Soloist: Distributed SNARK for R1CS with Constant Proof Size

Weihan Li[1,2*], Zongyang Zhang[1,3✉], Yun Li[2✉], Pengfei Zhu[4], Cheng Hong[2], and Jianwei Liu[1]

[1] School of Cyber Science and Technology, Beihang University
{leeweihan, zongyangzhang, liujianwei}@buaa.edu.cn
[2] Ant Group {liweihan.lwh, liyun24, vince.hc}@antgroup.com
[3] Beijing Advanced Innovation Center for Future Blockchain and Privacy Computing
[4] Tsinghua University zhupf321@gmail.com

Abstract. Succinct non-interactive arguments of knowledge (SNARK) is a powerful cryptographic primitive with diverse real-world applications. The rank-one constraint system (R1CS), an intermediate representation of SNARK, has been widely used for proving arithmetic circuits. Distributed SNARKs allow multiple provers to jointly generate proofs for improving prover efficiency. However, existing distributed SNARKs for R1CS, *i.e.*, DIZK (USENIX Sec. '18) and Hekaton (CCS '24), fail to simultaneously achieve constant proof size and scalable prover efficiency. In this paper, we propose Soloist, a distributed SNARK for R1CS with constant proof size, amortized communication and verification. For a size-$O(n)$ R1CS, its prover complexity is $O(n/\ell \cdot \log(n/\ell))$ given ℓ sub-provers. Experiments show that the concrete prover time of Soloist is $\sim\ell\times$ as fast as the non-distributed R1CS-targeted Marlin (Eurocrypt '20) given ℓ sub-provers. Compared with Hekaton, Soloist features a 100× smaller communication overhead, and has a 7× faster prover time when proving general circuits. For R1CS-friendly zkRollups, Soloist outperforms the state-of-the-art Plonk-targeted Pianist (S&P '24) with a 2.5× smaller memory cost, a 2.8× faster preprocessing, and a 1.8× faster prover.

To build Soloist, we design a distributed polynomial oracle proof (PIOP) for R1CS. Its core techniques include an improved (and distributed) inner product PIOP, and a distributed preprocessing PIOP via lookup tables. To instantiate the PIOPs, we build a (distributed) batch scheme for bivariate KZG, which enables opening multiple points on multiple polynomials with a proof size irrelevant to polynomial size or point number.

Keywords: SNARK · Distributed argument · R1CS · Polynomial interactive oracle proof · Polynomial commitment

1 Introduction

Succinct non-interactive argument of knowledge (SNARK) [43,24] allows a prover to convince a verifier of the validity of some public statement with small proof

* Work done during an internship at Ant Group.
✉ Corresponding authors.

size and fast verification. Benefiting from the practical performance, they have found various applications such as blockchain and machine learning.

SNARKs typically compile the statement to be proven into an intermediate representation (IR) and then prove the IR. Popular IRs for NP-complete problems include R1CS [5,14,48], Plonk [13,19], customizable constraint system [49], and algebraic intermediate representation [4]. Among them, R1CS [24] and Plonk [44,64] are widely applied with distinct advantages: R1CS can handle addition gates for free [24], while Plonk has gained popularity for supporting custom high-degree gates and lookup schemes (though R1CS can support the latter as well [53]). While some applications can be efficiently encoded by either two IRs, certain tasks are preferred for R1CS, *e.g.*, SHA-256 membership (Small witness values make R1CS more efficient[5]), and verifiable fully homomorphic encryption (Using Plonk's high-degree gates can be less efficient [29]). At the front-end, automatic compilers from statements to IRs have emerged to facilitate proving. There exist several popular circuit-to-R1CS compilers (*e.g.*, Circom [3], Circ [39]) with formal verification methods [40,41]. However, circuit-to-Plonk compilers currently still remain less mature and efficient [51]. This is probably one reason why several Plonk-targeted schemes [36,54] firstly use compilers to transform a circuit into R1CS, and then to Plonk, leading to an expansion in statement size and additional overhead (*e.g.*, up to 10× in [36]). To sum up, constructing efficient SNARKs for various IRs remains a meaningful and active research direction.

Prover time is currently a bottleneck for SNARKs. Many efforts have been made to reduce it at the scheme level [22,48,59], but few simultaneously deliver extremely small proof size or fast verification (*e.g.*, constant complexities in [14,19,24]) and fast prover (*e.g.*, linear complexity in [48,59]). Recent works explore distributed SNARKs [33,36,54,55,57], which employ multiple provers to generate proofs collaboratively. We say a distributed scheme has *linear speedup* if its per-prover work is $1/\ell$ of the original one given ℓ provers. It further achieves *full* linear speedup [34] if maintaining the same proof size and verifier complexity as the original scheme. Distributed SNARKs with full linear speedup offer a promising path toward attaining fast proof generation, extremely small proof and fast verification (though needing more computational resources).

Different from non-distributed SNARKs for R1CS or Plonk, *e.g.*, (univariate) Groth16 [24]/Marlin *v.s.* PLONK [19], and (multivariate) Spartan [48] *v.s.* HyperPlonk [13], current distributed SNARKs for R1CS are less efficient than those for Plonk. The Plonk-targeted Pianist [36] built upon PLONK achieves full linear speedup with constant proof size, verification, and amortized communication. In contrast, neither the R1CS-targeted DIZK [55] nor Hekaton [46] achieves this. This inefficiency would impact many R1CS-friendly applications.

Such a state of issues motivates us to ask: *Can we build a distributed SNARK for R1CS with constant proof size, verification, and amortized communication?*

Challenges. Compared with Plonk handling vectors of size linear in circuit size, R1CS is inherently more complex for handling (sparse) matrices of linear

[5] aztec.network/blog/ plonk-benchmarks-i—-2-5x-faster-than-groth16-on-mimc.

dimensions. Various lines of research on (non-)distributed SNARKs for R1CS have been proposed, but they all face challenges in achieving our goal above.

Ligero-based works [1] arrange the witness into a matrix. The prover operates over all rows, and then all columns. Assigning each sub-prover with several rows is natural for distribution, but sub-provers need to communicate all rows for column-wise operations, leading to a *linear communication*. Building distributed SNARKs from multilinear sum-check is also feasible [33,57], but their proof size or verification is *logarithmic*. SNARKs from univariate sum-check [5,14] originally feature a constant proof size; however, although it is possible to split a large-size sum-check into multiple smaller ones, how to aggregate them together while maintaining the constant proof size remains unknown.

For distributed schemes, DIZK [55] designs a series of distributed components in Groth16 [24] such as distributed FFT and distributed multi-scalar multiplication. However, the distributed FFT incurs a linear communication and fails to obtain linear speedup. Hekaton [46] partitions the circuit into multiple sub-circuits, which are proved via Mirage [30] (a commit-and-carry version of Groth16) by sub-provers and then aggregated by the master prover. However, this aggregation leads to a logarithmic proof size and verifier cost in the number of sub-provers, and the amortized communication cost can be several MBs.

1.1 Other Related Work

There are two kinds of distributed SNARKs. One category [38,21,15,37,60] typically considers the scenario where a *resource-limited* prover delegates proof generation to multiple servers while preserving privacy of the witness via secure multi-party computation. Due to the usage of secure computation primitives (such as secret sharing), achieving proof acceleration can be challenging.

Another category, which we focus on, considers a *resource-rich* prover utilizing multiple machines to accelerate proof generation. We use a distributed model where each sub-prover only communicates with the master prover, and each sub-prover is initialized with a part of the true secret witness.

Most distributed SNARKs [33,36,57] employ the paradigm of combining a distributed polynomial interactive oracle proof (PIOP) and a distributed polynomial commitment scheme (PCS), with varied properties according to the underlying PIOP and PCS. DeVirgo [57] uses a PIOP for data-parallel circuits and a trustless hash-based PCS [62], incurring linear communication and poly-logarithmic proof size or verification. HyperPianist [33] and Cirrus [54] use multilinear Plonk-targeted PIOPs and group-based PCSs, leading to a linear prover complexity over group (still super-linear over field) but logarithmic proof size, verification, and communication. Pianist [36] uses a Plonk-targeted bivariate PIOP and a group-based PCS for constant proof size, verification, and amortized communication, but it is not favorable for R1CS-friendly circuits.

Existing R1CS-targeted schemes, DIZK [55] and Hekaton [46], adapt different paradigms as recalled, but fail to achieve *full linear speedup*. Another limitation of them is circuit-specific setups, which require a new trusted ceremony for every new circuit. While the state-of-the-art Hekaton supports general circuits, it is

efficient mainly for *(nearly-)data-parallel* circuits with few shared wires across sub-circuits (*e.g.*, Merkle tree membership proofs), and when proving general circuits, its efficiency would degrade rapidly as the number of shared wires increases. We believe efficiently proving for general circuits is important. Even data-parallel circuits themselves become general ones if each sub-circuit is partitioned across multiple machines, in which case Hekaton may perform worse.

A concurrent work DFS [27] proposes distributed SNARKs for R1CS in two categories, though it mainly focuses on the former one. Its scheme in the second category matches HyperPianist in asymptotic complexity, as both rely on multilinear PIOPs and PCSs. DFS also leaves open whether a distributed univariate sum-check protocol is feasible, which we answer affirmatively. While (distributed) SNARKs built from multilinear primitives typically outperform those based on univariate PIOPs and PCSs in prover efficiency, DFS has a 2× slower prover concretely than Soloist with the same vCPUs according to its reported figures.

1.2 Technical Contribution

We follow the "distributed PIOP + distributed PCS" paradigm as in [33,36,57]. Our key observation is that R1CS can be reduced to inner products [5,14], and inner products are friendly for distribution: a size-$m\ell$ inner product instance can be split into the sum of ℓ sets of size-m ones, which are basically independent with only summation as a weak inter-connection. This motivates us to build a *constant-proof inner product PIOP* with improved efficiency than prior works [5,14] (*cf.*, Section 3.1). We then extend our PIOP into a *distributed inner product PIOP*, while maintaining the constant proof size (*cf.*, Section 3.2).

Next, we reduce R1CS into inner products by improving the method in Aurora [5], leading to a *distributed PIOP for R1CS* with constant proof size but linear verifier complexity (*cf.*, Section 4.1). We then further achieve constant verification by delegating polynomial oracles *w.r.t.* public R1CS matrices to a trusted indexer, via building a *distributed preprocessing* PIOP based on lookup tables with novel *online* table-validity check techniques (*cf.*, Section 4.2).

Our last step is to use a distributed PCS to instantiate our distributed PIOP, which requires to open multiple *bivariate* polynomial oracles at multiple points. We choose the bivariate KZG [42] and its distributed variant from Pianist [36] considering their constant proof size. However, this PCS has no efficient batch version in either non-distributed or distributed setting. This motivates us to build an efficient batch scheme for bivariate KZG with small proof size (*cf.*, Section 5.1), and then extend it into a distributed version (*cf.*, Section 5.2).

Below we elaborate on our main technical contributions.

(1) Improved inner product PIOP and distribution. Inner product arguments (IPA) allow showing the inner product of two vectors equals a claimed value. Classic IPAs from discrete-log [9] are trustless but with logarithmic proof sizes. IPAs from the "PIOP + PCS" paradigm [11,14,52] could achieve distinct trade-offs; those using a constant-proof PIOP and the KZG PCS [28] feature constant proof sizes. We build an inner product PIOP with constant proof size

Table 1. Inner product PIOPs for size-$m\ell$ vectors, all with $O(1)$ proof size

Scheme	SRS size	Secret oracle number	FFT number	Proof size
Marlin [14]	$m\ell$	5	5	$5\,\lvert\mathbb{F}\rvert$
Dark [11]	$m\ell$	5	3	$6\,\lvert\mathbb{F}\rvert$
Ours	$m\ell$	4	3	$4\,\lvert\mathbb{F}\rvert$
Below is the distributed inner product PIOP, assuming ℓ provers				
Scheme	Per prover	Amortized communication	#Oracle & Proof size	Verifier
Ours	$O(m \log m)$	2 oracles, $3\,\lvert\mathbb{F}\rvert$	4 oracles & $5\,\lvert\mathbb{F}\rvert$	$O(1)^*$

* Similarly to Marlin [14] and Plonk [19], the $O(1)$ verifier complexity is in the preprocessing model feasible by polynomial delegation. Without this, it is $O(\log m\ell)$.

and verification, featuring the smallest proof size, fewest polynomial oracles, and fewest FFTs in proving compared with existing ones (*cf.*, Table 1). The core technique is a newly explored relation between inner product and univariate sum-check of coefficient-based polynomials (instead of evaluation-based ones [5,14]).

We then distribute our inner product PIOP with full linear speedup and constant amortized communication (*cf.*, Table 1). Technically, we design a *two-dimension univariate sum-check*, which uses the univariate sum-check twice to prove a bivariate sum-check with constant proof size and verification. It outperforms the existing bivariate sum-check protocol [10] which has a linear proof size and verification cost in the total degree of the two variables.

Our schemes may be of independent interest, as IPAs are widely used in range proofs [9], PCSs [32,62], and SNARKs [63]. For example, our distributed inner product PIOP can naturally return back to a non-distributed batch IPA with constant proof size regardless of the number of IPA instances, in contrast to linear or logarithmic ones to the instance number as in existing methods [5,9].

(2) Succinct Distributed PIOP for R1CS. We build a distributed PIOP for R1CS, featuring constant proof size, amortized communication and verification. It inherently supports general circuits, unlike Pianist [36] which incurs extra cost extending from data-parallel circuits to general ones.

To build this PIOP, we first reduce R1CS to our (distributed) inner product PIOPs with constant proof size. Compared with Aurora [5], our PIOP for R1CS has fewer polynomial oracles and hence superior efficiency. To further achieve constant verification, we delegate the polynomial oracles of R1CS matrices to a trusted indexer. We use the sparse matrix representation from Marlin; however, the adaptation of Marlin incurs an $O(\ell)$ proof size given ℓ provers. We thus design a new distributed preprocessing PIOP based on our coefficient-based polynomials. This requires handling exponential polynomial oracles that are determined by online randomness, and we exploit lookup table schemes with novel online table-validity check as well as table decomposition techniques to achieve linear speedup and constant verification. Note that most existing lookup schemes [16,18,61] assume a correct table that is pre-constructed by a trusted

Table 2. Batch KZG for opening t points on each of n size-$m\ell$ *Uni*variate polynomials or *Bi*variate polynomials possibly reduced from *Multi*variate polynomials$^{\|}$

Scheme	Poly.	Open (over $\mathbb{G}$)	Proof size	Verify (over $\mathbb{G}$)
BDF^{+}20 [7]	Uni.	$O(m\ell)\,\mathbb{G}$	$2\,\lvert\mathbb{G}\rvert$	$O(n)\,\mathbb{G}$, $2\,P$
HyperPlonk [13]	Multi. to Bi.	$O(m\ell)\,\mathbb{G}$	$O((m+\ell)\cdot\log t)\,\lvert\mathbb{F}\rvert$, $2\,\lvert\mathbb{G}\rvert$	$O(n)\,\mathbb{G}$, $2\,P$
Boomy* [31]	Multi. to Bi.	$O(nm\ell)\,\mathbb{G}$	$2n\,\lvert\mathbb{G}\rvert$	$O(nt)\,\mathbb{G}$, $3n\,P$
Ours†	Bi.	$O(m\ell)\,\mathbb{G}$	$(n+4)\,\lvert\mathbb{G}\rvert$	$O(n)\,\mathbb{G}$, $4\,P$
Ours (same-Y points)	Bi.	$O(m\ell)\,\mathbb{G}$	$4\,\lvert\mathbb{G}\rvert$, $(n+1)\,\lvert\mathbb{F}\rvert^{\ddagger}$	$O(n)\,\mathbb{G}$, $5\,P$
Below is the distributed batch bivariate KZG, assuming ℓ provers				
Scheme	Open	Total commu.	Proof size	Verifier
Ours	$O(m)\,\mathbb{G}$	$O(\ell n)$	$(n+4)\,\lvert\mathbb{G}\rvert$	$O(n)\,\mathbb{G}$, $4\,P$

$\|$: Typically, n and t are small constants in SNARKs [14,19], while $m\ell$ is linear in the circuit size.
$*$: Boomy only supports the case where $n = 1$, so we multiply all complexity terms by n for repeating the scheme n times when handling n polynomials.
$\dagger$: Here we assume the t points come from a Cartesian-product of two sets.
$\ddagger$: The field element size $|\mathbb{F}|$ can be concretely $3\times$ smaller than the group element size $|\mathbb{G}|$ [32].

indexer and the verifier does not need to check its validity. Our approach may inspire other lookup schemes when the tables cannot be pre-determined.

(3) Batch bivariate KZG and its distribution. Batch PCSs enable efficient openings for multiple points on multiple polynomials. For the univariate KZG [28] (which is the simplest case), Boneh *et al.* [7] propose a batch scheme with constant proof size. For the multivariate KZG [42], Boomy [31] explores batch opening multiple points on a *single* polynomial, and only a special case (where the points are all different in at least one dimension) achieves a proof size linear in the number of polynomial variables. HyperPlonk [13] gives a general batch scheme, but its proof size is linear in the total degree of the variables and logarithmic to the point number due to a high-degree sum-check protocol.[6] The concurrent Samaritan [20] proposes a new bivariate KZG from univariate KZG, which can be naturally extended to a batch version based on Boneh *et al.* [7].

We propose an efficient batch scheme for bivariate KZG (*cf.*, Table 2), which supports opening multiple points on multiple polynomials. We start from the univariate batch KZG [7], which is challenging to be extended into a bivariate one as the quotient polynomials over the two dimensions (*i.e.*, X and Y) cannot be merged directly. For evaluation points forming a Cartesian-product of two evaluation sets over the two dimensions, we rely on auxiliary bivariate polynomials to help "split" the two dimensions and batch them separately, yielding a proof size of $(n+4)\,|\mathbb{G}|$ given n polynomials (irrelevant to the point number). For the more general non-Cartesian-product case where the verifier cannot compute a public well-formed bivariate polynomial from the given evaluations, we

[6] It is feasible to reduce the proof size by committing and opening intermediate polynomials via PCSs in the sum-check instead of sending polynomials plainly. However, this would worsen the prover complexity and the verifier efficiency.

Table 3. Distributed SNARKs for size-$n = m\ell$ statements and ℓ sub-provers. (All schemes, if for general computations, require trusted setups and/or trusted preprocessing for sublinear verification. "Accoun." means accountability)

Scheme	Circuit	Setup	Per SRS	Prover (over field)‡	Total commu.	$\lvert\pi\rvert/\mathcal{V}$	Accoun.
DeVirgo [57]	Parallel	Transparent	N.A.	$O(m \log m)$	$O(n)$	$O(\log^2 n)$	✗
Pianist [36]	General (Plonk)	Universal & Updatable (U&U)	$O(m)$	$O(m \log m)$	$O(\ell)$	$O(1)$	✓
HyperPianist [33]				$O(m \frac{\log \lvert\mathbb{F}\rvert}{\log m})$	$O(\ell \log n)$	$O(\log n)$	✗
Cirrus [54]							✓
DIZK [55]	General (R1CS)	Circuit-specific	$O(n)$	$O(m \log^2 m)$	$O(n)$	$O(1)$	✗
Hekaton [46]			$O(m)$	$O(m \log m)$	$O(\ell)$	$O(\log \ell)$	✓
Soloist		U&U	$O(m)$	$O(m \log m)$	$O(\ell)$	$O(1)$	✓

‡: The super-linear terms for prover can stem from distributed FFTs [55] of $O(m \log^2 m)$, FFTs of $O(m \log m)$, and group multi-scalar exponentiations of $O(m \frac{\log \lvert\mathbb{F}\rvert}{\log m})$ where $\log \lvert\mathbb{F}\rvert = \omega(\log m)$ [22].

introduce additional univariate polynomials for the verifier to validate the evaluations. This increases the proof size to $(3n+6)\,|\mathbb{G}|$, but the prover and verifier remain efficient. We further observe that our PIOP exhibits a distinctive property: all queried evaluation points share the same Y-coordinate. In this case, we propose a PCS variant with a concretely smaller proof size of $4\,|\mathbb{G}| + (n+1)\,|\mathbb{F}|$. Compared with prior batch PCSs (seeing bivariate as a special case of multivariate), our PCS outperforms Boomy for supporting multiple polynomials with faster prover and verifier complexities, and has a smaller proof size than HyperPlonk. Our PCSs also have smaller proof sizes than Samaritan, which is around $2n\,|\mathbb{G}| + 3n\,|\mathbb{F}|$.

Based on above, we further build a distributed batch KZG with full linear speedup. Its core technique, inspired by Pianist [36], is to use Lagrange polynomials to split the computation over two dimensions, enabling the sub-provers and master prover to handle X-related and Y-related polynomials separately.

(4) Distributed SNARK and evaluation. By combining our distributed PIOP for R1CS and distributed batch bivariate KZG, we obtain Soloist[7] (*cf.*, Table 3). It has a constant proof size, verifier complexity, and amortized communication, with the same complexities as the Plonk-targeted Pianist [36]. Compared with DeVirgo [57] which is limited to data-parallel circuits, Soloist supports general ones with superior communication, proof size and verification complexities. Compared with Plonk-targeted HyperPianist [33] and Cirrus [54], Soloist achieves trade-offs between prover efficiency and all other metrics. Among R1CS-targeted schemes, Soloist has a universal and updatable setup, in contrast to DIZK [55] and Hekaton [46] with circuit-specific ones. In addition, Soloist outperforms DIZK in prover and communication complexities, and beats Hekaton in proof size and verifier time. Although the total SRS size of Soloist is $O(m\ell)$, each prover only needs to store a sub-SRS of size $O(m)$, achieving a similar advantage

[7] A.k.a, Succinct and Optimized Linear-speedup SNARK for rank-One constraint system via dISTribution.

as Hekaton. This owes to the construction of our distributed batch KZG PCS, which naturally applies to other distributed SNARKs (*e.g.*, Pianist and Hyper-Pianist) from "distributed PIOP + distributed PCS". This feature, to the best of our knowledge, is the first time that has been explicitly identified. Besides, Soloist can attain accountability without sacrificing asymptotic complexities for general circuits, while this property is limited to data-parallel circuits in Pianist.

Experiments show the full linear speedup of Soloist for both data-parallel and general circuits, in contrast to Hekaton, where the prover time may worsen for general circuits with the increase of prover number (due to more shared wires among sub-circuits). Concretely, while Soloist can be up to 10× slower on data-parallel circuits, it achieves up to a 7× speedup on general circuits, and the advantage will be larger with more provers. More importantly, Soloist has a 2.5× smaller proof size, a 20× faster verifier time, and a 1000× smaller communication cost. When proving the R1CS-friendly zkRollups for blockchain scalability, Soloist outperforms Pianist with a 2.8× and 1.8× speedup in preprocessing and proving, respectively, and has the capability to handle 10× more transactions.

2 Preliminaries

For an integer n, let $[n] = \{1, 2, \ldots, n\}$. Vectors are denoted in bold like $\boldsymbol{a}$, with a_i being the i-th element. For vectors $\boldsymbol{a}$ and $\boldsymbol{b}$, $\langle \boldsymbol{a}, \boldsymbol{b} \rangle$ is their inner product, and $\boldsymbol{a} \circ \boldsymbol{b}$ is their entry-wise product. Uppercase letters like P mean matrices, where $P[j]$ is the j-th column, $P[j][k]$ is the entry at j-th column and k-th row, and $||P||$ is its non-zero entry number. $f \in \mathbb{F}_m[X]$ is a univariate polynomial $f(X)$ with degree (strictly) less than m, and $g \in \mathbb{F}_{m,\ell}[X, Y]$ means a bivariate polynomial $g(X, Y)$ with degrees (strictly) less than m over X and ℓ over Y. Denote polynomial size as the possible maximum number of its monomial terms.

Vanishing, Lagrange, and bivariate polynomials. Given a set $S \subset \mathbb{F}$, denote a size-$|S|$ polynomial $Z_S(X)$ as the vanishing polynomial of S where $Z_S(x) = 0$ for all $x \in S$. Given a set $(\alpha_1, \ldots, \alpha_\ell)$, let $L_i(X)$ be the i-th Lagrange polynomial s.t. $L_i(\alpha_i) = 1$ and for $k \in [\ell] \wedge k \neq i$, $L_i(\alpha_k) = 0$. Use $\mathbb{H}, \mathbb{M}, \mathbb{L}$ to represent smooth multiplicative subgroups over field. For such subgroups (*e.g.*, $\mathbb{L} = (\eta^0, \ldots, \eta^{\ell-1})$), we have $Z_{\mathbb{L}}(X) = X^\ell - 1$ and $L_i(Y) = \frac{\eta^{i-1}}{\ell} \cdot \frac{Y^\ell - 1}{Y - \eta^{i-1}}$. Further, any evaluation on $Z_{\mathbb{L}}(X)$ or $\{L_i(Y)\}$ can be computed in time $O(\log \ell)$.

In this paper, *all bivariate polynomials* are defined using Lagrange polynomials of $\mathbb{L}$. For example, given univariate polynomials $\{f_i(X)\}$ or $\{\mathsf{col}_i(X)\}$ for $i \in [\ell]$, we have $f(X, Y) = \sum_{i \in [\ell]} f_i(X) L_i(Y)$ or $\mathsf{col}(X, Y) = \sum_{i \in [\ell]} \mathsf{col}_i(X) L_i(Y)$.

Interactive oracle proof (IOP). IOP [6] is a multi-round interactive proof where in each round, the verifier sends a challenge and the prover replies with a message or oracle. At the end, the verifier queries these oracles and accepts or rejects. PIOP [14] is an IOP where all oracles are polynomials and queries are their evaluations. Denote proof size as the sum of messages and query sizes.

Polynomial commitment scheme (PCS). PCS [22,28] allows a prover to commit a polynomial via a commitment algorithm, and then convince the verifier of the validity of some public evaluation at this polynomial via a open protocol.

It is well known that SNARKs can be built from "PIOP + PCS", where the polynomial oracles in PIOP are instantiated by PCS. Appendix A recalls more details and definitions.

Low-degree tests (LDTs). LDTs enable a prover to show the degree of some polynomial is less than a public d, given a size-D system parameter ($d < D$). LDT is important in recent SNARKs [5,12,14,45]. A general LDT can be built by "PIOP + PCS", and we recall the PIOP in Lunar [12]. Given a secret $f \in \mathbb{F}_d[X]$, the prover sends an oracle $f'(X) = f(X) \cdot X^{D-d}$ to the verifier. Then, the verifier queries $f(X), f'(X)$ at a random α, and accepts iff $f'(\alpha) = f(\alpha) \cdot \alpha^{D-d}$.

Univariate sum-check. Given a size-d system parameter, 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 [5] shows $\sum_{x \in \mathbb{H}} f(x) = y$. 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. $f(X) = X \cdot g(X) + y/m + Z_{\mathbb{H}}(X) \cdot h(X)$. $\mathcal{P}$ sends polynomials oracles $g(X), h(X)$ to the verifier $\mathcal{V}$, who then queries $f(\alpha), g(\alpha), h(\alpha)$ for a random α to check the validity. $\mathcal{P}$ and $\mathcal{V}$ also run an LDT to show the low-degree property of $g(X)$.

