reduced into $\sum_{i \in [\ell]} \langle P'[i], W[i] \rangle = \sum_{i \in [\ell]} \langle A[i], R[i] \rangle$, where $A[i]$ and $R[i]$ have similar meanings as $P'[i]$ and $W[i]$.

To prove this inner product equation, a direct approach is seeing $W[i], P'[i], A[i], R[i]$ as coefficient-based polynomials $f_{W[i]}, f_{P'[i]}, f_{A[i]}, f_{R[i]}$ respectively, and invoking Protocol 1 to prove

$$\sum_{x \in \mathbb{H}} \sum_{i \in [\ell]} f_{P'[i]}(x) f'_{W[i]}(x) = \sum_{x \in \mathbb{H}} \sum_{i \in [\ell]} f_{A[i]}(x) f_{R[i]}(x^{-1}), \tag{6}$$

where $f'_{W[i]}$ is a standard polynomial modified by $f_{W[i]}$ like modifying s into s' as in Equation (2), which involves 4 sets of polynomial oracles. We next eliminate the oracle $f_{R[i]}$ via structured challenge $\boldsymbol{r}$, *i.e.*, $(r^0, \ldots, r^{m\ell-1})$. As the coefficient of $f_{A[i]}$ is $A[i]$, for $i \in [\ell]$, we have $\langle A[i], R[i] \rangle = r^{(i-1)m} \langle A[i], \boldsymbol{r} \rangle = r^{(i-1)m} f_{A[i]}(r)$. Now by Equation (1), it suffices to prove using 3 sets of oracles:

$$\sum_{x \in \mathbb{H}} \sum_{i \in [\ell]} f_{P'[i]}(x) \cdot f'_{W[i]}(x) = m \cdot \sum_{i \in [\ell]} r^{(i-1)m} \cdot f_{A[i]}(r). \tag{7}$$

Testing quadratic constraints. Similar to testing linear constraints, we transform a size-$m\ell$ entry-wise product into ℓ size-m entry-wise products to achieve distribution. Split $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}$ into ℓ sub-vectors,

arrange them into matrices, and define $A[i], B[i], C[i]$ as the i-th sub-vector. The quadratic constraint is then transformed into $\forall i \in [\ell], A[i] \circ B[i] = C[i]$. We next reduce it into inner products via random linear combination. Given a challenge vector $\boldsymbol{s} = (s^0, \ldots, s^{m-1})$, the constraint becomes $\forall i \in [\ell]$, $\langle A[i] \circ B[i], \boldsymbol{s}\rangle = \langle C[i], \boldsymbol{s}\rangle$. To prove it, we transform equivalently $\langle A[i] \circ B[i], \boldsymbol{s}\rangle$ into $\langle A[i] \circ \boldsymbol{s}, B[i]\rangle$, and then use a random linear combination to transform the ℓ inner products into one by reusing the challenge s, *i.e.*, $\sum_{i\in[\ell]} s^{m(i-1)}\langle A[i] \circ \boldsymbol{s}, B[i]\rangle = \sum_{i\in[\ell]} s^{m(i-1)} \cdot \langle C[i], \boldsymbol{s}\rangle$. As the exponent of s in each term varies, the soundness does not affect.

Now, the length of $\boldsymbol{s}$ equals $\boldsymbol{r}$ in testing linear constraints. We can further reuse $\boldsymbol{r}$ to replace $\boldsymbol{s}$ as these challenges are generated in the same round after the verifier receiving secret polynomial oracles. Following Protocol 1, the quadratic constraint becomes

$$\sum_{x\in\mathbb{H}}\sum_{i\in[\ell]} r^{m(i-1)} f_{A[i]}(rx) f'_{B[i]}(x) = m \sum_{i\in[\ell]} r^{m(i-1)} f_{C[i]}(r). \quad (8)$$

Reducing one polynomial oracle. With these methods, we can build 4 univariate sum-check relations for R1CS. For R1CS$(\mathbb{F}, m\ell, P_a, P_b, P_c; \boldsymbol{w}, \boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c})$, $v \in \{a, b, c\}$ and $V \in \{A, B, C\}$, we prove

$$\begin{aligned}\sum_{x\in\mathbb{H}}\sum_{i\in[\ell]} f_{P'_v[i]}(x) f'_{W[i]}(x) &= m \sum_{i\in[\ell]} r^{(i-1)m} f_{V[i]}(r),\\ \sum_{x\in\mathbb{H}}\sum_{i\in[\ell]} r^{(i-1)m} f_{A[i]}(rx) f'_{B[i]}(x) &= m \sum_{i\in[\ell]} r^{(i-1)m} f_{C[i]}(r).\end{aligned} \quad (9)$$

In the above, two polynomial oracles for secret vector $\boldsymbol{b}$ are needed, *i.e.*, $f'_{B[i]}$ and $f_{B[i]}$. We eliminate the oracle $f'_{B[i]}$ by constructing a virtual $f'_{B[i]}$ from $f_{B[i]}$. According to Equation (2), for any query r, $f'_{B[i]}(r) = r^m \cdot f_{B[i]}(r^{-1}) + (1 - r^m) \cdot f_{B[i]}(0)$. Then, the verifier can obtain $f'_{B[i]}(r)$ by querying $f_{B[i]}(r^{-1})$ and $f_{B[i]}(0)$.

4.2 Distributed PIOP with Constant Proof Size

To construct a distributed PIOP for R1CS, a direct approach is to assign the i-th sub-prover $\mathcal{P}_i$ with $\{f'_{W[i]}, f_{A[i]}, f'_{B[i]}, f_{C[i]}\}_{i\in[\ell]}$ and invoke Equation (9). However, the proof size is at least $O(\ell)$ as the verifier queries $O(1)$ evaluations on each of the $O(\ell)$ polynomials.

This section reduces the proof size into constant. We first propose Lemma 1, which shows the equivalence of: 1) the sum of multiple univariate polynomial multiplications; 2) constant-number bivariate polynomial multiplications summed over a Lagrange-domain.

Lemma 1. *Given order-m and order-ℓ multiplicative cosets $\mathbb{H}$ and $\mathbb{L}$, for any univariate polynomials $f_i^{(1)}, f_i^{(2)} \in \mathbb{F}_m[X]$, we have $\sum_{y\in\mathbb{L}} f^{(1)}(X, y) f^{(2)}(X, y) = \sum_{i\in[\ell]} f_i^{(1)}(X) f_i^{(2)}(X)$, and further $\sum_{x\in\mathbb{H}}\sum_{y\in\mathbb{L}} f^{(1)}(x, y) f^{(2)}(x, y) = \sum_{x\in\mathbb{H}}\sum_{i\in[\ell]} f_i^{(1)}(x) f_i^{(2)}(x)$, where $f_j(X, Y) = \sum_{i\in[\ell]} f_i^{(j)}(X) L_i(Y)$. This can be naturally extended to cases with more polynomial multiplications or additions.*

Proof of Lemma 1. Denote the elements in $\mathbb{L}$ by $(1, \eta, \ldots, \eta^{\ell-1})$. By the property of Lagrange polynomials, we have $\sum_{y\in\mathbb{L}} f^{(1)}(X, y) = \sum_{i\in[\ell]} f_i^{(j)}(X)$. This is because for any $i, k \in [\ell]$ and any $k \neq i$, only $L_i(\eta^{i-1}) = 1$ and $L_k(\eta^{i-1}) = 0$.

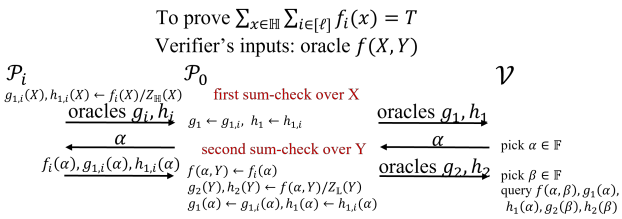

Figure 1: Visualization of the distributed sum-check

Further, for $\sum_{y\in\mathbb{L}} f^{(1)}(X, y) f^{(2)}(X, y)$ and $i, j \in [\ell]$ s.t. $i \neq j$, the cross term $L_i(y)L_j(y) = 0$ for all $y \in \mathbb{L}$. Then, we have

$$\begin{aligned}\sum_{y\in\mathbb{L}} f^{(1)}(X, y) f^{(2)}(X, y) &= \sum_{y\in\mathbb{L}}\sum_{i\in[\ell]} f_i^{(1)}(X) f_i^{(2)}(X) L_i^2(y)\\ &= \sum_{i\in[\ell]} f_i^{(1)}(X) f_i^{(2)}(X).\end{aligned}$$

The first equation in Lemma 1 hence follows.

Summing this equation over $\mathbb{H}$, we have

$$\sum_{x\in\mathbb{H}}\sum_{y\in\mathbb{L}} f^{(1)}(x, y) f^{(2)}(x, y) = \sum_{x\in\mathbb{H}}\sum_{i\in[\ell]} f_i^{(1)}(x) f_i^{(2)}(x).$$

The above proof naturally holds for more polynomials, including both more polynomial multiplications and more polynomial additions. □

We start by introducing auxiliary polynomials $R_i(X) = X^{i-1}$ and $R(X, Y) = \sum_{i\in[\ell]} R_i(X)L_i(Y)$ to describe relations in Equation (9) uniformly. Assume sub-prover $\mathcal{P}_i$ holds $f_i^{(1)}, f_i^{(2)}, f_i^{(3)}$. By moving the right hand into the left, Equation (9) can be described as

$$\sum_{x\in\mathbb{H}}\sum_{i\in[\ell]} f_i^{(1)}(x) f_i^{(2)}(x) R_i(z_1) - f_i^{(3)}(r) R_i(z_2) = 0 \quad (10)$$

In Equation (7), $R_i(z_1)$ is not needed, $r = r$ or r^{-1}, and $z_2 = r^m$. In Equation (8), $f_i^{(1)}(x) = f_{A[i]}(rx)$, and $z_1 = z_2 = r^m$.

Define $f_i(X) = f_i^{(1)}(X) f_i^{(2)}(X) R_i(z_1) - f_i^{(3)}(r) R_i(z_2)$ as the i-th summed polynomial in Equation (10). For $j \in [3]$, define polynomials $f^{(j)}(X, Y) = \sum_{i\in[\ell]} f_i^{(j)}(X) L_i(Y)$. Define bivariate polynomial $f(X, Y) = f^{(1)}(X, Y) f^{(2)}(X, Y) R(z_1, Y) - f^{(3)}(r, Y) R(z_2, Y)$.

Two times of sum-check. By the univariate sum-check over Equation (10), there exist $g_1 \in \mathbb{F}_{m-1}[X], h_1(X)$ s.t. $\sum_{i\in[\ell]} f_i(X) = X \cdot g_1(X) + Z_{\mathbb{H}}(X) h_1(X)$. Given a challenge α, it can be checked by $\sum_{i\in[\ell]} f_i(\alpha) = \alpha \cdot g_1(\alpha) + Z_{\mathbb{H}}(\alpha) h_1(\alpha)$. This is the first univariate sum-check, corresponding to Step 2. in Protocol 3.

Let $T_2 = \alpha g_1(\alpha) + Z_{\mathbb{H}}(\alpha) h_1(\alpha)$. By Lemma 1 we have $\sum_{i\in[\ell]} f_i(\alpha) = \sum_{y\in\mathbb{L}} f(\alpha, y) = T_2$. As α has been fixed, $f(\alpha, Y)$ is a degree-$(\ell-1)$ univariate polynomial over Y, and can be invoked in the univariate sum-check again. Hence, there exist $g_2 \in \mathbb{F}_{\ell-1}[Y], h_2(Y)$ s.t.

$$f(\alpha, Y) = Y \cdot g_2(Y) + T_2/\ell + Z_{\mathbb{L}}(Y) h_2(Y). \quad (11)$$

This is the second univariate sum-check, as in Step 4. of Protocol 3.

Scalability and constant proof size. The above procedure is scalable with constant proof size. Assume the i-th prover $\mathcal{P}_i$ holds $f_i(X)$.

After receiving the challenge r, $\mathcal{P}_i$ can build a distributed sum-check by dividing $Z_{\mathbb{H}}(X)$ s.t. $f_i(X) = X \cdot g_{1,i}(X) + T_{1,i} + Z_{\mathbb{H}}(X) h_{1,i}(X)$. $\mathcal{P}_i$ then sends oracles $g_{1,i}, h_{1,i}$ to $\mathcal{P}_0$. Next, $\mathcal{P}_0$ can compute the aggregated oracles $g_1 = \sum_{i\in[\ell]} g_{1,i}$ and $h_1 = \sum_{i\in[\ell]} h_{1,i}$.[1] Completeness holds by the uniqueness of polynomial division.

Given the challenge α, as $\mathcal{P}_i$ holds $f_i(X)$, she can compute $g_{1,i}(\alpha), h_{1,i}(\alpha), f_i(\alpha)$ locally. $\mathcal{P}_i$ sends these evaluations to $\mathcal{P}_0$. Next, $\mathcal{P}_0$ computes T_2 and $f(\alpha, Y) = \sum_{i\in[\ell]} f_i(\alpha) L_i(Y)$. Now, $\mathcal{P}_0$ can compute $g_2(Y), h_2(Y)$, and run the sum-check in Equation (11) locally.

In the query phase, the verifier $\mathcal{V}$ queries $g_1(\alpha), h_1(\alpha)$ for the first-time sum-check, and queries $f(\alpha, \beta), g_2(\beta), h_2(\beta)$ for the second-time sum-check. The proof size is hence $O(1)$, non-related to ℓ.

We visualize the above procedure in Figure 1. We find that each sub-prover only handles degree-$O(m)$ polynomials, and $\mathcal{P}_0$ only handles degree-$O(\ell)$ polynomials. The scalability hence follows.

Decomposing large-degree polynomials. We discuss the degrees of h_2 in Equation (11). For linear constraints in Equation (7), $R(z_1, Y)$ is not needed, so $\deg(h_2) \le \ell - 2$. However, for quadratic constraints in Equation (8), $R(z_1, Y)$ is needed and $\deg(h_2) \ge 2\ell - 3$. If using size-2ℓ system parameters, additional LDTs for secret polynomials like $f(\alpha, y)$ would be required. We decompose h_2 uniquely into $h_{2,\mathsf{low}}, h_{2,\mathsf{high}} \in \mathbb{F}_\ell[Y]$ s.t. $h_2(Y) = h_{2,\mathsf{low}}(Y) + Y^\ell \cdot h_{2,\mathsf{high}}(Y)$. Now we can still use size-ℓ system parameters.

The formal protocol. We propose the formal distributed PIOP for R1CS in two protocols. Given the original R1CS sub-witness vectors $W[i], A[i], B[i], C[i]$ of $\mathcal{P}_i$, Protocol 2 describes how to generate corresponding sub-polynomials $f'_{W[i]}, f_{A[i]}, f'_{B[i]}, f_{C[i]}$ locally and how to generate collaboratively the bivariate polynomial oracles $f'_W(X,Y), f_A(X,Y), f'_B(X,Y), f_C(X,Y)$. Protocol 3 proves R1CS, assuming each sub-prover is assigned these secret sub-polynomials.

R1CS involves 4 sum-check relations as in Equation (9), where The first three relations are similar. For simplicity, Protocol 3 discusses two relations: a general linear constraint $P_v \boldsymbol{w} = \boldsymbol{v}$ (v can belong to $\{a, b, c\}$) and a quadratic constraint $\boldsymbol{a} \circ \boldsymbol{b} = \boldsymbol{c}$. Introducing more relations is natural. For multiple sum-checks (2 in Protocol 3) and the formal R1CS with more linear constraints, we follow Aurora [9] to combine them into one sum-check using random linear combination (s in Protocol 3).

Theorem 2. *Protocol 3 is a distributed PIOP. For size-$m\ell$ R1CS, the sub-prover complexity is $O(m \log m)$, the master prover complexity is $O(\ell \log \ell)$, the proof size and amortized communication complexity are $O(1)$, and the verifier complexity is $O(m\ell)$.*

Proof of Theorem 2. **Completeness.** The completeness of the transformation from the R1CS instance into Equation (9) holds following that of linear constraint tests and quadratic constraint tests in Section 4.1. Starting from Equation (9), completeness holds due to the amortized univariate sum-check [9] to prove multiple sum-check relations at one time, and the technique to reduce LDT number in Basilisk [51].

[1] We assume the additive-homomorphic oracle size is $O(1)$. Also, assume an oracle can be obtained from the sum of ℓ oracles in $O(\ell)$ time. Looking ahead, we use univariate and bivariate KZG [35, 46] with these properties to instantiate polynomial oracles.

Protocol 2 (Distributed secret polynomial oracle generation)**.** *Suppose an R1CS instance $(\mathbb{F}, m\ell, P_a, P_b, P_c; \boldsymbol{w}, \boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c})$. Split secret vectors $\boldsymbol{w}, \boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}$ into ℓ sub-vectors, and denote the i-th sub-vector as $W[i], A[i], B[i], C[i]$. Suppose sub-provers $\mathcal{P}_1, \ldots, \mathcal{P}_\ell$ and master prover $\mathcal{P}_0$. For $i \in [\ell]$, $\mathcal{P}_i$ is assigned with $W[i], A[i], B[i], C[i]$.*

Provers distributedly generate oracles of secret bivariate polynomials $f'_W, f_A, f'_B, f_C \in \mathbb{F}_{m,\ell}[X, Y]$. For $i \in [\ell]$ and $U \in \{W, A, B, C\}$:

1. *$\mathcal{P}_i$ computes coefficient-based $f_{U[i]}$ or $f'_{U[i]} \in \mathbb{F}_m[X]$ from $U[i]$, where $f_{U[i]}(X) = \sum_{j\in[m]} U[i]_j \cdot X^{j-1}$, and $f'_{U[i]}(X) = \sum_{j\in[m]} X^m (U[i]_j \cdot X^{1-j}) - U[i]_1 \cdot X^m + U[i]_1$ like Equation (2).*
2. *$\mathcal{P}_i$ sends the oracle $f_{U[i]}(X)$ to $\mathcal{P}_0$. $\mathcal{P}_0$ computes the oracle $f_U(X, Y) = \sum_{i\in[\ell]} f_{U[i]}(X) L_i(Y)$. $f'_U(X, Y)$ is similar.*

Soundness. Checking $P_v \boldsymbol{w} = \boldsymbol{v}$ by introducing a random challenge $\boldsymbol{r}$ holds a soundness error of $(m\ell)/|\mathbb{F}|$. Checking $\boldsymbol{a} \circ \boldsymbol{b} = \boldsymbol{c}$ by checking $\sum_{i=1}^{\ell} r^{m(i-1)} \langle A[i] \circ \boldsymbol{s}, B[i] \rangle = \sum_{i=1}^{\ell} r^{m(i-1)} \langle C[i], \boldsymbol{s} \rangle$ for $i \in [\ell]$ has a soundness error bounded by $(m\ell)/|\mathbb{F}|$. Using s to combine multiple sum-check relations has a soundness error of $1/|\mathbb{F}|$. Checking the two equations over $X = \alpha$, $Y = \beta$ has a soundness error of $2\ell/|\mathbb{F}|, 2m/|\mathbb{F}|, 3\ell/|\mathbb{F}|$, respectively. By the union bound argument, the total soundness error is $(2m + 2\ell + 2m\ell)/|\mathbb{F}|$.

Complexity. We present the complexity analysis below.

Sub-prover. The time complexity of $\mathcal{P}_i$ is $O(m)$ for running Protocol 2 to compute $\hat{f}_{U[i]}(X)$ or $\hat{f}'_{U[i]}(X)$. Assume each m column of the (sparse) P_v has $O(m)$ non-zero entries.[2] Then, $\mathcal{P}_i$ spends $O(m)$ time to compute the $O(m)$ non-zero entries on $\boldsymbol{r}^\top P_v$ and $\hat{f}_{P'_v[i]}(X)$. Computing $g_{1,i}, h_{1,i}$ costs $O(m \log m)$ time. Computing and sending $O(1)$ evaluations on degree-m univariate polynomials costs $O(m)$ time. Hence, the total sub-prover complexity is $O(m \log m)$.