R1CS. We use the definition in Aurora [5]. Given public and sparse matrices $P_a, P_b, P_c \in \mathbb{F}^{m\ell \times m\ell}$ with $O(m\ell)$ non-zero entries, 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 $\boldsymbol{w}, \boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c} \in \mathbb{F}^{m\ell}$ s.t. $P_a\boldsymbol{w} = \boldsymbol{a}, P_b\boldsymbol{w} = \boldsymbol{b}, P_c\boldsymbol{w} = \boldsymbol{c}$, and $\boldsymbol{a} \circ \boldsymbol{b} = \boldsymbol{c}$. Denote its size by $n = O(m\ell) = \max\{||P_a||, ||P_b||, ||P_c||\}$.

3 Improved Inner Product PIOP and its Distribution

We present an improved inner product PIOP with constant proof size, and then extend it into a distributed PIOP with full linear speedup.

3.1 Improved Inner Product PIOP

Previous techniques. An inner product PIOP shows the following relation: given two secret vectors $\boldsymbol{f}, \boldsymbol{s} \in \mathbb{F}^m$ and a claimed sum T, it holds that $\langle \boldsymbol{f}, \boldsymbol{s} \rangle = T$. Below we recall two constructions from univariate sum-check [5,14] and Laurent polynomials [52,63], respectively. Assume the system parameter is of size m.

Univariate sum-check. Such PIOPs compute secret polynomials $f, s \in \mathbb{F}_m[X]$ from $\boldsymbol{f}, \boldsymbol{s}$ by interpolation over an order-m subgroup $\mathbb{H}$, and build a sum-check relation $\sum_{x \in \mathbb{H}} f(x) \cdot s(x) = T$. The prover and verifier then run the univariate sum-check PIOP [5] (as recalled in Section 2) to prove this relation.

The PIOP requires 4 polynomial oracles for $f(X), s(X), g(X)$ and $h(X)$, and 1 oracle for the LDT of $g(X)$. The proof size is 5 field elements due to querying each oracle once. The prover needs 2 FFTs to compute $f(X)$ and $s(X)$ by polynomial interpolation, and 3 FFTs to compute $f(X) \cdot s(X)$.

Laurent polynomials. Such PIOPs use secret polynomials $f, s \in \mathbb{F}_m[X]$ by directly seeing $\boldsymbol{f}, \boldsymbol{s}$ as coefficients. Specifically, they set $f(X) = \sum_{i \in [m]} f_{i-1} \cdot X^{i-1}$ and $s(X^{-1}) = \sum_{i \in [m]} s_{i-1} \cdot X^{-i+1}$. Then, $T = \langle \boldsymbol{f}, \boldsymbol{s} \rangle$ equals the coefficient of the

middle-term in $f(X) \cdot X^{m-1} \cdot s(X^{-1})$. This is further reduced into the existence of $p, q \in \mathbb{F}_{m-1}[X]$ s.t. $f(X) \cdot X^{m-1} \cdot s(X^{-1}) = p(X) + X^{m-1} \cdot T + X^m \cdot q(X)$.

The prover needs no FFTs to compute the coefficient-based $f(X)$ and $X^{m-1} \cdot s(X^{-1})$. It needs 3 FFTs to compute $f(X) \cdot X^{m-1} \cdot s(X^{-1})$. The PIOP needs 6 polynomial oracles: 4 for $f(X), s(X), p(X)$ and $q(X)$, and 2 for the LDT of $p(X)$ and $q(X)$. The proof size is 6 elements due to querying each oracle once.

To sum up, inner product PIOPs from univariate sum-check have fewer oracles and smaller proof sizes with fewer LDTs, while PIOPs from Laurent polynomials need fewer FFTs for viewing vectors as coefficients other than evaluations.

Our construction. We achieve the best of both worlds by combining univariate sum-check and coefficient-based polynomials via a newly explored relation between inner product and univariate sum-check, which we show in Lemma 1.

Lemma 1. *Given vectors $\boldsymbol{f}, \boldsymbol{s} \in \mathbb{F}^m$ and an order-m subgroup $\mathbb{H}$, define $f, s \in \mathbb{F}_m[X]$ s.t. $f(X) = \sum_{i\in[m]} f_{i-1} \cdot X^{i-1}$ and $s(X) = \sum_{i\in[m]} s_{i-1} \cdot X^{i-1}$. Then, the inner product relation $\langle \boldsymbol{f}, \boldsymbol{s} \rangle = T$ holds iff $\sum_{x\in\mathbb{H}} f(x) \cdot s(x^{-1}) = m \cdot T$.*

Using Lemma 1, we express the inner product as a *univariate sum-check* of *coefficient-based polynomials*. However, directly running the sum-check is infeasible as $s(X^{-1})$ is evaluated at inverses of X. We then try to convert it to a standard polynomial. Our first attempt substitutes $s(X^{-1})$ with $X^m \cdot s(X^{-1})$, using the identity $x^m = 1$ for $x \in \mathbb{H}$. However, the resulting polynomial size is $m + 1$, which increases the system parameter size from m to $m + 1$, and hence requires additional LDTs for $f(X)$ and $s(X)$ to restrict the vector length.

To address this problem, we instead build a size-m standard polynomial $s'(X)$ that is equivalent to $s(X^{-1})$ for each $x \in \mathbb{H}$. Given $s(X)$ as in Lemma 1, we set

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

inspired by [56]. Note that for all $x \in \mathbb{H}$, we have $s(x^{-1}) = s'(x)$ as $x^m = 1$; more importantly, $s'(X)$ is now of size m as the term $s_0 X^m$ is eliminated. Thus,

$$\langle \boldsymbol{f}, \boldsymbol{s} \rangle = T \Leftrightarrow \sum\nolimits_{x\in\mathbb{H}} f(x) \cdot s'(x) = m \cdot T, \tag{2}$$

which is a standard univariate sum-check. While the sum-check initially needs an LDT for an auxiliary polynomial $g(X)$, we adapt the techniques in Basilisk [45] to remove it with slight modifications. In Equation (2), it suffices to handle $s'(X)$ directly, and extracting $s'(X)$ is equivalent to extracting $s(X)$.

Based on Equation (2), univariate sum-check [5] and the LDT-elimination technique from Basilisk, we have the following theorem. The formal protocol and security proofs for Lemma 1 and the PIOP are presented in Appendix B.1.

Theorem 1. *Given size-m vectors, there exists an inner product PIOP with perfect completeness and a soundness error of $(2m-1)/|\mathbb{F}|$. The proof size is $O(1)$. The prover complexity is $O(m \log m)$ for requiring 3 FFTs. The verifier complexity is $O(\log m)$, and can be reduced to $O(1)$ via preprocessing.*

3.2 Distributed Inner Product PIOP

This section distributes our improved inner product PIOP via a method named as *two-dimension univariate sum-check*. In the distributed setting, we consider proving $\langle \boldsymbol{f}, \boldsymbol{s} \rangle = T$ for secret vectors $\boldsymbol{f}, \boldsymbol{s} \in \mathbb{F}^{m\ell}$ across ℓ sub-provers $\mathcal{P}_1, \ldots, \mathcal{P}_\ell$ and a master prover $\mathcal{P}_0$. We start by splitting $\boldsymbol{f}$ and $\boldsymbol{s}$ as $(\boldsymbol{f}_1, \ldots, \boldsymbol{f}_\ell)$ and $(\boldsymbol{s}_1, \ldots, \boldsymbol{s}_\ell)$, and assigning $\mathcal{P}_i$ with the i-th sub-vectors $\boldsymbol{f}_i$ and $\boldsymbol{s}_i$. Then, it suffices to prove $\sum_{i\in[\ell]} \langle \boldsymbol{f}_i, \boldsymbol{s}_i \rangle = T$ and hence, relying on Equation (2), to prove

$$\sum\nolimits_{i\in[\ell]} \sum\nolimits_{x\in\mathbb{H}} f_i(x) \cdot s'_i(x) = m \cdot T. \tag{3}$$

By the *(first) X-dimension univariate sum-check* over Equation (3), it requires the prover to show the existence of $g_1 \in \mathbb{F}_{m-1}[X]$ and $h_1(X)$ such that

$$\sum\nolimits_{i\in[\ell]} f_i(X) \cdot s'_i(X) = X \cdot g_1(X) + T + Z_{\mathbb{H}}(X) \cdot h_1(X). \tag{4}$$

Proving Equation (4) is distribution-friendly. Each prover $\mathcal{P}_i$ can compute $g_{1,i}(X)$, $h_{1,i}(X)$, and T_i locally by dividing $Z_{\mathbb{H}}(X)$ over $f_i(X) \cdot s'_i(X)$. Due to the uniqueness of polynomial division, $\mathcal{P}_0$ can compute $g_1(X) = \sum_{i\in[\ell]} g_{1,i}(X)$, $h_1(X) = \sum_{i\in[\ell]} h_{1,i}(X)$, and $T = \sum_{i\in[\ell]} T_i$. Given oracle access to $g_1(X), h_1(X)$ and $\{f_i(X), s'_i(X)\}_{i\in[\ell]}$, the verifier checks if Equation (4) holds at a random $X = \alpha$.

However, such a method leads to at least an $O(\ell)$ proof size, as the prover would send 2ℓ polynomial evaluations to the verifier. To achieve a constant proof size, our key idea is to aggregate the univariate polynomials $\{f_i(X), s'_i(X)\}_{i\in[\ell]}$ by introducing another variable Y and the following newly explored lemma.

Lemma 2. *Suppose multiplicative subgroups $\mathbb{H}$ and size-ℓ $\mathbb{L}$ with Lagrange polynomials $\{L_i(Y)\}_{i\in[\ell]}$. For each $k \in [n]$, given polynomials $\{f_i^{(k)}(X)\}_{i\in[\ell]}$ (without assuming any degree bound) and $f^{(k)}(X,Y) = \sum_{i\in[\ell]} f_i^{(k)}(X) L_i(Y)$, we have*

$$\sum\nolimits_{i\in[\ell]} f_i^{(1)}(X) \cdot f_i^{(2)}(X) \cdots f_i^{(n)}(X) = \sum\nolimits_{y\in\mathbb{L}} f^{(1)}(X,y) \cdot f^{(2)}(X,y) \cdots f^{(n)}(X,y),$$
$$\sum\nolimits_{x\in\mathbb{H}, i\in[\ell]} f_i^{(1)}(x) \cdot f_i^{(2)}(x) \cdots f_i^{(n)}(x) = \sum\nolimits_{x\in\mathbb{H}, y\in\mathbb{L}} f^{(1)}(x,y) \cdot f^{(2)}(x,y) \cdots f^{(n)}(x,y).$$

The first equation in Lemma 2 holds as $L_i(Y)$ only evaluates to 1 at the i-th point in $\mathbb{L}$ (*i.e.*, $y = \eta^{i-1}$) and 0 otherwise. This usage of Lagrange polynomials is inspired by Pianist [36]. Thus, $f^{(k)}(X,y) = f_i^{(k)}(X)$ at $y = \eta^{i-1}$ for all $i \in [\ell]$. The second equation then follows naturally. Using this lemma, it is feasible to aggregate a univariate sum-check of *multiple* sets of univariate polynomial products into a two-dimension sum-check of *one* set of bivariate polynomial products, as shown below. We now handle Equation (4) at $X = \alpha$. From Lemma 2 we know that $\sum_{i\in[\ell]} f_i(\alpha) \cdot s'_i(\alpha) = \sum_{y\in\mathbb{L}} f(\alpha, y) \cdot s'(\alpha, y)$. Then, Equation (4) becomes

$$\sum\nolimits_{i\in[\ell]} f_i(\alpha) \cdot s'_i(\alpha) = \sum\nolimits_{y\in\mathbb{L}} f(\alpha, y) \cdot s'(\alpha, y) = \alpha \cdot g_1(\alpha) + T + Z_{\mathbb{H}}(\alpha) \cdot h_1(\alpha). \tag{5}$$

Let $T_2 = \alpha \cdot g_1(\alpha) + T + Z_{\mathbb{H}}(\alpha) \cdot h_1(\alpha)$. Equation (5) then turns to $\sum_{y \in \mathbb{L}} f(\alpha, y) \cdot s'(\alpha, y) = T_2$, which is a *(second) Y-dimension univariate sum-check*. To prove it, it suffices for the prover to show the existence of $g_2 \in \mathbb{F}_{\ell-1}[Y]$ and $h_2(Y)$ s.t.

$$f(\alpha, Y) \cdot s'(\alpha, Y) = Y \cdot g_2(Y) + T_2/\ell + Z_{\mathbb{L}}(Y) \cdot h_2(Y). \tag{6}$$

Given a random challenge β, the verifier only needs to query $f(\alpha, \beta), s'(\alpha, \beta), g_1(\alpha), h_1(\alpha), g_2(\beta), h_2(\beta)$. The proof size is hence $O(1)$ and irrelevant to ℓ.

Proving Equation (5) is also distribution-friendly. Given α, $\mathcal{P}_i$ can compute $g_{1,i}(\alpha), h_{1,i}(\alpha), f_i(\alpha), s'_i(\alpha)$ locally, and send these evaluations to $\mathcal{P}_0$. Next, $\mathcal{P}_0$ can compute $f(\alpha, Y)$ and invoke Equation (6) locally. The sub-prover complexity is only related to m, thus achieving linear speedup.

Based on the univariate sum-check [5] and Equations (4), (5) and (6), we have the following theorem. The formal protocol and security proofs for Lemma 2 and the distributed PIOP are given in Appendix B.2.

Theorem 2. *Given size-$m\ell$ vectors and ℓ provers, there exists a distributed inner product PIOP with perfect completness and a soundness error of $(2m + 2\ell - 2)/|\mathbb{F}|$. The sub-prover complexity is $O(m \log m)$, and the master prover complexity is $O(\ell \log \ell)$. The proof size and communication per prover are $O(1)$. The verifier complexity is $O(\log m\ell)$, and can be reduced to $O(1)$ via preprocessing.*[8]

Remark 1. Our distributed inner product PIOP may be of independent interest. First, based on Lemma 2, Equation (3) can be generalized as a bivariate sum-check PIOP to prove $\sum_{x \in \mathbb{H}} \sum_{y \in \mathbb{L}} f(x, y) = T$ via the double-dimension univariate sum-check. Its proof size is $O(1)$, in contrast to BC24 [10] with a proof size of $O(m + \ell)$. Our scheme also has faster prover complexity and verification.

Second, our PIOP can be naturally extended as a (non-distributed) batch scheme which proves ℓ sets of size-m inner product relations, *e.g.*, $\langle \boldsymbol{f}_i, \boldsymbol{s}_i \rangle = T_i$ for $i \in [\ell]$. A trivial approach to prove these relations is reducing ℓ relations into sum-check relations and then invoking an amortized univariate sum-check [5]. However, the proof size is $O(\ell)$ for sending evaluations of $O(\ell)$ polynomials to describe $\{\boldsymbol{f}_i, \boldsymbol{s}_i\}_{i \in [\ell]}$. Using our technique, the proof size is reduced to $O(1)$.

More details are given in Appendix B.3.

4 Succinct Distributed PIOP for R1CS

This section first constructs a distributed PIOP for R1CS with constant proof size on top of our distributed inner product PIOP, and then proposes a distributed preprocessing PIOP to achieve a constant verifier complexity.

[8] In this scheme, we make several assumptions about polynomial oracles. Given oracles $\{f_i(X)\}_{i \in [\ell]}$ or $\{g_i(X)\}_{i \in [\ell]}$, $\mathcal{P}_0$ can build oracles $f(X, Y) = \sum_{i \in [\ell]} f_i(X) L_i(Y)$ or $g(X) = \sum_{i \in [\ell]} g_i(X)$ in time $O(\ell)$. In addition, each polynomial oracle is assumed wich size $O(1)$. This is feasible by using the bivariate and univariate KZG PCSs.

4.1 Reducing R1CS into Distributed Inner Product PIOPs

We use the folklore techniques to transform an R1CS (*cf.*, Section 2) into the following two types of constraints:

1. linear constraints: witness vectors $\boldsymbol{w}, \boldsymbol{v} \in \mathbb{F}^{m\ell}$ satisfy $P_v \boldsymbol{w} = \boldsymbol{v}$ for public sparse matrix $P_v \in \mathbb{F}^{m\ell \times m\ell}$ and secret $\boldsymbol{v} \in \{\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}\}$, and
2. quadratic constraints: witness vectors $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c} \in \mathbb{F}^{m\ell}$ satisfy $\boldsymbol{a} \circ \boldsymbol{b} - \boldsymbol{c} = \boldsymbol{0}$.

We will show that each type of constraints can be reduced to the following Equation, which is similar to Equation (3) and thus can be proved distributively via our two-dimension univariate sum-check as in Section 3.2.

$$\sum\nolimits_{x\in\mathbb{H}} \Big(\sum\nolimits_{i\in[\ell]} f_i^{(1)}(x) \cdot f_i^{(2)}(x) \cdot R_i(z_1) - f_i^{(3)}(r) \cdot R_i(z_2) \Big) = 0, \tag{7}$$

where $f_i^{(1)}, f_i^{(2)}, f_i^{(3)} \in \mathbb{F}_m[X]$ are secret or public polynomials, $R_i(X) = X^{i-1}$ for all $i \in [\ell]$, and r, z_1, z_2 are constants possibly related to random challenges.

Linear constraints. For a linear constraint $P_v \boldsymbol{w} = \boldsymbol{v}$, we rely on a random challenge vector $\boldsymbol{r} = (1, r, \ldots, r^{m\ell-1})$ to transform it to $(\boldsymbol{r}^\top P_v)\boldsymbol{w} = \boldsymbol{r}^\top \boldsymbol{v}$ using the classic Freivalds' algorithm. Let $\boldsymbol{p} = \boldsymbol{r}^\top P_v$ be a size-$m\ell$ vector; then the above constraint turns into an inner product relation $\langle \boldsymbol{p}, \boldsymbol{w} \rangle = \langle \boldsymbol{r}, \boldsymbol{v} \rangle$. For distribution, we split $\boldsymbol{w}, \boldsymbol{r}, \boldsymbol{v}$ each into ℓ size-m sub-vectors $\{\boldsymbol{w}_i, \boldsymbol{r}_i, \boldsymbol{v}_i\}$, and split the matrix P_v into ℓ sub-matrices by columns: specifically, $\mathcal{P}_i$ holds $P_v^{(i)} \in \mathbb{F}^{m\ell \times m}$ consisting of m columns of P_v s.t. $P_v^{(i)}[j] = P_v[(i-1)m + j]$ for $j \in [m]$. Given the challenge $\boldsymbol{r}$, $\mathcal{P}_i$ can locally compute a sub-vector $\boldsymbol{p}_i = \boldsymbol{r}^\top P_v^{(i)}$. Then, $\mathcal{P}_i$ can invoke the distributed inner product PIOP to prove $\sum_{i\in[\ell]} \langle \boldsymbol{p}_i, \boldsymbol{w}_i \rangle = \sum_{i\in[\ell]} \langle \boldsymbol{r}_i, \boldsymbol{v}_i \rangle$.

For $i \in [\ell]$, we define coefficient-based polynomials $f_{p_i}(X), f_{w_i}(X)$ and $f_{v_i}(X)$ from $\boldsymbol{p}_i, \boldsymbol{w}_i$ and $\boldsymbol{v}_i$, respectively. Let $T_i = \langle \boldsymbol{r}_i, \boldsymbol{v}_i \rangle = \sum_{j\in[m]} r^{(i-1)m+j-1} \cdot \boldsymbol{v}_{i,j}$ be the inner-product result each P_i holds, where $\boldsymbol{v}_{i,j}$ is the j-th element of $\boldsymbol{v}_i$. Due to the coefficient-based construction, we have $f_{v_i}(r) = \sum_{j\in[m]} \boldsymbol{v}_{i,j} \cdot r^{j-1}$, and thus $T_i = r^{(i-1)m} \cdot f_{v_i}(r)$. Meanwhile, to prove $\langle \boldsymbol{p}_i, \boldsymbol{w}_i \rangle = T_i$, it suffices to prove $\sum_{x\in\mathbb{H}} f_{p_i}(x) \cdot f'_{w_i}(x) = m \cdot T_i$ by Equation (2) (where $f'_{w_i}(X)$ is built from $f_{w_i}(X)$ by Equation (1)). Thus, to prove $\sum_{i\in[\ell]} \langle \boldsymbol{p}_i, \boldsymbol{w}_i \rangle = \sum_{i\in[\ell]} \langle \boldsymbol{r}_i, \boldsymbol{v}_i \rangle$, we show

$$\sum\nolimits_{x\in\mathbb{H}} \sum\nolimits_{i\in[\ell]} f_{p_i}(x) \cdot f'_{w_i}(x) = m \cdot \sum\nolimits_{i\in[\ell]} T_i = m \cdot \sum\nolimits_{i\in[\ell]} r^{(i-1)m} \cdot f_{v_i}(r). \tag{8}$$

As $\mathbb{H}$ is of order m, the above equation then turns into $\sum_{x\in\mathbb{H}} (\sum_{i\in[\ell]} f_{p_i}(x) \cdot f'_{w_i}(x) - f_{v_i}(r) \cdot (r^m)^{i-1}) = 0$, which aligns with Equation (7) by setting $f_i^{(1)}(X) = f_{p_i}(X), f_i^{(2)}(X) = f'_{w_i}(X), f_i^{(3)}(r) = f_{v_i}(r), z_1 = 0$, and $z_2 = r^m$.

Remark 2. Equation (8) naturally falls back to a non-distributed PIOP for linear constraints. Compared with the PIOP for linear constraints in Aurora [5] which also uses the Freivalds' algorithm and univariate sum-check but adopts *evaluation-based* polynomial constructions, our reduction needs fewer FFTs with fewer polynomial interpolations, and further eliminates one oracle about $\boldsymbol{r}$, leading to superior efficiency in all aspects. Appendix C.1 presents more details.

Quadratic constraints. We split $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}$ into ℓ size-m sub-vectors, *i.e.*, $\boldsymbol{a} = (\boldsymbol{a}_1, \ldots, \boldsymbol{a}_\ell)$, and similarly for $\boldsymbol{b}$ and $\boldsymbol{c}$. Then, the constraint is transformed into $\boldsymbol{a}_i \circ \boldsymbol{b}_i = \boldsymbol{c}_i$ for all $i \in [\ell]$. Given a challenge vector $\boldsymbol{r} = (r^0, \ldots, r^{m-1})$,[9] we transform it into an inner product relation $\langle \boldsymbol{a}_i \circ \boldsymbol{r}, \boldsymbol{b}_i \rangle = \langle \boldsymbol{c}_i, \boldsymbol{r} \rangle$ for each $i \in [\ell]$; then with another linear combination, we further reduce the ℓ inner products into one equation by reusing the challenge r, *i.e.*, $\sum_{i\in[\ell]} r^{m(i-1)} \cdot \langle \boldsymbol{a}_i \circ \boldsymbol{r}, \boldsymbol{b}_i \rangle = \sum_{i\in[\ell]} r^{m(i-1)} \cdot \langle \boldsymbol{c}_i, \boldsymbol{r} \rangle$. Soundness is guaranteed as the exponent of r in each term varies.

Similarly to the linear constraints, we build coefficient-based polynomials $f_{a_i}(X), f_{b_i}(X)$ and $f_{c_i}(X)$ from $\boldsymbol{a}_i, \boldsymbol{b}_i$ and $\boldsymbol{c}_i$, respectively. Then, it holds that $\sum_{i\in[\ell]} r^{m(i-1)} \cdot \langle \boldsymbol{c}_i, \boldsymbol{r} \rangle = \sum_{i\in[\ell]} r^{m(i-1)} \cdot f_{c_i}(r)$. Further, by definition, we have $f_{a_i}(sX) = \sum_{j\in[m]} \boldsymbol{a}_{ij} \cdot r^{j-1} \cdot X^{j-1}$, whose coefficients are in accord with $\boldsymbol{a}_i \circ \boldsymbol{r}$. Define $f'_{b_i}(X)$ from $\boldsymbol{b}_i$ by Equation (1). Then, based on Equation (2), to prove $\sum_{i\in[\ell]} r^{m(i-1)} \cdot \langle \boldsymbol{a}_i \circ \boldsymbol{r}, \boldsymbol{b}_i \rangle = \sum_{i\in[\ell]} r^{m(i-1)} \cdot \langle \boldsymbol{c}_i, \boldsymbol{r} \rangle$, it suffices to show

$$\sum\nolimits_{x\in\mathbb{H}} \sum\nolimits_{i\in[\ell]} r^{m(i-1)} \cdot f_{a_i}(rx) \cdot f'_{b_i}(x) = m \cdot \sum\nolimits_{i\in[\ell]} r^{m(i-1)} \cdot f_{c_i}(r). \tag{9}$$

Moving the right hand to the left side, Equation (9) aligns with Equation (7) by setting $f_i^{(1)}(X) = f_{a_i}(rX), f_i^{(2)}(X) = f'_{b_i}(X), z_1 = z_2 = r^m, f_i^{(3)}(r) = f_{c_i}(r)$.

Combining two constraints. With the reductions above, for R1CS$(\mathbb{F}, m\ell, P_a, P_b, P_c; \boldsymbol{w}, \boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c})$ and for all $v \in \{a, b, c\}$, it suffices to prove

$$\begin{aligned} \sum\nolimits_{x\in\mathbb{H}} \Big(\sum\nolimits_{i\in[\ell]} f_{p_{v,i}}(x) \cdot f'_{w_i}(x) - f_{v_i}(r) \cdot R_i(r) \Big) = 0, \\ \sum\nolimits_{x\in\mathbb{H}} \Big(\sum\nolimits_{i\in[\ell]} f_{a_i}(rx) \cdot f'_{b_i}(x) \cdot R_i(r) - f_{c_i}(r) \cdot R_i(r) \Big) = 0. \end{aligned} \tag{10}$$

Note that two polynomial oracles for the secret vector $\boldsymbol{b}$ are needed, *i.e.*, $f'_{b_i}(X)$ and $f_{b_i}(X)$. We eliminate the oracle $f'_{b_i}(X)$ by building it from $f_{b_i}(X)$. By Equation (1), for any query x, it holds that $f'_{b_i}(x) = x^m \cdot f_{b_i}(x^{-1}) + (1 - x^m) \cdot f_{b_i}(0)$. Thus, the verifier can obtain $f'_{b_i}(x)$ by querying $f_{b_i}(x^{-1})$ and $f_{b_i}(0)$. That is to say, the secret polynomials are $\{f'_{w_i}(X), f_{a_i}(X), f_{b_i}(X), f_{c_i}(X)\}_{i\in[\ell]}$.

Using an amortized univariate sum-check [5], it is feasible to combine four relations in Equation (10) into one, and then construct a distributed PIOP for R1CS with constant proof size. Theorem 3 describes its properties. The formal protocol and security proof are given in Appendix C.2.

Theorem 3. *There exists a distributed PIOP for R1CS. Given ℓ provers, for a 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 $O(m\ell)$.*

4.2 Achieving Constant Verification via Distributed Preprocessing

The distributed PIOP in Theorem 3 has an $O(m\ell)$ verifier complexity for computing $f_p(\alpha, \beta) = \sum_{i\in[\ell]} f_{p_i}(\alpha) L_i(\beta)$ related to the public matrix P_v. We re-

[9] The challenge $\boldsymbol{r}$ would be extended into $(r^0, \ldots, r^{m\ell-1})$ later, which is the same as the one in linear constraints and can be reused.

duce the verifier complexity by delegating the oracle construction $f_p(X, Y)$ to a trusted indexer $\mathcal{I}$. The prover then proves to the verifier the validity of $f_p(\alpha, \beta)$.

A naive (but inefficient) solution works as follows. As $\boldsymbol{p} = \boldsymbol{r}^\top P_v$, $f_p(X, Y)$ depends on both the matrix P_v and the challenge r. Since the indexer $\mathcal{I}$ (in the preprocessing phase) does not know r in advance, we introduce an additional variable Z to describe it. Recall that each $\mathcal{P}_i$ holds the sub-matrix $P_v^{(i)} \in \mathbb{F}^{m\ell \times m}$ and sub-vector $\boldsymbol{p}_i = \boldsymbol{r}^\top P_v^{(i)}$; then we have the k-th ($k \in [m]$) element of $\boldsymbol{p}_i$, *i.e.*, $\boldsymbol{p}_{i,k}$, satisfies that $\boldsymbol{p}_{i,k} = \langle \boldsymbol{r}, P_v^{(i)}[k] \rangle = \sum_{j \in [m\ell]} r^{j-1} \cdot P_v^{(i)}[k][j]$. Then, we can construct a bivariate polynomial $f_{p_i}(Z, X)$ whose coefficients encode $\boldsymbol{p}_i$ with the variable Z describing r, *i.e.*, $f_{p_i}(Z, X) = \sum_{k \in [m]} (\sum_{j \in [m\ell]} Z^{j-1} \cdot P_v^{(i)}[k][j]) \cdot X^{k-1}$. However, its polynomial size is m^2, incurring an $O(m^2)$ sub-prover complexity.

To address this, we use the techniques in Marlin [14] to present P_v via low-degree polynomials describing its non-zero entries. The indexer $\mathcal{I}$ can then encode P_v in the preprocessing phase without knowing r. However, Marlin's techniques do not work in the distributed setting. We explain the reasons below.

Preliminaries in Marlin [14]. For a sparse $P_v \in \mathbb{F}^{m\ell \times m\ell}$, define an order-$k$ multiplicative subgroup $\mathbb{K}$ with generator κ, where $k \geq ||P_v||$. Define $\mathsf{val} \in \mathbb{F}_k[X]$ s.t. $\mathsf{val}(\kappa^{j-1})$ is the value of the j-th non-zero entry in P_v. The non-zero entries are arranged in some canonical order (*e.g.*, row-wise or column-wise). Following the same order, define $\mathsf{row} \in \mathbb{F}_k[X]$ (resp., col) s.t. $\mathsf{row}(\kappa^{j-1})$ (resp., $\mathsf{col}(\kappa^{j-1})$) be the row (resp., column) index (counting from 0) of the j-th non-zero entry in P_v. The key observation of Marlin is that the matrix P_v can be represented as a bivariate polynomial $P_v(X, Y) = \sum_{\kappa \in \mathbb{K}} u(X, \mathsf{row}(\kappa)) \cdot u(Y, \mathsf{col}(\kappa)) \cdot \mathsf{val}(\kappa)$ where $u \in \mathbb{F}_{m\ell, m\ell}[X, Y]$ is the formal derivative of the vanishing polynomial $Z_S(X)$ (*cf.*, see Marlin for more details). Given this representation, Marlin reduces the linear constraint into the validity of $P_v(\gamma_1, \gamma_2)$ and then $\mathsf{row}(\delta), \mathsf{col}(\delta), \mathsf{val}(\delta)$, where γ_1, γ_2 and δ are random challenges. In the distributed setting, although we can generalize $P_v(X, Y)$ with distributed variants of $\mathsf{row}(X), \mathsf{col}(X)$ and $\mathsf{val}(X)$, it is not natural to maintain the constant proof size as shown below.

Why Marlin does not work in the distributed setting. We first give some generalized algebraic notations. Given the sub-matrix $P_v^{(i)}$ held by $\mathcal{P}_i$, define a size-$\tilde{m}$ multiplicative subgroup $\mathbb{M}$ with generator θ s.t. $\tilde{m} \geq \max\{||P_v^{(i)}||\}$. For all $i \in [\ell]$, define polynomials $\{\mathsf{val}_i, \mathsf{row}_i, \mathsf{col}_i\} \in \mathbb{F}_{\tilde{m}}[X]$ similarly to those in Marlin, but corresponding to the sub-matrix $P_v^{(i)}$. Specifically, $\mathsf{row}(\theta^{j-1}) \in 0 \cup [m\ell - 1]$ is the row index of the j-th non-zero entry in $P_v^{(i)}$, and $\mathsf{col}(\theta^{j-1}) \in 0 \cup [m-1]$ is the column index of this entry. If $j > ||P_v^{(i)}||$, then set $\mathsf{val}_i(\theta^{j-1}) = 0$.

Still, $\mathcal{P}_i$ can build a bivariate polynomial to describe $P_v^{(i)}$, *e.g.*, $P_v^{(i)}(X, Y) = \sum_{x \in \mathbb{M}} u(X, \mathsf{row}_i(x)) \cdot u(Y, \mathsf{col}_i(x)) \cdot \mathsf{val}_i(x)$. However, even if there exists a method similar to Marlin to reduce the validity of $f_{p_i}(X)$ into the validity of $P_v^{(i)}(\gamma_1, \gamma_2)$ (which is not known), the verifier at least needs to query $\{P_v^{(i)}(\gamma_1, \gamma_2), \mathsf{row}_i(\delta), \mathsf{col}_i(\delta), \mathsf{val}_i(\delta)\}_{i \in [\ell]}$. This sacrifices the proof size from $O(1)$ to $O(\ell)$, due to the lack of an efficient aggregation approach for these polynomial evaluations.

Our approach. We start by using $\mathsf{row}_i(X), \mathsf{col}_i(X), \mathsf{val}_i(X)$ and r to *re-describe* $f_{p_i}(X)$. Our key insight is that, $f_{p_i}(X)$ can be expressed as $\sum_{x\in\mathbb{M}} \mathsf{val}_i(x)\cdot r^{\mathsf{row}_i(x)}\cdot X^{\mathsf{col}_i(x)}$ due to our coefficient-based representation: for each term $X^{\mathsf{col}_i(x)}$ where $\mathsf{col}_i(x)$ corresponds with some column index $j \in [m]$, its final coefficient in the polynomial representation is the random linear combination of all the non-zero entries $\mathsf{val}_i(x)$ within this column using the corresponding coefficient $r^{\mathsf{row}_i(x)}$. Based on this, define $f_p(X,Y) = \sum_{i\in[\ell]} f_{p_i}(X)L_i(Y)$ and then we have

$$f_p(\alpha,\beta) = \sum_{i\in[\ell]} f_{p_i}(\alpha)L_i(\beta) = \sum_{x\in\mathbb{M}}\sum_{i\in[\ell]} \mathsf{val}_i(x)\cdot r^{\mathsf{row}_i(x)}\cdot \alpha^{\mathsf{col}_i(x)}\cdot L_i(\beta). \tag{11}$$

Note that $r^{\mathsf{row}_i(X)}$ and $\alpha^{\mathsf{col}_i(X)}$ are exponential polynomials. To aggregate these polynomials, we introduce standard polynomials $A_i, B_i \in \mathbb{F}_{\tilde{m}}[X]$ to equivalently describe $r^{\mathsf{row}_i(X)}$ and $\alpha^{\mathsf{col}_i(X)}$, and then rely on Lemma 2 to reduce Equation (11) into a relation of constant-number bivariate polynomials.

Specifically, for any $i \in [\ell]$ and $x \in \mathbb{M}$, let $A_i(x) = r^{\mathsf{row}_i(x)}$ and $B_i(x) = \alpha^{\mathsf{col}_i(x)}$. Let $\mathsf{row}(X,Y), \mathsf{col}(X,Y), \mathsf{val}(X,Y)$ and $A(X,Y), B(X,Y)$ be the corresponding bivariate polynomials of $\mathsf{row}_i(X), \mathsf{col}_i(X), \mathsf{val}_i(X)$ and $A_i(X), B_i(X)$; let $L(X,Y) = \sum_{i\in[\ell]} L_i(X)L_i(Y)$ where $L_i(X), L_i(Y)$ are the i-th Lagrange polynomial of $\mathbb{L}$. Then by definition, for any $x \in \mathbb{M}$ and $y = \eta^{i-1} \in \mathbb{L}$, we have $r^{\mathsf{row}(x,y)} = r^{\sum_{i\in[\ell]}\mathsf{row}_i(x)L_i(y)} = r^{\mathsf{row}_i(x)} = A_i(x) = \sum_{i\in[\ell]} A_i(x)L_i(y) = A(x,y)$, and similarly, $\alpha^{\mathsf{col}(x,y)} = B(x,y)$. By Lemma 2, Equation (11) can be written as

$$f_p(\alpha,\beta) = \sum\nolimits_{x\in\mathbb{M}}\sum\nolimits_{y\in\mathbb{L}} \mathsf{val}(x,y)\cdot A(x,y)\cdot B(x,y)\cdot L(\beta,y). \tag{12}$$

Using the *two-dimension univariate sum-check* in Section 3.2, the validity of Equation (12) can be finally reduced into that of $\mathsf{val}(\delta,\zeta), A(\delta,\zeta), B(\delta,\zeta), L(\beta,\zeta)$ for random challenges δ and ζ. Thus, $O(\ell)$ evaluations on $\{\mathsf{val}_i, \mathsf{row}_i, \mathsf{col}_i\}_{i\in[\ell]}$ are aggregated into $O(1)$ ones of *bivariate* polynomials, and the proof size remains $O(1)$. As the validity proof of $f_p(\alpha,\beta)$ has to start after α and β are given, this adds two additional rounds to the original protocol.

New challenges and solutions. The above approach still faces an issue. During the preprocessing, the indexer $\mathcal{I}$ constructs oracles for $\mathsf{val}(X,Y)$, $\mathsf{row}(X,Y)$, and $\mathsf{col}(X,Y)$, but in the online phase, the verifier cannot directly compute $A(X,Y), B(X,Y)$ from $\mathsf{row}(X,Y), \mathsf{col}(X,Y)$ due to the exponential definitions. Thus, the prover must send oracles for $A(X,Y)$ and $B(X,Y)$ online, which requires the verifier to check their consistency with $\mathsf{row}(X,Y)$ and $\mathsf{col}(X,Y)$.

A typical approach to process non-linear functions is lookup schemes, which prove that elements are valid for belonging to a table. Recall that $B(x,y) = \alpha^{\mathsf{col}(x,y)}$ and $A(x,y) = r^{\mathsf{row}(x,y)}$. Thus it suffices for the prover to show

$$\begin{aligned}\{(\mathsf{col}(x,y), B(x,y))\}_{x\in\mathbb{M},y\in\mathbb{L}} &\subseteq \{(j-1, \alpha^{j-1})\}_{j\in[m]},\\ \{(\mathsf{row}(x,y), A(x,y))\}_{x\in\mathbb{M},y\in\mathbb{L}} &\subseteq \{(k-1, r^{k-1})\}_{k\in[m\ell]}.\end{aligned} \tag{13}$$

Existing lookup schemes [16,18,61] typically assume that the lookup table is correctly pre-constructed by a trusted party. In our setting, however, the tables

encode the sets $\{(j-1, \alpha^{j-1})\}_{j\in[m]}$ and $\{(k-1, r^{k-1})\}_{k\in[m\ell]}$, which depend on online challenges r, α and cannot be pre-constructed. This requires the verifier to perform additional validity checks on the tables in the online phase, but a full table scan would incur an unacceptable $O(m\ell)$ verifier cost. Additionally, the lookup scheme must be distribution-friendly: each prover $\mathcal{P}_i$ should participate only using its size-$\tilde{m}$ sub-polynomials (*e.g.*, $\mathsf{col}_i(X)$) for the linear speedup. A recent lookup scheme Lasso [50] allows tables to be constructed online via table decomposition. It is still unknown how to extend Lasso into a distributed version.

To resolve these issues, we build a distributed lookup PIOP via the following steps. First, we build an online table-validity check with sublinear verification in the table size. Second, we extend the state-of-the-art univariate lookup argument logup [26] to a bivariate and distributed version. Such a scheme has a sub-prover complexity linear in the table size, which violates the linear speedup when proving the row relation in Equation (13) with a size-$m\ell$ table. We then, finally, achieve linear speedup via table decomposition. Our techniques may inspire other lookup schemes without pre-determined tables or with distributed requirements.

Online table-validity check PIOP. We start by proving the column lookup relation $\{(\mathsf{col}(x,y), B(x,y))\}_{x\in\mathbb{M},y\in\mathbb{L}} \subseteq \{(j-1, \alpha^{j-1})\}_{j\in[m]}$. To facilitate verification, our insight is to make the table structured by equivalently proving $\{(\omega^{\mathsf{col}(x,y)}, B(x,y))\}_{x\in\mathbb{M},y\in\mathbb{L}} \subseteq \{(\omega^{j-1}, \alpha^{j-1})\}_{j\in[m]}$, where ω is the generator of $\mathbb{H}$.

We define a polynomial $T_{\mathsf{col}} \in \mathbb{F}_m[X]$ to describe the table s.t. $T_{\mathsf{col}}(\omega^{j-1}) = \alpha^{j-1}$ for $j \in [m]$. Notice that to prove its validity, it suffices to show $T_{\mathsf{col}}(1) = 1$ and further, $T_{\mathsf{col}}(\omega x) = \alpha \cdot T_{\mathsf{col}}(x)$ for $x \in \{\omega^{j-1}\}_{j\in[m-1]}$. A typical approach for the latter relation is the zero-check (*e.g.*, in PLONK [19]). However, it requires showing the existence of an additional secret quotient polynomial, and thus incurs additional overhead. We prove the validity of $T_{\mathsf{col}}(X)$ without introducing any additional polynomial, by instead showing the following relation

$$p(x) = T_{\mathsf{col}}(\omega x) - \alpha \cdot T_{\mathsf{col}}(x) + L_{\mathbb{H},m}(x) \cdot (\alpha^m - 1) = 0, \quad \forall x \in \mathbb{H}, \tag{14}$$

where $L_{\mathbb{H},m} \in \mathbb{F}_m[X]$ is the m-th Lagrange polynomial of $\mathbb{H}$. This holds as: for any $j \in [m-1]$, we have $L_{\mathbb{H},m}(x) = 0$, and Equation (14) indicates $T_{\mathsf{col}}(\omega x) = \alpha \cdot T_{\mathsf{col}}(x)$, and thus $T_{\mathsf{col}}(\omega^{m-1}) = \alpha^{m-1} \cdot T_{\mathsf{col}}(1)$; for $j = m$, we have $L_{\mathbb{H},m}(\omega^{m-1}) = 1$, and Equation (14) then becomes $T_{\mathsf{col}}(1) - \alpha \cdot (\alpha^{m-1} \cdot T_{\mathsf{col}}(1)) + (\alpha^m - 1) = 0$, indicating $T_{\mathsf{col}}(1) = 1$. Note that the polynomial $p(X)$ has at most m roots given $T_{\mathsf{col}}, L_{\mathbb{H},m} \in \mathbb{F}_m[X]$. As Equation (14) already gives m roots, $p(X)$ is a zero polynomial. Hence, to check Equation (14), it suffices to check if $p(\delta) = 0$ at a random δ by querying $T_{\mathsf{col}}(\delta)$ and $T_{\mathsf{col}}(\omega\delta)$. Different from the trivial linear-verification approach, our verifier complexity is $O(\log m)$ for computing $L_{\mathbb{H},m}(\delta)$.

Distributed bivariate lookup PIOP. We now extend the univariate lookup PIOP from logup [26] to a distributed bivariate one. We first review the main theorem used in logup [26] below in the univariate case.

Lemma 3 ([26]). *$\{a_i\}_{i\in[\tilde{m}]} \subseteq \{b_j\}_{j\in[m]}$ iff there exists $\{n_j\}_{j\in[m]}$ where $n_j \in 0 \cup [\tilde{m}]$ such that $\sum_{i\in[\tilde{m}]} \frac{1}{X+a_i} = \sum_{j\in[m]} \frac{n_j}{X+b_j}$. Here, for each b_j, the integer n_j describes the frequency number of elements in $\{a_i\}_{i\in[\tilde{m}]}$ equaling b_j.*

As we aim to prove lookup relations on tuples each consisting of two field elements (rather than a single one as in logup), we first use Reed-Solomon fingerprints [50] to transform each tuple to a field element. Specifically, for a tuple relation like $\{(v_i, V_i)\}_{i\in[\tilde{m}]} \subseteq \{(u_j, U_j)\}_{j\in[m]}$, given a random challenge ξ, the prover can equivalently show the single-element relation $\{\xi \cdot v_i + V_i\}_{i\in[\tilde{m}]} \subseteq \{\xi \cdot u_j + U_j\}_{j\in[m]}$ except with $m/|\mathbb{F}|$ soundness error.

For notation simplicity, for each $i \in [\ell]$, we define a size-$\tilde{m}$ polynomial $b_i(X)$ s.t. $b_i(x) = \omega^{\mathsf{col}_i(x)}$ for all $x \in \mathbb{M}$, and also a polynomial $b \in \mathbb{F}_{\tilde{m},\ell}[X, Y]$ s.t. $b(x, y) = \omega^{\mathsf{col}(x,y)}$ for $x \in \mathbb{M}$ and $y \in \mathbb{L}$. By definition, $b(X, Y) = \sum_{i\in[\ell]} b_i(X)L_i(Y)$. We now prove $\{(b(x, y), B(x, y))\}_{x\in\mathbb{M},y\in\mathbb{L}} \subseteq \{(\omega^{j-1}, \alpha^{j-1})\}_{j\in[m]}$ distributively. Define $n_{\mathsf{col}} \in \mathbb{F}_m[Y]$ s.t. for any $j \in [m]$, $n_{\mathsf{col}}(\omega^{j-1})$ is the frequency times of $b(x, y) = \omega^{j-1}$ for $x \in \mathbb{M}$ and $y \in \mathbb{L}$. From Lemma 3 and the Reed-Solomon fingerprints technique, it suffices to prove

$$\sum_{x\in\mathbb{M}} \sum_{y\in\mathbb{L}} \frac{1}{\gamma + \xi \cdot b(x, y) + B(x, y)} = \sum_{y\in\mathbb{H}} \frac{n_{\mathsf{col}}(y)}{\gamma + \xi \cdot y + T_{\mathsf{col}}(y)}. \tag{15}$$

In Equation (15), $b(X, Y)$ and $n_{\mathsf{col}}(Y)$ can be generated by the indexer. Given $b_i(X)$, $\mathcal{P}_i$ can locally compute $B_i(X)$, and all provers can generate $B(X, Y)$ jointly. Also, the size-m polynomial $T_{\mathsf{col}}(Y)$ can be computed locally by $\mathcal{P}_0$.

We follow logup to prove Equation (15) by showing its validity at a random $X = \gamma$. Specifically, define $g \in \mathbb{F}_{\tilde{m},\ell}[X, Y]$ s.t. $g(x, y) = \frac{1}{\gamma+\xi\cdot b(x,y)+B(x,y)}$ for $x \in \mathbb{M}$ and $y \in \mathbb{L}$. Define $h \in \mathbb{F}_{\tilde{m}}[Y]$ s.t. $h(y) = \frac{n_{\mathsf{col}}(y)}{\gamma+\xi\cdot y+T_{\mathsf{col}}(y)}$ for $y \in \mathbb{H}$. Then, Equation (15) turns into $\sum_{x\in\mathbb{M},y\in\mathbb{L}} g(x, y) = \sum_{y\in\mathbb{H}} h(y)$. To prove it, we argue that it suffices to prove $\tilde{m}\ell \cdot g(0, 0) = m \cdot h(0)$ (*cf.*, Appendix C.3). Here, $h(Y)$ can be generated by the master prover locally in time $O(m \log m)$, while the size-$O(m\ell)$ $g(X, Y)$ needs to be computed distributively. Note that $g(X, Y)$ can be written as $\sum_{i\in[\ell]} g_i(X)L_i(Y)$ s.t. $g_i \in \mathbb{F}_{\tilde{m}}[X]$ and $g_i(x) = \frac{1}{\gamma+\xi\cdot b_i(x)+B_i(x)}$ for $x \in \mathbb{M}$. As $\mathcal{P}_i$ holds $b_i(X)$ and $B_i(X)$, it can compute $g_i(X)$ locally, and all provers can then compute $g(X, Y)$ jointly while maintaining the linear speedup.

The remaining issue is that the prover needs to show the well-formation of $g(X, Y)$ and $h(Y)$. This is solvable by a zero-check. For $h(\mathrm{Y})$, the master prover can show the existence of $q_h(Y)$ s.t. $h(Y)\cdot(\gamma+\xi\cdot Y+T_{\mathsf{col}}(Y))-n_{\mathsf{col}}(Y) = q_h(Y)\cdot Z_{\mathbb{H}}(Y)$. However, for $g(X, Y)$, this is not natural due to the lack of (distributed) zero-check for bivariate polynomials. A direct approach is to show the existence of $q_g(X, Y), q_h(X, Y)$ s.t. $g(X, Y)\cdot(\gamma+\xi\cdot b(X, Y)+B(X, Y))-1 = q_g(X, Y)\cdot Z_{\mathbb{M}}(X) + q_h(X, Y)\cdot Z_{\mathbb{L}}(Y)$, but it is unknown how to compute the quotient polynomials distributively. Also, their sizes are larger than $m\ell$, violating the linear speedup.

We address this issue by reducing the bivariate relation back to a univariate one, which admits a size-m and distribution-friendly univariate quotient polynomial. To show the validity of $g(X, Y)$ s.t. $\{g(x, y) = \frac{1}{\gamma+\xi\cdot b(x,y)+B(x,y)}\}_{x\in\mathbb{M},y\in\mathbb{L}}$, we prove that for any $y = \eta^{i-1} \in \mathbb{L}$, there exists a quotient polynomial $q_{g,i}(X)$ s.t. $g(X, y)\cdot(\gamma+\xi\cdot b(X, y)+B(X, y))-1 = q_{g,y}(X)\cdot Z_{\mathbb{M}}(X)$. As $g(X, y) = g_i(X)$, $b(X, y) = b_i(X)$ and $B(X, Y) = B_i(X)$, we can ask each $\mathcal{P}_i$, who exactly holds $g_i(X), b_i(X)$ and $B_i(X)$, to compute $q_{g,i}(X)$ s.t. $g_i(X)\cdot(\gamma+\xi\cdot b_i(X)+B_i(X))-1 =$

$q_{g,i}(X) \cdot Z_{\mathbb{M}}(X)$, and then combine the ℓ quotient relations from all provers into a single one. Inspired by [7], given a challenge r, it suffices to show the existence of *a single* quotient polynomial $q_g(X) = \sum_{i\in[\ell]} r^{i-1} \cdot q_{g,i}(X)$ s.t.

$$\sum\nolimits_{i\in[\ell]} r^{i-1} \cdot \big(g_i(X) \cdot (\gamma + \xi \cdot b_i(X) + B_i(X)) - 1\big) = q_g(X) \cdot Z_{\mathbb{M}}(X). \tag{16}$$

Given a random challenge δ, the verifier can check Equation (16) at a random point $X = \delta$. We describe r^{i-1} using a polynomial $R_i(X) = X^{i-1}$ with $R(X, Y) = \sum_{i\in[\ell]} R_i(X)L_i(Y)$. Then by Lemma 2, this is further equivalent to

$$\sum\nolimits_{y\in\mathbb{L}} R(r, y) \cdot (g(\delta, y) \cdot (\gamma + \xi \cdot b(\delta, y) + B(\delta, y)) - 1) = q_g(\delta) \cdot Z_{\mathbb{M}}(\delta), \tag{17}$$

which can be proved distributively via a univariate sum-check over Y-dimension. This yields a distributed lookup PIOP for the column relation in the following theorem. The formal protocol and security proof are given in Appendix C.3.

Theorem 4. *There exists a distributed lookup PIOP to prove the first equation in Equation (13). The sub-prover complexity is $O(m \log m)$. The master prover complexity is $O(m \log m + \ell \log \ell)$. The proof size and amortized communication complexity are $O(1)$. The verifier complexity is $O(\log m\ell)$, and can be reduced to $O(1)$ via preprocessing.*

Handling the row relation. It remains to build a lookup PIOP for the row relation, *i.e.*, $\{(\mathsf{row}(x, y), A(x, y))\}_{x\in\mathbb{M},y\in\mathbb{L}} \subseteq \{(k-1, r^{k-1})\}_{k\in[m\ell]}$ in Equation (13).

Different from the size-m columns, each row in the table is of size $m\ell$, and a naive use of the above lookup PIOP leads to an unacceptable $O(m\ell \log m\ell)$ sub-prover complexity, which violates the linear speedup. To tackle this, our key insight is to decompose the size-$m\ell$ table into two smaller size-$\sqrt{m\ell}$ tables. As the number of sub-provers, *i.e.*, ℓ, is usually smaller than m, we have $\sqrt{m\ell} < m$. Hence the sub-prover complexity would still be bounded by $O(m \log m)$.

Specifically, we define polynomials $\mathsf{row}_{\mathsf{low},i}, \mathsf{row}_{\mathsf{high},i} \in \mathbb{F}_{\tilde{m}}[X]$ that maps $\mathbb{M}$ to $0 \cup [\sqrt{m\ell} - 1]$. Then $\mathsf{row}_i(x)$ can be expressed as $\mathsf{row}_i(x) = \mathsf{row}_{\mathsf{high},i}(x) \cdot \sqrt{m\ell} + \mathsf{row}_{\mathsf{low},i}(x)$ for all $x \in \mathbb{M}$. By definition, this decomposition is unique. Now we define bivariate polynomials $\mathsf{row}_{\mathsf{high}}(X, Y) = \sum_{i\in[\ell]} \mathsf{row}_{\mathsf{high},i}(X)L_i(Y)$ and $\mathsf{row}_{\mathsf{low}}(X, Y)$ similarly. Then we have $r^{\mathsf{row}(X,Y)} = (r^{\sqrt{m\ell}})^{\mathsf{row}_{\mathsf{high}}(X,Y)} \cdot r^{\mathsf{row}_{\mathsf{low}}(X,Y)}$. Thus, the row lookup relation is transformed into two independent sub-relations

$$\begin{aligned} \{\mathsf{row}_{\mathsf{high}}(x, y), A_{\mathsf{high}}(x, y)\}_{x\in\mathbb{M},y\in\mathbb{L}} &\subseteq \{(j-1, (r^{\sqrt{m\ell}})^{j-1})\}_{j\in[m]}, \\ \{\mathsf{row}_{\mathsf{low}}(x, y), A_{\mathsf{low}}(x, y)\}_{x\in\mathbb{M},y\in\mathbb{L}} &\subseteq \{(j-1, r^{j-1}\}_{j\in[m]}, \end{aligned} \tag{18}$$

which can be proved distributively in a similar way to the column relation.

Putting everything together. Using the distributed lookup PIOP, we build a distributed PIOP for R1CS with constant proof size and verification as shown below. The formal protocol and security proof are given in Appendix C.3.

Theorem 5. *There exists a distributed PIOP for R1CS. Given ℓ provers, for a size-$m\ell$ R1CS, the sub-prover complexity is $O(m \log m)$. The master prover complexity is $O(m \log m + \ell \log \ell)$. The proof size or amortized communication is $O(1)$. Verifier complexity is $O(\log m\ell)$, and can be $O(1)$ via preprocessing.*

An "making-even" algorithm for R1CS matrices. The sub-prover complexity of our distributed PIOP for R1CS, to be more specifically, is $O(\tilde{m} \log \tilde{m})$ where $\tilde{m} = O(m)$ is the maximum non-zero entries in any sub-matrix. Ideally, we hope the non-zero entries are evenly distributed to each sub-matrix. In practice, however, a skewed distribution may make some sub-matrices "dense", leading to a large concrete $\tilde{m}$ and worsening the concrete prover efficiency.