Master prover. The prover complexity of $\mathcal{P}_0$ is $O(\ell)$ for computing oracles f_U (or f'_U) and g_2, h_2. Given evaluations in Step 4.a, $\mathcal{P}_0$ requires $O(\ell)$ time to obtain size-ℓ polynomial evaluations in Step 4.b. Then, $\mathcal{P}_0$ interpolates these polynomials via IFFT to obtain the polynomials $\{f_j(\alpha, Y)\}$. Hence, computing g_2, h_2 costs $O(\ell \log \ell)$ time. In addition, computing oracles g_1, h_1 from $\{g_{1,i}, h_{1,i}\}$ costs $O(\ell)$ time. Therefore, the total master prover complexity is $O(\ell \log \ell)$.

Proof size. The involved polynomial oracles include $f'_W(X,Y), f_A(X,Y), f'_B(X,Y), f_C(X,Y), g_1, h_1, g_2, h_{2,\mathsf{low}}, h_{2,\mathsf{high}}$, costing $O(1)$. The proof size comes from values in Step 5., costing $O(1)$.

Communication complexity. The communication between $\mathcal{P}_i$ and $\mathcal{P}_0$ includes: 1) oracles $f'_W(X,Y), f_A(X,Y), f'_B(X,Y), f_C(X,Y)$; 2) oracles $g_{1,i}(X), h_{1,i}(X)$; 3) $O(1)$ evaluations on $\hat{f}_{P'_v[i]}, \hat{f}'_{W[i]}, \hat{f}_{V[i]}, \hat{f}_{A[i]}(X), \hat{f}_{B[i]}(X), \hat{f}_{C[i]}(X)$, and $R_i(X)$. Assuming the oracle size is $O(1)$, the amortized communication complexity is $O(1)$.

[2] This is feasible as, taking P_a as an example, the non-zero entry number of each column describes the times that one specific wire acts as gate left inputs, which is constant typically. If not, we can introduce more constraints and variables to achieve this. Further, computing the inner product over finite field is concretely faster compared with other operations such as FFTs and multi-scalar exponentiations.

Protocol 3 (Distributed PIOP for R1CS). *Secret inputs:* $\mathcal{P}_i$ *holds* $f'_{W[i]}, f_{V[i]}, f_{A[i]}, f'_{B[i]}, f_{C[i]} \in \mathbb{F}_m[X]$ *after running Protocol 2. Public inputs: oracles of* $f'_W, f_V, f_A, f'_B, f_C \in \mathbb{F}_{m,\ell}[X,Y]$. *Statement: The provers proves to* $\mathcal{V}$ *that: 1) given a public matrix* $P_v \in \mathbb{F}^{m\ell\times m\ell}$, *there exists secret vectors* $\boldsymbol{w}, \boldsymbol{v} \in \mathbb{F}^{m\ell}$ *s.t.* $P_v\boldsymbol{w} = \boldsymbol{v}$; *2) there exists secret vectors* $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c} \in \mathbb{F}^{m\ell}$ *s.t.* $\boldsymbol{a} \circ \boldsymbol{b} = \boldsymbol{c}$.

1. (a) $\mathcal{V}$: *send challenges* $r, s, u_1 \in \mathbb{F}$ *to* $\mathcal{P}_0$, *who transfers to* $\mathcal{P}_i$.
   (b) $\mathcal{V}$: *computes* $\boldsymbol{r} = (r^0, \ldots, r^{m\ell-1})$ *and* $\boldsymbol{r}^\top P_v = \boldsymbol{v}$. *Arrange* $\boldsymbol{v}$ *into matrix* $P'_v \in \mathbb{F}^{m\times\ell}$ *s.t.* $P'_v[i] = (v_{(i-1)m+1}, \ldots, v_{im})$. *Compute* $f_{P'_v}(X,Y) = \sum_{i\in[\ell]} f_{P'_v[i]}(X)L_i(Y)$.
2. *Provers compute and send polynomial oracles* $g_1, h_1 \in \mathbb{F}_m[X]$ *of the first-round univariate sum-check to* $\mathcal{V}$.
   (a) $\mathcal{P}_i$: *computes public* $f_{P'_v[i]} \in \mathbb{F}_m[X]$ *from* $P_v[i]$.
   (b) $\mathcal{P}_i$: *computes polynomials* $f_{1,i}, f_{2,i}, f_i \in \mathbb{F}_m[X]$ *s.t.*
   $$f_{1,i}(X) = f_{P'_v[i]}(X)f'_{W[i]}(X) - R_i(r^m)f_{V[i]}(r),$$
   $$f_{2,i}(X) = R_i(r^m)f_{A[i]}(rX)f'_{B[i]}(X) - R_i(r^m)f_{C[i]}(r).$$
   Let $f_i(X) = f_{1,i}(X) + s \cdot f_{2,i}(X)$.
   (c) $\mathcal{P}_i$: *computes* $g_{1,i}(X), h_{1,i}(X)$ *by dividing* $Z_{\mathbb{H}}(X)$ *s.t.*
   $$(X-u_1)f_i(X) = X\cdot g_{1,i}(X) + (X-u_1)T_{1,i} + (X-u_1)Z_{\mathbb{H}}(X)h_{1,i}(X).$$
   $\mathcal{P}_i$ *sends polynomial oracles* $g_{1,i}, h_{1,i}$ *to* $\mathcal{P}_0$.
   (d) $\mathcal{P}_0$: *computes oracles* $g_1 = \sum_{i\in[\ell]} g_{1,i}$ *and* $h_1 = \sum_{i\in[\ell]} h_{1,i}$.
3. $\mathcal{V}$ *sends challenges* $\alpha, u_2 \in \mathbb{F}$ *to* $\mathcal{P}_0$, *who transfers to* $\mathcal{P}_i$.
4. *Provers compute and send polynomial oracles* $g_2, h_{2,\mathsf{low}}, h_{2,\mathsf{high}} \in \mathbb{F}_\ell[Y]$ *of the second-round univariate sum-check to* $\mathcal{V}$.
   (a) $\mathcal{P}_i$: *computes and sends* $f_{P'_v[i]}(\alpha), f'_{W[i]}(\alpha), f_{V[i]}(r), f_{A[i]}(r\alpha), f'_{B[i]}(\alpha), f_{C[i]}(r), R_i(r^m)$ *to* $\mathcal{P}_0$.
   (b) $\mathcal{P}_0$: *computes* $f_{P'_v}(\alpha,Y), f'_W(\alpha,Y), f_V(r,Y), f_A(r\alpha,Y), f'_B(\alpha,Y), f_C(r,Y), R(r^m,Y)$. *Let* $f(\alpha,Y) = f_1(\alpha,Y) + sf_2(\alpha,Y)$ *s.t.*
   $$f_1(\alpha,Y) = f_{P'_v}(\alpha,Y)f'_W(\alpha,Y) - R(r^m,Y)f_V(r,Y),$$
   $$f_2(\alpha,Y) = R(r^m,Y)\cdot(f_A(r\alpha,Y)f'_B(\alpha,Y) - f_C(r,Y)).$$
   (c) $\mathcal{P}_0$: *computes* $g_2, h_{2,\mathsf{low}}, h_{2,\mathsf{high}} \in \mathbb{F}_\ell[Y]$ *by dividing* $Z_{\mathbb{L}}$ *s.t.*
   $$f(\alpha,Y) = Y/(Y-u_2)\cdot g_2(Y) + T_2/\ell + Z_{\mathbb{L}}(Y)(h_{2,\mathsf{low}} + Y^\ell \cdot h_{2,\mathsf{high}}), \quad (12)$$
   where $T_2 = \alpha g_1(\alpha) + (\alpha - u_1)Z_{\mathbb{H}}(\alpha)h_1(\alpha)$.
5. (a) $\mathcal{V}$: *queries* $g_1(\alpha), h_1(\alpha)$ *to compute* T_2. *Pick a random* $\beta \in \mathbb{F}$.
   (b) $\mathcal{V}$: *computes* $f_{P'_v}(\alpha,\beta), R(r^m,\beta)$. *Query* $f'_W(\alpha,\beta), f_V(r,\beta), f_A(r\alpha,\beta), f'_B(\alpha,\beta), f_C(r,\beta)$. *Query* $g_2, h_{2,\mathsf{low}}, h_{2,\mathsf{high}}$ *at* β.
   (c) $\mathcal{V}$: *computes* $f(\alpha,\beta)$. *Accept iff Equation* (12) *holds at* $Y = \beta$.

Verifier complexity. The verifier complexity is $O(m\ell)$ due to computing $f_{P'_v}(\alpha,\beta)$ and $R(r^m,\beta)$. The verifier computation other than this is $O(\log m + \log\ell)$ for computing $Z_{\mathbb{H}}(\alpha), Z_{\mathbb{L}}(\beta)$, respectively. □

4.3 Distributed PIOP with Sublinear Verifier

Protocol 3 has a linear verifier complexity due to evaluating linear-size public polynomials $f_{P'_v}(X,Y)$ related to public matrix P_v. This section reduces it to sublinear via preprocessing.

In the preprocessing phase, an indexer $\mathcal{I}$ encodes the public matrices into oracles, and the verifier $\mathcal{V}$ takes these oracles as inputs of the online phase. $\mathcal{P}$ then proves to $\mathcal{V}$ the evaluation validity of $f_{P'_v}(\alpha,\beta)$. Recall that $P'_v = \boldsymbol{r}P_v$, and hence $f_{P'_v}(X,Y)$ is determined by both the public matrix P_v and the challenge $\boldsymbol{r}$. However, $\mathcal{I}$ can not know the online challenge $\boldsymbol{r}$ ahead. A direct approach is to introduce an additional variant Z to describe $\boldsymbol{r}$ in $f_{P'_v[i]}(X)$ to construct $f_{P'_v[i]}(X,Z)$. However, the polynomial size would be m^2, incurring a $O(m^2)$ sub-prover complexity.

We let $\mathcal{I}$ encode the public *sparse* matrices by low-degree univariate polynomials row, col, val, representing the row indexes, column indexes, and values of the *non-zero* entries in the matrices. Now the encoding polynomials are only of sizes $O(m\ell)$, linear to the R1CS size. Below we show how to construct $f_{P'_v}(\alpha,\beta)$ relying on the encoding polynomials and to prove its validity distributedly.

Algebraic preliminaries. For a sparse matrix $P_v \in \mathbb{F}^{m\ell\times m\ell}$, split it into ℓ sub-matrices $P_v^{(1)}, \ldots, P_v^{(\ell)} \in \mathbb{F}^{m\ell\times m}$ s.t. $P_v^{(i)}[j] = P_v[(i-1)\ell + j]$ for $j \in [m]$. Define a size-$\tilde{m}$ multiplicative subgroup $\mathbb{M}$ with generator θ s.t. $\tilde{m} = \max\{||P_v^{(i)}||\}$, *i.e.*, the maximum non-zero entry number among all $P_v^{(i)}$. Define polynomials $\{\mathsf{val}_i\} \in \mathbb{F}_{\tilde{m}}[X]$ s.t. $\mathsf{val}_i(\theta^{j-1})$ is the value of the j-th non-zero entry in $P_v^{(i)}$. The non-zero entries are assumed in some canonical order (*e.g.*, row-wise or column-wise). Similarly, define polynomials $\mathsf{row}_i \in \mathbb{F}_{\tilde{m}}[X]$ (resp., col_i) s.t. $\mathsf{row}_i(\theta^{j-1})$ (resp., $\mathsf{col}_i(\theta^{j-1})$) is the row (resp., column) index (counting from 0) of the j-th non-zero entry in $P_v^{(i)}$. If $j > ||P_v^{(i)}||$, set $\mathsf{val}_i(\theta^{j-1}) = 0$. row, col, val suffice to describe a matrix. Define polynomials $\mathsf{val}(X,Y) = \sum_{i\in[\ell]} \mathsf{val}_i(X)L_i(Y)$, and similar for $\mathsf{col}(X,Y)$, $\mathsf{row}(X,Y)$. Define $L(X,Y) = \sum_{i\in[\ell]} L_i(X)L_i(Y)$.

Remark 1. *After splitting the R1CS public matrix* $\mathcal{P}_v$ *into sub-matrices, the non-zero entries of each sub-matrix may not be even. Looking ahead, our prover complexity is* $O(\tilde{m}\log\tilde{m})$. *Although* $\tilde{m} = O(m)$, *the uneven sub-matrices may lead to large concrete* $\tilde{m}$ *and worsens the performance. To solve this problem, we observe that the non-zero entries in R1CS matrices represent the left, right, or output wires of circuit gates. Given a fixed circuit, the entries' locations are only determined by gate indexes, which can be manually adjusted. Hence, the indexer, delegated by the verifier to do the matrix-related public computation, can adjust these locations to build more even sub-matrices. We propose such an algorithm in Appendix B.*

Main idea. The verifier needs $f_{P'_v}(\alpha,\beta) = \sum_{i\in[\ell]} f_{P'_v[i]}(\alpha)L_i(\beta)$ in Protocol 3. Thanks to our coefficient-based representations, we have $f_{P'_v[i]}(X) = \sum_{x\in\mathbb{M}} \mathsf{val}_i(x)r^{\mathsf{row}_i(x)}X^{\mathsf{col}_i(x)}$. Further, $f_{P'_V}(\alpha,\beta) = \sum_{x\in\mathbb{M}}\sum_{i\in[\ell]} \mathsf{val}_i(x)r^{\mathsf{row}_i(x)}\cdot\alpha^{\mathsf{col}_i(x)}L_i(\beta)$. By Lemma 1, we have

$$f_{P'_v}(\alpha,\beta) = \sum_{x\in\mathbb{M}}\sum_{y\in\mathbb{L}} \mathsf{val}(x,y)\cdot r^{\mathsf{row}(x,y)}\cdot\alpha^{\mathsf{col}(x,y)}\cdot L(\beta,y). \quad (13)$$

Suppose bivariate polynomials $A, B \in \mathbb{F}_{\tilde{m},\ell}[X,Y]$ defined by $\{r^{\mathsf{row}(x,y)}, \alpha^{\mathsf{col}(x,y)}\}_{x\in\mathbb{M},y\in\mathbb{L}}$, respectively. Now, Equation (13) becomes

$$f_{P'_v}(\alpha,\beta) = \sum_{x\in\mathbb{M}}\sum_{y\in\mathbb{L}} \mathsf{val}(x,y)\cdot A(x,y)\cdot B(x,y)\cdot L(\beta,y). \quad (14)$$

By Lemma 1, we further have

$$f_{P'_v}(\alpha,\beta)=\sum_{x\in\mathbb{M}}\sum_{i\in[\ell]}\mathsf{val}_i(x)\cdot A_i(x)\cdot B_i(x)\cdot L_i(\beta). \quad (15)$$

Indeed, for all $x\in\mathbb{M}$, we exactly have $A_i(x)=r^{\mathsf{row}_i(x)}, B_i(x)=\alpha^{\mathsf{col}_i(x)}$. Taking the first as an example, this is because, by definition for all $x\in\mathbb{M}$ and $y\in\mathbb{L}$, it holds that $A(x,y)=r^{\mathsf{row}(x,y)}$. Setting $y=\eta^{i-1}$, then $A(x,y)=A_i(x)=r^{\mathsf{row}(x,y)}=r^{\mathsf{row}_i(x)}$.

Similar to Equation (10), Equation (15) can be proved distributedly. The sub-prover $\mathcal{P}_i$ run a first-time univariate sum-check over X for target polynomial $\mathsf{val}_i(X)\cdot A_i(X)\cdot B_i(X)\cdot L_i(\beta)$. Given the challenge $\alpha,\mathcal{P}_i$ and $\mathcal{P}_0$ collaboratively run a second-time sum-check over Y for target polynomial $\mathsf{val}(\alpha,Y)\cdot A(\alpha,Y)\cdot B(\alpha,Y)\cdot L(\beta,Y)$.

One problem remains to check the validity of non-linear functions $A(X,Y),B(X,Y)$ given their oracles. This is typically handled by lookup arguments. Taking the column as an example, we can prove $\{(\mathsf{col}(x,y),B(x,y))\}_{x\in\mathbb{M},y\in\mathbb{L}}\subseteq\{(k-1,\alpha^{k-1})\}_{k\in[m]}$. However, building the lookup arguments can be challenging in four aspects. We delve into these challenges and give solutions below.

4.3.1 Checking the table validity online. We start by handling the simpler univariate lookup relation about $P_v[i]$'s column

$$\{(\mathsf{col}_i(x),B_i(x))\}_{x\in\mathbb{M}}\subseteq\{(j-1,\alpha^{j-1})\}_{j\in[m]} \quad (16)$$

Here, $\mathsf{col}_i,B_i\in\mathbb{F}_{\tilde{m}}[X]$ such that $\{B_i(x)=\alpha^{\mathsf{col}_i(x)}\}_{x\in\mathbb{M}}$.

Most lookup arguments [23, 25, 63] assume a trusted party to commit the table prescribedly. However, our table $\{(j-1,\alpha^{j-1})\}_{j\in[m]}$, is determined by the online challenge α. Then, a direct verifier complexity to check the table is linear to the table size.

To solve this problem, we modify our tables with a special structure to allow an online table validity check with sublinear verification. Assume ω is the generator of $\mathbb{H}$, we can transform the Equation (16) into $\{(\omega^{\mathsf{col}_i(x)},B_i(x))\}_{x\in\mathbb{M}}\subseteq\{(\omega^{j-1},\alpha^{j-1})\}_{j\in[m]}$.

Define $T_{\mathsf{col}}\in\mathbb{F}_m[X]$ s.t. $T_{\mathsf{col}}(\omega^{j-1})=\alpha^{j-1}$. As T_{col} includes all the table information, the prover can send a polynomial oracle of T_{col} to the verifier, who then check the table validity by checking the evaluation validity of T_{col}. To complete this, our key observation is that it suffices to prove the following equations:

$$T_{\mathsf{col}}(1)=1,\quad T_{\mathsf{col}}(\omega x)=\alpha\cdot T_{\mathsf{col}}(x),\text{for } x\in\{\omega^j\}_{j\in[0,m-2]}. \quad (17)$$

A typical approach for the latter relation is to show the existence of q, which is the division polynomial of $p(X)=T_{\mathsf{col}}(\omega X)-\alpha T_{\mathsf{col}}(X)$ divided by $\prod_{j\in[0,m-2]}(X-\omega^j)$. However, it incurs a new oracle q.

We observe that the degree-$(m-1)$ polynomial $p(X)$ equals to zero on $m-1$ distinct points, where $x\in\{w^j\}_{j\in[0,m-2]}$. If there exists a degree-$(m-1)$ polynomial $p'(X)$ built from $p(X)$ s.t. $p'(\omega^{m-1})=0$ and $p'(\omega^j)=p(\omega^j)$ for $j\in[0,m-2]$, then $p'(X)$ is a zero polynomial. We introduce the auxiliary public Lagrange polynomial $L_m\in\mathbb{F}_m[X]$ to build p' from p. Specifically, $L_m(x)=0$ for $x\in\{\omega^j\}_{j\in[0,m-2]}$ and $L_m(\omega^{m-1})=1$, and any point on $L_m(X)$ can be computed in logarithmic time. Equation (17) then becomes: for all $x\in\mathbb{H}$, it holds that $T_{\mathsf{col}}(\omega x)=\alpha\cdot T_{\mathsf{col}}(x)+L_{\mathbb{H},m}(x)\cdot(1-\alpha^m)$. As $T_{\mathsf{col}},L_{\mathbb{H},m}\in\mathbb{F}_{m-1}[X]$, we have

$$p'(X)=T_{\mathsf{col}}(\omega X)-\alpha\cdot T_{\mathsf{col}}(X)-L_{\mathbb{H},m-1}(X)\cdot(1-\alpha^m) \quad (18)$$

is a zero polynomial. This can be checked by querying a random point on $T_{\mathsf{col}}(X)$ and checking if $p'(X)$ is zero at this point. Further, our method does not require any new polynomial oracle.