To solve this, we observe that given a fixed circuit, the non-zero entries' locations are only determined by gate indexes, which can be manually adjusted. Hence, it is feasible to run a public "making-even" algorithm to adjust their locations for buiding more even sub-matrices before proving. We propose such an algorithm in Appendix D. This "making-even" algorithm is important for Soloist's full linear speedup over general circuits, which may also be helpful for other distributed SNARKs targeting R1CS.

Remark 3. The indexer's computation can be distributed similarly to the proving phase. This is reasonable, as the verifier is assumed to delegate the public computation to a trusted indexer, who is naturally expected to be powerful in computational resources. Also, the preprocessing can be time-consuming in practice. For example, the indexer time is nearly the same as the prover time in Marlin [14]. Appendix C.3 presents more details.

5 Efficient Batch for Bivariate KZG and its Distribution

To instantiate our distributed PIOP with bivariate polynomials, we give a bivariate batch KZG allowing to open multiple bivariate polynomials at multiple points in Section 5.1, and then present its distributed version in Section 5.2.

5.1 Batch Scheme for Bivariate KZG

Previous batch scheme for univariate polynomials. We first recall the batch scheme [7] for univariate KZG [28] as below.

Opening a single polynomial at multiple points. Given a secret polynomial $f(X)$, to prove the validity of $\{f(\alpha)\}_{\alpha \in A}$ over a size-t set A, $\mathcal{P}$ and $\mathcal{V}$ compute a public size-t polynomial $r(X)$ s.t. $r(\alpha) = f(\alpha)$ for all $\alpha \in A$. Then, $\{f(\alpha)\}_{\alpha \in A}$ are valid iff each α is a root of the polynomial $f(X) - r(X)$. Thus, it suffices for the prover to show the existence of a *single* polynomial $q(X)$ s.t. $f(X) - r(X) = q(X) \cdot Z_A(X)$. The proof size is constant and irrelevant to the point number t.

Opening multiple polynomials at multiple points. Assume there are n polynomials $f_1(X), \ldots, f_n(X)$, where for simplicity, each $f_k(X)$ $(k \in [n])$ is evaluated over a set A_k of size t. For $k \in [n]$, $\mathcal{P}$ and $\mathcal{V}$ compute a public degree-$(t-1)$

polynomial $r_k(X)$ s.t. $r_k(\alpha) = f_k(\alpha)$. Similarly, there exists a polynomial $q_k(X)$ s.t. $f_k(X) - r_k(X) = q_k(X) \cdot Z_{A_k}(X)$. To batch these n quotient relations, Boneh *et al.* [7] define $A = \bigcup_{k\in[n]} A_k$ and perform a linear combination with a random challenge γ to obtain a *single* quotient polynomial $q(X)$ s.t. $\sum_{k\in[n]} \gamma^{k-1} \cdot (f_k(X) - r_k(X)) \cdot Z_{A\setminus A_k}(X) = q(X) \cdot Z_A(X)$, where $Z_{A\setminus A_k}(X) = Z_A(X)/Z_{A_k}(X)$, and $q(X) = \sum_{k\in[n]} \gamma^{k-1} \cdot q_k(X)$. This suggests $Z_{A_k}(X) \mid f_k(X) - r_k(X)$ for $k \in [n]$. The proof size remains constant and irrelevant to n or t.

Directly proving the validity of $q(X)$ has a verifier complexity of $O(n)$ pairings. Boneh *et al.* [7] further reduce it to $O(n)$ exponentiations (which are cheaper) and $O(1)$ pairings. We omit it here and kindly refer to [7] for details.

Review of the bivariate KZG [42]. The bivariate KZG relies on a key insight: given a secret polynomial $f(X, Y)$, if the claimed evaluation $f(\alpha, \beta)$ is valid, then (α, β) must be a root of the polynomial $f(X, Y) - f(\alpha, \beta)$, and hence there must exist quotient polynomials $p \in \mathbb{F}_{m-1,\ell}[X, Y]$ and $q \in \mathbb{F}_{m,\ell-1}[X, Y]$[10] s.t.

$$f(X, Y) - f(\alpha, \beta) = p(X, Y) \cdot (X - \alpha) + q(X, Y) \cdot (Y - \beta). \tag{19}$$

Our batch bivariate KZG over Cartesian-product points. We first consider the case where each polynomial $f(X, Y)$ is evaluated over the Cartesian product of two size-t sets A and B, *e.g.*, $(\alpha, \beta) \in A \times B$ for $X = \alpha$ and $Y = \beta$. We then extend the PCS over non-Cartesian-product points, and finally over points with shared Y-dimension ones, which is used in Soloist.

Opening a single polynomial at multiple points. Inspired by the univariate case, we define a public polynomial $r \in \mathbb{F}_{t,t}[X, Y]$ s.t. $r(\alpha, \beta) = f(\alpha, \beta)$ for all $(\alpha, \beta) \in A \times B$. Then, the key observation is that, if all $\{f(\alpha, \beta)\}_{(\alpha,\beta)\in A\times B}$ are valid evaluations, then all $(\alpha, \beta) \in A \times B$ are the roots of the bivariate polynomial $f(X, Y) - r(X, Y)$. Then, it suffices for the prover to show the existence of *two* quotient polynomials $p \in \mathbb{F}_{m-t,\ell}[X, Y]$ and $q \in \mathbb{F}_{m,\ell-t}[X, Y]$ s.t.

$$f(X, Y) - r(X, Y) = p(X, Y) \cdot Z_A(X) + q(X, Y) \cdot Z_B(Y). \tag{20}$$

Opening multiple polynomials at multiple points. Given $\{f_k(X, Y)\}_{k\in[n]}$, assume each $f_k(X, Y)$ is evaluated over $A_k \times B_k$ for size-t sets A_k and B_k. Let $A = \bigcup_{k\in[n]} A_k$ and $B = \bigcup_{k\in[n]} B_k$. We then expect to build a public $r_k(X, Y)$ for each $f_k(X, Y)$ and combine multiple equations as in the univariate case. However, it is not natural as each equation has diverse $Z_{A_k}(X)$ and $Z_{B_k}(Y)$, and we cannot find a *common* multiple (like $Z_A(X)$ in the univariate case) to combine them.

To address this, we introduce an intermediate polynomial to split the quotient polynomials over the two dimensions. For each $k \in [n]$, define an auxiliary polynomial $s_k \in \mathbb{F}_{t,\ell}[X, Y]$ s.t. $s_k(\alpha, Y) = f_k(\alpha, Y)$ for all $\alpha \in A_k$. [11] By definition, $s_k(X, Y)$ also satisfies $s_k(X, \beta) = r_k(X, \beta)$ for all $\beta \in B_k$. Thus, Equation (20) can be divided into two equations by introducing $s(X, Y)$ where one is

[10] Note that there exist multiple choices of valid $p(X, Y)$ and $q(X, Y)$.

[11] $s_k(X, Y)$ can be computed in time $O(tm\ell + t^2\ell)$ given $f_k(X, Y)$ (*cf.*, Appendix E.2).

only related to $Z_{A_k}(X)$, and the other is only related to $Z_{B_k}(Y)$. This enables combining $p_k(X,Y)$ or $q_k(X,Y)$ separately. Specifically, we have

$$\begin{aligned}\sum\nolimits_{k\in[n]} \gamma^{k-1}\cdot(f_k(X,Y)-s_k(X,Y))\cdot Z_{A\setminus A_k}(X) &= p(X,Y)\cdot Z_A(X)\\ \sum\nolimits_{k\in[n]} \gamma^{k-1}\cdot(s_k(X,Y)-r_k(X,Y))\cdot Z_{B\setminus B_k}(Y) &= q(X,Y)\cdot Z_B(Y).\end{aligned} \tag{21}$$

Following Equation (21), we can build a batch scheme for bivariate KZG with a proof size linear in the number of polynomials (as the prover needs to commit all $s_k(X,Y)$). Verification requires $O(n)$ pairings, and similarly to Boneh *et al.* [7], we give a method to reduce it to $O(n)$ exponentiations and $O(1)$ pairings.

Theorem 6. *There exists a batch bivariate KZG as in Protocol 8, Appendix E.2. Committing n polynomials of $\mathbb{F}_{m,\ell}[X,Y]$ costs $O(nm\ell)\,\mathbb{G}$. Opening t^2 points on each polynomial costs $O(m\ell+nt\ell)\,\mathbb{G}+O(ntm\ell+nt^2\ell+nt^3)\,\mathbb{F}$. The verifier complexity is $O(n)\,\mathbb{G}+4\,P+O(nt^2\ell+nt^3)\,\mathbb{F}$. The proof size is $(n+4)\,|\mathbb{G}|$.*

Appendix E.2 gives the formal protocol and proofs.

Generalization to non-Cartesian-product points. We generalize the above PCS to a more common setting, where the evaluation points are not necessarily from a Cartesian product of two sets, as long as they differ from each other over one dimension (which is also one assumption in Boomy [31]).

WLOG, assume the t evaluation points are $\{(\alpha_i,\beta_i)\}_{i\in[t]}$, where $\alpha_i\neq\alpha_j$ for all $i,j\in[t]$. We first discuss the single-polynomial case. Now the main challenge is that the verifier cannot obtain a public $r\in\mathbb{F}_{t,t}[X,Y]$ as in Equation (20) from only t evaluations. Thus, we instead let the prover extend the t points into t^2 points over the corresponding Cartesian-product set, then compute and commit to the *secret* $r(X,Y)$. Now the verifier needs to additionally check the validity of $r(X,Y)$. To this end, we introduce two public univariate polynomials that encode necessary information from the given t points and evaluations to help check $r(X,Y)$. Specifically, we define $u,v\in\mathbb{F}_t[X]$ s.t. $u(\alpha_i)=\beta_i$ and $v(\alpha_i)=f(\alpha_i,\beta_i)$. Then there must exist secret quotient polynomials $p(X,Y)$ and $q(X)$ s.t. $r(X,Y)-v(X)=(Y-u(X))\cdot p(X,Y)+q(X)\cdot Z_A(X)$. For the multiple-polynomial case, we further introduce an auxiliary polynomial $R(X)=r(X,u(X))$ to split $p(X,Y)$ and $q(X)$ separately into two equations. The proof size increases to $(3n+6)\,|\mathbb{G}|$. Appendix E.3 presents more details.

Reducing the proof size with shared Y-dimension point. We notice that the proof size can be further reduced if the Y-dimension of the evaluation points are all the same, which is exactly the case in our PIOP.[12] The key idea is to reduce *different* evaluations on multiple bivariate polynomials into *the same* evaluations on these polynomials, which makes it easier to batch.

Specifically, for $k\in[n]$, any polynomial $f_k(X,Y)$ evaluated at A_k of size t over X and $B_k=\{\beta\}$ over Y, there exist $r_k\in\mathbb{F}_t[X]$ and $q_k(X)$ s.t. $f_k(X,\beta)-$

[12] For example, recall that in Equation (10), the verifier would at least query $f_{p_{v,i}}(\alpha,\beta)$, $f_{a_i}(r\alpha,\beta)$ and $R(r,\beta)$, all with a shared β.

$r_k(X) = q_k(X) \cdot Z_{A_k}(X)$, where $r_k(X)$ is publicly defined by $\{(\alpha, f_k(\alpha, \beta))\}_{\alpha \in A_k}$. When handling n polynomials $\{f_k(X, Y)\}_{k \in [n]}$, this quotient relation can be batched via a random linear combination over a challenge γ s.t. $\sum_{k \in [n]} \gamma^{k-1} \cdot (f_k(X, \beta) - r_k(X)) \cdot Z_{A \setminus A_k}(X) = q(X) \cdot Z_A(X)$. To prove it, the prover commits all $\{f_k(X, Y)\}_{k \in [n]}$ and $q(X)$; the verifier opens them and checks if this equation holds at a random $X = \eta$ by opening all $\{f_k(\eta, \beta)\}_{k \in [n]}$ and $q(\eta)$. As all $f_k(X, Y)$ are opened at the same point, they have common vanishing terms (*i.e.*, $(X - \eta)$ and $(Y - \beta)$) and are natural to batch. The proof size is reduced to $4\,|\mathbb{G}| + (n + 1)\,|\mathbb{F}|$. Appendix E.4 gives the construction details and security proofs.

5.2 Distributed Batch Scheme for Bivariate KZG

This section extends the batch bivariate KZG into a distributed version, where the Lagrange-based SRS is used other than monomial-based. Each prover $\mathcal{P}_i$ holds $f_i(X)$ and an SRS related to $\{X^{j-1} \cdot L_i(Y)\}_{j \in [m]}$, enabling it to compute the commitment to $f_i(X)L_i(Y)$. The master prover holds an SRS related to $\{L_i(Y)\}_{i \in [\ell]}$, enabling it to build the commitment to $f(\alpha, Y)$ given $\{f_i(\alpha)\}_{i \in [\ell]}$.

To build a distributed batch bivariate KZG, recall that the (non-batch) Pianist [36] uses a distribution-friendly choice of $p(X, Y)$ and $q(X, Y)$ in Equation (19) by setting $p(X, Y) = \frac{f(X,Y) - f(\alpha,Y)}{X - \alpha}$ and $q(X, Y) = \frac{f(\alpha,Y) - f(\alpha,\beta)}{Y - \beta}$. By definition, $q(X, Y)$ is only related to the variable Y (*i.e.*, $q(Y)$). Further, given $\{f_i(\alpha)\}_{i \in [\ell]}$, $q(Y)$ can be computed by $\mathcal{P}_0$ locally in $O(\ell \log \ell)$ time. Meanwhile, we have $p(X, Y) = \sum_{i \in [\ell]} \frac{f_i(X) - f_i(\alpha)}{X - \alpha} \cdot L_i(Y)$. Hence, $p(X, Y)$ can be obtained by $\mathcal{P}_0$ locally aggregating all terms with each $\frac{f_i(X) - f_i(\alpha)}{X - \alpha}$ computed by $\mathcal{P}_i$.

With this idea in mind, we generalize the variant of Equation (20) (after introducing $s(X, Y)$) into a distributed version. In the second half (*i.e.*, $s(X, Y) - r(X, Y)$), as $s \in \mathbb{F}_{t,\ell}[X, Y]$ and $r \in \mathbb{F}_{t,t}[X, Y]$, we have $q \in \mathbb{F}_{t,\ell-1}[X, Y]$, which can be computed by $\mathcal{P}_0$ solely in $O(t^3 + t^2\ell)$ time. For the first half (*i.e.*, $f(X, Y) - s(X, Y)$), assume $A = \{\alpha_1, \ldots, \alpha_t\}$. As $s(\alpha, Y) = f(\alpha, Y)$ for $\alpha \in A$, we have $s(X, Y) = \sum_{j \in [t]} f(\alpha_j, Y) \cdot N_j(X)$, where $N_j \in \mathbb{F}_t[X]$ is the j-th Lagrange polynomial of A. In our case, t is a constant, so directly computing $N_j(X)$ is efficient. By the properties of $N_j(X)$, we have $\sum_{j \in [t]} N_j(x) = 1$ for any x, and

$$\begin{aligned} f(X, Y) - s(X, Y) &= \sum\nolimits_{j \in [t]} f(X, Y) \cdot N_j(X) - \sum\nolimits_{j \in [t]} f(\alpha_j, Y) \cdot N_j(X) \\ &= \sum\nolimits_{j \in [t]} \big(\sum\nolimits_{i \in [\ell]} (f_i(X) - f_i(\alpha_j)) \cdot L_i(Y) \big) \cdot N_j(X) \\ &= \sum\nolimits_{i \in [\ell]} \big(\sum\nolimits_{j \in [t]} (f_i(X) - f_i(\alpha_j)) \cdot N_j(X) \big) \cdot L_i(Y). \end{aligned} \tag{22}$$

For any j with fixed i, we have $(X - \alpha_j) \mid (f_i(X) - f_i(\alpha_j))$ and $Z_{A \setminus \alpha_j}(X) \mid N_j(X)$ by the properties of $N_j(X)$. Hence, it holds that $Z_A(X) \mid (\sum_{j \in [t]} (f_i(X) - f_i(\alpha_j)) \cdot N_j(X))$, and further, $p_i(X) = (\sum_{j \in [t]} (f_i(X) - f_i(\alpha_j)) \cdot N_j(X)) \cdot 1/Z_A(X)$ is a polynomial. Then, $p(X, Y)$ can be generated distributively by $\mathcal{P}_i$ locally computing $p_i(X)$ and $\mathcal{P}_0$ aggregating them as $p(X, Y) = \sum_{i \in [\ell]} p_i(X) L_i(Y)$.

Extending Equation (22) into a batch version is natural following how Equation (21) extends Equation (20). With this idea, we can build a distributed batch scheme for bivariate KZG PCS, which is presented in Protocol 10, Appendix E.5.

Theorem 7. *There exists a distributed batch bivariate KZG. When opening t^2 points on each of the t bivariate size-(m, ℓ) polynomials, the sub-prover complexity is $O(m + nt)\,\mathbb{G} + O(ntm + nt^2)\,\mathbb{F}$. The master prover complexity is $O(t\ell)\,\mathbb{G} + O(nt^2\ell + nt^3)\,\mathbb{F}$. The communication per prover is $O(n)\,|\mathbb{G}|$. The proof size and verifier complexity remain the same as in Theorem 6.*

Reducing the SRS size per prover. In our distributed PCSs, the total SRS size is $O(m\ell)$. However, as each sub-prover $\mathcal{P}_i$ only precesses the i-th part of bivariate polynomials with term $L_i(Y)$, it only needs to store a size-$O(m)$ SRS.

6 Construction of Soloist

Combining our distributed PIOP for R1CS and the distributed batch bivariate KZG, we build our distributed SNARK Soloist with the following properties.

Theorem 8. *There exists a distributed SNARK such that given a size-$O(m\ell)$ R1CS and ℓ provers, each prover needs to store an SRS of size $O(m)$, and the prover complexity is $O(m)\,\mathbb{G} + O(m \log m)\,\mathbb{F}$. Choosing one prover as the master prover, its time complexity is $O(m + \ell)\,\mathbb{G} + O(m \log m + \ell \log \ell)\,\mathbb{F}$. The proof size and amortized communication complexity are $O(1)\,|\mathbb{G}|$. The verifier complexity is $O(\log m\ell)\,\mathbb{F}$, and can be reduced to $O(1)\,\mathbb{G}$ via preprocessing.*

We provide a proof sketch here. Completeness holds directly. Complexities are implied by that of our distributed PIOP (*cf.*, Theorem 5) and our distributed PCS (*cf.*, Theorem 7), where the polynomial number n and the evaluation point number t are both constants in our PIOP. For knowledge soundness, it has been proven that if a PCS satisfies knowledge soundness and a PIOP has (knowledge) soundness, then the compiled interactive argument from "PIOP + PCS" inherits this property [11]. The security proof hence follows from the soundness of our distributed PIOP and the knowledge soundness of our distributed PCS.

Universal & updatable setup and sublinear SRS. The SRS of Soloist, which is used in our distributed bivariate KZG, is determined by the system parameter, including the R1CS size and the number of sub-provers. In addition, the standard SRS (with only monomials) is updatable [25], and the Lagrange-based SRS (as used in our PCS) can be constructed from the standard SRS. Thus, our setup is universal and updatable [25]. In Soloist, each sub-prover handles size-$O(m)$ univariate polynomials or size-$O(m\ell)$ bivariate ones. Thanks to our distributed PCS, each sub-prover only needs to store an SRS of size $O(m)$.

Accountability. Distributed SNARKs typically assume honest sub-provers, but malicious ones may submit invalid sub-proofs. Recent works [36,54] introduce accountability, allowing an honest master prover to detect and isolate misbehaving sub-provers. Soloist achieves the same accountability with minimal modifications

and no asymptotic overhead. This is feasible as each sub-proof is an independent proof for some public relation. Consequently, each sub-prover can generate, alongside its main sub-proof, an accountability sub-proof to the master prover, enabling on-demand verification Appendix F gives more details.

Adding zero knowledge. We currently do not add zero knowledge, which suffices for blockchain scalability as in [36,57]. We additionally achieve zero knowledge following [14,58]. More details can be found in Appendix G.

7 Implementation and Evaluation

7.1 Experimental Setup

Hardware and software. We implement Soloist using $\sim 10,000$ lines of codes based on the arkworks ecosystem in Rust.[13] All experiments are run on up to 64 machines within one cluster, each with 16 cores and 256GB of RAM. Each machine acts as a sub-prover, and we randomly choose one sub-prover to act as the master prover. The elliptic curve is BN254 except explicitly stated.

Experimental goals. We aim to demonstrate the following benefits of Soloist.

1. Soloist has full linear speedup over non-distributed schemes. We choose to compare with *Marlin* [14], the state-of-the-art SNARK for R1CS also with constant proof size and verification. PLONK [19] is omitted as it is specified for the Plonk constraint system and provides trade-offs with Marlin.
2. Soloist has a faster prover time than the Plonk-targeted *Pianist* [36] for R1CS-friendly applications. Soloist shows trade-offs with Plonk-targeted HyperPianist [33] among the prover time and all other figures. We omit Cirrus [54] (with the same asymptotic complexities as HyperPianist) as it is unreleased.
3. Soloist outperforms R1CS-targeted distributed SNARKs over (nearly-)data-parallel and general circuits. We compare with state-of-the-art *Hekaton* [46], which specifically fits (nearly-)data-parallel circuits. DIZK [55] is omitted, as its prover time is $60\times$ slower than Marlin in the non-distributed setting.

Benchmarks and modifications. We implement or use multi-core parallelization for Soloist, Marlin, Pianist and HyperPianist. As Hekaton does not support multi-core parallelization, to conduct a fair comparison, we manually run 16 Hekaton programs with 16 cores in parallel on each machine.

Hekaton only offers benchmarks for nearly-data-parallel circuits, which can be divided into sub-circuits with very few shared wires. Its performance worsens significantly when the proportion of shared wires increases. Thus, to run Hekaton over general circuits, we design a circuit division algorithm with the goal of *minimized* shared wires. We present the algorithm in Appendix I, which may be of independent interest to implement Hekaton over general circuits.

Metrics. We evaluate the indexer (preprocessing) time, prover time, verifier time, proof size, amortized communication overhead, and amortized memory

[13] We plan to open-source our implementation soon.

cost. We omit to report the memory cost when it is several GBs (*e.g.*, < 6GB), as it may be inaccurate due to noises such as program building.

Circuit choice. We run two kinds of circuits in our experiments. The first one is an R1CS-friendly zkRollup circuit used for blockchain scalability [36]. Same as Pianist, we extract it from Hermez[14], use Circom [3] to compile it into R1CS, and further transform it into Plonk using the compiler in Pianist. The R1CS has $229,847$ ($\sim 2^{18}$) constraints for a batch of 3 transactions, and Plonk has $2,215,685$ ($\sim 2^{21}$) constraints, both in the same order as in Pianist.[15]

The second one is a nearly-data-parallel Merkle tree hash membership circuit applied in verifiable key directory [46]. We use the circuit from Hekaton's benchmark, which considers a SHA-256-based Merkle tree of depth 128 and verifies *diverse* batches of update operations on the leaves. Same as Hekaton, we use 7 kinds of sub-circuits to compose the target circuit, each having $\sim 2^{14}$ constraints. The number of shared wires among sub-circuits is at most 4, so we call the circuit nearly-data-parallel. Our largest tested circuit has $\sim 2^{27}$ constraints.

7.2 Evaluation of Non-Distributed Schemes

We implement and evaluate our IPA from our inner product PIOP (*cf.*, Section 3.1) and the batch KZG [7] (*cf.*, Section 5.1). Experiments show that compared with IPAs from existing PIOPs (*cf.*, Table 1) and batch KZG, our IPA has a 20% faster prover, 5%-80% faster verifier, and 30%-200% smaller proof size. In addition, compared with PST13 [42] which directly opens all points, our batch bivariate KZG scheme has a 7-10× faster prover time, 5× faster verifier time, and a 6-7× smaller proof size. Appendix H.1 presents the details.

7.3 Evaluation of Distributed SNARKs

Comparison with Marlin. Fig. 1 compares Soloist and Marlin on random R1CS adapted from Marlin's implementations. The elliptic curve is BLS12-381.

Soloist shows a linear speedup. Utilizing ℓ provers, its indexer time, prover time, and memory cost are $1.5\ell\times$ faster, $\ell\times$ faster, and $\ell\times$ smaller than Marlin, respectively. Marlin fails to run for size-2^{26} circuits due to insufficient memory, while the *scalable* Soloist operates successfully with ≥ 2 machines.

For other metrics, our verifier time is 4.6ms, slightly faster than Marlin's 6.5ms. The communication per prover is 8.7KB. One trade-off is that Soloist has a 13.2KB proof size, which is 20× larger than Marlin. However, it is at least competitive among distributed SNARKs, which we delve into below.

Comparison with Pianist. We compare Soloist with Pianist over zkRollup circuits. For such applications, a constant proof size and verifier complexity

[14] https://github.com/hermeznetwork/circuits, last accessed 2024-11-16.

[15] To prove zkRollup transactions with $261,833$ constraints encoded as an R1CS, Pianist transforms it into a Plonk with $2,544,486$ constraints before proving. Hekaton also accounts this statement expansion from IR transformation when compared with Pianist. It may be reduced by optimizing circuit-to-Plonk compilers.

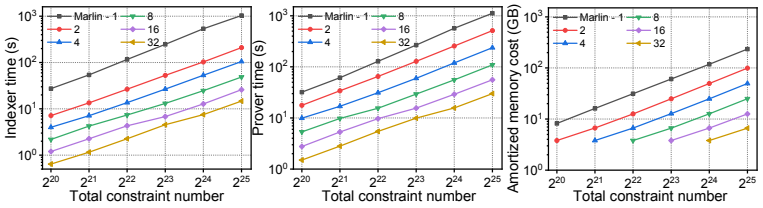

Fig. 1. Comparison of Marlin and Soloist with 2-32 provers

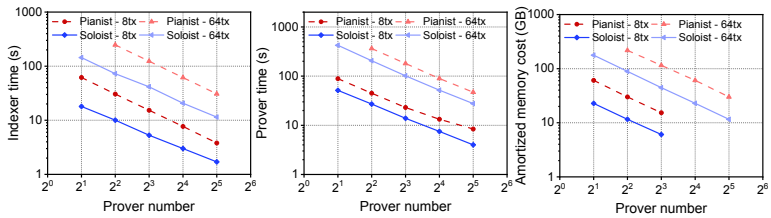

Fig. 2. Comparison of Soloist and Pianist for proving zkRollups (in batches of 3)

are important for blockchain scalability. Also, the provers (*i.e.*, miners) can be distributed [36] all over the world, so the communication overhead also matters.

Fig. 2 shows the concrete performances with circuits verifying 8 ("-8tx") or 64 ("-64tx") batches of transactions. With 2 - 2^6 provers, our evaluation covers data-parallel and general circuits, where each prover handles 2^{-2} - 2^5 batches of transactions. For general circuits, we use the algorithm in to make the R1CS matrices "even". In Fig. 2, each prover in Pianist can handle up to 16 batches of transactions, as for more batches, the constraint number would exceed the largest multiplicative subgroup size in BN254. In contrast, Soloist can handle up to 320 batches of transactions per prover in theory if given sufficient memory.

As shown, the indexer and prover times of Soloist are 2.8× and 1.8× faster than Pianist, respectively. The memory costs are 3× smaller. For other metrics, our verifier time is 3ms, competitive with Pianist's 2.8ms. The proof size and communication cost per prover are 10.2KB and 6.8KB, while Pianist's are 2.2KB and 2.1KB, respectively. Although our proof size and communication cost are larger, they are constant and reasonably practical in the real world, and can be reduced in an amortized manner as Soloist is capable to handle more transactions.

Particularly, Soloist shows linear speedup when the circuit is *general*, *i.e.*, a *single* batch of transactions. It owes to our "making-even" algorithm to reduce $\tilde{m}$, *i.e.*, the maximum number of non-zero entries in R1CS sub-matrices. This indicates that Soloist can prove both data-parallel and general circuits efficiently.

Comparison with HyperPianist. Fig. 3 compares Soloist and HyperPianist over zkRollup circuits with 16 (*i.e.*, "16tx") or 32 (*i.e.*, "32tx") batches of transactions. Soloist shows trade-offs between prover time and all of the other metrics. Specifically, Soloist can achieve 1.5× smaller proof size, 3 - 4× smaller communi-

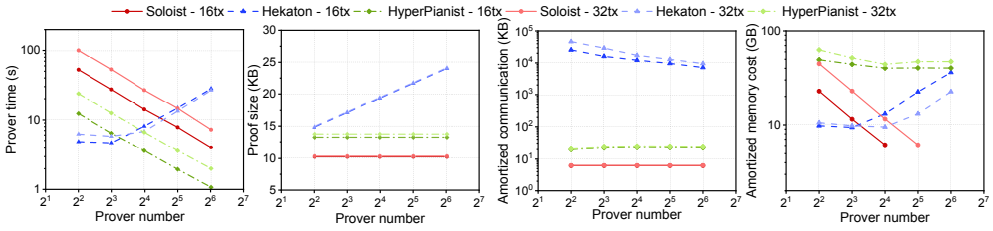

Fig. 3. Comparison of Soloist, Hekaton, and HyperPianist for proving zkRollups

cation cost, and 1.3× faster verifier time than HyperPianist, at the cost of 4× slower prover time. Note that our proof size, communication cost and verifier time are all constant, while HyperPianist's grow logarithmically in the circuit size. The memory cost of Soloist is up to 10× smaller than HyperPianist.[16]

Comparison with Hekaton. We compare Soloist with Hekaton over two circuits, the zkRollup circuit (covering data-parallel and general circuits) and the Merkle tree membership circuit (nearly-data-parallel). For the former, we run our circuit division algorithm to divide the circuit into multiple sub-circuits. For the latter, we directly use the sub-circuits from its benchmarks.

Fig. 3 shows the evaluation results over zkRollup circuits. As shown, Soloist is up to 10× slower in proving with 4 provers, but can be 7× faster with 64 provers. This is because Hekaton's performance is heavily affected by the number of shared wires among provers. When the prover number increases such that multiple provers handle a single batch of transactions, its prover efficiency degrades significantly since the number of shared wires also increases. This shows the scalability of Soloist for general circuits, and we would expect a larger performance advantage of Soloist over Hekaton with more provers (*e.g.*, 128).

Considering other metrics, Hekaton's logarithmic proof size is 1.5 - 2.5× larger than our constant one. We failed to measure the verifier time from Hekaton's implementation and thus we use their reported data, which shows to be 83ms and is 20× slower than ours. Although Hekaton's amortized communication is $O(1)$, it is concretely 10MB and varies with the circuit type. In contrast, ours remains a constant size of 6.8KB, which is 1000× smaller. Finally, the memory costs of the two SNARKs demonstrate similar tendencies to the prover time: Soloist's decreases linearly while Hekaton's increases with the number of provers.

Fig. 4 shows results on Merkle tree membership circuits, where "x/prover" means that *each prover* handles x sub-circuits, *i.e.*, the circuit *per prover* is fixed for each line. As shown, our prover time is 5× slower than Hekaton with 4 provers, but is competitive or even better with 64 provers. This is possibly because the aggregation time of sub-proofs increases with the number of provers in Hekaton. It is predictable that Soloist can outperform Hekaton with more

[16] The memory cost of HyperPianist does not decrease linearly with an increasing number of provers, as expected. This may be optimized at the implementation level.

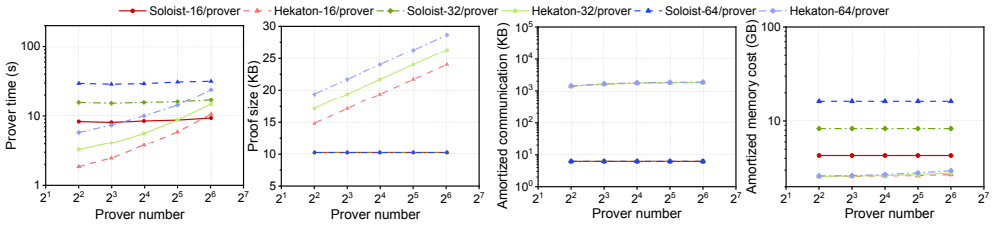

Fig. 4. Comparison of Soloist and Hekaton for proving Merkle tree memberships

provers (*e.g.*, 128). Besides, Hekaton's proof size is 1.5 - 3× larger than Soloist, and our verifier time can be 20× faster than Hekaton's logarithmic one. Our amortized communication cost is 100× smaller than Hekaton, which is several MBs. A drawback of our scheme is the memory cost, which is up to 10× larger.[17]

In summary, our experiments suggest that Soloist is preferable for proving general circuits, or in scenarios prioritizing prover efficiency with sufficient computational resources. These scenarios are natural: by deploying sufficient provers with rich resources, one would reasonably expect improved performance instead of degraded ones. Also, the constant and low amortized communication overhead makes Soloist suitable for low-bandwidth or wide-area-network applications.

8 Conclusion and Future Work

We propose Soloist, a distributed SNARK for R1CS with constant proof size, verification, and amortized communication. Soloist uses inner product and lookup PIOPs to build a distributed PIOP with constant proof size and verification. Soloist also extends the bivariate KZG into a batch and distributed version.

One limitation of Soloist is the concretely large proof size. To reduce it, future work may develop a more efficient distributed preprocessing to reduce the number of polynomial oracles. It is also promising to construct distributed SNARKs for customizable constraint system [49], which is compatible with most existing constraint systems such as R1CS, Plonk, and AIR.

Acknowledgments. This work was supported by the National Cryptologic Science Fund of China (2025NCSF02022), the National Natural Science Foundation of China (62372020), and the Fundamental Research Funds for the Central Universities. We thank anonymous reviewers of S&P 2025, CCS 2025, and Eurocrypt 2026 for their helpful comments and suggestions. We thank Yanpei Guo for his insights about lookup schemes. We thank Sherman S. M. Chow for his advice in the early stage of this work.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

[17] Hekaton's memory costs may be mainly determined by the number of shared wires, so it does not increase much even when the number of sub-circuits per prover increases.

References

1. Ames, S., Hazay, C., Ishai, Y., Venkitasubramaniam, M.: Ligero: Lightweight sublinear arguments without a trusted setup. In: CCS (2017)
2. Arun, A., Ganesh, C., Lokam, S.V., Mopuri, T., Sridhar, S.: Dew: A transparent constant-sized polynomial commitment scheme. In: PKC (2023)
3. Bellés-Muñoz, M., Isabel, M., Muñoz-Tapia, J.L., Rubio, A., Melé, J.B.: Circom: A circuit description language for building zero-knowledge applications. IEEE Trans. Dependable Secur. Comput. **20**(6), 4733–4751 (2023)
4. Ben-Sasson, E., Bentov, I., Horesh, Y., Riabzev, M.: Scalable, transparent, and post-quantum secure computational integrity. IACR ePrint 2018/046
5. Ben-Sasson, E., Chiesa, A., Riabzev, M., Spooner, N., Virza, M., Ward, N.P.: Aurora: Transparent succinct arguments for R1CS. In: EUROCRYPT (2019)
6. Ben-Sasson, E., Chiesa, A., Spooner, N.: Interactive oracle proofs. In: TCC (2016)
7. Boneh, D., Drake, J., Fisch, B., Gabizon, A.: Halo infinite: Proof-carrying data from additive polynomial commitments. In: CRYPTO (2021)
8. Bui, D., Chu, H., Couteau, G., Wang, X., Weng, C., Yang, K., Yu, Y.: An efficient ZK compiler from SIMD circuits to general circuits. J. Cryptol. **38**(1), 10 (2025)
9. Bünz, B., Bootle, J., Boneh, D., Poelstra, A., Wuille, P., Maxwell, G.: Bulletproofs: Short proofs for confidential transactions and more. In: S&P (2018)
10. Bünz, B., Chen, J.: Proofs for deep thought: Accumulation for large memories and deterministic computations. In: ASIACRYPT (2024)
11. Bünz, B., Fisch, B., Szepieniec, A.: Transparent SNARKs from DARK compilers. In: EUROCRYPT (2020)
12. Campanelli, M., Faonio, A., Fiore, D., Querol, A., Rodríguez, H.: Lunar: A toolbox for more efficient universal and updatable zkSNARKs and commit-and-prove extensions. In: ASIACRYPT (2021)
13. Chen, B., Bünz, B., Boneh, D., Zhang, Z.: Hyperplonk: Plonk with linear-time prover and high-degree custom gates. In: EUROCRYPT (2023)
14. Chiesa, A., Hu, Y., Maller, M., Mishra, P., Vesely, N., Ward, N.P.: Marlin: Preprocessing zkSNARKs with universal and updatable SRS. In: EUROCRYPT (2020)
15. Chiesa, A., Lehmkuhl, R., Mishra, P., Zhang, Y.: Eos: Efficient private delegation of zkSNARK provers. In: USENIX Security (2023)
16. Eagen, L., Fiore, D., Gabizon, A.: cq: Cached quotients for fast lookups. IACR ePrint 2022/1763 (2022)
17. Fuchsbauer, G., Kiltz, E., Loss, J.: The algebraic group model and its applications. In: CRYPTO (2018)
18. Gabizon, A., Williamson, Z.J.: plookup: A simplified polynomial protocol for lookup tables. IACR ePrint 2020/315 (2020)
19. Gabizon, A., Williamson, Z.J., Ciobotaru, O.: PLONK: Permutations over Lagrange-bases for oecumenical noninteractive arguments of knowledge. IACR ePrint 2019/953 (2019)
20. Ganesh, C., Patranabis, S., Singh, N.: Samaritan: Linear-time prover SNARK from new multilinear polynomial commitments. In: ASIACRYPT (2025)
21. Garg, S., Goel, A., Jain, A., Policharla, G., Sekar, S.: zkSaaS: Zero-knowledge SNARKs as a service. In: USENIX Security (2023)
22. Golovnev, A., Lee, J., Setty, S.T.V., Thaler, J., Wahby, R.S.: Brakedown: Linear-time and field-agnostic SNARKs for R1CS. In: CRYPTO (2023)
23. Graham, R.L.: Bounds on multiprocessing timing anomalies. SIAM Journal of Applied Mathematics **17**(2), 416–429 (1969)

24. Groth, J.: On the size of pairing-based non-interactive arguments. In: EUROCRYPT (2016)
25. Groth, J., Kohlweiss, M., Maller, M., Meiklejohn, S., Miers, I.: Updatable and universal common reference strings with applications to zk-SNARKs. In: CRYPTO (2018)
26. Haböck, U.: Multivariate lookups based on logarithmic derivatives. IACR ePrint 2022/1530 (2022)
27. Hu, Y., Mishra, P., Wang, X., Xie, J., Yang, K., Yu, Y., Zhang, Y.: DFS: delegation-friendly zksnark and private delegation of provers. In: USENIX Security (2025)
28. Kate, A., Zaverucha, G.M., Goldberg, I.: Constant-size commitments to polynomials and their applications. In: ASIACRYPT (2010)
29. Knabenhans, C., Viand, A., Merino-Gallardo, A., Hithnawi, A.: vFHE: Verifiable fully homomorphic encryption. In: WAHC (2024)
30. Kosba, A.E., Papadopoulos, D., Papamanthou, C., Song, D.: MIRAGE: succinct arguments for randomized algorithms with applications to universal zk-SNARKs. In: USENIX Security (2020)
31. Lavaur, T., Lacan, J.: Boomy: Batch opening of multivariate polynomial commitment. IACR Commun. Cryptol. **2**(1), 7 (2025)
32. Lee, J.: Dory: Efficient, transparent arguments for generalised inner products and polynomial commitments. In: TCC (2021)
33. Li, C., Li, Y., Zhu, P., Qu, W., Zhang, J.: Hyperpianist: Pianist with linear-time prover via fully distributed hyperplonk. In: S&P (2025), https://github.com/AntCPLab/HyperPianist
34. Li, W., Zhang, Z., Chow, S.S.M., Guo, Y., Gao, B., Song, X., Deng, Y., Liu, J.: Shred-to-shine metamorphosis of (distributed) polynomial commitments. IACR ePrint 2025/1354 (2025), to appear at USENIX Security 2026
35. Lipmaa, H., Siim, J., Zajac, M.: Counting Vampires: From univariate sumcheck to updatable ZK-SNARK. In: ASIACRYPT (2022)
36. Liu, T., Xie, T., Zhang, J., Song, D., Zhang, Y.: Pianist: Scalable zkrollups via fully distributed zero-knowledge proofs. In: S&P (2024)
37. Liu, X., Zhou, Z., Wang, Y., Pang, Y., He, J., Zhang, B., Yang, X., Zhang, J.: Scalable collaborative zk-snark and its application to fully distributed proof delegation. In: USENIX Security (2025)
38. Ozdemir, A., Boneh, D.: Experimenting with collaborative zk-SNARKs: Zero-Knowledge proofs for distributed secrets. In: USENIX Security (2022)
39. Ozdemir, A., Brown, F., Wahby, R.S.: Circ: Compiler infrastructure for proof systems, software verification, and more. In: S&P (2022)
40. Ozdemir, A., Wahby, R.S., Brown, F., Barrett, C.W.: Bounded verification for finite-field-blasting - in a compiler for zero knowledge proofs. In: CAV (2023)
41. Pailoor, S., Chen, Y., Wang, F., Rodríguez-Núñez, C., Geffen, J.V., Morton, J., Chu, M., Gu, B., Feng, Y., Dillig, I.: Automated detection of under-constrained circuits in zero-knowledge proofs. Proc. ACM Program. Lang. **7**(PLDI), 1510–1532 (2023)
42. Papamanthou, C., Shi, E., Tamassia, R.: Signatures of correct computation. In: TCC (2013)
43. Parno, B., Howell, J., Gentry, C., Raykova, M.: Pinocchio: Nearly practical verifiable computation. In: S&P (2013)
44. Polygon Zero: Plonky2: Fast recursive arguments with PLONK and FRI (2022), https://github.com/mir-protocol/plonky2, last accessed 2024-11-16
45. Ràfols, C., Zapico, A.: An algebraic framework for universal and updatable SNARKs. In: CRYPTO (2021)

46. Rosenberg, M., Mopuri, T., Hafezi, H., Miers, I., Mishra, P.: Hekaton: Horizontally-scalable zkSNARKs via proof aggregation. In: CCS (2024), https://github.com/Pratyush/hekaton-system
47. Schlag, S., Heuer, T., Gottesbüren, L., Akhremtsev, Y., Schulz, C., Sanders, P.: High-quality hypergraph partitioning. ACM J. Exp. Algorithmics **27**, 1.9:1–1.9:39 (2022)
48. Setty, S.T.V.: Spartan: Efficient and general-purpose zkSNARKs without trusted setup. In: CRYPTO (2020)
49. Setty, S.T.V., Thaler, J., Wahby, R.S.: Customizable constraint systems for succinct arguments. IACR ePrint 2023/552 (2023)
50. Setty, S.T.V., Thaler, J., Wahby, R.S.: Unlocking the lookup singularity with lasso. In: EUROCRYPT (2024)
51. Stronati, M., Firsov, D., Locascio, A., Livshits, B.: Clap: a semantic-preserving optimizing eDSL for Plonkish proof systems. Arxiv 2405/12115 (2023)
52. Szepieniec, A., Zhang, Y.: Polynomial IOPs for linear algebra relations. In: PKC (2022)
53. Thaler, J.: 17 misconceptions about SNARKs (and why they hold us back). https://a16zcrypto.com/posts/article/17-misconceptions-about-snarks/
54. Wang, W., Shi, F., Vilardell, D., Zhang, F.: Cirrus: Performant and accountable distributed SNARK. IACR ePrint 2024/1873 (2024), to appear at NDSS 2026
55. Wu, H., Zheng, W., Chiesa, A., Popa, R.A., Stoica, I.: DIZK: A distributed zero knowledge proof system. In: USENIX Security (2018)
56. Xie, J., Hu, Y., Yu, Y.: Hadamard product argument from lagrange-based univariate polynomials. In: ACISP (2024)
57. Xie, T., Zhang, J., Cheng, Z., Zhang, F., Zhang, Y., Jia, Y., Boneh, D., Song, D.: zkBridge: Trustless cross-chain bridges made practical. In: CCS (2022)
58. Xie, T., Zhang, J., Zhang, Y., Papamanthou, C., Song, D.: Libra: Succinct zero-knowledge proofs with optimal prover computation. In: CRYPTO (2019)
59. Xie, T., Zhang, Y., Song, D.: Orion: Zero knowledge proof with linear prover time. In: CRYPTO (2022)
60. Yang, Y., Cheng, Y., Wang, K., Li, X., Sun, J., Shen, J., Dong, X., Cao, Z., Yang, G., Deng, R.H.: Siniel: Distributed privacy-preserving zkSNARK. In: NDSS (2025)
61. Zapico, A., Buterin, V., Khovratovich, D., Maller, M., Nitulescu, A., Simkin, M.: Caulk: Lookup arguments in sublinear time. In: CCS (2022)
62. Zhang, J., Xie, T., Zhang, Y., Song, D.: Transparent polynomial delegation and its applications to zero knowledge proof. In: S&P (2020)
63. Zhang, Y., Szepieniec, A., Zhang, R., Sun, S., Wang, G., Gu, D.: Voproof: Efficient zkSNARKs from vector oracle compilers. In: CCS (2022)
64. Zk-Sync: https://www.zksync.io/, last accessed 2024-11-16

A Supplementary Preliminaries

A.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.*** *Given any valid* pp*, for any PPT adversary* $\mathcal{P}^*$ *with any* $(\mathtt{x}, \mathtt{z}) \leftarrow \mathcal{P}^*(\mathsf{pp})$*, there exists an expected polynomial time extractor* $\mathcal{E}^{\mathcal{P}^*}$ *such that*

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

$\mathcal{E}^{\mathcal{P}^*}$ *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* $\mathsf{poly}(\lambda, |\mathtt{x}|, \log n)$.

A.2 Polynomial Commitment Scheme

We adapt the PCS definitions from Brakedown [22], which allows the prover and verifier to run an interactive argument of knowledge (*i.e.*, the Eval protocol) for proving the validity of a polynomial evaluation.

Definition 2 (Polynomial commitment scheme [22]). *A general 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*, and* d*, and then 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 a public-coin 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 a polynomial* f *with bounded size* d^μ *corresponding to* C *such that* $f(\boldsymbol{x}) = y$*.* $\mathcal{V}$ *outputs* $b \in \{0, 1\}$ *at the end.*

In addition, a PCS should satisfy the following security properties:

• ***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 that, if an PIOP satisfies soundness and a PCS satisfies knowledge soundness, then the compiled interactive argument from the "PIOP + PCS" also inherits the knowledge soundness property [36,11].

In this paper, we use or build PCSs from bilinear pairings, where the knowledge soundness hold in the algebraic group model. Appendix E.1 gives details.

Protocol 1 (An inner product PIOP with improved efficiency).
A prover $\mathcal{P}$ has witness $\boldsymbol{f}, \boldsymbol{s} \in \mathbb{F}^m$. $\mathcal{P}$ and the verifier $\mathcal{V}$ hold $T \in \mathbb{F}$, a multiplicative subgroup $\mathbb{H}$ with order m and $u \in \mathbb{F} \setminus \{\mathbb{H}, 0\}$. $\mathcal{P}$ proves to $\mathcal{V}$ it holds that $\langle \boldsymbol{f}, \boldsymbol{s} \rangle = T$.
1. *$\mathcal{P} \to \mathcal{V}$: coefficient-based polynomial oracles $f(X)$ and $s'(X)$ (cf., Equation (1)).*
2. *$\mathcal{P} \to \mathcal{V}$: polynomial oracles $g'(X)$ and $h(X)$ such that*

$$f(X) \cdot s'(X) \cdot (X - u) = X \cdot g'(X) + (T + Z_{\mathbb{H}}(X) \cdot h(X)) \cdot (X - u). \tag{24}$$

3. *$\mathcal{V}$: queries $f(\alpha), s'(\alpha), g'(\alpha), h(\alpha)$ at a random $\alpha \in \mathbb{F}$. Compute $Z_{\mathbb{H}}(\alpha)$. Accept iff Equation (24) holds when $X = \alpha$.*

B Supplementary Details of Section 3

B.1 Construction of the Improved Inner Product PIOP

This section presents the construction details, formal protocol, and security proof of the improved inner product PIOP as described in Section 3.1.

We prove Equation (2) based on the univariate sum-check protocol [5]. Its validity is transformed into the existence of $g \in \mathbb{F}_{m-1}[X]$ and $h(X)$ such that

$$f(X) \cdot s'(X) = X \cdot g(X) + T + Z_{\mathbb{H}}(X) \cdot h(X). \tag{23}$$

Such a method requires an LDT for $g(X)$. We eliminate it following Basilisk [45]. Given a prescribed $u \in \mathbb{F} \setminus \{0, \mathbb{H}\}$ (not required to be random), Equation (23) is equivalent to the existence of $g' \in \mathbb{F}_m[X]$ and $h(X)$ such that Equation (24) holds. Now, if given a size-m system parameter (as in our case), the LDT for $g'(X)$ is free. After the prover sends polynomial oracles $f(X), s'(X), g'(X), h(X)$, the verifier picks a random challenge $\alpha \in \mathbb{F} \setminus u$, queries $f(\alpha), s'(\alpha), g'(\alpha), h(\alpha)$, computes $Z_{\mathbb{H}}(\alpha)$, and accepts iff Equation (24) holds at $X = \alpha$.

We propose our inner product PIOP in Protocol 1. Note that the protocol is secure requiring $u \in \mathbb{F} \setminus \{0, \mathbb{H}\}$, and we will explain this in the proof.

Proof (Proof of Theorem 1). **Completeness.** Completeness holds by the completeness of univariate sum-check [5] and Lemma 1, which we prove below.

Proof of Lemma 1. We first show that, for any order-m multiplicative subgroup $\mathbb{H} = \{\omega^0, \ldots, \omega^{m-1}\}$ with generator ω, it holds that

$$\text{for any } \ i \in [-m+1, m-1] \setminus \{0\}, \quad \sum_{x \in \mathbb{H}} x^i = 0, \tag{25}$$

where $[-m+1, m-1]$ means the integer set $\{-m+1, -m+2, \ldots, -1, 0, 1, \ldots, m-2, m-1\}$. As $\mathbb{H}$ is of order m, for any integer i, we have $(\omega^i)^m = (\omega^m)^i = 1^i = 1$. Hence, it holds that

$$(\omega^i)^m - 1 = (\omega^i - 1) \cdot ((\omega^i)^{m-1} + (\omega^i)^{m-2} + \cdots + (\omega^i)^0) = 0. \tag{26}$$

For any $i \in [-m+1, m-1] \setminus \{0\}$, we have $\omega^i \neq 1$ and $\omega^i - 1 \neq 0$. Hence by Equation (26), we have

$$(\omega^i)^{m-1} + (\omega^i)^{m-2} + \cdots + (\omega^i)^0 = \sum_{j\in[m]} (\omega^i)^{j-1} = \sum_{j\in[m]} (\omega^{j-1})^i = 0,$$

which proves Equation (25) as $\mathbb{H}$ is exactly $\{\omega^{j-1}\}_{j\in[m]}$.

As the degrees of $f(X)$ and $s(X)$ are $m-1$, all possible terms in $f(X) \cdot s(X^{-1})$ include $\{X^{-m+1}, X^{-m+2}, \ldots, 1, X, \ldots X^{m-1}\}$. By Equation (25), in the sum $\sum_{x\in\mathbb{H}} f(x) \cdot s(x^{-1})$, only the constant terms contribute to the final sum. Thus, given $f(X) = \sum_{i\in[m]} f_{i-1} \cdot X^{i-1}$ and $s(X) = \sum_{i\in[m]} s_{i-1} \cdot X^{i-1}$, we have

$$\sum_{x\in\mathbb{H}} f(x) \cdot s(x^{-1}) = |\mathbb{H}| \cdot (f_0 \cdot s_0 + f_1 \cdot s_1 + \cdots f_{m-1} \cdot s_{m-1}) = m \cdot \langle \boldsymbol{f}, \boldsymbol{s} \rangle.$$

Lemma 1 hence follows.

By definition, for all $x \in \mathbb{H}$, it holds that $s'(x) = s(x^{-1})$. Thus, based on Lemma 1 we have Equation (2) follows. By the univariate sum-check over Equation (2), Equation (23) is valid. By multiplying $(X-u)$ to both sides of Equation (23) and setting $g'(X) = (X-u) \cdot g(X)$, Equation (24) follows.

Hence, the completeness of Lemma 1 follows.

Soundness. This property follows due to the soundness of univariate sum-check, Schwartz-Zippel lemma, and the deduction from Equation (24) to Equation (23).

Below we show the validity of Equation (23) if Equation (24) is valid given an auxiliary polynomial $g' \in \mathbb{F}_m[X]$. When $X = u$, we have $u \cdot g'(u) = 0$ by Equation (24). As $u \neq 0$, this implies that $g'(u) = 0$. Hence, we have $g'(X) = (X-u) \cdot g(X)$ for some polynomial $g(X)$ with a degree strictly less than $m-1$. For all $x \in \mathbb{H}$, as $u \notin \mathbb{H}$, we have $x - u \neq 0$. Thus, we can divide Equation (24) by $X - u$ and evaluate this equation at some $x \in \mathbb{H}$ to obtain the following equation

$$f(x) \cdot s'(x) - T = x \cdot g(x). \tag{27}$$

As Equation (27) holds for all $x \in \mathbb{H}$, there exists a quotient polynomial $h(X)$ such that

$$f(X) \cdot s'(X) - T - X \cdot g(X) = Z_{\mathbb{H}}(X) \cdot h(X). \tag{28}$$

Hence, Equation (23) follows. Note that if $u \in \mathbb{H}$, we cannot conclude Equation (27) when $x = u$ and hence cannot conclude Equation (23) from Equation (24). Also note that the validity of Equation (23) does not depend on the randomness of u, so we do not need u to be a random challenge.

We next analyze the soundness when checking Equation (24) at $X = \alpha$. By the Schwartz-Zippel lemma, if Equation (24) holds at $X = \alpha$, then except with a probability of $(2m-1)/|\mathbb{F}|$ that Equation (24) holds. This further means that Equation (23) holds. Then, by the perfect soundness of univariate sum-check, we have $\sum_{x\in\mathbb{H}} f(x) \cdot s'(x) = m \cdot T$ and hence $\langle \boldsymbol{f}, \boldsymbol{s} \rangle = T$ by Lemma 1. Hence, the total soundness error is $(2m-1)/|\mathbb{F}|$.