4.3.2 Building double-dimension lookup PIOP. We use the lookup PIOP in logup [33] to build our scheme. Lemma 2 recalls this PIOP.

Lemma 2 ([33]). *$\{a_i\}_{i\in[\tilde{m}]}\subseteq\{b_j\}_{j\in[m]}$ iff there exists a sequence $\{n_j\}_{j\in[m]}$ where $n_j\in[0,\tilde{m}]$ s.t. $\sum_{i\in[\tilde{m}]}\frac{1}{X+a_i}=\sum_{j\in[m]}\frac{n_j}{X+b_j}$. Here, n_j describes the number of elements in $\{a_i\}_{i\in[\tilde{m}]}$ equaling b_j.*

The PIOP above is for one-dimension relation, while Equation (16) is double-dimension. For a two-dimension relation $\{(a_i,A_i)\}_{i\in[\tilde{m}]}\subseteq\{(b_j,B_j)\}_{j\in[m]}$, given a challenge β, the prover can show $\{\beta\cdot a_i+A_i\}_{i\in[\tilde{m}]}\subseteq\{\beta\cdot b_j+B_j\}_{j\in[m]}$. If some $(a_k,A_k)\notin\{(b_j,B_j)\}$, then $\beta\cdot a_k+A_k\in\{\beta\cdot b_j+B_j\}$ holds with an error probability of $m/|\mathbb{F}|$.

With this generalization at hand, we prove Equation (16). Define polynomial $b_i\in\mathbb{F}_{\tilde{m}}[X]$ s.t. $b_i(x)=\omega^{\mathsf{col}_i(x)}$ for all $x\in\mathbb{M}$. Define $n_{\mathsf{col}}\in\mathbb{F}_m[Y]$ s.t. $n(\omega^{j-1})$ equals how many times $\mathsf{col}_i(x)=j-1$, or $b_i(x)=\omega^{j-1}$ for $x\in\mathbb{M}$. Now, we prove Equation (16) by

$$\sum_{x\in\mathbb{M}}\frac{1}{X+\beta\cdot b_i(x)+B_i(x)}=\sum_{y\in\mathbb{H}}\frac{n_{\mathsf{col}}(y)}{X+\beta\cdot y+T_{\mathsf{col}}(y)}. \quad (19)$$

To prove Equation (19), given a challenge γ, the prover can show its validity at a random γ. Define $f_1,f_2\in\mathbb{F}_{\tilde{m}}[X]$ s.t. $\{f_1(x)=\frac{1}{\gamma+\beta\cdot b_i(x)+B_i(x)}\}_{x\in\mathbb{M}}$ and $\{f_2(y)=\frac{n_{\mathsf{col}}(y)}{\gamma+\beta\cdot y+T_{\mathsf{col}}(y)}\}_{y\in\mathbb{H}}$. The prover sends polynomial oracles of f_1,f_2 to the verifier. Then, Equation (19) becomes a univariate sum-check, *i.e.*, $\sum_{x\in\mathbb{M}}f_1(x)=\sum_{y\in\mathbb{H}}f_2(y)$.

The remaining issue is how to prove the validity of f_1,f_2 given their oracles , which can be solved by a standard zero check [27]. For example, for f_2, the prover can show the existence of $q_2(Y)$ s.t.

$$f_2(Y)\cdot(\gamma+\beta\cdot Y+T_{\mathsf{col}}(Y))-n_{\mathsf{col}}(Y)=q_2(Y)Z_{\mathbb{H}}(Y). \quad (20)$$

4.3.3 Achieving distribution. The PIOP above is non-distributed. We now prove the modified lookup relation $\{(\omega^{\mathsf{col}(x,y)},\alpha^{\mathsf{col}(x,y)})\}\subseteq\{(\omega^{j-1},T_{\mathsf{col}}(\omega^{j-1}))\}$ distributedly. Assume that the sub-prover $\mathcal{P}_i$ holds $\mathsf{col}_i(X)$. By definition, she can compute $b_i(X)$ and $B_i(X)$ s.t. $\{b_i(x)=\omega^{\mathsf{col}_i(x)}\}_{x\in\mathbb{M}}$ and $\{B_i(x)=\alpha^{\mathsf{col}_i(x)}\}_{x\in\mathbb{M}}$. Define $b(X,Y)=\sum_{i\in[\ell]}b_i(X)L_i(Y)$ and $B(X,Y)=\sum_{i\in[\ell]}B_i(X)L_i(Y)$. By definition, $b(x,y)=\omega^{\mathsf{col}(x,y)}$ and $B(x,y)=\alpha^{\mathsf{col}(x,y)}$ for any $x\in\mathbb{M}$ and $y\in\mathbb{L}$. Define $n_{\mathsf{col}}\in\mathbb{F}_{\tilde{m}}[Y]$ s.t. $n_{\mathsf{col}}(\omega^{j-1})$ equals the times of $\{\mathsf{col}(x,y)\}_{x\in\mathbb{M},y\in\mathbb{L}}=j-1$. Equation (19) becomes

$$\sum_{x\in\mathbb{M},y\in\mathbb{L}}\frac{1}{\gamma+\beta\cdot b(x,y)+B(x,y)}=\sum_{y\in\mathbb{H}}\frac{n_{\mathsf{col}}(y)}{\gamma+\beta\cdot y+T_{\mathsf{col}}(y)}. \quad (21)$$

Let $f_{1,i}\in\mathbb{F}_{\tilde{m}}[X]$ s.t. $f_{1,i}(x)=\frac{1}{\gamma+\beta\cdot b_i(x)+B_i(x)}$ for $x\in\mathbb{M}$. Define $f_1(X,Y)=\sum_{i\in[\ell]}f_{1,i}(X)L_i(Y)$. By definition, we have for any $x\in\mathbb{M}$ and $y\in\mathbb{L}$, $f_1(x,y)=\frac{1}{\gamma+\beta\cdot b(x,y)+B(x,y)}$. Hence, the sum-check relation becomes $\sum_{x\in\mathbb{M},y\in\mathbb{L}}f_1(x,y)=\sum_{y\in\mathbb{H}}f_2(y)$. By the univariate sum-check and the low-degree properties of f_1 and f_2, we stress that it suffices to prove $\tilde{m}\ell f_1(0,0)=mf_2(0)$, which can be distributedly proved similar to Equation (10) by Lemma 1.

However, given the oracle of $f_1(X,Y)$, the verifier has to check its validity, which can be challenging due to the lack of (distributed)

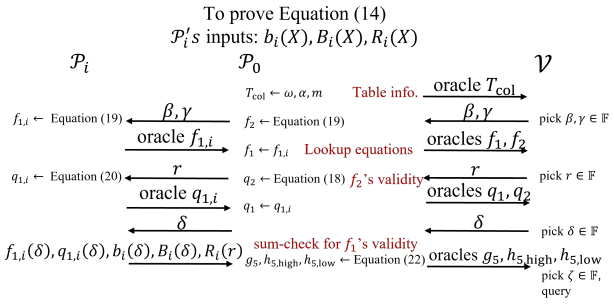

Figure 2: Visualization of the distributed lookup PIOP

zero-check for bivariate polynomials. To see this, although there exists q_1, q_2 s.t. $f_1(X,Y)(\gamma+\beta\cdot b(X,Y)+B(X,Y))-1 = q_1(X,Y)Z_{\mathbb{M}}(X) + q_2(X,Y)Z_{\mathbb{L}}(Y)$, the degrees of q_1, q_2 are larger than $m\ell$, and generating q_1, q_2 distributedly may be hard.

We find that to prove $\{f_1(x,y) = \frac{1}{\gamma+\beta\cdot b(x,y)+B(x,y)}\}_{x\in\mathbb{M},y\in\mathbb{L}}$, it suffices to prove that for all $y \in \mathbb{L}$, there exists $q_i(X)$ such that $f_1(X,y)\cdot(\gamma+\beta\cdot b(X,y)+B(X,y))-1 = q_{1,i}(X)Z_{\mathbb{M}}(X)$. For $\mathcal{P}_i$, we have $y = \eta^{i-1}$. By the definition of f_1, b, B, we further have

$$f_{1,i}(X)\cdot(\gamma+\beta\cdot b_i(X)+B_i(X))-1 = q_{1,i}(X)Z_{\mathbb{M}}(X), \tag{22}$$

which can be handled by $\mathcal{P}_i$ locally. Inspired by [11], we use random linear combination of r to reduce the validity of ℓ quotient relations of all sub-provers in Equation (22) into the existence of $q_1(X)$ s.t.

$$\sum_{i\in[\ell]} r^{i-1}\cdot\big(f_{1,i}(X)\cdot(\gamma+\beta\cdot b_i(X)+B_i(X))-1\big) = q_1(X)Z_{\mathbb{M}}(X). \tag{23}$$

By definition, $q_1(X) = \sum_{i\in[\ell]} r^{i-1}\cdot q_{1,i}(X)$. Define $R(X,Y) = \sum_{i\in[\ell]} X^{i-1}L_i(Y)$. By Lemma 1, Equation (23) is equivalent to

$$\sum_{y\in\mathbb{L}} R(r,y)\big(f_1(X,y)(\gamma+\beta\cdot b(X,y)+B(X,y))-1\big) = q_1\cdot Z_{\mathbb{M}}. \tag{24}$$

By Schwartz-Zippel lemma, the verifier can pick a random $X=\delta$ and check its validity. After fixing a δ, this become a univariate sum-check over Y. Note that the computation of bivariate polynomials like $f_1(\delta,y)$ can be distributed similar to Equation (10) by Lemma 1.

Combining Sections 4.3.1- 4.3.3, we present the lookup PIOP for columns in Protocol 4. Figure 2 presents a visualization.

Theorem 3. *Protocol 4 is a distributed lookup PIOP. The prover complexity of $\mathcal{P}_0$ is $O(m\log m+\ell)$. The prover complexity of $\mathcal{P}_i$ is $O(\tilde{m}\log\tilde{m})$. The proof size and amortized communication complexity are $O(1)$. The verifier complexity is $O(\log m\ell)$, and can be $O(1)$*[3].

Proof of Theorem 3. Completeness holds directly. For soundness, if some $(b(x,y),B(x,y)) \notin \{(\omega^{j-1},\alpha^{j-1})\}_{j\in[m]}$ for some $x\in\mathbb{M}$ and $y\in\mathbb{L}$, then for the random challenge β picked by the verifier, with a probability of at most $m/|\mathbb{F}|$, $\beta\cdot b(x,y)+B(x,y) \in \{\beta\cdot\omega^{j-1}+\alpha^{j-1}\}_{j\in[m]}$. For $\tilde{m}\ell$ instances, the soundness error is at most $\tilde{m}m\ell/|\mathbb{F}|$ by the union bound argument. The soundness error for checking the validity of T_{col}, f_2, f_1 is at most $(2\tilde{m}\ell)/|\mathbb{F}|$ according to the Schwartz-Zippel lemma. For the validity check of f_1, according to Lemma 3 [11], with at most a soundness error of $\ell/|\mathbb{F}|$, there exists $f_{1,k,k\in[\ell]}$ s.t. $Z_{\mathbb{M}} \nmid f_{1,k}$ while $\sum_{i\in[\ell]} r^{i-1}f_{1,i} \mid Z_{\mathbb{M}}$.

Lemma 3 ([11]). *Fix $F_1,\dots,F_k \in \mathbb{F}_d[X]$. Fix $Z\in\mathbb{F}_d[X]$ that decomposes to distinct linear factors over $\mathbb{F}$. Suppose for some $i\in[n]$, $Z\nmid F_i$. Then, except with probability $k/|\mathbb{F}|$ over the choice of random $\gamma\in\mathbb{F}$, it holds that $Z\nmid G=\sum_{j=1}^{k}\gamma^{j-1}\cdot F_j$.*

We next prove the equivalence of

$$\sum_{x\in\mathbb{M}}\sum_{y\in\mathbb{L}} f_1(x,y) = \sum_{x\in\mathbb{L}} f_2(x) \quad \text{and} \quad \tilde{m}\ell f_1(0,0) = m f_2(0).$$

By Lemma 1, we have

$$\sum_{x\in\mathbb{M}}\sum_{y\in\mathbb{L}} f_1(x,y) = \sum_{x\in\mathbb{M}}\sum_{i\in[\ell]} f_{1,i}(x).$$

As $\deg(f_{1,i}) < \tilde{m}$, by the univariate sum-check, this sum equals

$$\tilde{m}\cdot\sum_{i\in[\ell]} f_{1,i}(0) = \tilde{m}\cdot\sum_{y\in\mathbb{L}} f_1(0,y).$$

Again as the Y-degree of f_1 is less than ℓ, again by the univariate sum-check, we have

$$\tilde{m}\cdot\sum_{y\in\mathbb{L}} f_1(0,y) = \tilde{m}\ell\cdot f_1(0,0).$$

Similarly, $\sum_{x\in\mathbb{L}} f_2(x) = m\cdot f_2(0)$. The equivalence hence holds.

Complexity. For the master prover, computing the polynomials T_{col}, f_2, q_2 requires $O(m\log m)$ time, and computing the polynomial oracles $f_1, q_1, g_5, h_{5,\text{low}}, h_{5,\text{high}}$ needs $O(\ell)$ time. For the sub-prover, computing oracles $f_{i,1}, q_{1,i}$ costs $O(\tilde{m}\log\tilde{m})$ time. The proof size includes 11 oracles and 15 polynomial evaluations.

The verifier's overhead includes: 1) time-$O(\ell^2)$ computation for computing $R(r,\zeta)$; 2) time-$O(\log m\ell)$ computation for $L_{\mathbb{H},m-1}(\delta)$, $Z_{\mathbb{M}}(\delta), Z_{\mathbb{L}}(\zeta)$. These polynomial evaluations can be delegated by the provers. The verifier complexity is then reduced to $O(1)$. Note that these at most introduces an additional overhead of $O(\ell)$ for provers, and do not affect the prover complexities. □

4.3.4 Building lookup PIOP for the row relation. We have built a distributed lookup PIOP for the column relation. Now we construct a PIOP for row, *i.e.*, $\{(\text{row}(x,y),A(x,y))\}_{x\in\mathbb{M},y\in\mathbb{L}} \subseteq \{(k-1, r^{k-1})\}_{k\in[m\ell]}$. These meanings can be found in Equation (14).

Different from the size-m columns, the table size of the row is $m\ell$. As our lookup PIOP has a sub-prover complexity linear to the table size, this would lead to a $O(m\ell)$ sub-prover complexity, which is not acceptable. To tackle this, we split the size-$m\ell$ table into two size-$\sqrt{m\ell}$ tables. As the number of sub-provers, *i.e.*, ℓ, is usually smaller than m, it holds that $\sqrt{m\ell} < m$. Now the provers handle 2 size-$O(m)$ tables, reducing the sub-prover complexity to $O(m)$.

Specifically, define the split polynomials $\text{row}_{\text{low},i}, \text{row}_{\text{high},i} \in \mathbb{F}_{\tilde{m}}[X]$ mapping $\mathbb{M}$ to $[0,\sqrt{m\ell}-1]$. Let $\text{row}_i(x) = \text{row}_{\text{high},i}(x)\sqrt{m\ell} + \text{row}_{\text{low},i}(x)$ for any $x\in\mathbb{M}$. By definition, this split is unique. Define $\text{row}_{\text{high}}(X,Y) = \sum_{i\in[\ell]} \text{row}_{\text{high},i}(X)L_i(Y)$ and $\text{row}_{\text{low}}(X,Y)$ similarly. Then we have $r^{\text{row}(X,Y)} = (r^{\sqrt{m\ell}})^{\text{row}_{\text{high}}(X,Y)}\cdot r^{\text{row}_{\text{low}}(X,Y)}$.

[3]The $O(\log m\ell)$ verifier complexity due to computing public vanishing polynomials can be further reduced to $O(1)$ pairings via polynomial delegation similar to [18, 27, 42]. Notably, the $O(\log m\ell)$ field operations are concretely faster than $O(1)$ pairings.

Assume bivariate polynomials $A_{\mathsf{high}}, A_{\mathsf{low}} \in \mathbb{F}_{\tilde{m},\ell}[X,Y]$ defined by $\{(r^{\sqrt{m\ell}})^{\mathsf{row}_{\mathsf{high}}(x,y)}\}_{x\in\mathbb{M},y\in\mathbb{L}}$ and $\{r^{\mathsf{row}_{\mathsf{high}}(x,y)}\}_{x\in\mathbb{M},y\in\mathbb{L}}$, respectively. Then, $f_{P'_v}(\alpha,\beta)$ in Equation (13) can be finally written as

$$\sum_{x\in\mathbb{M},y\in\mathbb{L}} \mathsf{val}(x,y)\cdot A_{\mathsf{high}}(x,y)\cdot A_{\mathsf{low}}(x,y)\cdot B(x,y)\cdot L(\beta,y). \quad (25)$$

Also, $\{(\mathsf{row}(x,y), A(x,y))\}_{x\in\mathbb{M},y\in\mathbb{L}} \subseteq \{(k-1, r^{k-1})\}_{k\in[m\ell]}$, the lookup relation, is transformed into two sub-relations

$$\{\mathsf{row}_{\mathsf{high}}(x,y), A_{\mathsf{high}}(x,y)\}_{x\in\mathbb{M},y\in\mathbb{L}} \subseteq \{(k-1,(r^{\sqrt{m\ell}})^{k-1})\}$$
$$\{\mathsf{row}_{\mathsf{low}}(x,y), A_{\mathsf{low}}(x,y)\}_{x\in\mathbb{M},y\in\mathbb{L}} \subseteq \{(k-1, r^{k-1}\},$$

which can be proved similar to the column relation.

Batch lookup PIOP. Protocol 4 can be extended to a batch version. For multiple relations with the same table, we can use the same T_{col}. Further, to prove the validity of multiple $f_{k,2}(Y)$ for t sets of polynomials $\{T_{k,\mathsf{col}}, n_{i,\mathsf{col}}\}_{k\in[t]}$, we can combine relations in Equation (20) using a random linear combination with challenge v:

$$\sum_{k\in[t]} v^{k-1}\cdot f_{k,2}(Y)(\gamma+\beta\cdot Y+T_{k,\mathsf{col}}(Y)) - n_{k,\mathsf{col}}(Y) = q_2(Y)Z_{\mathbb{H}}(Y).$$

Similarly, for multiple $f_{k,1}(X,Y)$ for t sets of bivariate polynomials $\{b_k(X,Y), B_k(X,Y)\}_{k\in[t]}$ as in Equation (24), let $f_{k,1}(X,y) = R(r,y)\cdot\big(f_1(X,y)\cdot(\gamma+\beta\cdot b(X,y)+B(X,y))-1\big)$ We use a random linear combination with challenge v to prove

$$\sum_{y\in\mathbb{L}}\sum_{k\in[t]} v^{k-1}\cdot f_{k,1}(X,y) = q_1(\delta)Z_{\mathbb{M}}(\delta).$$

According to [11], if any polynomial does not divide q_1 or q_2, then with a soundness error of $t/|\mathbb{F}|$, the random linear combination of this polynomials does not divide q_1 or q_2. Now, only 2 polynomials $q_1(X), q_2(Y)$ are required instead of t polynomials.

Putting everything together. Using the lookup PIOP in Protocol 4, we can modify the PIOP for R1CS in Protocol 3 to achieve a sublinear verifier complexity. We present it in Protocol 5.

Protocol 5 is a combination of the PIOP for R1CS without preprocessing in Protocol 3 and the lookup PIOP in Protocol 4. The security and complexity properties inherit from these two protocols.