Complexity. Oracle number and query size hold by count. The prover requires 2 FFTs for obtaining size-$2m$ evaluations of $f(X), s'(X)$, and 1 FFT for computing $f(X) \cdot s'(X)$. The prover's computation other than FFTs is $O(m)$. The verifier complexity is $O(\log m)$ for computing $Z_{\mathbb{H}}(\alpha)$. As $Z_{\mathbb{H}}(X)$ is only related to $\mathbb{H}$ other than $\boldsymbol{f}, \boldsymbol{s}$, we can let an indexer to construct the polynomial oracle $Z_{\mathbb{H}}(X)$ in the preprocessing phase. In the online phase, the verifier queries $Z_{\mathbb{H}}(\alpha)$ instead of computing it. The verifier complexity is then reduced to $O(1)$. □

Remark 4. The LDT elimination technique we used cannot be applied to PIOPs from Laurent polynomials. Specifically, from Section 3.1, we know to prove $\langle \boldsymbol{f}, \boldsymbol{s} \rangle = T$, it suffices to prove the existence of $p, q \in \mathbb{F}_{m-1}[X]$ such that

$$f(X) \cdot X^{m-1} \cdot s(X^{-1}) = p(X) + X^{m-1} \cdot T + X^m \cdot q(X). \tag{29}$$

Although we can still multiply $(X - u)$ to both sides of the equation and set $p'(X) = p(X) \cdot (X - u)$ and $q'(X) = q(X) \cdot (X - u)$, the existence of $p', q' \in \mathbb{F}_m[X]$ such that

$$f(X) \cdot X^{m-1} \cdot s(X^{-1}) \cdot (X - u) = p'(X) + X^{m-1} \cdot T \cdot (X - u) + X^m \cdot q'(X)$$

can only deduce $(X - u) \mid p'(X) + X^m \cdot q'(X)$ instead of Equation (29).

Remark 5. When instantiated with the standard univariate KZG, our inner product PIOP contributes to an IPA without any FFTs during commitment due to our coefficient-based constructions. Despite this, if with evaluation-based constructions as in Marlin, one could still avoid such FFTs by using Lagrange-based KZG [56] at the cost of a larger SRS.

B.2 Construction of the Distributed Inner Product PIOP

This section gives the formal protocol to prove $\langle \boldsymbol{f}, \boldsymbol{s} \rangle = T$ distributively. The core technique is a two-dimension univariate sum-check, and refer to Section 3.2 for high-level descriptions. As recalled, to prove $\langle \boldsymbol{f}, \boldsymbol{s} \rangle = T$, it suffices for the prover to prove Equation (3), or, by Lemma 2, equivalently to prove

$$\sum_{x \in \mathbb{H}} \sum_{y \in \mathbb{L}} f(x, y) \cdot s'(x, y) = m \cdot T. \tag{30}$$

Protocol 2 presents the formal scheme of distributed inner product PIOP, which involves two univariate sum-check schemes: one over the dimension X and subgroup $\mathbb{H}$, and the other over the dimension Y and subgroup $\mathbb{L}$. A visualization of proving an simplified statement $\sum_{x \in \mathbb{H}} \sum_{i \in [\ell]} f_i(x) = T$ (or equivalently, $\sum_{x \in \mathbb{H}, y \in \mathbb{L}} f(x, y) = T$) is given in Figure 5.

Below we give the security proofs for Lemma 2 and Theorem 2.

Proof (Proof of Lemma 2). Denote the elements in $\mathbb{L}$ by $(1, \eta, \ldots, \eta^{\ell-1})$. By the properties of Lagrange polynomials, fixing any $i \in [\ell]$, we have $L_i(\eta^{i-1}) = 1$ and $L_i(\eta^{j-1}) = 0$ for all $j \in [\ell]$ and $j \neq i$. Hence, for any $k \in [n]$, it holds that

$$\sum_{y \in \mathbb{L}} f^{(k)}(X, y) = \sum_{y \in \mathbb{L}} \sum_{i \in [\ell]} f_i^{(k)}(X) \cdot L_i(y) = \sum_{j \in [\ell]} \sum_{i \in [\ell]} f_i^{(k)}(X) \cdot L_i(\eta^{j-1}) = \sum_{i \in [\ell]} f_i^{(k)}(X).$$

Protocol 2 (Distributed PIOP for Inner Products).
Statement: *The master prover $\mathcal{P}_0$ and sub-provers $\mathcal{P}_1, \ldots, \mathcal{P}_\ell$ show that $\langle \boldsymbol{f}, \boldsymbol{s} \rangle = T$ for secret $\boldsymbol{f}, \boldsymbol{s} \in \mathbb{F}^{m\ell}$ and public T, or show equivalently Equation (3) or Equation (30).*
Secret inputs: *Suppose $\boldsymbol{f}$ and $\boldsymbol{s}$ are split into ℓ sub-vectors $(\boldsymbol{f_1}, \ldots, \boldsymbol{f_\ell})$ and $(\boldsymbol{s_1}, \ldots, \boldsymbol{s_\ell})$, respectively, each of size m. $\mathcal{P}_i$ holds the sub-vectors $\boldsymbol{f_i}$ and $\boldsymbol{s_i}$ as well as coefficient-based sub-polynomials $f_i(X)$ and $s'_i(X)$, where $s'_i(X)$ is constructed from $\boldsymbol{s_i}$ according to Equation (1).*
Public inputs: *The claimed sum T. Multiplicative subgroups $\mathbb{H}$ of order m and $\mathbb{L}$ of order ℓ. Prescribed elements $u_1 \in \mathbb{F} \setminus \{0, \mathbb{H}\}$ and $u_2 \in \mathbb{F} \setminus \{0, \mathbb{L}\}$.*

1. *Provers compute and send polynomial oracles $f(X, Y)$ and $s'(X, Y)$ to $\mathcal{V}$.*
   (a) *$\mathcal{P}_i$: computes polynomials $f_i(X)$ and $s'_i(X)$, and sends their oracles to $\mathcal{P}_0$.*
   (b) *$\mathcal{P}_0$: computes polynomial oracles $f(X, Y) = \sum_{i\in[\ell]} f_i(X)L_i(Y)$ and $s'(X, Y) = \sum_{i\in[\ell]} s'_i(X)L_i(Y)$.*
2. *Provers compute and send polynomial oracles $g_1, h_1 \in \mathbb{F}_m[X]$ of the first-dimension univariate sum-check over X to $\mathcal{V}$.*
   (a) *$\mathcal{P}_i$: computes $g_{1,i}, h_{1,i} \in \mathbb{F}_m[X]$ by dividing $Z_{\mathbb{H}}(X)$ over $f_i(X) \cdot s'_i(X)$ s.t.*

$$(X - u_1) \cdot f_i(X) \cdot s'_i(X) = X \cdot g_{1,i}(X) + (X - u_1) \cdot (T_{1,i} + Z_{\mathbb{H}}(X) \cdot h_{1,i}(X)). \quad (31)$$

   By the uniqueness of polynomial division, we have $\sum_{i\in[\ell]} T_{1,i} = T$. Then, $\mathcal{P}_i$ sends oracles $g_{1,i}(X)$ and $h_{1,i}(X)$ to $\mathcal{P}_0$.
   (b) *$\mathcal{P}_0$: computes oracles $g_1(X) = \sum_{i\in[\ell]} g_{1,i}(X)$ and $h_1(X) = \sum_{i\in[\ell]} h_{1,i}(X)$.*
3. *$\mathcal{V}$ sends a random challenge $\alpha \in \mathbb{F}$ to $\mathcal{P}_0$, who transfers to all $\mathcal{P}_i$.*
4. *Provers compute and send polynomial oracles $g_2, h_2 \in \mathbb{F}_\ell[Y]$ of the second-dimension univariate sum-check over Y to $\mathcal{V}$.*
   (a) *$\mathcal{P}_i$: computes and sends $f_i(\alpha), s'_i(\alpha), g_{1,i}(\alpha)$, and $h_{1,i}(\alpha)$ to $\mathcal{P}_0$.*
   (b) *$\mathcal{P}_0$: computes $f(\alpha, Y), s'(\alpha, Y), g_1(\alpha)$, and $h_1(\alpha)$. Compute $T_2 = \alpha \cdot g_1(\alpha)/(\alpha - u_1) + T + Z_{\mathbb{H}}(\alpha) \cdot h_1(\alpha)$. Compute $g_2, h_2 \in \mathbb{F}_\ell[Y]$ by dividing $Z_{\mathbb{L}}(Y)$ over $(Y - u_2) \cdot f(\alpha, Y) \cdot s'(\alpha, Y)$ s.t.*

$$(Y - u_2) \cdot f(\alpha, Y) \cdot s'(\alpha, Y) = Y \cdot g_2(Y) + (Y - u_2) \cdot (T_2/\ell + Z_{\mathbb{L}}(Y) \cdot h_2(Y)). \quad (32)$$

5. *$\mathcal{V}$: queries $g_1(\alpha), h_1(\alpha)$, and compute $Z_{\mathbb{H}}(\alpha)$ and T_2. Pick a random $\beta \in \mathbb{F}$ and compute $Z_{\mathbb{L}}(\beta)$. Query $f(\alpha, \beta), s'(\alpha, \beta), g_2(\beta)$, and $h_2(\beta)$. Accept iff Equation (32) holds at $Y = \beta$.*

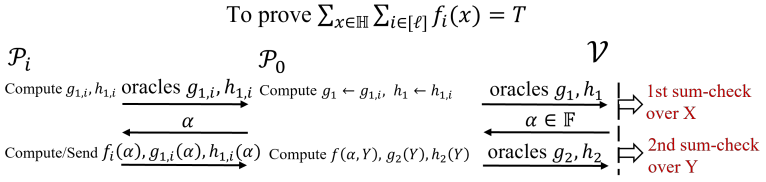

Fig. 5. Visualization of the distributed PIOP in Protocol 2

Further, fixing any $j \in [\ell]$ and $y = \eta^{j-1}$, we have

$$\begin{aligned} & f^{(1)}(X,y) \cdot f^{(2)}(X,y) \cdots f^{(n)}(X,y) \\ =&(\sum_{i\in[\ell]} f_i^{(1)}(X) \cdot L_i(y)) \cdot (\sum_{i\in[\ell]} f_i^{(2)}(X) \cdot L_i(y)) \cdots (\sum_{i\in[\ell]} f_i^{(n)}(X) \cdot L_i(y)) \\ =&f_j^{(1)}(X) \cdot f_j^{(2)}(X) \cdots f_j^{(n)}(X). \end{aligned} \tag{33}$$

Hence, we have

$$\begin{aligned} & \sum_{y\in\mathbb{L}} f^{(1)}(X,y) \cdot f^{(2)}(X,y) \cdots f^{(n)}(X,y) \\ =& \sum_{j\in[\ell]} f^{(1)}(X,\eta^{j-1}) \cdot f^{(2)}(X,\eta^{j-1}) \cdots f^{(n)}(X,\eta^{j-1}) \\ =& \sum_{j\in[\ell]} f_j^{(1)}(X) \cdot f_j^{(2)}(X) \cdots f_j^{(n)}(X) \Leftarrow \text{Equation (33)} \\ =& \sum_{i\in[\ell]} f_i^{(1)}(X) \cdot f_i^{(2)}(X) \cdots f_i^{(n)}(X). \end{aligned} \tag{34}$$

Therefore, the first equation in Lemma 4 follows. Summing up Equation (34) over $x \in \mathbb{H}$, we further have

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

Thus, the second equation in Lemma 2 also follows. □

We give the security proof of Theorem 2 as below.

Proof (Proof of Theorem 2). **Completeness.** Completeness holds by Lemma 2, univariate sum-check, and the completeness of Protocol 1.

Soundness. Assume that all sub-provers are honest here. The target inner product relation $\langle \boldsymbol{f}, \boldsymbol{s} \rangle = T$ is equivalent to the sum of inner product relations, *i.e.*, $\sum_{i\in[\ell]} \langle \boldsymbol{f}_i, \boldsymbol{s}_i \rangle = T$. According to Theorem 1, we know if $\sum_{i\in[\ell]} \langle \boldsymbol{f}_i, \boldsymbol{s}_i \rangle \neq T$, then Equation (31) holds at a random $\alpha \in \mathbb{F}$ with a probability of at most $(2m-1)/|\mathbb{F}|$. From Lemma 2 we know that $\sum_{i\in[\ell]} f_i(X) \cdot s_i'(X) = \sum_{y\in\mathbb{L}} f(X,y) \cdot s'(X,y)$. Similarly, the following equation holds with a probability of at most $(2m-1)/|\mathbb{F}|$:

$$\sum_{y\in\mathbb{L}} (\alpha - u_1) \cdot f(\alpha, y) \cdot s'(\alpha, y) = \alpha \cdot g_1(\alpha) + (\alpha - u_1) \cdot (T + Z_{\mathbb{H}}(\alpha) \cdot h_1(\alpha)).$$

Assume we are not in this case. Then, by the univariate sum-check protocol, there do not exist polynomials $g_2, h_2 \in \mathbb{F}_\ell[Y]$ satisfying Equation (32). By the Schwartz-Zippel lemma, for a random challenge $\beta \in \mathbb{F}$, the probability that Equation (32) holds at $Y = \beta$ is $(2\ell-1)/|\mathbb{F}|$. By the union bound argument, the total soundness error is at most $(2m + 2\ell - 2)/|\mathbb{F}|$.

Complexity. We present the complexity analysis below.

Sub-prover. The time complexity of $\mathcal{P}_i$ is $O(m)$ for computing $f_i(X)$ and $s'_i(X)$ from $\boldsymbol{f}_i$ and $\boldsymbol{s}_i$, respectively. Computing $g_{1,i}(X)$ and $h_{1,i}(X)$ costs $O(m \log m)$ time by running FFTs to obtain $f_i(X) \cdot s'_i(X)$ and dividing $Z_{\mathbb{H}}(X)$. Computing and sending $f_i(\alpha), s'_i(\alpha), g_{1,i}(\alpha)$ and $h_{1,i}(\alpha)$ to $\mathcal{P}_0$ 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 $f(X,Y)$, $s'(X,Y)$, $g_1(X)$, and $h_1(X)$. Given evaluations $f_i(\alpha)$ and $s'_i(\alpha)$, $\mathcal{P}_0$ requires $O(\ell \log \ell)$ time to obtain size-ℓ polynomials $f(\alpha, Y)$ and $s'(\alpha, Y)$. Then, it takes $O(\ell \log \ell)$ time to compute $f(\alpha, Y) \cdot s'(\alpha, Y)$ by FFTs and run the polynomial division over $Z_{\mathbb{L}}(Y)$ to obtain $g_2(Y)$ and $h_2(Y)$. Computing $g_2(\beta)$ and $h_2(\beta)$ costs $O(\ell)$ time. Therefore, the total master prover complexity is $O(\ell \log \ell)$.

Proof size. The involved polynomial oracles include $f(X,Y), s'(X,Y), g_1(X)$, $h_1(X), g_2(Y)$, and $h_2(Y)$. Thus, the oracle number is 6. The proof size comes from sending each evaluation once on these polynomials, costing $O(1)$ and 6 field elements in total.

Communication complexity. The communication between $\mathcal{P}_i$ and $\mathcal{P}_0$ includes: 1) oracles $f_i(X)$ and $s'_i(X)$; 2) oracles $g_{1,i}(X)$ and $h_{1,i}(X)$; 3) $f_i(\alpha), s'_i(\alpha), g_{1,i}(\alpha)$, and $h_{1,i}(\alpha)$. Assuming the oracle size is $O(1)$, the amortized communication complexity is $O(1)$.

Verifier complexity. The verifier complexity is $O(\log m + \log \ell)$ mainly due to computing $Z_{\mathbb{H}}(\alpha)$ and $Z_{\mathbb{L}}(\beta)$, respectively. □

B.3 Further Discussions of our Distributed Inner Product PIOP

This section gives further discussions of our distributed inner product PIOP.

Bivariate sum-check protocol and comparison. If combining Equation (3) and Lemma 2, our distributed inner product PIOP technically offers a non-distributed PIOP to prove a bivariate sum-check relation $\sum_{x\in\mathbb{H}} \sum_{y\in\mathbb{L}} f(x,y) \cdot s'(x,y) = m \cdot T$, or simply, to prove $\sum_{x\in\mathbb{H}} \sum_{y\in\mathbb{L}} f(x,y) = T$ where $f \in \mathbb{F}_{d,\ell}[X,Y]$.

The protocol procedure is as follows. Public inputs include the polynomial oracle $f(X,Y)$, the claimed sum T, and multiplicative subgroups $\mathbb{H}$ (of order m) and $\mathbb{L}$. The secret input is the polynomial $f(X,Y)$. We assume here it has the form $f(X,Y) = \sum_{i\in[\ell]} f_i(X) L_i(Y)$ where $f_i \in \mathbb{F}_d[X]$ for all $i \in [\ell]$ and $L_i \in \mathbb{F}_\ell[Y]$ is the i-th Lagrange polynomial of $\mathbb{L}$. WLOG, assume $d > m$. The form of $f(X,Y)$ would be generalized later. For simplicity, we omit to use the LDT elimination technique as in Protocol 2.

1. $\mathcal{P}$: computes and sends polynomial oracles $g_1 \in \mathbb{F}_{m-1}[X]$ and $h_1(X)$ to $\mathcal{V}$. They are computed by dividing $Z_{\mathbb{H}}(X)$ over $\sum_{i\in[\ell]} f_i(X)$, which satisfy

$$\sum_{i\in[\ell]} f_i(X) = X \cdot g_1(X) + T/m + Z_{\mathbb{H}}(X) \cdot h_1(X).$$

2. $\mathcal{V}$: sends a random challenge α to $\mathcal{P}$.

3. $\mathcal{P}$: computes and sends polynomial oracles $g_2 \in \mathbb{F}_{\ell-1}[Y]$ and $h_2(Y)$ to $\mathcal{V}$. They are computed by dividing $Z_{\mathbb{L}}(Y)$ over $f(\alpha, Y)$. Let $T_1 = \alpha \cdot g_1(\alpha) + T/m + Z_{\mathbb{H}}(\alpha) \cdot h_1(\alpha)$. These polynomials satisfy

$$f(\alpha, Y) = Y \cdot g_2(Y) + T_1/\ell + Z_{\mathbb{L}}(Y) \cdot h_2(Y). \tag{35}$$

4. $\mathcal{V}$: queries $g_1(\alpha), h_1(\alpha)$ to compute T_1. Pick a random challenge β. Query $f(\alpha, \beta), g_2(\beta), h_2(\beta)$, and check if Equation (35) holds at $Y = \beta$.

The security properties of this bivariate sum-check protocol follow directly from Theorem 2. We analyze the complexities below.

The prover's computation include: 1) computing $\sum_{i\in[\ell]} f_i(X), g_1(X)$, and $h_1(X)$ by polynomial division over $Z_{\mathbb{H}}(X)$, which costs at most $O(d\ell)$; 2) computing $f(\alpha, Y), g_2(Y)$ and $h_2(Y)$ by polynomial division, which costs at most $O(d\ell)$. Hence, the total prover complexity is $O(d\ell)$. The proof size is $O(1)$. The verifier complexity is $O(\log m\ell)$ for computing $Z_{\mathbb{H}}(\alpha)$ and $Z_{\mathbb{L}}(\beta)$.

Our bivariate sum-check protocol achieves a constant proof size by splitting the bivariate sum-check into two univariate sum-checks over two dimensions. This is feasible by Lemma 2, but also incurs a limitation: the sum-domain of the Y-dimension has to be the subgroup corresponding to the Lagrange polynomials $\{L_i(Y)\}_{i\in[\ell]}$ as in the target polynomial $f(X, Y)$. Although the form of $f(X, Y)$ can be generalized a little, which is not limited to be $\sum_{i\in[\ell]} f_i(X)L_i(Y)$, it has to be $\sum_{i\in[\ell]} f_i(X)L_i^c(Y)$ for some positive integer c. This guarantees that $\sum_{y\in\mathbb{L}} f(X, y) = \sum_{i\in[\ell]} f_i(X)$, and hence supports the split of the single two-dimension sum-check into two univariate sum-checks.

Comparison with BC24. BC24 [10] also gives a bivariate sum-check PIOP to prove the relation $\sum_{x\in\mathbb{H}} \sum_{y\in\mathbb{L}} f(x, y) = T$, where $f \in \mathbb{F}_{d,\ell}[X, Y]$, $\mathbb{H}$ and $\mathbb{L}$ are public multiplicative subgroups, and T is a claimed sum. Like ours, BC24 makes several (non-general) assumptions of the sum-domains and the target polynomial $f(X, Y)$. Specifically, it requires that there exist multiplicative subgroups $|\mathbb{M}|$ and $|\mathbb{N}|$ such that $\mathbb{H} \subseteq \mathbb{M}$, $\mathbb{L} \subseteq \mathbb{N}$, $|\mathbb{M}| \geq d$ and $|\mathbb{N}| \geq \ell$. This guarantees that given the evaluations of $g(X) = \sum_{y\in\mathbb{L}} f(X, y)$ over $\mathbb{M}$, the verifier can directly compute $\sum_{x\in\mathbb{H}} g(x)$ in time $O(|\mathbb{H}|)$. Similarly, for any fixed $X = a$, given the evaluations of $h(Y) = f(a, Y)$ over $\mathbb{N}$, the verifier can directly compute $\sum_{y\in\mathbb{L}} h(y)$ in time $O(|\mathbb{L}|)$. Its detailed protocol procedures are as follows.

1. $\mathcal{P}$: computes $g(X) = \sum_{y\in\mathbb{L}} f(X, y)$ and the evaluations of $g(X)$ over $\mathbb{M}$. Send to $\mathcal{V}$ the oracle $g(X)$ and its evaluations over $\mathbb{M}$.
2. $\mathcal{V}$: sends a random challenge $a \in \mathbb{F} \setminus \mathbb{H}$ to $\mathcal{P}$.
3. $\mathcal{P}$: computes $h(Y) = f(a, Y)$ and its evaluations over $\mathbb{N}$. Send the oracle $h(Y)$ and its evaluations over $\mathbb{N}$ to $\mathcal{V}$.
4. $\mathcal{V}$: sends a random challenge $b \in \mathbb{F} \setminus \mathbb{L}$ to $\mathcal{P}$.
5. $\mathcal{P}$: computes and sends $T^* = h(b)$ to $\mathcal{V}$.
6. $\mathcal{V}$: queries $g(a), h(b), f(a, b)$. Check if $\sum_{x\in\mathbb{H}} g(x) = T$. Check if $\sum_{y\in\mathbb{L}} h(y) = g(a)$. Check if $h(b) = T^*$. Check if $f(a, b) = f(a, b)$. Accept iff all checks pass.

We analyze the complexities below. The prover's computation includes: 1) $O(d\ell \cdot |\mathbb{L}|)$ for computing $g(X)$, and at least $O(d \log d)$ for computing its evalua-

Table 4. Comparison of two bivariate sum-check PIOPs in a common setting

Scheme	#Round	Prover	Proof size	Verifier
BC24 [10]	3	$O(d\ell + d\log d + \ell\log\ell)$	$O(m+\ell)$	$O(m+\ell)$
Ours	2	$O(d\ell)$	$O(1)$	$O(\log m\ell)$

tions over $\mathbb{M}$; 2) $O(d\ell)$ for computing $h(Y)$, and at least $O(\ell\log\ell)$ for computing its evaluations over $\mathbb{N}$. Hence, the prover complexity is $O(d\ell|\mathbb{L}|+d\log d+\ell\log\ell)$. The proof size is at least $O(d+\ell)$ for sending and receiving $O(|\mathbb{M}|)$ evaluations of $g(X)$ and $O(|\mathbb{N}|)$ evaluations of $h(Y)$. The verifier complexity is at least $O(|\mathbb{H}|+|\mathbb{L}|)$ for computing $\sum_{x\in\mathbb{H}} g(x)$ and $\sum_{y\in\mathbb{L}} h(y)$, respectively.

To give an intuitive comparison, we adapt the assumptions of both, *i.e.*, to prove $\sum_{x\in\mathbb{H}}\sum_{y\in\mathbb{L}} f(x,y) = T$, where $f \in \mathbb{F}_{d,\ell}[X,Y]$, $|\mathbb{L}| = \ell$, $|\mathbb{H}| = m < d$, $|\mathbb{M}| = d$, and $f(X,Y)$ can be written as $f(X,Y) = \sum_{i\in[\ell]} f_i(X)L_i(Y)$. The complexities of our bivariate sum-check remain unaffected, while the complexities of BC24 would change. Specifically, the term $O(d\ell|\mathbb{L}|)$ is improved into $O(d\ell)$ for computing $g(X)$. This is because for each $y = \eta^{j-1}$, $f(X,y)$ is exactly $f_i(X)$;

Table 4 gives a comparison. As shown, our bivariate sum-check PIOP has fewer rounds, faster prover, smaller proof size, and faster verifier complexity.

A batch inner product PIOP and comparison. Our distributed inner product PIOP can be naturally extended as a batch scheme for inner products with constant proof size, which is irrelevant to instance number and vector length. Specifically, the PIOP proves $\langle \boldsymbol{f}_i, \boldsymbol{s}_i\rangle = T_i$ for $i \in [\ell]$ at one time, where for each i, $\boldsymbol{f}_i$ and $\boldsymbol{s}_i$ are secret size-m vectors and T_i is a claimed sub-sum.

We reduce $\{\langle \boldsymbol{f}_i, \boldsymbol{s}_i\rangle = T_i\}_{i\in[\ell]}$ into a univariate sum-check relation similar to Equation (3), which can then be proved by the non-distributed version of our distributed inner product PIOP (*i.e.*, Protocol 2).

Given a random challenge r, to prove $\{\langle \boldsymbol{f}_i, \boldsymbol{s}_i\rangle = T_i\}_{i\in[\ell]}$, it suffices to prove $\sum_{i\in[\ell]} r^{i-1}\cdot\langle \boldsymbol{f}_i, \boldsymbol{s}_i\rangle = \sum_{i\in[\ell]} r^{i-1}\cdot T_i$ with a slight soundness error. For each $i \in [\ell]$, define an auxiliary polynomial $R_i(X) = X^{i-1}$. By Equation (2), we have

$$\sum_{i\in[\ell]}\sum_{x\in\mathbb{H}} r^{i-1}\cdot f_i(x)\cdot s_i'(x) = \sum_{i\in[\ell]}\sum_{x\in\mathbb{H}} R_i(r)\cdot f_i(x)\cdot s_i'(x) = m\cdot\sum_{i\in[\ell]} R_i(r)\cdot T_i. \quad (36)$$

In Equation (36), the right side is a constant, and we can move it to the left and inside the summation. Then, Equation (36) becomes

$$\sum_{i\in[\ell]}\sum_{x\in\mathbb{H}}(R_i(r)\cdot f_i(x)\cdot s_i'(x) - R_i(r)\cdot T_i) = 0. \quad (37)$$

Equation (37) is similar to Equation (3), and can be proved by the (non-distributed) bivariate sum-check PIOP. Such a batch inner product PIOP has a prover complexity of $O(\ell\cdot m\log m + \ell\log\ell)$, as a single prover does the work of all sub-provers and the master prover. The proof size and verifier complexity inherit from Protocol 1, which is $O(1)$ and $O(\log m\ell)$, respectively.