Theorem 4. *Protocol 5 is a distributed PIOP for R1CS. The sub-prover complexity is $O(m\log m)$, and the master prover complexity is $O(\ell\log\ell + m\log m)$. The proof size and amortized communication complexity are $O(1)$. The verifier complexity can be $O(1)$.*

Distributed preprocessing. The preprocessing phase can be distributed similar to the proving phase. This is reasonable as the verifier is assumed to delegate the public computation to a trusted indexer, who is expected to be powerful in computational resources. Also, the preprocessing can be time-consuming in practice. In Marlin [18], the indexer time is nearly the same as the prover time.

In Protocol 5, the indexer needs to compute degree-$(\tilde{m},\ell)$ bivariate polynomials and degree-m univariate polynomials. The former can be handled distributedly similar to Protocol 2. The latter costs $O(m\log m)$ time and can be handled directly. Hence, given ℓ machines, the sub-machine complexity of indexer is $O(m\log m)$, and the master-machine complexity is $O(\ell\log\ell + m\log m)$.

Protocol 4 (Distributed lookup PIOP for column). *Public inputs: public α. Order-$\tilde{m}$ multiplicative subgroup $\mathbb{M}$, Order-ℓ multiplicative subgroup $\mathbb{L}$. Generator ω of order-m subgroup $\mathbb{H}$.*
Secret inputs: $\{\mathsf{col}_i(X)\}_{i\in[\ell]}$, $\mathsf{col}(X,Y) = \sum_{i\in[\ell]}\mathsf{col}_i(X)L_i(Y)$.
Statement: The prover proves to $\mathcal{V}$ the validity of Equation (16).

Offline phase:

1. *Indexer $\mathcal{I}$ computes $\{b_i(X)\}_{i\in[\ell]}$ s.t. $b_i(x) = \omega^{\mathsf{col}_i(x)}$ for $x\in\mathbb{M}$. Compute $b(X,Y) = \sum_{i\in[\ell]} b_i(X)L_i(Y)$. Similarly, compute $B_i(X)$ and $B(X,Y)$. Compute $R(X,Y) = \sum_{i\in[\ell]} X^{i-1}L_i(Y)$.*
2. *$\mathcal{I}$ computes polynomial $n_{\mathsf{col}}\in\mathbb{F}_m[Y]$ s.t. $n_{\mathsf{col}}(\omega^{j-1})$ equals the times of $\mathsf{col}(x,y) = j-1$ for $x\in\mathbb{M}$ and $y\in\mathbb{L}$.*
3. *$\mathcal{I}$ sends $n_{\mathsf{col}}(Y)$ to $\mathcal{P}_0$, sends $\mathsf{col}_i(X), b_i(X), B_i(X)$ to $\mathcal{P}_i$, and sends the oracles of $b(X,Y), B(X,Y), n_{\mathsf{col}}(Y)$ to $\mathcal{V}$.*

Online phase:

1. *$\mathcal{P}_0$: computes table-related $T_{\mathsf{col}}\in\mathbb{F}_m[X]$ s.t. $T_{\mathsf{col}}(\omega^{j-1}) = \alpha^{j-1}$ for $j\in[m]$. Send the polynomial oracle T_{col} to $\mathcal{V}$.*
2. *$\mathcal{V}$: sends challenges $\beta,\gamma\in\mathbb{F}$ to $\mathcal{P}_0$, who transfers to $\mathcal{P}_i$.*
3. *Provers compute and send polynomial oracles f_2, f_1 to $\mathcal{V}$. The lookup relation is reduced to relations of f_1, f_2 as in Equation (21).*
   (a) *$\mathcal{P}_0$: computes $f_2\in\mathbb{F}_m[Y]$ s.t. $\{f_2(y) = \frac{n_{\mathsf{col}}(y)}{\gamma+\beta\cdot y+T_{\mathsf{col}}(y)}\}_{y\in\mathbb{H}}$.*
   (b) *$\mathcal{P}_i$: computes $f_{i,1}\in\mathbb{F}_{\tilde{m}}[X]$ s.t. $f_{i,1}(x) = \frac{1}{\gamma+\beta\cdot b_i(x)+B_i(x)}$ for $x\in\mathbb{M}$. Send polynomial oracle $f_{i,1}(X)$ to $\mathcal{V}$.*
   (c) *$\mathcal{P}_0$: computes the oracle $f_1(X,Y) = \sum_{i\in[\ell]} f_{i,1}(X)L_i(Y)$.*
4. *$\mathcal{V}$: sends a random challenge $r\in\mathbb{F}$ to $\mathcal{P}_0$, who transfers to $\mathcal{P}_i$.*
5. *Provers compute and send polynomial oracles q_1, q_2 to $\mathcal{V}$.*
   (a) *$\mathcal{P}_0$: computes $q_2\in\mathbb{F}_m[Y]$ according to Equation (20).*
   (b) *$\mathcal{P}_i$: computes $q_{1,i}\in\mathbb{F}_{\tilde{m}}[X]$ according to Equation (22). Send polynomial oracle $q_{1,i}(X)$ to $\mathcal{P}_0$.*
   (c) *$\mathcal{P}_0$: computes the oracle $q_1(X) = \sum_{i\in[\ell]} r^{i-1}\cdot q_{1,i}(X)$.*
6. *$\mathcal{V}$: sends a random challenge $\delta\in\mathbb{F}$ to $\mathcal{P}_0$, who transfers to $\mathcal{P}_i$.*
7. *$\mathcal{P}_0$: computes and sends oracles $g_5, h_{5,\mathsf{low}}, h_{5,\mathsf{high}}\in\mathbb{F}_\ell[Y]$ to $\mathcal{V}$. These are used to prove the validity of f_1 as in Equation (24). $\mathcal{P}_i$ first sends $R_i(r), f_{1,i}(\delta), b_i(\delta), B_i(\delta), q_{1,i}(\delta)$ to $\mathcal{P}_0$. $\mathcal{P}_0$ then compute polynomials $g_5, h_{5,\mathsf{low}}, h_{5,\mathsf{high}}$ s.t.*

$$R(r,Y)\cdot f_1(\delta,Y)\big(\gamma+\beta\cdot b(\delta,Y)+B(\delta,Y)\big) - R(r,Y) =$$
$$Yg_5(Y) + q_1(\delta)Z_{\mathbb{M}}(\delta)/\ell + Z_{\mathbb{L}}(Y)(h_{5,\mathsf{low}}(Y)+\delta^\ell h_{5,\mathsf{high}}(Y)).$$

8. (a) *$\mathcal{V}$: checks table validity. Query $T_{\mathsf{col}}(\omega\delta), T_{\mathsf{col}}(\delta), T_{\mathsf{col}}(1)$. Check if Equation (18) holds and $T_{\mathsf{col}}(1) = 1$.*
   (b) *$\mathcal{V}$: checks the validity of $f_2(Y)$. Query $f_2, n_{\mathsf{col}}, q_2$ at δ, and check if Equation (20) holds.*
   (c) *$\mathcal{V}$: checks the validity of $f_1(X,Y)$. Pick a random $\zeta\in\mathbb{F}$. Query f_1, b, B at (δ,ζ). Query $R(r,\zeta)$. Query $q_1, g_5, h_{5,\mathsf{low}}, h_{5,\mathsf{high}}$ at ζ. Check if Step 7. holds.*
   (d) *$\mathcal{V}$: checks the validity of lookup relation as in Equation (21). Query $f_1(0,0), f_2(0)$. Check if $\tilde{m}\ell f_1(0,0) = mf_2(0)$.*

Protocol 5 (Distributed PIOP for R1CS with sublinear verifier). *Suppose the same setting as Protocol 2. Suppose an indexer $\mathcal{I}$.*

Offline phase:

1. *$\mathcal{I}$: reads the public matrix P_v. Compute 3 sets of polynomials $\{\mathsf{val}_i, \mathsf{col}_i, \mathsf{row}_{\mathsf{high},i}, \mathsf{row}_{\mathsf{low},i}\}_{i\in[\ell]} \in \mathbb{F}_{\tilde{m}}[X]$. These polynomials describe the information of the i-th sub-matrix $P_v[i]$. Compute bivariate polynomials $\mathsf{val}(X,Y), \mathsf{col}(X,Y), \mathsf{row}_{\mathsf{high}}(X,Y), \mathsf{row}_{\mathsf{low}}(X,Y)$.*
2. *$\mathcal{I}$: computes n_{col}, $n_{\mathsf{row,high}}$, $n_{\mathsf{row,low}} \in \mathbb{F}_m[X]$, which count the evaluation frequency of $\{\mathsf{col}(x,y), \mathsf{row}_{\mathsf{high}}(x,y), \mathsf{row}_{\mathsf{low}}(x,y)\}_{x\in\mathbb{M}, y\in\mathbb{L}}$.*
3. *$\mathcal{I}$: computes polynomials $a_{\mathsf{row,high},i}(X)$, $a_{\mathsf{row,low},i}(X)$, $b_{\mathsf{col},i}(X) \in \mathbb{F}_{\tilde{m}}[X]$ defined by $\{\omega^{\mathsf{row}_{\mathsf{high},i}(x)}, \omega^{\mathsf{row}_{\mathsf{low},i}(x)}, \omega^{\mathsf{col}_i(x)}\}_{x\in\mathbb{M}}$. Compute bivariate polynomials $a_{\mathsf{row,high}}(X,Y)$, $a_{\mathsf{row,low}}(X,Y)$, $b_{\mathsf{col}}(X,Y) \in \mathbb{F}_{\tilde{m},\ell}[X,Y]$*
4. *$\mathcal{I}$: computes bivariate polynomials $R(X,Y) = \sum_{i\in[\ell]} R_i(X)L_i(Y)$ and $L(X,Y) = \sum_{i\in[\ell]} L_i(X)L_i(Y)$, where $R_i(X) = X^{i-1}$.*

Online phase:

Input: *$\mathcal{P}_i$ holds univariate polynomials over X, including: $\mathsf{val}_i, \mathsf{col}_i, \mathsf{row}_{\mathsf{high},i}, \mathsf{row}_{\mathsf{low},i}, a_{\mathsf{row,high},i}, a_{\mathsf{row,low},i}, b_{\mathsf{col},i}, R_i, L_i$.*

$\mathcal{P}_0$ holds univariate polynomials over Y, including: $n_{\mathsf{col}}, n_{\mathsf{row,high}}, n_{\mathsf{row,low}}$.

$\mathcal{V}$ holds bivariate polynomial oracles $\mathsf{val}, a_{\mathsf{row,high}}, a_{\mathsf{row,low}}, b_{\mathsf{col}}, L, R$. $\mathcal{V}$ holds polynomial oracles over Y, including $n_{\mathsf{col}}, n_{\mathsf{row,high}}, n_{\mathsf{row,low}}$.

This protocol is a modification of to Protocol 3. Below we only describe the changes made to Protocol 3.

1. *In Step 2., provers additionally computes and sends to $\mathcal{V}$ the oracles $A_{\mathsf{high}}, A_{\mathsf{low}} \in \mathbb{F}_{\tilde{m},\ell}[X,Y]$ and oracles $T_{\mathsf{high}}, T_{\mathsf{low}} \in \mathbb{F}_m[X]$.*
   (a) *$\mathcal{P}_i$: computes $A_{\mathsf{high},i}, A_{\mathsf{low},i} \in \mathbb{F}_{\tilde{m}}[X]$ defined by $\{(r^{\sqrt{m\ell}})^{\mathsf{row}_{\mathsf{high},i}(x)}, r^{\mathsf{row}_{\mathsf{low},i}(x)}\}_{x\in\mathbb{M}}$. Send oracles $A_{\mathsf{high},i}(X), A_{\mathsf{low},i}(X)$ to $\mathcal{V}$.*
   (b) *$\mathcal{P}_0$: computes oracles $A_{\mathsf{high}}(X,Y) = \sum_{i\in[\ell]} A_{\mathsf{high},i}(X)L_i(Y)$ and $A_{\mathsf{low}}(X,Y) = \sum_{i\in[\ell]} A_{\mathsf{low},i}(X)L_i(Y)$.*
   (c) *$\mathcal{P}_0$: computes $T_{\mathsf{high}}, T_{\mathsf{low}} \in \mathbb{F}_m[X]$ s.t. $T_{\mathsf{high}}(\omega^{j-1}) = (r^{\sqrt{m\ell}})^{j-1}$ and $T_{\mathsf{low}}(\omega^{j-1}) = r^{j-1}$ for $j \in [m]$.*
2. *In Step 4., provers additionally computes and sends to $\mathcal{V}$ the oracle $B \in \mathbb{F}_{\tilde{m},\ell}[X,Y]$ and oracle $T_{\mathsf{col}} \in \mathbb{F}_m[X]$.*
   (a) *$\mathcal{P}_i$: computes $B_i \in \mathbb{F}_{\tilde{m}}[X]$ defined by $\{\alpha^{\mathsf{col}_i(x)}\}_{x\in\mathbb{M}}$. Send oracle $B_i(X)$ to $\mathcal{P}_0$.*
   (b) *$\mathcal{P}_0$: computes oracles $B(X,Y) = \sum_{i\in[\ell]} B_i(X)L_i(Y)$.*
   (c) *Compute $T_{\mathsf{col}} \in \mathbb{F}_m[X]$ s.t. $T_{\mathsf{col}}(\omega^{j-1}) = \alpha^{j-1}$ for $j \in [m]$.*
3. *In Step 5., after receiving β, the provers compute and sends $f_{P'_v}(\alpha,\beta)$ to $\mathcal{V}$. Provers and verifiers then prove its evaluation validity.*
   (a) *$\mathcal{P}$ and $\mathcal{V}$: invoke the first seven steps of Protocol 4 to prove the validity of $A_{\mathsf{high}}, A_{\mathsf{low}}, B$. They should satisfy for $x \in \mathbb{M}, y \in \mathbb{L}$, it holds that $\{(b_{\mathsf{row,high}}(x,y), A_{\mathsf{high}}(x,y))\} \subseteq \{(\omega^{k-1}, (r^{\sqrt{m\ell}})^{k-1})\}_{k\in[\sqrt{m\ell}]}$, $\{(b_{\mathsf{row,low}}(x,y), A_{\mathsf{low}}(x,y))\} \subseteq \{(\omega^{k-1}, r^{k-1})\}_{k\in[\sqrt{m\ell}]}$, and $\{(b_{\mathsf{col}}(x,y), B(x,y))\} \subseteq \{(\omega^{j-1}, \alpha^{j-1})\}_{j\in[m]}$. At the end, $\mathcal{V}$ receives univariate polynomial oracles $q_1 \in \mathbb{F}_{\tilde{m}}[X], q_2 \in \mathbb{F}_m[Y]$, and multiple polynomial oracles $f_1 \in \mathbb{F}_{\tilde{m},\ell}[X,Y], f_2 \in \mathbb{F}_m[Y]$.*
   (b) *$\mathcal{V}$: sends a random challenge $u_3 \in \mathbb{F}$ to $\mathcal{P}_0$.*
   (c) *Provers compute and send polynomial oracles $g_3, \{h_{3,j}\}_{j\in[3]} \in \mathbb{F}_{\tilde{m}}[X]$ of the third-round univariate sum-check to $\mathcal{V}$.*
      (i) *$\mathcal{P}_0$: transfers challenge u_3 to $\mathcal{P}_i$.*
      (ii) *$\mathcal{P}_i$: computes polynomials $g_{3,i}, \{h_{3,j,i}\}_{j\in[3]} \in \mathbb{F}_{\tilde{m}}[X]$ by dividing $Z_{\mathbb{M}}(X)$ of Equation (25) s.t.*

$$(X-u_3)\cdot \mathsf{val}_i(X)\cdot A_{\mathsf{high},i}(X)\cdot A_{\mathsf{low},i}(X)\cdot B_i(X)\cdot L_i(\beta) = X\cdot g_{3,i}(X) + (X-u_3)\cdot T_{3,i}/\tilde{m} + (X-u_3)\cdot Z_{\mathbb{M}}(X)\sum_{j\in[3]} X^{(j-1)\tilde{m}} h_{3,j,i}(X).$$

      Send polynomial oracles $g_{3,i}, \{h_{3,j,i}\}_{j\in[3]}$ to $\mathcal{P}_0$.
      (iii) *$\mathcal{P}_0$: computes oracles $g_3 = \sum_{i\in[\ell]} g_{3,i}$ and $h_{3,j} = \sum_{i\in[\ell]} h_{3,j,i}$ for $j \in [3]$.*
   (d) *$\mathcal{V}$: sends random challenges $\delta \in \mathbb{F}, u_4 \in \mathbb{F}$ to $\mathcal{P}_0$.*
   (e) *Provers compute and send polynomial oracles $g_4, \{h_{4,j}\}_{j\in[4]} \in \mathbb{F}_\ell[Y]$ of the fourth-round univariate sum-check to $\mathcal{V}$.*
      (i) *$\mathcal{P}_0$: transfers δ and u_4 to $\mathcal{P}_i$.*
      (ii) *$\mathcal{P}_i$: computes and sends $\mathsf{val}_i(\delta), A_{\mathsf{high},i}(\delta), A_{\mathsf{low},i}(\delta), B_i(\delta), L_i(\beta)$ to $\mathcal{P}_0$.*
      (iii) *$\mathcal{P}_0$: computes $\mathsf{val}(\delta,Y), A_{\mathsf{high}}(\delta,Y), A_{\mathsf{low}}(\delta,Y), B(\delta,Y), L(\beta,Y)$ Compute $g_4, \{h_{4,j}\}_{j\in[4]}$ such that*

$$(\delta-u_3)(Y-u_4)\cdot \mathsf{val}(\delta,Y)A_{\mathsf{high}}(\delta,Y)A_{\mathsf{low}}(\delta,Y)B(\delta,Y)\cdot L(\beta,Y) = Y\cdot g_4(Y) + (Y-u_4)T_4/\ell + (Y-u_4)Z_{\mathbb{L}}(Y)\sum_{j\in[4]} Y^{(j-1)\cdot\ell} h_{4,j}(Y), \quad (26)$$

   (f) *$\mathcal{V}$ outputs accept if and only if all the following checks pass:*
      (i) *Pick a random $\zeta \in \mathbb{F}$. Invoke Step 8. of Protocol 4 to verify the validity of $A_{\mathsf{high}}(X,Y), A_{\mathsf{low}}(X,Y), B(X,Y)$.*
      (ii) *Query $g_3(\delta), \{h_{3,j}(\delta)\}_{j\in[3]}$. Compute $T_4 = \delta\cdot g_3(\delta) + (\delta-u_3)\cdot f_{P'_v}(\alpha,\beta)/\tilde{m} + (\delta-u_3)Z_{\mathbb{M}}(\delta)(h_{3,1}(\delta) + \delta^{\tilde{m}} h_{3,2}(\delta) + \delta^{2\tilde{m}} h_{3,3}(\delta))$.*
      (iii) *Query $\mathsf{val}(\delta,\zeta), A_{\mathsf{high}}(\delta,\zeta), A_{\mathsf{low}}(\delta,\zeta), B(\delta,\zeta), L(\beta,\zeta), g_4(\zeta), \{h_{4,j}(\zeta)\}_{j\in[4]}$. Check if Equation (26) holds when $Y = \zeta$.*
      (iv) *Different from Step 5. in Protocol 3, query $R(r^m,\beta)$ instead of computing it himself. Invoke Step 5. using the evaluation $f_{P'_v}(\alpha,\beta)$.*

5 CONSTRUCTION OF DISTRIBUTED SNARK

5.1 Batch Bivariate KZG

Protocols 3 and 5 require opening multiple points on multiple bivariate polynomials. We use bivariate KZG as the PCS for its constant proof size and verification. However, directly running it incurs a large overhead. We propose a batch bivariate KZG for efficiency.

Building a batch bivariate KZG can be challenging even given a novel univariate batch approach as in [11], which we recall below. Given a set $S \subset \mathbb{F}$ of size t, denote Z_S as the vanishing polynomial on S. To prove the validity of $\{f(\alpha)\}_{\alpha\in S}$, the prover and verifier compute a public polynomial $r \in \mathbb{F}_t[X]$ s.t. $r(\alpha) = f(\alpha)$ for all α. Then, the prover shows the existence of polynomial $h(X)$ s.t.

$$f(X) - r(X) = h(X) \cdot Z_S(X). \tag{27}$$

However, the bivariate case can be more complex. For example, to prove $f(\alpha_1, \beta_1), f(\alpha_2, \beta_2)$, we can also build a bivariate r, but the quotient relation turns: there exists bivariate polynomials p, q s.t.

$$f(X,Y)-r(X,Y) = p(X,Y)\cdot(X-\alpha_1)(Y-\beta_2)+q(X,Y)\cdot(X-\alpha_2)(Y-\beta_1).$$

For more evaluations points, such as an additional $f(\alpha_3, \beta_3)$, it seems necessary to use more bivariate polynomials instead of two.

Suppose the evaluation points are $(\alpha_1, \beta_1), \ldots (\alpha_t, \beta_t)$. For simplicity, assume each α or β is distinct. Let $R = \{\alpha_1, \ldots, \alpha_t\}$ and $S = \{\beta_1, \ldots, \beta_t\}$. To construct a uniform quotient equation, we enlarge the evaluation points as $R \times S$, or say, $\{(\alpha, \beta)\}_{\alpha\in R,\beta\in S}$. Under this convention, the validity of all $\{f(\alpha, \beta)\}_{\alpha\in R,\beta\in S}$ is equivalent to the existence of bivariate polynomials p and q s.t.

$$f(X,Y) - r(X,Y) = p(X,Y) \cdot Z_R(X) + q(X,Y) \cdot Z_S(Y). \tag{28}$$

This equation suffices for opening the same set of evaluation points on multiple polynomials, but we still face challenges for distinct evaluation points. In the univariate case [11], each quotient polynomial is multiplied by some public polynomial, making them aggregatable by a random linear combination. For example, to open $f_1(X), f_2(X)$ on distinct S_1, S_2 with auxiliary polynomials $r_1(X), r_2(X)$, given a random challenge γ, we have

$$(f_1 - r_1) \cdot Z_{S_2} + \gamma \cdot (f_2 - r_2) \cdot Z_{S_1} = (h_1 + \gamma \cdot h_2) \cdot Z_{S_1} Z_{S_2}.$$

This combines $h_1(X)$ and $h_2(X)$, leading to a proof size non-related to the polynomial number. In Equation (28), however, finding such public polynomials is not straightforward as each equation has *two and possibly different* quotient polynomials, *i.e.*, $Z_R(X)$ and $Z_S(Y)$.

To address this problem, we introduce an auxiliary polynomial $s \in \mathbb{F}_{t,\ell}[X, Y]$ s.t. $s(\alpha, Y) = f(\alpha, Y)$ for all $\alpha \in R$. Further, let $r \in \mathbb{F}_{t,t}[X, Y]$ s.t. $r(X, \beta) = s(X, \beta)$ for all $\beta \in S$. By definition, $r(X, Y)$ satisfies $r(\alpha, \beta) = f(\alpha, \beta)$ for all $\alpha \in R$ and $\beta \in S$. Further, $r(X, Y)$ is public and can be computed from all evaluations of $f(\alpha, \beta)$.

By definition, $f(X, Y) - s(X, Y)$ equals zero on any $X = \alpha$ for $\alpha \in R$, hence $Z_R \mid f - s$. Similarly, we have $Z_S \mid s - r$. we further set p and q in Equation (28) to be $(f-s)/Z_R$ and $(s-r)/Z_S$, respectively. As $f - r = f - s + s - r$, Equation (28) can be divided into two parts: one related to Z_R and independent of Z_S; the other one related to Z_S and independent of Z_R. This enables the combination of quotient polynomials over X or Y separately. For example, for any two f_1, f_2, two s_1, s_2, two r_1, r_2, two Z_{R_1}, Z_{R_2}, and two Z_{S_1}, Z_{S_2}, we have

$$\begin{aligned}(f_1 - s_1) \cdot Z_{R_2} + \gamma \cdot (f_2 - s_2) \cdot Z_{R_1} &= (p_1 + \gamma \cdot p_2) \cdot Z_{R_1} Z_{R_2},\\ (s_1 - r_1) \cdot Z_{S_2} + \gamma \cdot (s_2 - r_2) \cdot Z_{S_1} &= (q_1 + \gamma \cdot q_2) \cdot Z_{S_1} Z_{S_2}.\end{aligned} \tag{29}$$

Using the evaluation padding and the auxiliary polynomial s, we follow the techniques in [11] to build a batch bivariate KZG. We consider n bivariate polynomials $\{f_k(X, Y)\}_{k\in[n]}$ with claimed evaluations $f_k(\alpha, \beta)$ for all $\alpha \in R_k$ and all $\beta \in S_k$. For simplicity, assume for all $k \in [n]$, $|R_k| = |S_k| = t$, and each polynomial is opened at t^2 points. Denote $T_1 = \bigcup_{k\in[n]} R_k$ and $T_2 = \bigcup_{k\in[n]} S_k$. We present the PCS in Protocol 6, and below we give the main idea.

First (in Step 1.), for $k \in [n]$, $\mathcal{P}$ computes auxiliary secret polynomial $s_k(X, Y)$ and sends its commitment to $\mathcal{V}$.

Second (in Step 3.), $\mathcal{P}$ splits $f_k(X, Y)$ as $P_k = f_k - s_k$ and $Q_k = s_k - r_k$ and handles these two polynomials separately like Equation (29). Given a random challenge γ, for the first half, $\mathcal{P}$ multiplies each P_k with an auxiliary polynomial $Z_{T_1\setminus R_k}(X)$ to allow a random linear combination of all P_k. The combined polynomial is

$$P(X,Y) = \sum_{k\in[n]} \gamma^{k-1} \cdot Z_{T_1\setminus R_k}(X) \cdot P_k(X,Y). \tag{30}$$

Similarly, $\mathcal{P}$ can compute $Q(X, Y)$, the second-half combined polynomial. By definition, $Z_{R_k} \mid P_k$, hence $Z_{T_1} \mid P$, and similarly $Z_{T_2} \mid Q$. $\mathcal{P}$ sends commitments $P(X, Y)/Z_{T_1}(X)$ and $Q(X, Y)/Z_{T_2}(Y)$.

Third (in Step 5.), $\mathcal{P}$ proves to $\mathcal{V}$ the validity of $P(X, Y)$ and $Q(X, Y)$ relying on challenges z_1 and z_2. $\mathcal{P}$ builds a polynomial $P'(X, Y)$ similar to Equation (30) but modifies $Z_{T_1\setminus R_k}(X)$ as $Z_{T_1\setminus R_k}(z_1)$. Let $W_1(X, Y) = P'(X, Y) - Z_{T_1}(z_1)P(X, Y)/Z_{T_1}(X)$. As $W_1(z_1, Y) = 0$, it holds that $(X - z_1) \mid W_1$. Similarly, $\mathcal{P}$ can build $Q'(X, Y)$ and $W_2(X, Y)$. Then, it holds that $(Y - z_2) \mid W_2$. $\mathcal{P}$ sends commitments $W_1(X, Y)/(X - z_1)$ and $W_2(X, Y)/(Y - z_2)$.

Finally (in Step 6.), $\mathcal{V}$ checks the validity of quotient relations $W_1(X, Y)/(X - z_1)$ and $W_2(X, Y)/(Y - z_2)$ at $X = \sigma$ and $Y = \tau$.

Theorem 5. *Protocol 6 is a batch bivariate PCS. Opening t^2 points on each of n bivariate polynomials with X-degree of m and Y-degree of ℓ, the opening complexity is $O(m\ell + nt\ell)$ group operations plus $O(nm\ell + nt\ell)$ field operations. The proof size is $n + 4$ group elements. The verifier complexity is $O(n)$ group operations plus 4 pairings.*

Proof of Theorem 5. Completeness holds by design. To prove knowledge soundness, we first recall some additional preliminaries in Appendix C. We prove the knowledge soundness in the algebraic group model (AGM), *i.e.*, the existence of an efficient extractor $\mathcal{E}$ s.t. an algebraic adversary $\mathcal{A}$ can only win the game with a probability of $\mathsf{negl}(n)(\lambda)$. We kindly refer to [11] for background about ideal and real pairing checks. We will use the following lemma for soundness proof.

Lemma 4 ([11]). *Fix subset $S \subset T \subset \mathbb{F}$, and $g \in \mathbb{F}_d[X]$. Then $Z_S(X) \mid g(X)$ iff $Z_T(X) \mid Z_{T\setminus S}(X) \cdot g(X)$.*

Let $\mathcal{A}$ be an algebraic adversary. $\mathcal{A}$ begins by outputting commitments $\mathsf{cm}_1, \ldots, \mathsf{cm}_n \in \mathbb{G}_1$. In the AGM, each cm_k is a linear combination of $\sum_{i=0}^{m-1}\sum_{j=0}^{\ell-1} f_{k,i,j}[\sigma^i\tau^j]_1$. The extractor $\mathcal{E}$, given the coefficients of $\{f_{k,i,j}\}$ for some k, outputs a polynomial $f_k(X, Y) =$

Protocol 6 (Batch bivariate KZG)**.**

1. $\mathsf{Gen}(m,\ell)$*: for* $\sigma,\tau \in \mathbb{F}$ *probably generated by ceremony [8], output* $\mathsf{srs} = (\{[\sigma^{i-1}\tau^{j-1}]_1\}_{i\in[m],j\in[\ell]}, [1]_1, [1]_2, [\sigma]_2, [\tau]_2)$.
2. $\mathsf{Com}(\mathsf{srs}, \{f_k(X,Y)\})$*: Given* $f_k \in \mathbb{F}_{m,\ell}[X,Y]$*, compute the commitment* $\mathsf{cm}_k = [f_k(\sigma,\tau)]_1$ *for* $k \in [n]$.

$\mathsf{Open}(\{\mathsf{cm}_k\}, \{R_k\}, \{S_k\}, \{f_k(\alpha,\beta)_{\alpha\in R_k,\beta\in S_k}\})$:

1. $\mathcal{P}$ *sends commitments to* $\{s_k\}$*, i.e.,* cm_{s_k} *to* $\mathcal{V}$.
   (a) *Compute* $\{r_k\}_{k\in[n]} \in \mathbb{F}_{t,t}[X,Y]$ *such that* $r_k(\alpha,\beta) = s_k(X,\beta)$ *for all* $\alpha \in R_k$ *and* $\beta \in S_k$.
   (b) *Compute* $\{s_k\}_{k\in[n]} \in \mathbb{F}_{t,\ell}[X,Y]$ *such that* $s_k(\alpha,Y) = f_k(\alpha,Y)$ *for any* $\alpha \in R_k$*. Compute commitments* $\{s_k[\sigma,\tau]_1\}$ *for* $k \in [n]$.
2. $\mathcal{V}$*: sends a random challenge* $\gamma \in \mathbb{F}$ *to* $\mathcal{P}$.
3. $\mathcal{P}$*: computes and sends* U_1, U_2 *to* $\mathcal{V}$*. Specifically,*
   (a) *For* $k \in [n]$*, compute bivariate polynomials* $P_k(X,Y)$, $Q_k(X,Y)$ *s.t.* $P_k = f_k - s_k$ *and* $Q_k = s_k - r_k$.
   (b) *Set* $T_1 = \bigcup_{k\in[n]} R_k$*. Set* $T_2 = \bigcup_{k\in[n]} S_k$*. Compute* $P(X,Y) = \sum_{k\in[n]} \gamma^{k-1} \cdot Z_{T_1\setminus R_k}(X) \cdot P_k(X,Y)$*, and* $Q(X,Y) = \sum_{k\in[n]} \gamma^{k-1} \cdot Z_{T_2\setminus S_k}(Y) \cdot Q_k(X,Y)$. *By construction, we have* $Z_{R_k}(X) \mid P_k(X,Y)$*, and hence* $Z_{T_1}(X) \mid P(X,Y)$*. Similarly, we have* $Z_{T_2}(Y) \mid Q(X,Y)$.
   (c) *Compute* $p(X,Y) = P(X,Y)/Z_{T_1}(X)$ *and* $q(X,Y) = Q(X,Y)/Z_{T_2}(Y)$*. Compute* $U_1 = [p(\sigma,\tau)]_1, U_2 = [q(\sigma,\tau)]_1$.
4. $\mathcal{V}$*: sends random challenges* $z_1, z_2 \in \mathbb{F}$ *to* $\mathcal{P}$.
5. $\mathcal{P}$ *computes and sends* V_1 *and* V_2 *to* $\mathcal{V}$.
   (a) *Compute* $W_1 \in \mathbb{F}_{m,\ell}[X,Y], W_2 \in \mathbb{F}_{t,\ell}[X,Y]$ *s.t.*

$$W_1 = \sum_{k\in[n]} \gamma^{k-1} \cdot Z_{T_1\setminus R_k}(z_1) \cdot (f_k - s_k) - Z_{T_1}(z_1) \cdot p,$$

$$W_2 = \sum_{k\in[n]} \gamma^{k-1} \cdot Z_{T_2\setminus S_k}(z_2) \cdot (s_k - r_k(z_1,z_2)) - Z_{T_2}(z_2) \cdot q.$$

   By definition, we have $(X - z_1) \mid W_1$ *and* $(Y - z_2) \mid W_2$.
   (b) *Compute* $W_1'(X,Y) = W_1(X,Y)/(X - z_1)$ *and* $W_2'(X,Y) = W_2(X,Y)/(X - z_2)$*. Compute* $V_1 = [W_1'(\sigma,\tau)]_1, V_2 = [W_2'(\sigma,\tau)]_1$.
6. $\mathcal{V}$*: computes* $r_k(X,Y)$ *for* $k \in [n]$*. Compute* F_1 *and* F_2 *s.t.*

$$F_1 = Z_{T_1\setminus R_k}(z_1) \cdot \Big(\sum_{k\in[n]} \gamma^{k-1}(\mathsf{cm}_k - \mathsf{cm}_{s_k}) - Z_{T_1}(z_1) \cdot U_1\Big),$$

$$F_2 = \sum_{k\in[k]} \gamma^{k-1} \cdot Z_{T_2\setminus S_k}(z_2)(\mathsf{cm}_{s_k} - [r_k(z_1,z_2)]_1) - Z_{T_2}(z_2) \cdot U_2.$$

   Accept iff

$$e(F_1, [1]_2) = e(V_1, [\sigma - z_1]_2) \wedge e(F_2, [1]_2) = e(V_2, [\tau - z_2]_2).$$

$\sum_{i=0}^{m-1}\sum_{j=0}^{\ell-1} f_{k,i,j}X^iY^j$. $\mathcal{A}$ also outputs $R_1,\ldots,R_n, S_1,\ldots S_n, T_1, T_2$, and polynomials $r_1,\ldots,r_n$.

Assume for some $k^* \in [n]$, we have $Z_{R_{k^*}} \nmid f_{k^*} - s_{k^*}$ or $Z_{S_{k^*}} \nmid s_{k^*} - r_{k^*}$. The assumption is feasible as otherwise we have for all $k \in [n]$, there exists p_k and q_k s.t. $f_k - r_k = p_k \cdot Z_{R_k}(X) + q_k \cdot Z_{S_k}(Y)$. WLOG, assume $Z_{R_{k^*}} \nmid f_{k^*} - s_{k^*}$ and $Z_{S_{k^*}} \mid s_{k^*} - r_{k^*}$. Other cases are similar. Then, we know from Lemma 4 that $Z_{T_1\setminus R_{k^*}}(X)(f_{k^*} - s_{k^*})$ is not divisible by $Z_{T_1}(X)$. Let

$$f(X,Y) = \sum_{k\in[n]} \gamma^{k-1} Z_{T_1\setminus R_k}(X) \cdot (f_k(X,Y) - s_k(X,Y)).$$

Using Lemma 3, we know that except with probability $n/|\mathbb{F}|$ over γ, f is not divisible by $Z_{T_1}(X)$. Now $\mathcal{A}$ outputs $U_1 = [p'(\sigma,\tau)]_1, U_2 = [q'(\sigma,\tau)]_1$ for some $p', q' \in \mathbb{F}_{m,\ell}[X,Y]$ followed by $\mathcal{V}$ sending uniform $z_1, z_2 \in \mathbb{F}$. Since f is not divisible by $Z_{T_1}(X)$, for any p', there are at most $2m$ values of $z_1 \in \mathbb{F}$ s.t. $f(z_1,Y) = p'(z_1,Y)Z_{T_1}(z_1)$; and thus z_1 of this form chosen by $\mathcal{V}$ is with probability of $\mathsf{negl}(n)(\lambda)$.

Assume z_1 is not in this form. $\mathcal{P}$ now outputs $W_1' = [W_1(\sigma,\tau)]_1$ and W_2'. According to [11, Lemma 2.2], it suffices to upper bound the probability that the ideal check corresponding to the real pairing check in Step 6. Denoting $W_1^*(X,Y) =$

$$\sum_{k\in[n]} \gamma^{k-1} \cdot Z_{T_1\setminus R_k}(z_1) \cdot (f_k(X,Y) - s_k(X,Y)) - Z_{T_1}(z_1) \cdot p'(X,Y).$$

The ideal check has the form $W_1^*(X,Y) = H(X,Y)(X - z_1)$ for the existence of $H(X,Y)$. It can pass iff $W_1^*(z_1,Y) = 0$, which means $W_1^*(z_1,Y) = f(z_1,Y) - p'(z_1,Y) \cdot Z_{T_1}(z_1) = 0$. This a contradiction.

The proof for $Z_{S_{k^*}} \nmid s_{k^*} - r_{k^*}$ is similar by setting $s_k(X,Y)$ as $r_k(z_1,z_2)$ in $f_k(X,Y)$ corresponding to variable Y, and we omit it here. In summary, the ideal check can only pass with $\mathsf{negl}(n)(\lambda)$ probability over the randomness of $\mathcal{V}$, which implies the same thing for the real pairing check.

Complexity. The srs consists of $O(m\ell)$ group elements. The commitment generation requires $O(n \cdot m\ell)$ group operations.

For the Open protocol, the computation of all s_k costs $O(n \cdot t\ell)$ field operations in total. The computation of all $s_k[\sigma,\tau]_1$ costs $O(nt\ell)$ group operations. The computation of $p \in \mathbb{F}_{m\ell}[X,Y]$ and $q \in \mathbb{F}_{t,\ell}[X,Y]$ costs at most $O(nm\ell + nt\ell)$ field operations as each polynomial is a linear combination of n polynomials. The computation of U_1 and U_2 costs $O(m\ell + t\ell)$ group operations. The computation of $W_1(X,Y)$ costs $O(nm\ell)$ field operations. The computation of V_1 costs $O(m\ell)$ group operations. Similarly, the computation of $W_2(X,Y)$ and V_2 costs $O(nt\ell)$ field operations and $O(t\ell)$ group operations. Hence, the total prover complexity is $O(nm\ell + nt\ell)$ field operations $O(m\ell + nt\ell)$ group operations. In contrast, the trivial prover complexity is $O(t^2n \cdot m\ell)$ group operations. Note that n and t are usually small constants in SNARKs [17] and also in our case.

The proof size is $n + 4$ group elements, in contrast to the trivial $2t^2n$ group elements. The verifier complexity includes $O(n)$ group operations due to computing F_1 and F_2, $O(n)$ group operations for all $[r_k(z_1,z_2)]_1$, and 4 pairings. Hence, the total verifier complexity is $O(n)$ group operations plus 4 pairings. In contrast, the trivial verifier complexity is $O(t^2n)$ pairings. □

Reducing proof size with the same evaluation point over Y. The evaluations points on bivariate polynomials in Protocol 3 share the same $Y = \beta$. To further reduce the proof size, we propose a PCS variant specified for same Y-dimension points in Protocol 7. We present the batch bivariate KZG variant with the same evaluation points over Y in Protocol 7. Below we discuss the high-level idea.