Table 5. Comparison of two batch inner product PIOPs

Scheme	#Round	Prover	Proof size	Verifier
Amortized sum-check [5]	2	$O(m\ell \log m)$	$O(\ell)$	$O(\log m + \ell)$
Ours	2	$O(m\ell \log m + \ell \log \ell)$	$O(1)$	$O(\log m\ell)$

Comparison with previous batch scheme. As each inner product relation $\langle \boldsymbol{f}_i, \boldsymbol{s}_i \rangle = T_i$ would be transformed into a univariate sum-check relation in our PIOP, a straightforward batch approach is using the amortized univariate sum-check [5] to combine multiple sum-check relations into one.

Specifically, the protocol procedures are as follows.

1. $\mathcal{P}$: sends polynomial oracles of $\{f_i(X), s'_i(X)\}_{i\in[\ell]}$ to $\mathcal{V}$.
2. $\mathcal{V}$: sends a random challenge $r \in \mathbb{F}$ to $\mathcal{P}$.
3. $\mathcal{P}$: sends polynomial oracles $g \in \mathbb{F}_{m-1}[X], h(X)$ to $\mathcal{V}$. They satisfy that

$$\sum_{i\in[\ell]} r^{i-1} \cdot f_i(X) \cdot s'_i(X) = X \cdot g(X) + \sum_{i\in[\ell]} r^{i-1} \cdot T_i + Z_{\mathbb{H}}(X) \cdot h(X). \tag{38}$$

4. $\mathcal{V}$: picks a random challenge $\alpha \in \mathbb{F}$. Query $\{f_i(\alpha), s'_i(\alpha)\}_{i\in[\ell]}$, $g(\alpha)$, and $h(\alpha)$. Compute $Z_{\mathbb{H}}(\alpha)$. Accept iff Equation (38) holds at $X = \alpha$.

We analyze its complexities below. The prover's computation includes: 1) computing $f_i(X) \cdot s'_i(X)$, which costs $O(m \log m)$ and totally $O(m\ell \log m)$; 2) invoking the polynomial division, which costs $O(m)$. Hence, the total prover complexity is $O(m\ell \log m)$. The proof size is $O(\ell)$ for sending and receiving $O(\ell)$ evaluations over $f_i(X)$ and $s'_i(X)$ for all $i \in [\ell]$. The verifier complexity is $O(\log m)$ for computing $Z_{\mathbb{H}}(\alpha)$, and is $O(\ell)$ for checking Equation (38).

Table 5 presents a complexity comparison of the two batch PIOPs for inner products. As shown, our PIOP achieves a constant proof size and faster verifier complexity, while sacrificing the prover complexity when the instance number ℓ is much larger than the vector length m.

C Supplementary Details of Section 4

C.1 Comparisons of the Reductions for Linear Constraints

In Section 4.1, we give a reduction from linear constraints of R1CS into a distributed inner product relation. It naturally falls back to a non-distributed version. Specifically, for the linear constraint $P_v \boldsymbol{w} = \boldsymbol{v}$, given the challenge vector $\boldsymbol{r} = (1, r, \ldots, r^{m\ell-1})$, we let $\boldsymbol{p} = \boldsymbol{r}^\top P_v$. Then, the linear constraint becomes an inner product relation $\langle \boldsymbol{p}, \boldsymbol{w} \rangle = \langle \boldsymbol{r}, \boldsymbol{v} \rangle$. Define coefficient-based polynomials $f_p(X), f_v(X)$ from $\boldsymbol{p}, \boldsymbol{v}$, and define a coefficient-based polynomial $f'_w(X)$ from $\boldsymbol{w}$ according to Equation (1). Define a multiplicative subgroup $\mathbb{H}'$ of size $m\ell$. Similarly to Equation (8), we rely on Lemma 1 to transform the inner product

relation into a univariate sum-check relation. Then, it suffices for the prover to show

$$\sum_{x\in\mathbb{H}'} f_p(x)\cdot f'_w(x) = m\cdot f_v(r). \tag{39}$$

This PIOP involves 3 polynomial oracles, and requires 3 FFTs to obtain the polynomial $f_p(X)\cdot f'_w(X)$ from polynomials $f_p(X)$ and $f'_w(X)$.

Aurora [5] uses the same method to transform the linear constraint into an inner product relation. Indeed, it uses the univariate sum-check to prove $\langle \boldsymbol{p}, \boldsymbol{w}\rangle = \langle \boldsymbol{r}, \boldsymbol{v}\rangle$, but adopts the *evaluation-based* polynomial constructions. Specifically, define evaluation-based polynomials $g_p \in \mathbb{F}_{m\ell}[X]$ such that the evaluations of $g_p(X)$ over $\mathbb{H}'$ is $\boldsymbol{p}$. Similarly, define evaluation-based polynomials $g_w, g_r, g_v \in \mathbb{F}_{m\ell}[X]$ such that their evaluations over $\mathbb{H}'$ are exactly the vectors $\boldsymbol{w}, \boldsymbol{r}, \boldsymbol{v}$, respectively. Then, the inner product relation is transformed into a univariate sum-check relation

$$\sum_{x\in\mathbb{H}'} g_p(x)\cdot g_w(x) - g_r(x)\cdot g_v(x) = 0. \tag{40}$$

This PIOP involves 4 polynomial oracles. It requires 4 FFTs to obtain $g_p(X)$, $g_w(X), g_r(X)$, and $g_v(X)$ by polynomial interpolations, and 6 FFTs to obtain the polynomials $g_p(X)\cdot g_w(X)$ and $g_r(X)\cdot g_v(X)$.

Compared with Aurora, our scheme involves fewer polynomial oracles and FFTs, hence featuring a faster prover. Our reduction also has a faster verifier complexity due to the elimination of the public polynomial oracle about $\boldsymbol{r}$. Now, the verifier does not need to compute $g_r(X)$, which costs $O(m\ell)$ originally.

C.2 Distributed PIOP for R1CS with Constant Proof Size

This section gives the formal construction of our constant-proof distributed PIOP for R1CS as described in Theorem 3. We next show the main idea by proving Equation (7) via the two-dimension univariate sum-check.

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 in Equation (7). For $j\in[3]$, define polynomials $f^{(j)}(X,Y) = \sum_{i\in[\ell]} f_i^{(j)}(X)L_i(Y)$. Define a bivariate polynomial $f(X,Y) = f^{(1)}(X,Y)\cdot f^{(2)}(X,Y)\cdot R(z_1,Y) - f^{(3)}(r,Y)\cdot R(z_2,Y)$.

By the univariate sum-check over Equation (7), there exist $g_1\in\mathbb{F}_{m-1}[X]$ and $h_1(X)$ s.t. $\sum_{i\in[\ell]} f_i(X) = X\cdot g_1(X) + Z_{\mathbb{H}}(X)\cdot h_1(X)$. Given a random challenge α, it can be checked by verifying if the following equation holds:

$$\sum_{i\in[\ell]} f_i(\alpha) = \alpha\cdot g_1(\alpha) + Z_{\mathbb{H}}(\alpha)\cdot h_1(\alpha). \tag{41}$$

The soundness error is $m/|\mathbb{F}|$ according to the Schwartz-Zippel lemma. This is the *first univariate sum-check* over X-dimension.

Let $T_2 = \alpha g_1(\alpha) + Z_{\mathbb{H}}(\alpha)h_1(\alpha)$. By Lemma 2, it holds that $\sum_{i\in[\ell]} f_i(\alpha) = \sum_{y\in\mathbb{L}} f(\alpha,y) = T_2$. As α has been fixed, $f(\alpha,Y)$ is at most a degree-$(3\ell-3)$ *univariate* polynomial over Y, and can be invoked in the univariate sum-check

Protocol 3 (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 *size-*m *sub-vectors, and denote the* i*-th sub-vector as* $\boldsymbol{w_i}, \boldsymbol{a_i}, \boldsymbol{b_i}$, *and* $\boldsymbol{c_i}$, *respectively. Suppose sub-provers* $\mathcal{P}_1, \ldots, \mathcal{P}_\ell$ *and a master prover* $\mathcal{P}_0$. *For* $i \in [\ell]$, $\mathcal{P}_i$ *is assigned with* $\boldsymbol{w_i}, \boldsymbol{a_i}, \boldsymbol{b_i}$, *and* $\boldsymbol{c_i}$. *Assume* m *and* ℓ *are power-of-two integers.*

Provers distributedly generate secret bivariate polynomial oracles $f'_w, f_a, f_b, f_c \in \mathbb{F}_{m,\ell}[X, Y]$. *Specifically, for* $i \in [\ell]$*:*

1. $\mathcal{P}_i$*: computes coefficient-based polynomials* $f_{a_i}, f_{b_i}, f_{c_i} \in \mathbb{F}_m[X]$ *from* $\boldsymbol{a_i}, \boldsymbol{b_i}$, *and* $\boldsymbol{c_i}$, *respectively. Compute the polynomial* $f'_{w_i}(X)$ *from* $\boldsymbol{w_i}$ *according to Equation* (1). *Send polynomial oracles* $f_{a_i}(X), f_{b_i}(X), f_{c_i}(X)$, *and* $f'_{w_i}(X)$ *to* $\mathcal{P}_0$.
2. $\mathcal{P}_0$*: computes the polynomial oracles* $f_u(X, Y) = \sum_{i\in[\ell]} f_{u_i}(X)L_i(Y)$ *for all* $u \in \{a, b, c\}$. *Compute the polynomial oracle* $f'_w(X, Y) = \sum_{i\in[\ell]} f'_{w_i}(X)L_i(Y)$.

again. Hence, to check the validity of Equation (41), it suffices for the prover to show the existence of polynomials $g_2 \in \mathbb{F}_{\ell-1}[Y]$ and $h_2(Y)$ s.t.

$$f(\alpha, Y) = Y \cdot g_2(Y) + T_2/\ell + Z_{\mathbb{L}}(Y) \cdot h_2(Y), \tag{42}$$

where $f(\alpha, Y) = f^{(1)}(\alpha, Y) \cdot f^{(2)}(\alpha, Y) \cdot R(z_1, Y) - f^{(3)}(r, Y) \cdot R(z_2, Y)$. This is the *second univariate sum-check* over Y-dimension.

Handling large polynomials with small system parameters. We analyze the degrees of $h_2(Y)$ in Equation (42). For linear constraints as in Equation (8), $R(z_1, Y)$ is not needed, so $\deg(h_2) \leq \ell - 2$, and using a size-ℓ system parameter naturally suffices. However, for quadratic constraints as in Equation (9), $R(z_1, Y)$ is needed and $\deg(h_2) \leq 2\ell - 3$. If directly using a size-2ℓ system parameter, the LDT for secret polynomials like $g_2(Y)$ would be required again. To minimize the degree bound of $h_2(Y)$, we decompose it into two polynomials $h_{2,\mathsf{low}}, h_{2,\mathsf{high}} \in \mathbb{F}_\ell[Y]$ s.t. $h_2(Y) = h_{2,\mathsf{low}}(Y) + Y^\ell \cdot h_{2,\mathsf{high}}(Y)$. By definition, this decomposition is unique, and now we can still use size-ℓ system parameters.

The formal protocols. We propose the distributed PIOP for R1CS in two protocols. Assume that each $\mathcal{P}_i$ is assigned with the original R1CS sub-witness $\boldsymbol{w}_i, \boldsymbol{a}_i, \boldsymbol{b}_i$ and $\boldsymbol{c}_i$. Protocol 3 describes how to generate the secret sub-polynomials $f'_{w_i}(X), f_{a_i}(X), f_{b_i}(X), f_{c_i}(X)$ locally and the secret bivariate polynomial oracles $f'_w(X, Y), f_a(X, Y), f_b(X, Y), f_c(X, Y)$ collaboratively. Protocol 4 proves R1CS, assuming each sub-prover is assigned with its sub-polynomials.

R1CS involves 4 sum-check relations as in Equation (10), including three linear constraint relations and one quadratic constraint relation. For simplicity, Protocol 4 involves 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 using an amortized univariate sum-check via the random linear combination (*i.e.*, corresponding to the challenge s in Protocol 4).

We next give the security proof of Theorem 3.

Proof (Proof of Theorem 3). **Completeness** holds by the transformation correctness from R1CS into inner product relations as in Equation (10). Starting

Protocol 4 (Distributed PIOP for R1CS (a simplified version)).

Secret inputs: For each $i \in [\ell]$, $\mathcal{P}_i$ holds $f'_{w_i}, f_{v_i}, f_{a_i}, f_{b_i}, f'_{b_i}, f_{c_i} \in \mathbb{F}_m[X]$, where $f'_{b_i}(X) = (X^m \cdot f_{b_i}(X^{-1}) + (1 - X^m) \cdot f_{b_i}(0))$ is virtual and constructable from $f_{b_i}(X)$.
Public inputs: bivariate polynomial oracles f'_w, f_v, f_a, f_b, f_c and a virtual oracle $f'_b(X, Y) = \sum_{i \in [\ell]} f'_{b_i}(X) \cdot L_i(Y)$. Multiplicative subgroups $\mathbb{H}$ of size m and $\mathbb{L}$ of size ℓ. Public elements $u_1 \in \mathbb{F} \setminus \{\mathbb{H}, 0\}$ and $u_2 \in \mathbb{F} \setminus \{\mathbb{L}, 0\}$.
Statement: 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) $\mathcal{V}$: sends random challenges $r, s \in \mathbb{F}$ to $\mathcal{P}_0$, who transfers to all $\mathcal{P}_i$.

(b) $\mathcal{V}$: computes $\boldsymbol{r} = (r^0, \ldots, r^{m\ell-1})$ and $\boldsymbol{r}^\top P_v = \boldsymbol{p}$. Compute coefficient-based polynomial $f_{p_i}(X)$ from $\boldsymbol{p}$ and $f_p(X, Y) = \sum_{i \in [\ell]} f_{p_i}(X) L_i(Y)$.

2. Provers compute and send polynomial oracles $g_1, h_1 \in \mathbb{F}_m[X]$ to $\mathcal{V}$. This is the ***first-time univariate sum-check*** over X-dimension.

(a) $\mathcal{P}_i$: computes the coefficient-based polynomial $f_{p_i} \in \mathbb{F}_m[X]$ from $\boldsymbol{p_i}$.

(b) $\mathcal{P}_i$: computes polynomials $f_{1,i}, f_{2,i}, f_i \in \mathbb{F}_{2m-1}[X]$ s.t.

$$f_{1,i}(X) = f_{p_i}(X) \cdot f'_{w_i}(X) - R_i(r^m) \cdot f_{v_i}(r),$$
$$f_{2,i}(X) = R_i(r^m) \cdot f_{a_i}(rX) \cdot f'_{b_i}(X) - R_i(r^m) \cdot f_{c_i}(r).$$

Let $f_i(X) = f_{1,i}(X) + s \cdot f_{2,i}(X)$.

(c) $\mathcal{P}_i$: computes $g_{1,i}(X)$ and $h_{1,i}(X)$ by dividing $Z_{\mathbb{H}}(X)$ over $f_i(X)$ s.t.

$$(X - u_1) \cdot f_i(X) = X \cdot g_{1,i}(X) + (X - u_1) \cdot T_{1,i} + (X - u_1) \cdot Z_{\mathbb{H}}(X) \cdot h_{1,i}(X).$$

$\mathcal{P}_i$ sends polynomial oracles $g_{1,i}(X)$ and $h_{1,i}(X)$ to $\mathcal{P}_0$.

(d) $\mathcal{P}_0$: computes oracles $g_1(X) = \sum_{i \in [\ell]} g_{1,i}(X)$ and $h_1(X) = \sum_{i \in [\ell]} h_{1,i}(X)$.

3. $\mathcal{V}$ sends a random challenge $\alpha \in \mathbb{F}$ to $\mathcal{P}_0$, who transfers to $\mathcal{P}_i$.

4. Provers compute and send polynomial oracles $g_2, h_{2,\mathsf{low}}, h_{2,\mathsf{high}} \in \mathbb{F}_\ell[Y]$ to $\mathcal{V}$. This is the ***second-time univariate sum-check*** over Y-dimension.

(a) $\mathcal{P}_i$: sends $f_{p_i}(\alpha), f'_{w_i}(\alpha), f_{v_i}(r), f_{a_i}(r\alpha), f'_{b_i}(\alpha), f_{c_i}(r), R_i(r^m), g_{1,i}(\alpha), h_{1,i}(\alpha)$ to $\mathcal{P}_0$.

(b) $\mathcal{P}_0$: computes $f_p(\alpha, Y), f'_w(\alpha, Y), f_v(r, Y), f_a(r\alpha, Y), f'_b(\alpha, Y), f_c(r, Y), R(r^m, Y)$, $g_1(\alpha)$ and $h_1(\alpha)$. Let $f(\alpha, Y) = f_1(\alpha, Y) + s \cdot f_2(\alpha, Y)$ s.t.

$$f_1(\alpha, Y) = f_p(\alpha, Y) \cdot f'_w(\alpha, Y) - R(r^m, Y) \cdot f_v(r, Y),$$
$$f_2(\alpha, Y) = R(r^m, Y) \cdot f_a(r\alpha, Y) \cdot f'_b(\alpha, Y) - R(r^m, Y) \cdot f_c(r, Y).$$

(c) $\mathcal{P}_0$: computes $g_2, h_{2,\mathsf{low}}, h_{2,\mathsf{high}} \in \mathbb{F}_\ell[Y]$ by dividing $Z_{\mathbb{L}}(Y)$ over $f(\alpha, Y)$ s.t.

$$(Y - u_2) \cdot f(\alpha, Y) = Y \cdot g_2(Y) + (Y - u_2) \cdot \big(T_2/\ell + Z_{\mathbb{L}}(Y) \cdot (h_{2,\mathsf{low}}(Y) + Y^\ell \cdot h_{2,\mathsf{high}}(Y))\big), \tag{43}$$

where $T_2 = \alpha \cdot g_1(\alpha)/(\alpha - u_1) + Z_{\mathbb{H}}(\alpha) \cdot h_1(\alpha)$ as $\sum_{i \in [\ell]} T_{1,i} = 0$.

5.(a) $\mathcal{V}$: queries $g_1(\alpha)$ and $h_1(\alpha)$ to compute T_2. Pick a random $\beta \in \mathbb{F}$.

(b) $\mathcal{V}$: queries $f'_w(\alpha, \beta), f_v(r, \beta), f_a(r\alpha, \beta), f'_b(\alpha, \beta)$ and $f_c(r, \beta)$. Query $g_2(\beta)$, $h_{2,\mathsf{low}}(\beta)$ and $h_{2,\mathsf{high}}(\beta)$.

(c) $\mathcal{V}$: computes $f_p(\alpha, \beta)$ and $R(r^m, \beta)$. Accept iff Equation (43) holds at $Y = \beta$.

from Equation (10), completeness holds due to the completeness of amortized univariate sum-check [5] to prove multiple sum-check relations at one time.

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

If the PIOP has knowledge soundness, *i.e.*, the polynomials $f'_w(X, Y)$, $f_v(X, Y)$, $f_a(X, Y)$, $f_b(X, Y)$ and $f_c(X, Y)$ are extractable, then the witness vectors $\boldsymbol{w}, \boldsymbol{v}, \boldsymbol{a}, \boldsymbol{b}$ and $\boldsymbol{c}$ are also extractable efficiently in $O(m\ell)$ time.

Complexity. We present the complexity analysis below.

Sub-prover. The time complexity of $\mathcal{P}_i$ is $O(m)$ for running Protocol 3 to compute size-$O(m)$ sub-polynomials. Assume each sub-matrix $P_v^{(i)}$ has $O(m)$ non-zero entries. [18] Then, $\mathcal{P}_i$ spends $O(m)$ time to compute the $O(m)$ non-zero entries on $\boldsymbol{p}_i = \boldsymbol{r}^\top P_v^{(i)}$ and $f_{p_i}(X)$. Computing $g_{1,i}(X)$ and $h_{1,i}(X)$ costs $O(m \log m)$ time due to FFTs for obtaining $f_i(X)$ and $O(m)$ time for polynomial division. Computing and sending $O(1)$ evaluations on size-m univariate polynomials $f_{p_i}(X), f'_{w_i}(X), f_{v_i}(X), f_{a_i}(X), f_{b_i}(X), f_{c_i}(X), R_i(X)$ 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 polynomials oracles $g_1(X)$ and $h_1(X)$. Given evaluations in Step 4a, $\mathcal{P}_0$ requires $O(\ell)$ time to obtain the size-ℓ polynomials (*e.g.*, $f_p(\alpha, Y)$) and evaluations on size-m polynomials (*e.g.*, $g_1(\alpha)$) in Step 4b. Then, $\mathcal{P}_0$ interpolates these polynomials via FFT to obtain the polynomial $f(\alpha, Y)$. Thus, computing $g_2(Y)$ and $h_2(Y)$ costs $O(\ell \log \ell)$ time. Hence, the total master prover complexity is $O(\ell \log \ell)$.

Proof size. The involved polynomial oracles include $f'_w(X, Y), f_a(X, Y), f_b(X, Y), f_c(X, Y), g_1(X), h_1(X), g_2(Y), h_{2,\mathsf{low}}(Y), h_{2,\mathsf{high}}(Y)$. The proof size comes from sending constant evaluations of these polynomials in Step 5, hence costing $O(1)$.

Communication complexity. Communication between $\mathcal{P}_i$ and $\mathcal{P}_0$ includes: 1) oracles $f'_w(X, Y), f_a(X, Y), f_b(X, Y)$ and $f_c(X, Y)$; 2) oracles $g_{1,i}(X)$ and $h_{1,i}(X)$; 3) constant-number evaluations on $f_{p_i}(X), f'_{w_i}(X), f_{v_i}(X), f_{a_i}(X), f_{b_i}(X), f_{c_i}(X), R_i(X), g_{1,i}(\alpha)$ and $h_{1,i}(\alpha)$. Assuming all oracle sizes are of $O(1)$ (this assumption is valid as they are instantiated with the univariate KZG or bivariate KZG), the amortized communication complexity is $O(1)$.

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

[18] Our "making-even" algorithm as in Algorithm 1 would guarantee this assumption.

C.3 Distributed PIOP for R1CS with Succinct Verification

This section presents the details of our distributed preprocessing. We first give the distributed lookup PIOP for column relations in Protocol 5. A visualization of this protocol is given in Figure 6.

Proof (Proof of Theorem 4). **Completeness.** Completeness holds directly by the protocol construction, if the following two equations are equivalent:

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

We next show the equivalence.

By Lemma 2, we have

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

As $\deg(g_i) < \tilde{m}$ for all $i \in [\ell]$, by the univariate sum-check [5], this sum equals $\tilde{m} \cdot \sum_{i\in[\ell]} g_i(0)$ and hence $\tilde{m} \cdot \sum_{y\in\mathbb{L}} g(0,y)$. Again, as the Y-degree of $g(X,Y)$ is less than ℓ, by the univariate sum-check we have

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

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

Soundness. If there exists $(b(x,y), B(x,y)) \notin \{(\omega^{j-1}, \alpha^{j-1})\}_{j\in[m]}$ for some $x \in \mathbb{M}$ and $y \in \mathbb{L}$, then for the random challenge ξ picked by the verifier, $\xi \cdot b(x,y) + B(x,y) \in \{\xi \cdot \omega^{j-1} + \alpha^{j-1}\}_{j\in[m]}$ holds with a probability of at most $m/|\mathbb{F}|$. 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 $T_{\mathsf{col}}(X), h(Y)$ is at most $3m/|\mathbb{F}|$ according to the Schwartz-Zippel lemma.

For the validity check of $g(X,Y)$, the soundness error of the transformation from Equation (16) to Equation (17) is $\ell/|\mathbb{F}|$ according to Lemma 4 [7]. Checking the validity at $Y = \zeta$ as in Step 8 has a soundness error of $3\ell/|\mathbb{F}|$.

Lemma 4 ([7]). *Fix $F_1, \ldots, F_k \in \mathbb{F}_d[X]$. Fix $Z \in \mathbb{F}_d[X]$ that decomposes to distinct linear factors over $\mathbb{F}$. Suppose for some $i \in [n]$, $Z(X) \nmid F_i(X)$. Then, except with probability $k/|\mathbb{F}|$ over the choice of random $\gamma \in \mathbb{F}$, we have $Z(X) \nmid G(X)$, where $G(X) = \sum_{j=1}^{k} \gamma^{j-1} \cdot F_j(X)$.*

By the union bound argument, the total soundness error is $(4m+\tilde{m}m\ell+3\ell)/|\mathbb{F}|$.

Complexity. For the master prover, computing the polynomials $T_{\mathsf{col}}(X), h(Y)$ and $q_h(Y)$ requires $O(m \log m)$ time, computing the polynomial oracles $B(X,Y)$, $g(X,Y), q_g(X)$ needs $O(\ell)$ time, and computing the polynomials $g_5(Y), h_{5,\mathsf{low}}(Y)$ and $h_{5,\mathsf{high}}(Y)$ needs $O(\ell \log \ell)$ time. Thus, the total master prover complexity is $O(m \log m + \ell \log \ell)$.

Protocol 5 (Distributed lookup PIOP for column).

Public inputs: An online challenge α. Multiplicative subgroups $\mathbb{M}$ of order $\tilde{m}$ and $\mathbb{L}$ of order ℓ. The generator ω of an order-m subgroup $\mathbb{H}$. Public $u_2 \in \mathbb{F} \setminus \{\mathbb{L}, 0\}$.

Statement: Provers proves to $\mathcal{V}$ the validity of the first equation in Equation (13).