Protocol 7 (Batch bivariate KZG evaluated at a single $Y = \beta$).

1. *Output* $\mathsf{srs} = (\{[\sigma^{i-1}\tau^{j-1}]_1\}_{i\in[m],j\in[\ell]}, [1]_2, [\sigma]_2, [\tau]_2)$ *for uniform* σ, τ.
2. $\mathsf{cm}_k = [f_k(\sigma,\tau)]_1$ *for each* $k \in [n]$.

$\mathsf{Open}(\{\mathsf{cm}_k\}, \{R_k\}, \beta; \{f_k(\alpha,\beta)_{\alpha\in R_k}\})$*:*

1. $\mathcal{V}$*: sends to* $\mathcal{P}$ *a random challenge* $\gamma \in \mathbb{F}$.
2. $\mathcal{P}$*: computes and sends* $Q = [q(\sigma)]_1$ *to* $\mathcal{V}$ *for* $q \in \mathbb{F}_{m+|T_1|}[X]$ *obtained from Equation* (32).
3. $\mathcal{V}$*: sends to* $\mathcal{P}$ *a random challenge* $\eta \in \mathbb{F}$.
4. *(a)* $\mathcal{P}$*: sends* $f_1(\eta,\beta), \ldots, f_n(\eta,\beta)$ *and* $q(\eta)$ *to* $\mathcal{V}$.
   (b) $\mathcal{V}$*: sends to* $\mathcal{P}$ *a random challenge* $\theta \in \mathbb{F}$.
   (c) $\mathcal{P}$*: sends to* $\mathcal{V}$ *the* $Q_1 = [q_1(\sigma,\tau)]_1, Q_2 = [q_2(\sigma,\tau)]_1, Q_3 = [q_3(\sigma)]_1$*, where* $q(X) - q(\eta) = q_3(X)\cdot(X-\eta)$*, and*

$$\sum_{k\in[n]} \theta^{k-1}\cdot(f_k(X,Y) - f_k(\eta,\beta)) = q_1(X)\cdot(X-\eta) + q_2(Y)\cdot(Y-\beta).$$

   (d) $\mathcal{V}$*: accepts iff:*
      (i) *Equation* (32) *holds when* $X = \eta$*;*
      (ii) $e(Q_3, [\sigma-\eta]_2) = e(Q - [q(\eta)]_1, [1]_2)$*;*
      (iii) $e(Q_1, [\sigma-\eta]_2)\cdot e(Q_2, [\tau-\eta]_2) = e(\sum_{k\in[n]}\theta^{i-1}(\mathsf{cm}_k - [f_k(\eta,\beta)]_1), [1]_2)$.

For any bivariate polynomial f_k and points $R_k \subset T_1$ and $S_k = \{\beta\}$, there exists univariate polynomials $r_i \in \mathbb{F}_k[X]$ and q_i s.t.

$$f_k(X,\beta) - r_k(X) = q_k(X)\cdot Z_{R_k}(X), \tag{31}$$

where r_k is publicly defined by $\{(\alpha, f_k(\alpha,\beta))\}$. For multiple $f_1, \ldots, f_n$, Equation (31) can be batched via random linear combination of γ

$$\sum_{k\in[n]} \gamma^{k-1}\cdot(f_k(X,\beta) - r_k(X))\cdot Z_{T_1\setminus R_k}(X) = q(X)Z_{T_1}(X). \tag{32}$$

To prove it, the prover sends the commitment to q, and the verifier opens $\{f_k(\eta,\beta)\}_{k\in[n]}, q(\eta)$ for a randomly picked η, and checks if Equation (32) holds when $X = \eta$. We still need to open evaluations on each polynomial, but they are evaluated on the same points and simple to batch via random linear combination. The proof size is reduced to 4 $\mathbb{G}_1$ elements and $n+1$ $\mathbb{F}$ elements. Note that group elements for bilinear pairing can be 4× larger than field ones.

Theorem 6. *Protocol 7 is a batch PCS as in Definition 2. Opening* t *points on each of the* n *bivariate polynomials with* X*-degree of* m *and* Y*-degree of* ℓ*, the prover complexity is* $O(t + m\ell)$ *group operations. The proof size is* 4 *group elements plus* $n+1$ *field elements. The verifier complexity is* $O(n)$ *group operations plus* 5 *pairings.*

Proof. Completeness holds by design. We argue the knowledge soundness in AGM below. Let $\mathcal{A}$ be such an algebraic adversary. $\mathcal{A}$ begins by outputting $\mathsf{cm}_1, \ldots, \mathsf{cm}_n$, and each cm_k is a linear combination $\sum_{i=0}^{m-1}\sum_{j=0}^{\ell-1} f_{k,i,j}[\sigma^i\tau^j]_1$. $\mathcal{E}$, given the coefficients of $\{f_{k,i,j}\}$ for some k, outputs a polynomial $f_k(X,Y) = \sum_{i=0}^{m-1}\sum_{j=0}^{\ell-1} f_{k,i,j}X^iY^j$. $\mathcal{A}$ also outputs $R_1, \ldots, R_n, \beta, T_1$.

Assume for some $k^* \in [n]$, we have $Z_{R_k} \nmid f_{k^*} - r_{k^*}$, which means there exists some $\alpha \in R_k$, $f_{k^*}(\alpha) \neq r_{k^*}(\alpha)$. We know from Lemma 4 that $Z_{T_1\setminus R_{k^*}}\cdot(f_{k^*} - r_{k^*})$ is not divisible by Z_{T_1}. Let

$$f(X) = \sum_{k\in[n]} \theta^{k-1}(f_k(X,\beta) - r_k(X))\cdot Z_{T_1\setminus R_k}(X).$$

By Lemma 3, except with probability $n/|\mathbb{F}|$ over γ, f is not divisible by $Z_{T_1}(X)$. Now $\mathcal{A}$ outputs $Q = [q'(\sigma)]$ for some $q' \in \mathbb{F}_m[X]$. Given a random $\eta \in \mathbb{F}$, for any q' there are at most $2m$ values of $\eta \in \mathbb{F}$ s.t. $f(\eta) = q(\eta)Z_{T_1}(\eta)$. Thus, picking a η of this form for $\mathcal{V}$ is of probability $\mathsf{negl}(n)(\lambda)$.

Assume we are not in this case. Assume that for some $k^* \in [n]$, $f_{k^*}(\eta,\beta)$ does not equal the claimed $f_k(\eta,\beta)$. By the randomness of θ, with a probability of $n/|\mathbb{F}|$, $\sum_{k\in[n]}\theta^{k-1}f_k(X,Y) - \theta^{k-1}f_k(\eta,\beta) \neq 0$. If the ideal check passes, it shows the existence of q_1, q_2, which means that $\sum_{k\in[n]}\theta^{k-1}f_k(X,Y) - \theta^{k-1}f_k(\eta,\beta) = 0$. This is a contradiction. Thus the ideal check can only pass with probability $\mathsf{negl}(n)(\lambda)$, which implies the same thing for the real check.

Complexity. The srs and commitment generation are the same as Protocol 6. Open requires $O(t + m\ell)$ group operations due to the degree bound of q, q_1, q_2 and q_3. The proof size is 4 $\mathbb{G}_1$ + $(n+1)$ $\mathbb{F}$ elements, in contrast to $n+4$ $\mathbb{G}_1$ elements in Protocol 6. The verifier complexity is n group operations and 5 pairings. □

5.2 Distributed Batch Bivariate PCS

To build a distributed batch bivariate PCS, we recall the idea in Pianist [42]. In the distributed setting, the prover $\mathcal{P}_i$ knows f_i and oracles $\{X^{j-1}\cdot L_i(Y)\}_{j\in[m]}$ s.t. $f(X,Y) = \sum_{i\in[\ell]} f_i(X)L_i(Y)$. To compute the quotient bivariate polynomials p, q distributedly, Pianist sets $p(X,Y) = (f(X,Y) - f(\alpha,Y))/(X-\alpha)$ and $q(X,Y) = (f(\alpha,Y) - f(\alpha,\beta))/(Y-\beta)$. By definition, q is only related to variable Y, and can be computed by $\mathcal{P}_0$ locally in $O(\ell)$ time. Further, p satisfies $p(X,Y) = \sum_{i\in[\ell]}\frac{f_i(X)-f_i(\alpha)}{X-\alpha}\cdot L_i(Y)$. Then, p can be computed by $\mathcal{P}_i$ computing $\frac{f_i(X)-f_i(\alpha)}{X-\alpha}\cdot L_i(Y)$ and $\mathcal{P}_0$ adding them.

We generalize Equation (28) into a distributed version. We still use $s \in \mathbb{F}_{|R|=t,\ell}[X,Y]$, the auxiliary polynomial in Protocol 6, to construct quotient polynomials p and q. As $s(\alpha,Y) = f(\alpha,Y)$ for all $\alpha \in R$, it can be written as $s(X,Y) = \sum_{j\in[|R|]} f(\alpha_j,Y)\cdot N_j(X)$, where $N_j \in \mathbb{F}_{|R|}[X]$ is the Lagrange polynomial corresponding to α_j. It satisfies $N_j(\alpha_j) = 1$ while for other $k \neq j, k \in R$, it holds that $N_j(\alpha_k) = 0$. In our case, $|R|$ is a small constant.

By the properties of N_j, we have $\sum_{j\in[|R|]} N_j(X) = 1$, and further

$$\begin{aligned} p(X,Y)\cdot Z_R(X) &= \sum_{j\in[|R|]}(f(X,Y) - f(\alpha_j,Y))\cdot N_j(X) \\ &= \sum_{j\in[|R|]}\Big(\sum_{i\in[\ell]}(f_i(X) - f_i(\alpha_j))\cdot L_i(Y)\Big)\cdot N_j(X) \\ &= \sum_{i\in[\ell]}\Big(\sum_{j\in[|R|]}(f_i(X) - f_i(\alpha_j))\cdot N_j(X)\Big)\cdot L_i(Y). \end{aligned}$$

As $(X-\alpha_j) \mid f_i(X) - f_i(\alpha_j)$ for any $i \in [\ell]$ and $Z_{R\setminus\alpha_j}(X) \mid N_j(X)$, it holds that $Z_R(X) \mid \sum_{j\in[|R|]}(f_i(X) - f_i(\alpha_j))\cdot N_j(X)$ for $Z_R(X) = (X-\alpha_j)\cdot Z_{R\setminus\alpha_j}(X)$. Also, the polynomial $\sum_{j\in[|R|]}(f_i(X) - f_i(\alpha_j))\cdot N_j(X)\cdot 1/Z_R(X)$ can be computed by $\mathcal{P}_i$ locally. Hence, $p(X,Y)$ can still be computed distributedly. Further, $q \in \mathbb{F}_{|R|,\ell-1}[X,Y]$ can be computed by $\mathcal{P}_0$ herself in $O(|R| + \ell)$ time.

Protocol 8 (Distributed batch bivariate KZG). *Suppose ℓ sub-provers $\mathcal{P}_1, \ldots, \mathcal{P}_\ell$ and master $\mathcal{P}_0$. Given $\{f_k\}_{k\in[n]} \in \mathbb{F}_{m,\ell}[X,Y]$ s.t. $f_k(X,Y) = \sum_{i\in[\ell]} f_{k,i}(X)L_i(Y)$, $\mathcal{P}_i$ holds $\{f_{k,i}\}_{k\in[n]}$.*

1. $\mathsf{Gen}(m,\ell)$*: choose the secret* $\sigma, \tau \in \mathbb{F}$. *Output* $\mathsf{srs} = (\{[\sigma^{h-1}L_j(\tau)]_1\}_{h\in[m],j\in[\ell]}, [1]_2, [\sigma]_2, [\tau]_2)$.
2. $\mathsf{DKZG.Com}(\{f_k\}, \mathsf{srs})$*: For* $k \in [n]$, $\mathcal{P}_i$ *sends to* $\mathcal{P}_0$ $\mathsf{cm}_{f_{k,i}} = [f_{k,i}(\sigma)L_j(\tau)]_1$, *who then computes* $\mathsf{cm}_k = \prod_{i\in[\ell]} \mathsf{cm}_{f_{k,i}}$.

$\mathsf{DKZG.Open}(\{\mathsf{cm}_k\}, \{R_k\}, \{S_k\}, \{f_k(\alpha,\beta)_{\alpha\in R_k,\beta\in S_k}\})$:

1. *Provers compute and sends* $\{\mathsf{cm}_{s_k}\}_{k\in[n]}$ *to* $\mathcal{V}$.
   (a) $\mathcal{P}_i$*: for all* $k \in [n]$, *computes* $s_{k,i}(X)$, $\{s_{k,i}(\alpha)\}_{\alpha\in R_k}$ *for* $s_{k,i} \in \mathbb{F}_{|R_k|}[X]$ *s.t. for* $\alpha \in R_k$, $s_{k,i}(\alpha) = f_{k,i}(\alpha)$. *Compute* $\mathsf{cm}_{s_{k,i}} = [s_{k,i}(\sigma)L_i(\tau)]_1$. *Send* $\mathsf{cm}_{s_{k,i}}$ *to* $\mathcal{P}_0$.
   (b) $\mathcal{P}_0$*: computes* $\mathsf{cm}_{s_k} = [s_k(\sigma,\tau)]_1 = \prod_{i\in[\ell]} \mathsf{cm}_{s_{k,i}}$.
2. $\mathcal{V}$*: sends a random challenge* $\gamma \in \mathbb{F}$ *to* $\mathcal{P}_0$, *who transfers to* $\mathcal{P}_i$.
3. *Provers compute* U_1, U_2 *as in Step 3., Protocol 6.*
   (a) $\mathcal{P}_i$*: computes* $p_i(X) = P_i(X)/Z_{T_1}(X)$ *and*
   $$P_i(X) = \sum_{k\in[n]} \gamma^{k-1} \cdot Z_{T_1\setminus R_k}(X) \cdot (f_{k,i}(X) - s_{k,i}(X)).$$
   Send $[p_i(\sigma)L_i(\tau)]_1$ *to* $\mathcal{P}_0$.
   (b) $\mathcal{P}_0$*: computes* $U_1 = \prod_{i\in[\ell]} p_i(\sigma)L_i(\tau)]_1$. *Compute* $q(X,Y)$ *and* U_2 *herself the same as Step 3., Protocol 6.*
4. $\mathcal{V}$*: sends challenges* $z_1, z_2 \in \mathbb{F}$ *to* $\mathcal{P}_0$, *who transfers to* $\mathcal{P}_i$.
5. *Provers compute and send* V_1 *and* V_2 *to* $\mathcal{V}$.
   (a) $\mathcal{P}_i$*: computes* $W'_{1,i}(X) = W_{1,i}(X)/(X - z_1)$ *where*
   $$W_{1,i}(X) = \sum_{k\in[n]} \gamma^{k-1} \cdot Z_{T_1\setminus R_k}(z_1) \cdot (f_{k,i} - s_{k,i} - Z_{T_1}(z_1) \cdot P_i.$$
   Send $[W'_{1,i}(\sigma)L_j(\tau)]_1$ *to* $\mathcal{P}_0$.
   (b) $\mathcal{P}_0$*: computes* $V_1 = \prod_{i\in[\ell]}[W_{1,i}(\sigma)L_i(\tau)]_1$. *Compute* $W_2(X,Y)$ *and* V_2 *according to Step 5., Protocol 6.*
6. $\mathcal{V}$*: acts the same as Step 6., Protocol 6.*

With the high-level idea above, we can build a distributed batch bivariate KZG, which is presented in Protocol 8. To adapt to our distributed PIOP for R1CS, we modify the $\{[\tau^{j-1}]_1\}_{j\in[\ell]}$ in srs as $\{[L_j(\tau)]_1\}_{j\in[\ell]}$.

Theorem 7. *Protocol 8 is a distributed batch bivariate PCS. The prover complexity of $\mathcal{P}_i$ is $O(m)$ group and $O(nm+nt)$ field operations. The prover complexity of $\mathcal{P}_0$ is $O(\ell t)$ group and $O(\ell nt)$ field operations. The communication per prover is $O(n)$ group elements. The proof size and verifier complexity remain with Theorem 5.*

Proof. The security of Protocol 8 follows directly from Protocol 6 assuming each sub-prover is honest.

Complexity. The proof size is $n+4$ group elements, the same as the non-distributed Protocol 6. The communication per prover is $O(n)$ group elements for sending $\mathsf{cm}_{s_{k,i}}$ for all $k \in [n]$. Other communication per prover is $O(1)$ for sending $[p_i(\sigma)L_i(\tau)]_1$ and $[W'_{1,i}(\sigma)L_i(\tau)]_1$. The verifier complexity is $O(n)$ group operations and 4 pairings, the same as the non-distributed Protocol 6. These hold directly by count.

Assume $|R_k| = |S_k| = t$ for all $k \in [n]$. For prover $\mathcal{P}_i$, the computation of $\{s_{k,i}(\alpha)\}_{\alpha\in R_k}$ costs $O(tn)$ field operations given evaluations $f_{k,i}(\alpha)$. For n polynomials, this costs $O(n^2t)$ field operations. Computing $s_{k,i}(X)$ from polynomial interpolation costs $O(t)$ field operations. For n polynomials, this costs $O(nt)$ field operations. Computing $\{\mathsf{cm}_{s_{k,i}}\}_{k\in[n]}$ needs $O(nt)$ group operations. Computing $p_i(X)$ and $[p_i(\sigma)L_i(\tau)]_1$ costs $O(nm)$ field and $O(m)$ group operations, respectively. This is because $p_i(X)$ is the linear combination of n polynomials with degrees of at most m. Computing $W'_{1,i}(X)$ and $[W_{1,i}(\sigma)L_i(\tau)]_1$ costs $O(nm)$ field and $O(m)$ group operations, respectively. Hence, the sub-prover complexity is $O(nm+nt)$ field operations plus $O(m)$ group operations.

For $\mathcal{P}_0$, computing $q(X,Y)$ costs $O(t\ell)$ field operations. Computing U_2 costs $O(t\ell)$ group operations. Computing $W'_2(X,Y)$ need $O(nt\ell)$ field operations. Computing V_2 costs $O(t\ell)$ group operations. Hence, the master prover complexity is $O(nt\ell)$ field operations plus $O(t\ell)$ group operations.

The complexities show a good scalability of our distributed batch KZG, as the sub-prover complexity is non-related to ℓ and the master prover complexity is non-related to m. □

Similar to how Protocol 8 generalizes Protocol 6, Protocol 7 can be extended into a distributed scheme. We omit the details here.

5.3 Putting Everything Together

Combining our distributed PIOP for R1CS and the distributed batch bivariate KZG, we have the following theorem.

Theorem 8. *There exists a distributed SNARK. For size-$O(m\ell)$ R1CS utilized with ℓ provers, the sub-prover complexity is $O(m)$ group operations plus $O(m\log m)$ field operations. Choosing one sub-prover as the master prover, its time complexity is $O(m+\ell)$ group plus $O(m\log m + \ell\log\ell)$ field operations. The communication complexity per prover is $O(1)$. The verifier complexity can be $O(1)$.*

We provide a proof sketch here. Completeness holds directly. Complexities are implied by Theorem 4 and Theorem 7. For knowledge soundness, it has been proven that if a PCS satisfies knowledge soundness and a PIOP has soundness, then the compiled interactive argument from "PIOP + PCS" inherits the knowledge soundness [14, 40]. The security proof hence follows from the soundness of Protocol 5 and the knowledge soundness of Protocol 8.

6 IMPLEMENTATION AND EVALUATION

We implement Soloist using ~10,000 lines based on the arkworks library [4] in Rust. All experiments are run on machines with 16 cores and 256GB of RAM. For distributed schemes, we open 2-32 machines belonging to one cluster, where each machine acts as a prover. Reported figures are averages over 10 executions.

6.1 Evaluation of IPA and Batch Bivariate KZG

Performance of IPAs. We implement IPAs in Marlin [18], Dark [14], and our scheme by instantiating corresponding PIOPs with the batch KZG of Boneh *et al.* [11]. Figure 3 shows that our IPA has a

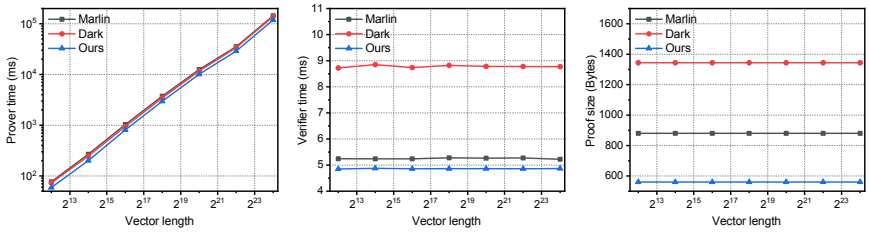

Figure 3: Performance of three inner product arguments from "PIOP + PCS" with constant proof size

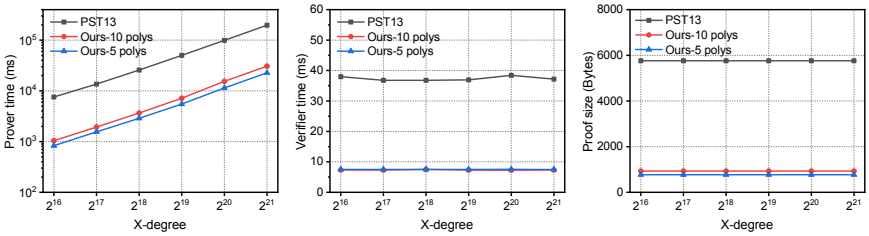

Figure 4: Performance of batch bivariate KZG with random X-dimension evaluation points and the same Y-dimension point

20% faster prover, 5%-80% faster verifier, and 30%-2× smaller proof size. This benefits from the superior complexities shown in Table 2.

There are also other potential IPAs from "PIOP + PCS" with constant proof size, such as Count [41], SZ22 [55], and Dew [5]. Due to the lack of open-sourced implementations, we compare them in theory. Count is an improved univariate sum-check and can be used to construct IPAs with smaller proof size than Marlin. However, it requires an SRS at least twice the size of vectors, incurring additional overhead due to SRS generation and additional low-degree tests. Similarly, SZ22, an IPA from Laurent polynomials, reduces a polynomial oracle over Dark but sacrifices the SRS size to twice of the vectors. Dew is a trustless PCS with constant proof size, but its security relies on the generic group model, which is viewed as stronger than our algebraic group model [24].

Performance of batch bivariate KZG. Figure 4 shows the performance of our batch bivariate KZG in Protocol 7 with the same evaluation point over Y, which is used in Soloist. We test opening 20 random points, 2 on each of 10 polynomials, or 4 on each of 5 polynomials, with the Y-degree as 7. These parameters are chosen or generalized from Soloist. Compared with PST13 [46] for directly opening all points, ours has a 7-10× faster prover time, 5× faster verifier time, and a 6-7× smaller proof size. When opening more points on one polynomial, our scheme would feature a smaller proof size, which is linear to the polynomial number and non-related to the point number on each polynomial.

6.2 Evaluation of Distributed SNARKs

We benchmark SNARKs on different circuits, including random R1CS, zkRollup, and ECDSA verification. The tested performance includes indexer time, prover time, verifier time, proof size, and memory costs. The indexer time is the running time of the offline and circuit-specific preprocessing phase. In the online phase, the provers and verifier would take some outputs of the preprocessing phase as inputs. The prover time is the slowest running time of all provers, and we choose one sub-prover as the master prover.

Table 4: Memory costs (GBs) of SNARKs for size-2^{25} R1CS

#Prover	Costs	#Prover	Costs	#Prover	Costs
Marlin-1	246.1	Soloist-2	103.9	Soloist-4	51.9
Soloist-8	26.2	Soloist-16	13.2	Soloist-32	6.9

Performance on R1CS. Figure 5 and Table 4 compares Soloist and the non-distributed Marlin [44] on random R1CS adapted from Marlin. The elliptic curve is BLS12-381. We choose the R1CS-targeted Marlin due to its similarity with Soloist like the constant proof size, a universal and updatable SRS, and constant verifier complexity.

Soloist shows good scalability. Its indexer time, prover time, and memory costs decrease linearly with the increase of sub-prover number. When utilizing ℓ provers, they are $1.5\ell\times$ faster, $\ell\times$ faster, and $\ell\times$ smaller than Marlin, respectively. For a size-2^{25} R1CS, Soloist costs 29s for proving with 32 provers, using 512 cores in total. Note that the prover time is end-to-end assuming that each prover holds the sub-witness of R1CS. The verifier time is 4.6ms, slightly faster than Marlin's 6.5ms. The communication per prover is 8KB. Soloist has a 13KB and 20× larger proof size than Marlin. However, it is still competitive among distributed SNARKs as shown below.

Comparisons with other distributed SNARKs. We do not compare DIZK, where a non-distributed prover time for a size-2^{20} R1CS is over 2,000s [59], which is 60× slower than Marlin and ours. We fail to compare with Hekaton thoroughly as we do not find any open-sourced implementation. Still, we can reference the results as both schemes are implemented using the arkworks library over the BLS12-381 curve. Different from [39, 42, 60], Hekaton counts the prover time using the number of total cores instead of the number of machines, where each core (instead of each machine) acts as

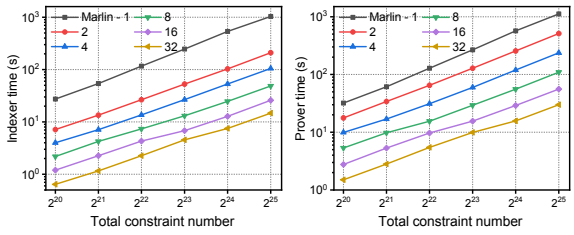

Figure 5: Comparison of non-distributed Marlin with 1 prover and Soloist with 2-32 provers for proving R1CS

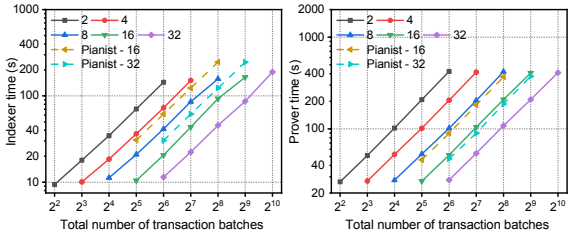

Figure 6: Comparison of Pianist with 16 or 32 provers and Soloist with 2-32 provers for zkRollups (in batches of 3)

a sub-prover assigned with a sub-circuit. Hekaton's prover time, utilized with 4096 cores, is 16s for circuits with 2^{29} constraints. Figure 5 has shown the scalability of Soloist; hence, our prover time in this setting is estimated to be about 58s, *i.e.*, 3.6× slower.

However, Hekaton's prover time can be sensitive to the circuit structure, especially the shared wire number of sub-circuits. The tested circuit in Hekaton has 10% constraints from shared wires, which may vary in practical applications. Cirrus [58] runs Hekaton over Pedersen hash function and SHA-256. Given 256 cores, the prover times of Hekaton for size-2^{25} circuits are 32s or 128s, instead of the estimated 16s. Then, Soloist is 2× slower to 2× faster.

Besides, Hekaton's proof size and verification are logarithmic to the sub-prover count, while Soloist's are $O(1)$. The only reported data in Hekaton indicates a 32KB proof size and an 83ms verifier time, which are 3× larger and 20× slower than ours. Also, Hekaton incurs a communication overhead of 900KB per prover, 100× larger than ours. Further, Hekaton needs a circuit-specific setup, while Soloist only requires a one-time universal setup like [42, 58]. To sum up, Soloist and Hekaton can be comparable in prover time, but ours has advantages in all of the other metrics.

Hekaton claims a 3× faster prover time advantage than Pianist. Hence, Soloist can be slower than Pianist in prover. This is probably due to Soloist involving more polynomial oracles, especially the lookup PIOP (Protocol 4) for preprocessing. Despite this, there are various practical applications more amenable to R1CS. For example, VKH23 [57], a verifiable fully homomorphic encryption scheme, shows a clear preference for R1CS over Plonk for efficiency. Groth16-based schemes like the R1SC0 ZKVM [50] can not use SNARKs for Plonk as Groth16 does not support Plonk. Reef [3] and zkPoT [1] choose R1CS due to the reliance for a specific SNARK [36]. To further demonstrate this, below we benchmark Soloist on proving zkRollups originally encoded as R1CS, same as in Pianist.

Performance of zkRollups. We extract the zkRollup circuit from Hermez [34], use Circom [21] to compile it into R1CS, and then use the compiler in Pianist to transform the R1CS into Plonk. The R1CS has 229, 847 constraints per batch of 3 transactions, and Plonk has 2, 215, 685 constraints, both in the same order as Pianist.[4]

Figure 6 shows the indexer and prover times of Soloist for proving zkRollups with 2-32 provers, compared with Pianist with 16 or 32 provers. The elliptic curve is BN254. Each prover in Pianist can handle up to 16 batches of transactions, as the constraint number would exceed the largest multiplicative subgroup size in the finite field of BN254 curve. In contrast, Soloist can handle up to 320 batches of transactions per prover given sufficiently large memory. For concrete efficiency, the indexer and prover times of Soloist are 2.8× and 1.8× faster than Pianist, respectively. We stress that this does not count the transformation time from R1CS to Plonk. For other metrics, our verifier time is 3ms, competitive with Pianist's 2.8ms. The proof size is 10.2KB, 4.5× larger than Pianist's 2.2KB. The communication per prover is 6.2KB, 3× larger than Pianist's 2.1KB. These constant-size overheads can be reduced in an amortized manner, as Soloist can handle more transactions.

[4]Pianist claims to use an R1CS with 86k constraints per transaction, but we only find an R1CS with 261k constraints without a proper witness [48]. Thus, we fail to reuse it.

Table 5: Memory costs (GBs) for zkRollups, where Per #Tx means the number of transaction batches per sub-prover

Per #Tx	Soloist	Pianist	Per #Tx	Soloist	Pianist
2	12.1	30	8	47	115
4	23.8	60.8	16	93	218

Table 6: Performance for non-data-parallel zkRollups

$\#\mathcal{P}$	$\tilde{m}$	Soloist-non-even	$\tilde{m}$	Soloist-even	Pianist
2	2^{20}	200s	2^{19}	109s	171s
4	2^{19}	101s	2^{18}	54s	87s
8	2^{18}	50s	2^{17}	27s	45s
16	2^{18}	49s	2^{16}	14s	23s

Table 5 shows the memory costs of Soloist and Pianist for proving zkRollups, which are only related to the number of transaction batches per sub-prover due to the good scalability. As shown, Soloist has a $>$ 2× smaller memory costs than Pianist. In summary, Soloist can achieve better performance for R1CS-based applications.

Evaluation of non-data-parallel applications. The circuits in Figure 6 are data-parallel: each sub-prover independently holds multiple batches of 3 transactions. We next implement Soloist over the non-data-parallel circuit: 1 batches of 3 zkRollup transactions. They have 229, 847 constraints, and the largest number of non-zero entries in R1CS matrices is $\sim 2^{20}$. We implement it on a single machine, where each prover is utilized with one core.

Table 6 presents the size of $\tilde{m}$ and prover time for zkRollups. Equipped with the "making-even" algorithm in Section 4.3, the size

Table 7: Performance for non-data-parallel ECDSA circuits

$\#\mathcal{P}$	$\tilde{m}$	Soloist-non-even	$\tilde{m}$	Soloist-even	Pianist
2	2^{20}	200s	2^{19}	108s	171s
4	2^{20}	193s	2^{18}	53s	86s
8	2^{20}	192s	2^{17}	26s	45s
16	2^{20}	192s	2^{16}	13s	23s

of $\tilde{m}$ is optimally minimized, *i.e.*, reduced by $\ell\times$, and 2-4× smaller than the uneven R1CS. The prover time is hence reduced by 2-4×, and is 1.6× faster than Pianist. The "making-even" time is only 0.5s. This result shows that our "making-even" algorithm can help to prove arbitrary R1CS well.

Apart from the zkRollup circuit, we also implement Soloist over the non-data-parallel ECDSA verification circuit from Circom [22], which proves knowledge of a private key corresponding to an Ethereum address. It has 247,380 constraints, and the largest number of non-zero entries in R1CS matrices is $\sim 2^{21}$. Table 7 presents the size of $\tilde{m}$ and prover time for this circuit. Equipped with the "making-even" algorithm, the size of $\tilde{m}$ is optimally minimized, *i.e.*, reduced by $\ell\times$, and 2-16× smaller than the uneven R1CS. The prover time is hence reduced by 2-16×. Besides, the prover time of Soloist is 1.7× faster than Pianist.

6.3 Comparisons with more distributed SNARKs

We present comparisons with more schemes like DeVirgo [60], HyperPianist [39], and Cirrus [58] for non-R1CS. We conclude that Soloist is 2-8× slower in prover time, but has 5-100× smaller proof size and 10-100× smaller communication overhead.

Comparison with DeVirgo. DeVirgo [60] is a distributed SNARK using FRI-based PCSs [60]. Such PCSs do not require group operations and can be concretely fast than group-based ones. Hence, DeVirgo can be faster than Soloist in prover. According to its figures, proving signatures with 2^{25} constraints utilized with 32 machines costs 5s, which is 6× faster than ours when proving random R1CS with constraints of the same number.

However, currently DeVirgo only supports data-parallel circuits, a special type of arithmetic circuit. In contrast, Soloist supports R1CS, an NP-complete problem. DeVirgo's proof size is about 1.9MB according to the reported figures, while ours is around 10KB. DeVirgo's communication complexity is linear to the statement size, and is up to GBs, which is 100× larger than ours and can be limited in specific applications.

Comparison with HyperPianist and Cirrus. HyperPianst [39] and Cirrus [58] are distributed SNARKs over the Plonk-targeted HyperPlonk [17]. Their sub-prover complexities are linear to the sub-statement, although in group operations and still costing quasi-linear field operations. According to the reported figures, the prover time of HyperPianist using mKZG for a size-2^{25} Plonk utilized with 8 sub-provers is 10s over BN254. The prover time over BLS12-381 is estimated to be $1.4 \times 10 = 14$s, which is 7.8× faster than ours. The prover time of Cirrus for a size-2^{25} Plonk utilized with 256 cores is larger than 32s, which is less than 2× faster ours.

However, Soloist outperforms these two schemes with constant proof size and amortized communication complexity. The proof size of HyperPianist is around 40-90KB, 4-9× larger than ours. Its communication per prover is around 60-120KB, 10-20× larger than ours. We do not find these two concrete figures of Cirrus.

7 CONCLUSION AND FUTURE WORK

We propose Soloist, a distributed SNARK for R1CS with constant proof size, verification, and amortized communication. Soloist uses a new inner-product-based approach to build a distributed PIOP with constant proof size. It uses the matrix sparsity of R1CS and distributed lookup arguments to achieve sublinear verification. Soloist also extends the bivariate KZG into a batch and distributed version.

Future work may develop a more efficient distributed preprocessing to minimize the polynomial oracles. Further, we are considering to build a distributed SNARK for customizable constraint system [54], which is efficiently compatible with all existing constraint systems such as R1CS, Plonk, and AIR [6].

REFERENCES

[1] Kasra Abbaszadeh, Christodoulos Pappas, Jonathan Katz, and Dimitrios Papadopoulos. Zero-knowledge proofs of training for deep neural networks. In *CCS*, 2024.

[2] Scott Ames, Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. Ligero: Lightweight sublinear arguments without a trusted setup. In *CCS*, 2017.

[3] Sebastian Angel, Eleftherios Ioannidis, Elizabeth Margolin, Srinath T. V. Setty, and Jess Woods. Reef: Fast succinct non-interactive zero-knowledge regex proofs. In *USENIX Security*, 2024.

[4] Arkworks. https://github.com/arkworks-rs.

[5] Arasu Arun, Chaya Ganesh, Satya V. Lokam, Tushar Mopuri, and Sriram Sridhar. Dew: A transparent constant-sized polynomial commitment scheme. In *PKC*, 2023.

[6] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Scalable, transparent, and post-quantum secure computational integrity. IACR ePrint 2018/046.

[7] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Fast Reed-Solomon interactive oracle proofs of proximity. In *ICALP*, 2018.

[8] Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *S&P*, 2014.

[9] Eli Ben-Sasson, Alessandro Chiesa, Michael Riabzev, Nicholas Spooner, Madars Virza, and Nicholas P. Ward. Aurora: Transparent succinct arguments for R1CS. In *EUROCRYPT*, 2019.

[10] Rishabh Bhadauria, Zhiyong Fang, Carmit Hazay, Muthuramakrishnan Venkitasubramaniam, Tiancheng Xie, and Yupeng Zhang. Ligero++: A new optimized sublinear IOP. In *CCS*, 2020.

[11] Dan Boneh, Justin Drake, Ben Fisch, and Ariel Gabizon. Efficient polynomial commitment schemes for multiple points and polynomials. IACR ePrint 2020/081.

[12] Jonathan Bootle, Andrea Cerulli, Pyrros Chaidos, Jens Groth, and Christophe Petit. Efficient zero-knowledge arguments for arithmetic circuits in the discrete log setting. In *EUROCRYPT*, 2016.

[13] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Gregory Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *S&P*, 2018.

[14] Benedikt Bünz, Ben Fisch, and Alan Szepieniec. Transparent SNARKs from DARK compilers. In *EUROCRYPT*, 2020.

[15] Benedikt Bünz, Mary Maller, Pratyush Mishra, Nirvan Tyagi, and Psi Vesely. Proofs for inner pairing products and applications. In *ASIACRYPT*, 2021.

[16] Matteo Campanelli, Antonio Faonio, Dario Fiore, Anaïs Querol, and Hadrián Rodríguez. Lunar: A toolbox for more efficient universal and updatable zkSNARKs and commit-and-prove extensions. In *ASIACRYPT*, 2021.

[17] Binyi Chen, Benedikt Bünz, Dan Boneh, and Zhenfei Zhang. Hyperplonk: Plonk with linear-time prover and high-degree custom gates. In *EUROCRYPT*, 2023.

[18] Alessandro Chiesa, Yuncong Hu, Mary Maller, Pratyush Mishra, Noah Vesely, and Nicholas P. Ward. Marlin: Preprocessing zkSNARKs with universal and updatable SRS. In *EUROCRYPT*, 2020.

[19] Alessandro Chiesa, Ryan Lehmkuhl, Pratyush Mishra, and Yinuo Zhang. Eos: Efficient private delegation of zksnark provers. In *USENIX Security*, 2023.

[20] Alessandro Chiesa, Dev Ojha, and Nicholas Spooner. Fractal: Post-quantum and transparent recursive proofs from holography. In *EUROCRYPT*, 2020.

[21] Circom. https://github.com/iden3/circom.

[22] Circom-ECDSA. https://github.com/0xPARC/circom-ecdsa.

[23] Liam Eagen, Dario Fiore, and Ariel Gabizon. cq: Cached quotients for fast lookups. IACR ePrint 2022/1763.

[24] Georg Fuchsbauer, Eike Kiltz, and Julian Loss. The algebraic group model and its applications. In *CRYPTO*, 2018.

[25] Ariel Gabizon and Zachary J. Williamson. plookup: A simplified polynomial protocol for lookup tables. IACR ePrint 2020/315.

[26] Ariel Gabizon and Zachary J. Williamson. Proposal: The Turbo-PLONK program syntax for specifying SNARK programs. https://docs.zkproof.org/pages/standards/accepted-workshop3/proposal-turbo_plonk.pdf.

[27] Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru. PLONK: Permutations over Lagrange-bases for oecumenical noninteractive arguments of knowledge. IACR ePrint 2019/953.

[28] Sanjam Garg, Aarushi Goel, Abhishek Jain, Guru-Vamsi Policharla, and Sruthi Sekar. zkSaaS: Zero-knowledge SNARKs as a service. In *USENIX Security*, 2023.

[29] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. Delegating computation: Interactive proofs for muggles. *J. ACM*, 62(4):27:1–27:64, 2015.

[30] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof-systems (extended abstract). In *STOC*, 1985.

[31] Alexander Golovnev, Jonathan Lee, Srinath T. V. Setty, Justin Thaler, and Riad S. Wahby. Brakedown: Linear-time and field-agnostic SNARKs for R1CS. In *CRYPTO*, 2023.

[32] Jens Groth. On the size of pairing-based non-interactive arguments. In *EUROCRYPT*, 2016.

[33] Ulrich Haböck. Multivariate lookups based on logarithmic derivatives. IACR ePrint 2022/1530.

[34] Polygon Hermez. https://github.com/hermeznetwork/circuits. 2024-11-16.

[35] Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. Constant-size commitments to polynomials and their applications. In *ASIACRYPT*, 2010.

[36] Abhiram Kothapalli, Srinath T. V. Setty, and Ioanna Tzialla. Nova: Recursive zero-knowledge arguments from folding schemes. In *CRYPTO*, 2022.

[37] Thomas Lavaur and Jérôme Lacan. Boomy: Batch opening of multivariate polynomial commitment. IACR ePrint 2023/1599.

[38] Jonathan Lee. Dory: Efficient, transparent arguments for generalised inner products and polynomial commitments. In *TCC*, 2021.

[39] Chongrong Li, Yun Li, Pengfei Zhu, Wenjie Qu, and Jiaheng Zhang. Hyperpianist: Pianist with linear-time prover via fully distributed hyperplonk. IACR ePrint 2024/1273.

[40] Yehuda Lindell. Parallel coin-tossing and constant-round secure two-party computation. In *CRYPTO*, 2021.

[41] Helger Lipmaa, Janno Siim, and Michal Zajac. Counting Vampires: From univariate sumcheck to updatable ZK-SNARK. In *ASIACRYPT*, 2022.

[42] Tianyi Liu, Tiancheng Xie, Jiaheng Zhang, Dawn Song, and Yupeng Zhang. Pianist: Scalable zkrollups via fully distributed zero-knowledge proofs. In *S&P*, 2024.

[43] Xuanming Liu, Zhelei Zhou, Yinghao Wang, Bingsheng Zhang, and Xiaohu Yang. Scalable collaborative zk-SNARK: Fully distributed proof generation and malicious security. IACR ePrint 2024/243.

[44] Marlin. https://github.com/arkworks-rs/marlin.

[45] Alex Ozdemir and Dan Boneh. Experimenting with collaborative zk-SNARKs: Zero-Knowledge proofs for distributed secrets. In *USENIX Security*, 2022.

[46] Charalampos Papamanthou, Elaine Shi, and Roberto Tamassia. Signatures of correct computation. In *TCC*, 2013.

[47] Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. Pinocchio: Nearly practical verifiable computation. In *S&P*, 2013.

[48] Pianist-gnark. https://github.com/dreamATD/pianist-gnark.

[49] Jim Posen and Assimakis A. Kattis. Caulk+: Table-independent lookup arguments. IACR ePrint 2022/957.

[50] R1CS Zero. https://github.com/risc0/risc0.

[51] Carla Ràfols and Arantxa Zapico. An algebraic framework for universal and updatable SNARKs. In *CRYPTO*, 2021.

[52] Michael Rosenberg, Tushar Mopuri, Hossein Hafezi, Ian Miers, and Pratyush Mishra. Hekaton: Horizontally-scalable zkSNARKs via proof aggregation. In *CCS*, 2024.

[53] Srinath T. V. Setty. Spartan: Efficient and general-purpose zkSNARKs without trusted setup. In *CRYPTO*, 2020.

[54] Srinath T. V. Setty, Justin Thaler, and Riad S. Wahby. Customizable constraint systems for succinct arguments. *IACR ePrint 2023/552*.

[55] Alan Szepieniec and Yuncong Zhang. Polynomial IOPs for linear algebra relations. In *PKC*, 2022.

[56] Justin Thaler. 17 misconceptions about SNARKs (and why they hold us back). https://a16zcrypto.com/posts/article/17-misconceptions-about-snarks/.

[57] Alexander Viand, Christian Knabenhans, and Anwar Hithnawi. Verifiable fully homomorphic encryption. *CoRR*, abs/2301.07041, 2023.

[58] Wenhao Wang, Fangyan Shi, Dani Vilardell, and Fan Zhang. Cirrus: Performant and accountable distributed SNARK. *IACR ePrint 2024/1873*.

[59] Howard Wu, Wenting Zheng, Alessandro Chiesa, Raluca Ada Popa, and Ion Stoica. DIZK: A distributed zero knowledge proof system. In *USENIX Security*, 2018.

[60] Tiancheng Xie, Jiaheng Zhang, Zerui Cheng, Fan Zhang, Yupeng Zhang, Yongzheng Jia, Dan Boneh, and Dawn Song. zkBridge: Trustless cross-chain bridges made practical. In *CCS*, 2022.

[61] Tiancheng Xie, Jiaheng Zhang, Yupeng Zhang, Charalampos Papamanthou, and Dawn Song. Libra: Succinct zero-knowledge proofs with optimal prover computation. In *CRYPTO*, 2019.

[62] Tiancheng Xie, Yupeng Zhang, and Dawn Song. Orion: Zero knowledge proof with linear prover time. In *CRYPTO*, 2022.

[63] Arantxa Zapico, Vitalik Buterin, Dmitry Khovratovich, Mary Maller, Anca Nitulescu, and Mark Simkin. Caulk: Lookup arguments in sublinear time. In *CCS*, 2022.

[64] Jiaheng Zhang, Tiancheng Xie, Yupeng Zhang, and Dawn Song. Transparent polynomial delegation and its applications to zero knowledge proof. In *S&P*, 2020.

[65] Jianning Zhang, Ming Su, Xiaoguang Liu, and Gang Wang. Springproofs: Efficient inner product arguments for vectors of arbitrary length. In *S&P*, 2024.

[66] Yuncong Zhang, Alan Szepieniec, Ren Zhang, Shifeng Sun, Geng Wang, and Dawu Gu. Voproof: Efficient zkSNARKs from vector oracle compilers. In *CCS*, 2022.

A EXISTING PIOPS FOR INNER PRODUCTS

A PIOP for inner products proves $\langle \boldsymbol{f}, \boldsymbol{s} \rangle = y$ for secret vectors $\boldsymbol{f}, \boldsymbol{s} \in \mathbb{F}^m$ and public $y \in \mathbb{F}$. Below we recall two univariate PIOPs for inner products from univariate sum-check and Laurent polynomials. As they both use the LDT PIOP, we first recall it in Lunar [16].

LDTs enable a prover to prove that the degree d of some polynomial f satisfies $d \leq D$, given size-D system parameters. Given oracle f, the prover sends a polynomial oracle $f'(X) = f(X) \cdot X^{D-d}$. The verifier then queries f, f' at random α, and accepts iff $f'(\alpha) = f(\alpha) \cdot \alpha^{D-d}$. The soundness error is $D/|\mathbb{F}|$ by the Schwartz-Zippel lemma.

Univariate sum-check. Such PIOPs [9, 18] compute secret polynomials $f, s \in \mathbb{F}_m[X]$ from $\boldsymbol{f}, \boldsymbol{s}$ by polynomial interpolation, and construct the univariate sum-check relation $\sum_{x \in \mathbb{H}} f(x)s(x) = y$ on an order-m multiplicative coset $\mathbb{H}$. The relation holds iff there exists polynomials $g \in \mathbb{F}_{(m-1)}[X]$ and h s.t.

$$f(X) \cdot s(X) = X \cdot g(X) + y/m + Z_{\mathbb{H}}(X)h(X),$$

where $Z_{\mathbb{H}}(X) = X^m - 1$ is the vanishing polynomial over $\mathbb{H}$. To prove it, the prover sends polynomial oracles f, s, g, h, and g^* for LDTs as discussed in Section 1.2. The verifier queries f, s, g, h, g^* evaluated at a random α. The verifier computes $Z_{\mathbb{H}}(\alpha)$, and accepts iff: 1) the equation holds for $X = \alpha$; 2) $\deg(g) < m-1$; 3) $\deg(f) < m$. Given size-m parameters, the check 3) is free.

For complexities, the oracle number and the query size are both 5 by count, and the prover requires 6 $\mathrm{FFT}(m, m)$ to compute f and s, and 6 $\mathrm{FFT}(m, m)$ to compute the polynomial multiplication $f \cdot s$.

Laurent polynomials. PIOPs [12, 55, 66] from Laurent polynomial construct IPAs by proving the coefficient of some term in a polynomial is zero. We recall the PIOP in [55] inspired by [12]. For size-m vectors $\boldsymbol{f}$ and $\boldsymbol{s}$, set size-m polynomials with coefficients $\boldsymbol{f}, \boldsymbol{s}$ as $\hat{f}, \hat{s}$, respectively. Then, $\langle \boldsymbol{f}, \boldsymbol{s} \rangle$ is the coefficient of middle term in $\hat{f}(X) \cdot X^{m-1} \cdot \hat{s}(X^{-1})$. This can be transformed into the existence of $p(X), q(X)$ such that

$$\hat{f}(X) \cdot X^{m-1} \cdot \hat{s}(X^{-1}) = p(X) + X^{m-1} \cdot y + X^m \cdot q(X), \qquad (33)$$

where they should satisfy $p, q \in \mathbb{F}_{(m-1)}[X]$. To prove it, the prover sends polynomial oracles $\hat{f}, \hat{s}, p, q$ and p^*, q^* for LDTs. The verifier queries $\hat{f}, p, q, p^*, q^*$ at random α and $\hat{s}$ at α^{-1}. The verifier accepts iff: 1) Equation (33) holds; 2) $\deg(p), \deg(q) < m-1$; 3) $\deg(f) < m$. Note that despite $\hat{s}(X^{-1})$ is a rational polynomial, the prover could compute $p(X), q(X)$ by handling $X^{m-1} \cdot \hat{s}(X^{-1})$.

For the complexity, the oracle number is 5, the query size is 6, and the prover requires 6 $\mathrm{FFT}(m, m)$ to compute $\hat{f}(X)X^{m-1}\hat{s}(X^{-1})$. The number of FFT is fewer than IPAs from univariate sum-check, as coefficient-based $\hat{f}, \hat{s}$ can be obtained directly from $\boldsymbol{f}, \boldsymbol{s}$.

B MAKING THE SUB-MATRICES "EVEN"

We propose the "making-even" algorithm in Algorithm 1. The high-level idea is to exchange the locations of columns in each of P_a, P_b, P_c and make them "even". The witness value in the witness vector $\boldsymbol{w}$ is also exchanged correspondingly. Note that if we exchange $P_a[i]$ to the location j, P_b and P_c are also changed. The number of non-zero entry in P_a, P_b, P_c can be different. We heuristically adjust the matrix with the most non-zero entries. WLOG, say this matrix is P_a, which is denoted as H in Algorithm 1.

Given the parameter ℓ and the matrix P_a, sort the number of non-zero values in each column of the matrix in ascending order to obtain the set $\mathcal{I}$. We then assign $P_a[\mathcal{I}_1]$ to $\mathcal{P}_1$, $P_a[\mathcal{I}_2]$ to $\mathcal{P}_2, \ldots$, and $P_a[\mathcal{I}_\ell]$ to $\mathcal{P}_\ell$. Then, we assign $P_a[\mathcal{I}_{\ell+1}]$ to $\mathcal{P}_\ell$, $P_a[\mathcal{I}_{\ell+2}]$ to $\mathcal{P}_{\ell-1}, \ldots$, and $P_a[\mathcal{I}_{2\ell}]$ to $\mathcal{P}_1$. Repeat the above procedures until each prover holds m columns.

The above algorithm works well except the first column in P_a, P_b, P_c, which describes constraints about the witness one, both known by the prover and the verifier. The problem lies in that the first column of these matrices may have too many non-zero entries. To solve this, we find that as this witness is "public", we can add the "one" witness to each sub-witness of sub-prover and add one additional column to sub-provers $\mathcal{P}_2, \ldots, \mathcal{P}_\ell$. Then, we can split the first column in P_a, P_b, P_c into ℓ parts in the meaning of non-zero entry number. For the i-th part, set the corresponding entry as 0 for $\mathcal{P}_1, \ldots, \mathcal{P}_{i-1}, \mathcal{P}_{i+1}, \ldots, \mathcal{P}_\ell$. As a result, we can also make the first column "even".

In Algorithm 1, it suffices to work over size-$O(m\ell)$ vectors where each entry describes the non-zero entry number instead of matrices. Hence, the algorithm costs a time complexity of at most $O(m\ell \log m\ell)$ for sorting.

C ADDITIONAL PRELIMINARIES OF KZG

In this paper, we use univariate or bivariate PCSs from pairing-based groups, We generalize the PCS definition in [11] from univariate polynomials to bivariate ones.

Definition 3. *A PCS is a triplet* (Gen, Com, Open) *such that*

- $\mathsf{Gen}(m, \ell)$ - *Given positive integers* m, ℓ*, output a structured reference string (SRS)* srs *of size* $m\ell$.
- $\mathsf{Com}(f, \mathsf{srs})$ - *Given a polynomial* $f \in \mathbb{F}_{m,\ell}[X, Y]$ *and an* srs *of* $\mathsf{Gen}(m, \ell)$*, return a commitment* cm *to* f.
- Open *is a public-coin interactive argument.* $\mathcal{P}$ *is given* $f_1, \ldots, f_n \in \mathbb{F}_{m,\ell}[X, Y]$. $\mathcal{P}$ *and* $\mathcal{V}$ *are both given:*
  1. m, ℓ, n *and* $t = \mathrm{poly}(\lambda)$.
  2. *Subsets* $R_1, \ldots, R_n$ *and* $S_1, \ldots S_n$*. For simplicity, assume* $|R_k| = |S_k| = t$ *for* $k \in [n]$*. Let* $T_1 = \bigcup_{k\in[n]} R_k$ *and* $T_2 = \bigcup_{k\in[n]} S_k$.
  3. $\mathsf{cm}_1, \ldots, \mathsf{cm}_n$ - *the KZG commitments to* $f_1, \ldots, f_n$.
  4. *claimed evaluations of* $f_k(\alpha, \beta)$ *for* $(\alpha, \beta) \in (R_k, S_k)$
  5. $\{r_k \in \mathbb{F}_{t,t}[X, Y]\}_{k\in[n]}$ - *the polynomials describing the alleged correct openings, i.e., having* $r_k(\alpha, \beta) = f_k(\alpha, \beta)$ *for each* $k \in [t]$, $\alpha \in R_k, \beta \in S_k$.

  This interactive argument satisfies:

- ***Completeness:*** *Fix any* $n, t, T_1, T_2, f_1, \ldots, f_n \in \mathbb{F}_{m,\ell}[X, Y]$, $\{r_k \in \mathbb{F}_{t,,t}[X, Y]\}_{k\in[n]}$. *Suppose for each* $k \in [t]$, $\mathsf{cm}_k = \mathsf{Com}(f_k, \mathsf{srs})$, *and we have* $Z_{R_k} \mid P_k(X, Y), Z_{S_k} \mid Q_k(X, Y)$ *for* $f_k - r_k = P_k + Q_k$. *Then for an honest* $\mathcal{P}, \mathcal{V}$ *outputs accept with probability one.*
- ***Knowledge soundness in the algebraic group model (AGM):*** *There exists an efficient extractor* $\mathcal{E}$ *s.t. for any algebraic adversary* $\mathcal{A}$ *and any choice of* $m, \ell = \mathrm{poly}(\lambda)$*, the probability of* $\mathcal{A}$ *winning*

Algorithm 1 The "making-even" algorithm for R1CS sub-matrices

Input: Public matrices of an R1CS instance $P_a, P_b, P_c \in \mathbb{F}^{m\ell \times m\ell}$. Witness $\boldsymbol{w} \in \mathbb{F}^m$. Number of sub-provers ℓ.

Output: The "Evened" distributed R1CS sub-matrices and sub-witness: $P_a^{(i)}, P_b^{(i)}, P_c^{(i)} \in \mathbb{F}^{m\ell \times (m+1)}, \boldsymbol{w}^{(i)}$, where $i \in [\ell]$.

1: Count the number of non-zero entries in $P_a[1], P_b[1], P_c[1]$. These entries denotes the constraints related to the witness one in $\boldsymbol{w}$. The witness one is public and known by both prover and verifier.
2: Split each of $P_a[1], P_b[1], P_c[1]$ into ℓ parts to make the number of non-zero entries "even". Generate $P_a^{(i)}[1], P_b^{(i)}[1], P_c^{(i)}[1]$ to describe the split constraints related to one for $i \in [\ell]$. Also, set $w_1^{(i)} = \text{one}$ for $i \in [\ell]$. As the witness one is public, these non-zero entries can be made "even" almost perfectly.
3: Find among P_a, P_b, P_c the matrix H with the largest number of non-zero entries.
4: Define a set $\mathcal{I} \leftarrow \{2, \cdots, m\ell\}$.
5: Sort $\mathcal{I}$ by the number of non-zero entries in $H[i]$ for $i \in \mathcal{I}$. $\mathcal{I}_{m\ell-1}$ has the most non-zero entries.
6: Let $j \leftarrow 1$.
7: **for** $i = m\ell - 1$ **downto** 1 **do**
8: $\quad k \leftarrow \mathcal{I}_i$
9: $\quad$ **if** $\lfloor \frac{ml-1-i}{\ell} \rfloor \mod 2 = 0$ **then**
10: $\quad\quad$ Give $P_a[k], P_b[k], P_c[k], \boldsymbol{w}_k$ to sub-prover $\mathcal{P}_j$
11: $\quad\quad j \leftarrow (j \mod \ell) + 1$
12: $\quad$ **else** $\lfloor \frac{ml-1-i}{\ell} \rfloor \mod 2 = 1$
13: $\quad\quad$ Give $P_a[k], P_b[k], P_c[k], \boldsymbol{w}_k$ to sub-prover $\mathcal{P}_{\ell-j+1}$
14: $\quad\quad j \leftarrow (j \mod \ell) + 1$
15: $\quad$ **end if**
16: **end for**

the following game is $\mathsf{negl}(n)(\lambda)$ *over randomness of* $\mathcal{A}, \mathcal{V}$, *and* Gen.

1. *Given* $\mathsf{srs} = \mathsf{Gen}(m, \ell)$, $\mathcal{A}$ *outputs* $\mathsf{cm}_1, \ldots, \mathsf{cm}_n \in \mathbb{G}_1$.
2. $\mathcal{E}$, *given access to the messages of* $\mathcal{A}$ *during the previous step, outputs* $f_1, \ldots, f_n \in \mathbb{F}_{m,\ell}[X, Y]$.
3. $\mathcal{A}$ *outputs* $T_1, T_2, R_1, \ldots, R_n, S_1, \ldots, S_n$, *and* $r_1, \ldots, r_n$.
4. $\mathcal{A}$ *takes the part of* $\mathcal{P}$ *in* Open *with inputs* $\mathsf{cm}_1, \ldots, \mathsf{cm}_n, T_1, T_2, R_1, \ldots, R_n, S_1, \ldots, S_n, r_1, \ldots, r_n$.
5. $\mathcal{A}$ *wins if* $\mathcal{V}$ *outputs accept, and for some* $k \in [n]$, $Z_{R_k} \nmid P_k(X, Y)$ *or* $Z_{S_k} \nmid Q_k(X, Y)$, *or* $f_k \neq P_k + Q_k$.