Offline phase:

1. *The indexer $\mathcal{I}$ computes $\{b_i(X)\}_{i\in[\ell]}$ s.t. $b_i(x) = \omega^{\mathsf{col}_i(x)}$ for all $x \in \mathbb{M}$. Compute $b(X,Y) = \sum_{i\in[\ell]} b_i(X)L_i(Y)$. Compute $R(X,Y) = \sum_{i\in[\ell]} X^{i-1} \cdot L_i(Y)$.*
2. *$\mathcal{I}$: computes a polynomial $n_{\mathsf{col}} \in \mathbb{F}_m[Y]$ s.t. $n_{\mathsf{col}}(\omega^{j-1})$ equals the frequency times of $b(x,y) = j-1$ (or say, $\mathsf{col}(x,y) = j-1$) for all $x \in \mathbb{M}$ and $y \in \mathbb{L}$.*
3. *$\mathcal{I}$: sends $n_{\mathsf{col}}(Y)$ to $\mathcal{P}_0$ as secret inputs, sends $\mathsf{col}_i(X), b_i(X), R_i(X)$ to $\mathcal{P}_i$ as secret inputs and sends the oracles $b(X,Y), n_{\mathsf{col}}(Y)$ and $R(X,Y)$ to $\mathcal{V}$.*

Online phase:

1. *Given α, provers compute the oracle $B(X,Y)$ to $\mathcal{V}$. Specifically, for each i, $\mathcal{P}_i$ computes $B_i(X)$ s.t. $B_i(x) = \alpha^{\mathsf{col}_i(x)}$ for all $x \in \mathbb{M}$. $\mathcal{P}_i$ sends oracle $B_i(X)$ to $\mathcal{P}_0$, who then computes the oracle $B(X,Y) = \sum_{i\in[\ell]} B_i(X)L_i(Y)$ and sends it to $\mathcal{V}$.*
2. *$\mathcal{P}_0$: computes the table-related polynomial $T_{\mathsf{col}} \in \mathbb{F}_m[X]$ s.t. $T_{\mathsf{col}}(\omega^{j-1}) = \alpha^{j-1}$ for $j \in [m]$. Send the polynomial oracle $T_{\mathsf{col}}(X)$ to $\mathcal{V}$.*
3. *$\mathcal{V}$: sends challenges $\xi, \gamma \in \mathbb{F}$ to $\mathcal{P}_0$, who transfers them to all $\mathcal{P}_i$.*
4. *Provers compute and send polynomial oracles $h(Y), g(X,Y)$ to $\mathcal{V}$. The lookup relation is then reduced into Equation (15).*
   (a) *$\mathcal{P}_0$: computes $h \in \mathbb{F}_m[Y]$ s.t. $h(y) = \frac{n_{\mathsf{col}}(y)}{\gamma+\xi\cdot y+T_{\mathsf{col}}(y)}$ for all $y \in \mathbb{H}$.*
   (b) *$\mathcal{P}_i$: computes $g_i \in \mathbb{F}_{\tilde{m}}[X]$ s.t. $g_i(x) = \frac{1}{\gamma+\xi\cdot b_i(x)+B_i(x)}$ for $x \in \mathbb{M}$. Send polynomial oracle $g_i(X)$ to $\mathcal{P}_0$.*
   (c) *$\mathcal{P}_0$: computes the oracle $g(X,Y) = \sum_{i\in[\ell]} g_i(X)L_i(Y)$.*
5. *$\mathcal{V}$: sends a random challenge $r \in \mathbb{F}$ to $\mathcal{P}_0$, who transfers to all $\mathcal{P}_i$.*
6. *Provers compute and send polynomial oracles $q_g(X), q_h(Y)$ to $\mathcal{V}$.*
   (a) *$\mathcal{P}_0$: computes $q_h \in \mathbb{F}_m[Y]$ s.t. $h(Y)\cdot(\gamma+\xi\cdot Y+T_{\mathsf{col}}(Y))-n_{\mathsf{col}}(Y) = q_h(Y)\cdot Z_{\mathbb{H}}(Y)$.*
   (b) *$\mathcal{P}_i$: computes $q_{g,i} \in \mathbb{F}_{\tilde{m}}[X]$ by Equation (49). Send the oracle $q_{g,i}(X)$ to $\mathcal{P}_0$.*
   (c) *$\mathcal{P}_0$: computes the oracle $q_g(X) = \sum_{i\in[\ell]} r^{i-1} \cdot q_{g,i}(X)$.*
7. *$\mathcal{V}$: sends a random challenge $\delta \in \mathbb{F}$ to $\mathcal{P}_0$, who transfers to all $\mathcal{P}_i$.*
8. *Provers compute and send polynomial 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 $g(X,Y)$ via a univariate sum-check as in Equation (17). To achieve this, $\mathcal{P}_i$ first computes and sends $R_i(r), g_i(\delta), b_i(\delta), B_i(\delta)$ and $q_{g,i}(\delta)$ to $\mathcal{P}_0$. $\mathcal{P}_0$ then computes polynomials $R(r,Y), g(\delta,Y), b(\delta,Y), B(\delta,Y), q_g(\delta,Y)$ from Lagrange polynomials, and $g_5(Y), h_{5,\mathsf{low}}(Y), h_{5,\mathsf{high}}(Y)$ s.t.*

$$\begin{aligned}&(Y-u_2)\cdot R(r,Y)\cdot g(\delta,Y)\cdot\big(\gamma+\xi\cdot b(\delta,Y)+B(\delta,Y)\big)-(Y-u_2)\cdot R(r,Y)\\ &= Y\cdot g_5(Y)+(Y-u_2)\cdot q_g(\delta)\cdot Z_{\mathbb{M}}(\delta)/\ell\\ &+(Y-u_2)\cdot Z_{\mathbb{L}}(Y)\cdot(h_{5,\mathsf{low}}(Y)+\delta^\ell\cdot h_{5,\mathsf{high}}(Y)).\end{aligned}$$

9. *$\mathcal{V}$ outputs accept if and only if all the following checks pass:*
   (a) *The table (i.e., $T_{\mathsf{col}}(X)$) is valid. Query $T_{\mathsf{col}}(\omega\delta)$ and $T_{\mathsf{col}}(\delta)$. Check if Equation (14) holds at $X = \delta$.*
   (b) *$h(Y)$ is valid. Query $h(\delta), n_{\mathsf{col}}(\delta), q_h(\delta)$. Check if the equation in Step 6a holds.*
   (c) *$g(X,Y)$ is valid. Pick a random $\zeta \in \mathbb{F}$. Query $g(\delta,\zeta), b(\delta,\zeta), B(\delta,\zeta)$ and $R(r,\zeta)$. Query $q_g(\zeta), g_5(\zeta), h_{5,\mathsf{low}}(\zeta), h_{5,\mathsf{high}}(\zeta)$. Check if the equation in Step 8 holds.*
   (d) *The lookup relation as in Equation (15) is valid. Query $g(0,0)$ and $h(0)$. Check if $\tilde{m}\ell\cdot g(0,0) = m\cdot h(0)$.*

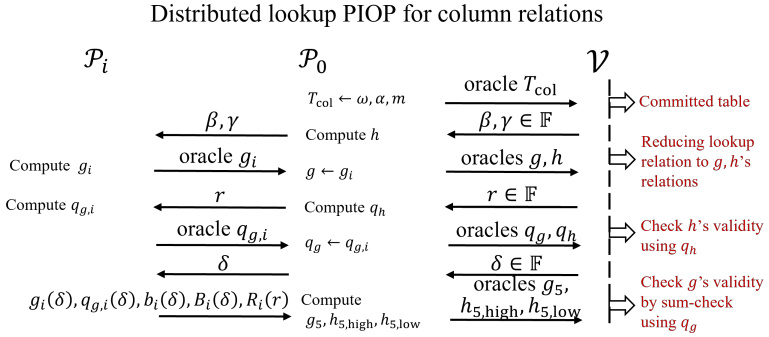

Fig. 6. Visualization of the distributed lookup PIOP

For sub-provers, computing the evaluations of $R_i(r), g_i(\delta), b_i(\delta), B_i(\delta)$ and $q_{g,i}(\delta)$ costs $O(\tilde{m})$ time, and computing polynomials $B_i(X), g_i(X)$ and $q_{g,i}(X)$ costs $O(\tilde{m}\log\tilde{m})$ time. Hence, the total sub-prover complexity is $O(m\log m)$ as $\tilde{m} = O(m)$.

The verifier's overhead is mainly due to the time-$O(\log m\ell)$ computation for $L_{\mathbb{H},m}(\delta), Z_{\mathbb{M}}(\delta)$ and $Z_{\mathbb{L}}(\zeta)$. These polynomial evaluations can be delegated by the provers. The verifier complexity is then reduced to $O(1)$. Note that these at most introduce an additional overhead of $O(m+\ell)$ for the master prover, and hence do not affect the total prover complexities. □

Batch distributed lookup PIOP. In our distributed PIOP for R1CS, there include three public matrices, each involving one row and one column lookup relation. Hence, we need a batch version of Protocol 5.

For multiple relations with the same table, we can reuse the same $T_{\mathsf{col}}(\mathrm{X})$. Further, to prove the validity of multiple polynomials $\{h_k(Y)\}_{k\in[t]}$ for t sets of polynomials $\{T_{k,\mathsf{col}}(Y), n_{k,\mathsf{col}}(Y)\}_{k\in[t]}$, we can combine the quotient relations using a random linear combination with a random challenge v, leading to a single quotient polynomial $q_h(Y)$:

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

Similarly, to prove the validity of multiple $g_k(X,Y)$ for t sets of bivariate polynomials $\{b_k(X,Y), B_k(X,Y)\}_{k\in[t]}$ as in Equation (17), we use a random linear combination to prove the existence of a single quotient polynomial $q_g(X)$ s.t.

$$\sum_{y\in\mathbb{L}}\sum_{k\in[t]} v^{k-1}\cdot R(r,y)\cdot\big(g_k(X,y)\cdot(\gamma+\xi\cdot b_k(X,y)+B_k(X,y))-1\big)=q_g(X)\cdot Z_{\mathbb{M}}(X).$$

According to [7], if any polynomial does not divide $q_g(X)$ (or $q_h(Y)$), then the probability that the random linear combination of these polynomials does not divide $q_g(X)$ (or $q_h(Y)$) is at most $t/|\mathbb{F}|$. Now, only 2 polynomial oracles (*i.e.*,

$q_g(X), q_h(Y)$) are required regardless of the lookup relation number, which is in contrast with the trivial t ones.

Modifications for Equation (12). After decomposing the table into two sub-ones, we need to modify the polynomial $A(X,Y)$ in Equation (12). 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, Equation (12) is finally written as

$$f_p(\alpha,\beta) = \sum_{x\in\mathbb{M},y\in\mathbb{L}} \mathsf{val}(x,y)\cdot A_{\mathsf{high}}(x,y)\cdot A_{\mathsf{low}}(x,y)\cdot B(x,y)\cdot L(\beta,y). \tag{44}$$

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 row lookup relation, is transformed into two sub-relations as in Equation (18), which can be proved distributively in a similar way to the column relation.

Distributed indexer. The indexer's computation can be distributed similarly to the proving phase. This is because the indexer only needs to compute degree-$(\tilde{m},\ell)$ bivariate polynomials and degree-m univariate polynomials. The former can be handled distributively in a similar way to Protocol 3. The latter costs $O(m\log m)$ time and can be handled by sub-workers directly. Hence, given ℓ workers, the sub-worker complexity is $O(m\log m)$, and the master-worker complexity is $O(\ell\log\ell + m\log m)$. Protocol 6 gives the distributed offline phase.

Putting everything together. Based on the distributed lookup PIOP, Protocol 6 (describing the offline phase) and Protocol 7 (describing the online phase) give our distributed PIOP for R1CS with sublinear verification. These protocols are the modification of Protocol 4, and we only describe the changes.

Our distributed PIOP for R1CS is a combination of Protocol 4 and the distributed lookup PIOP in Protocol 5. The security and complexity properties inherit from these two protocols, and we omit the proof for Theorem 5.

D Algorithm to Make the R1CS Sub-Matrices "Even"

This sections proposes the "making-even" algorithm for R1CS matrices in Algorithm 1, which takes the number of sub-provers ℓ and the R1CS public matrices P_a, P_b, P_c as inputs, and outputs modified sub-matrices $\{P_a^{(i)}, P_b^{(i)}, P_c^{(i)}\}_{i\in[\ell]}$ such that their non-zero entries are evenly distributed. It does not take the witness as inputs. This is feasible as the locations of entries in R1CS matrices are only determined by the gate indexes of a circuit, and these indexes are adjustable.

An ideal target is a "truly"-even distribution. Its main challenge is that, for example, if we exchange $P_a[i]$ to the j-th column, the i-th columns of P_b and P_c would be also changed, which may prevent achieving a "truly"-even distribution of non-zero entries across all matrices simultaneously. The problem is indeed Multiway-number-partitioning [23], which is NP-hard. We then target practical results instead of theoretical guarantees. Also, we observe that we can adopt a more relaxed balancing criterion as $\tilde{m}$ (*i.e.*, the maximum non-zero number in all sub-matrices) must be a power-of-two integer for being an order of a multiplicative subgroup. We then define a "loosely"-even distribution

Protocol 6 (Distributed PIOP for R1CS with sublinear verifier, offline).
Suppose the same setting as Protocol 3. Suppose an indexer $\mathcal{I}$ with ℓ sub-indexers $\mathcal{I}_1, \ldots, \mathcal{I}_\ell$ and a master indexer $\mathcal{I}_0$.

***Input**: public R1CS matrices P_v.*

***Output**:*

- *polynomials* $\mathsf{val}_i(X), \mathsf{col}_i(X), \mathsf{row}_{\mathsf{high},i}(X), \mathsf{row}_{\mathsf{low},i}(X), a_{\mathsf{high},i}(X), a_{\mathsf{low},i}(X), b_i(X), R_i(X)$ *and* $L_i(X)$. *These would be given to* $\mathcal{P}_i$.
- *polynomials and oracles* $n_{\mathsf{col}}(Y), n_{\mathsf{high}}(Y), n_{\mathsf{low}}(Y)$, *which would be given to* $\mathcal{P}_0$.
- *oracles* $\mathsf{val}(X,Y), a_{\mathsf{high}}(X,Y), a_{\mathsf{low}}(X,Y), b(X,Y), L(X,Y), R(X,Y), n_{\mathsf{col}}(Y), n_{\mathsf{row},\mathsf{high}}(Y)$ *and* $n_{\mathsf{row},\mathsf{low}}(Y)$. *These would be given to* $\mathcal{V}$.

1. *$\mathcal{I}$: reads the public matrix P_v. For all $i \in [\ell]$, compute 4 polynomials $\mathsf{val}_i, \mathsf{col}_i, \mathsf{row}_{\mathsf{high},i}, \mathsf{row}_{\mathsf{low},i} \in \mathbb{F}_{\tilde{m}}[X]$, which describe $P_v^{(i)}$, i.e., the i-th sub-matrix of P_v. Compute bivariate polynomial oracles $\mathsf{val}(X,Y), \mathsf{col}(X,Y), \mathsf{row}_{\mathsf{high}}(X,Y)$ and $\mathsf{row}_{\mathsf{low}}(X,Y)$.*
   (a) *$\mathcal{I}_i$: reads the sub-matrix $P_v^{(i)}$. Compute polynomials $\mathsf{val}_i(X), \mathsf{col}_i(X), \mathsf{row}_{\mathsf{high},i}(X)$ and $\mathsf{row}_{\mathsf{low},i}(X)$, and send their oracles to $\mathcal{I}_0$.*
   (b) *$\mathcal{I}_0$: computes polynomial oracles $\mathsf{val}(X,Y) = \sum_{i\in[\ell]} \mathsf{val}_i(X)L_i(Y)$, $\mathsf{col}(X,Y) = \sum_{i\in[\ell]} \mathsf{col}_i(X)L_i(Y)$, $\mathsf{row}_{\mathsf{high}}(X,Y) = \sum_{i\in[\ell]} \mathsf{row}_{\mathsf{high},i}(X)L_i(Y)$, and $\mathsf{row}_{\mathsf{low}}(X,Y) = \sum_{i\in[\ell]} \mathsf{row}_{\mathsf{low},i}(X)L_i(Y)$.*
2. *$\mathcal{I}$: computes polynomial oracles $n_{\mathsf{col}}, n_{\mathsf{high}}, n_{\mathsf{low}} \in \mathbb{F}_m[X]$, which count the evaluation frequency of $\{\mathsf{col}(x,y), \mathsf{row}_{\mathsf{high}}(x,y), \mathsf{row}_{\mathsf{low}}(x,y)\}_{x\in\mathbb{M},y\in\mathbb{L}}$, respectively.*
   (a) *$\mathcal{I}_i$: counts the evaluation frequency of $\{\mathsf{col}_i(x), \mathsf{row}_{\mathsf{high},i}(x), \mathsf{row}_{\mathsf{low},i}(x)\}_{x\in\mathbb{M}}$. We use $\mathsf{col}_i(X)$ as an example here, and $\mathsf{row}_{\mathsf{high},i}(x)$ and $\mathsf{row}_{\mathsf{low},i}(X)$ are similar. The frequency form for $\mathsf{col}_i(X)$ is $\{(0, n_{\mathsf{col},i,0}), (1, n_{\mathsf{col},i,1}), \ldots, (m-1, n_{\mathsf{col},i,m-1})\}$, where $n_{\mathsf{col},i,j-1} (j \in [m])$ equals to the times of $\mathsf{col}_i(x) = j-1$ for $x \in \mathbb{M}$. Note that by definition, the evaluations of $\mathsf{col}_i(x)$ exactly belong to $0 \cup [m-1]$ for $x \in \mathbb{M}$. Send $\{n_{\mathsf{col},j-1}\}_{j\in[m]}$ to $\mathcal{I}_0$.*
   (b) *$\mathcal{I}_0$: computes $n_{\mathsf{col},j-1} = \sum_{i\in[\ell]} n_{\mathsf{col},i,j-1}$ for $j \in [m]$. Compute $n_{\mathsf{col}}(X)$ via polynomial interpolation by seeing $\{n_{\mathsf{col},j-1}\}_{j\in[m]}$ as their evaluations over $\mathbb{H}$. Compute $n_{\mathsf{high}}(X), n_{\mathsf{low}}(X)$ and their oracles similarly.*
3. *$\mathcal{I}$: computes polynomials $a_{\mathsf{high},i}(X), a_{\mathsf{low},i}(X), b_i(X) \in \mathbb{F}_{\tilde{m}}[X]$ defined by $\{\omega^{\mathsf{row}_{\mathsf{high},i}(x)}, \omega^{\mathsf{row}_{\mathsf{low},i}(x)}, \omega^{\mathsf{col}_i(x)}\}_{x\in\mathbb{M}}$, respectively. Compute bivariate polynomial oracles $a_{\mathsf{high}}, a_{\mathsf{low}}, b \in \mathbb{F}_{\tilde{m},\ell}[X,Y]$.*
   (a) *$\mathcal{I}_i$: computes $a_{\mathsf{high},i}(X), a_{\mathsf{low},i}(X), b_i(X) \in \mathbb{F}_{\tilde{m}}[X]$ defined by $\{\omega^{\mathsf{row}_{\mathsf{high},i}(x)}, \omega^{\mathsf{row}_{\mathsf{low},i}(x)}, \omega^{\mathsf{col}_i(x)}\}_{x\in\mathbb{M}}$, respectively. Send their oracles to $\mathcal{I}_0$.*
   (b) *$\mathcal{I}_0$: computes bivariate polynomial oracles $a_{\mathsf{high}}(X,Y) = \sum_{i\in[\ell]} a_{\mathsf{high},i}(X)L_i(Y)$, $a_{\mathsf{low}}(X,Y) = \sum_{i\in[\ell]} a_{\mathsf{low},i}(X)L_i(Y)$, and $b(X,Y) = \sum_{i\in[\ell]} b_i(X)L_i(Y)$.*
4. *$\mathcal{I}$: computes bivariate polynomial oracles $R(X,Y) = \sum_{i\in[\ell]} R_i(X)L_i(Y)$ and $L(X,Y) = \sum_{i\in[\ell]} L_i(X)L_i(Y)$, where $R_i(X) = X^{i-1}$.*
   (a) *$\mathcal{I}_i$: computes polynomials $R_i(X)$ and $L_i(X)$. Send their oracles to $\mathcal{I}_0$.*
   (b) *$\mathcal{I}_0$: computes polynomial oracles $R(X,Y)$ and $L(X,Y)$.*

Protocol 7 (Distributed PIOP for R1CS with sublinear verifier, online).

Input: *$\mathcal{P}_i$ holds polynomials $\mathsf{val}_i(X), \mathsf{col}_i(X), \mathsf{row}_{\mathsf{high},i}(X)$, $\mathsf{row}_{\mathsf{low},i}(X)$, $a_{\mathsf{high},i}(X)$, $a_{\mathsf{low},i}(X)$, $b_i(X), R_i(X)$ and $L_i(X)$. $\mathcal{P}_0$ holds polynomials $n_{\mathsf{col}}(Y)$, $n_{\mathsf{high}}(Y)$ and $n_{\mathsf{low}}(Y)$. $\mathcal{V}$ holds polynomial oracles $\mathsf{val}(X,Y), a_{\mathsf{high}}(X,Y), a_{\mathsf{low}}(X,Y), b(X,Y)$, $L(X,Y), R(X,Y), n_{\mathsf{col}}(Y)$, $n_{\mathsf{row,high}}(Y)$ and $n_{\mathsf{row,low}}(Y)$. All these polynomial (oracle) inputs are from the indexer $\mathcal{I}$. All provers and $\mathcal{V}$ know multiplicative subgroups $\mathbb{H}, \mathbb{M}$ and $\mathbb{L}$, elements $u_3 \in \mathbb{F} \setminus \{\mathbb{M}, 0\}$ and $u_4 \in \mathbb{F} \setminus \{\mathbb{L}, 0\}$.*

1. *In Step 2 of Protocol 4, after receiving r, provers sends to $\mathcal{V}$ the oracles $A_{\mathsf{high}}, A_{\mathsf{low}} \in \mathbb{F}_{\tilde{m},\ell}[X,Y]$ and $T_{\mathsf{high}}, T_{\mathsf{low}} \in \mathbb{F}_m[X]$ for the lookup relations of the rows.*
   (a) *$\mathcal{P}_i$: computes $A_{\mathsf{high},i}, A_{\mathsf{low},i} \in \mathbb{F}_{\tilde{m}}[X]$ defined by $\{(r^{\sqrt{m\ell}})^{\mathsf{row}_{\mathsf{high},i}(x)}, r^{\mathsf{row}_{\mathsf{low},i}(x)}\}_{x\in\mathbb{M}}$, respectively. Send oracles $A_{\mathsf{high},i}(X)$ and $A_{\mathsf{low},i}(X)$ to $\mathcal{P}_0$.*
   (b) *$\mathcal{P}_0$: computes bivariate oracles $A_{\mathsf{high}}(X,Y)$ and $A_{\mathsf{low}}(X,Y)$. Compute $T_{\mathsf{high}}, T_{\mathsf{low}} \in \mathbb{F}_m[X]$ 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 of Protocol 4, provers additionally computes and sends to $\mathcal{V}$ the oracle $B \in \mathbb{F}_{\tilde{m},\ell}[X,Y]$ and oracle $T_{\mathsf{col}} \in \mathbb{F}_m[X]$ for the lookup relations of the column.*
   (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 oracle $B(X,Y)$ and $T_{\mathsf{col}} \in \mathbb{F}_m[X]$ s.t. $\{T_{\mathsf{col}}(\omega^{j-1}) = \alpha^{j-1}\}_{j\in[m]}$.*
3. *In Step 5 of Protocol 4, the provers sends $f_p(\alpha,\beta)$ and proves its validity to $\mathcal{V}$.*
   (a) *$\mathcal{P}\&\mathcal{V}$: invoke Protocol 5 to show $A_{\mathsf{high}}(X,Y), A_{\mathsf{low}}(X,Y)$ and $B(X,Y)$ are well-formed as in Equations (13) and (18). In this protocol, polynomial oracles $T_{\mathsf{high}}(X), n_{\mathsf{high}}(X), A_{\mathsf{high}}(X,Y), a_{\mathsf{high}}(X,Y); T_{\mathsf{low}}(X), n_{\mathsf{low}}(X), A_{\mathsf{low}}(X,Y)$, $a_{\mathsf{low}}(X,Y)$, and $T_{\mathsf{col}}(X,Y), n_{\mathsf{col}}(X), B(X,Y), b(X,Y)$ would be queried.*
   (b) *Provers compute and send polynomial oracles $g_3, \{h_{3,j}\}_{j\in[3]} \in \mathbb{F}_{\tilde{m}}[X]$ to $\mathcal{V}$.*
      i. *$\mathcal{P}_i$: computes and sends oracles $g_{3,i}, \{h_{3,j,i}\}_{j\in[3]} \in \mathbb{F}_{\tilde{m}}[X]$ to $\mathcal{P}_0$ s.t.*

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

      ii. *$\mathcal{P}_0$: computes oracles $g_3(X) = \sum_{i\in[\ell]} g_{3,i}(X)$, $h_{3,j}(X) = \sum_{i\in[\ell]} h_{3,j,i}(X)$.*
   (c) *$\mathcal{V}$: sends a random challenge $\delta \in \mathbb{F}$ to $\mathcal{P}_0$, who transfers to $\mathcal{P}_i$.*
   (d) *Provers compute and send polynomial oracles $g_4, \{h_{4,j}\}_{j\in[4]} \in \mathbb{F}_\ell[Y]$ to $\mathcal{V}$.*
      i. *$\mathcal{P}_i$: sends $\mathsf{val}_i(\delta)$,$A_{\mathsf{high},i}(\delta)$,$A_{\mathsf{low},i}(\delta)$,$B_i(\delta)$, $L_i(\beta), g_{3,i}(\delta), \{h_{3,j,i}(\delta)\}_{j\in[3]}$ to $\mathcal{P}_0$.*
      ii. *$\mathcal{P}_0$: computes $\mathsf{val}(\delta,Y), A_{\mathsf{high}}(\delta,Y), A_{\mathsf{low}}(\delta,Y), B(\delta,Y)$ and $L(\beta,Y)$. Compute $g_3(\delta)$ and $h_{3,j}(\delta)$. Compute polynomials $g_4(Y)$ and $\{h_{4,j}(Y)\}_{j\in[4]}$ s.t.*

$$\begin{aligned}&(Y-u_4)\cdot \mathsf{val}(\delta,Y)\cdot A_{\mathsf{high}}(\delta,Y)\cdot A_{\mathsf{low}}(\delta,Y)\cdot B(\delta,Y)\cdot L(\beta,Y)\\ =&Y\cdot g_4(Y)+(Y-u_4)\cdot T_4/\ell+(Y-u_4)\cdot Z_{\mathbb{L}}(Y)\cdot\sum\nolimits_{j\in[4]} Y^{(j-1)\cdot\ell}\cdot h_{4,j}(Y),\end{aligned}$$

      where $T_4 = \delta\cdot g_3(\delta)/(\delta-u_3) + f_p(\alpha,\beta)/\tilde{m} + Z_{\mathbb{M}}(\delta)\cdot\sum_{j\in[3]}\delta^{(j-1)\tilde{m}}\cdot h_{3,j,i}(\delta)$.
   (e) *$\mathcal{V}$ invokes Step 5 of Protocol 4 using the received $f_p(\alpha,\beta)$ and queried $R(r^m,\beta)$ instead of computing them itself. Reject if any check does not pass:*
      i. *Pick a random $\zeta \in \mathbb{F}$. Invoke Step 9 of Protocol 5 to verify the validity of $A_{\mathsf{high}}(X,Y), A_{\mathsf{low}}(X,Y)$ and $B(X,Y)$ by the lookup PIOP.*
      ii. *Query $R(r^m,\beta)$. Query $g_3(\delta)$ and $\{h_{3,j}(\delta)\}_{j\in[3]}$. Compute T_4 itself. Query $\mathsf{val}(\delta,\zeta), A_{\mathsf{high}}(\delta,\zeta), A_{\mathsf{low}}(\delta,\zeta), B(\delta,\zeta), L(\beta,\zeta), g_4(\zeta)$ and $\{h_{4,j}(\zeta)\}_{j\in[4]}$. Check if the equation in Step 3(d)ii holds at ζ.*
      iii. *Invoke Step 5 of Protocol 4.*

such that $\tilde{m}$ does not exceed the smallest power of two that is greater than or equal to $\max\{||P_a||, ||P_b||, ||P_c||\}/\ell$. To achieve this "loosely"-even distribution, we heuristically adjust the matrix with the most non-zero entries as a priority (The number of non-zero entries in P_a, P_b, P_c can be different.) Assume this matrix is H as in Algorithm 1. This simplifies the problem and experiments show that it suffices to achieve the "loosely"-even distribution (*cf.*, Section 7.3).

We next describe how to make the non-zero entries in the prior single matrix even. Given the parameter ℓ and the matrix P_a, we sort the number of non-zero values in each column of the matrix in an ascending order to obtain the set $\mathcal{I}$. That is to say, the column labeled with $\mathcal{I}_1$ has the minimal non-zero entries, and the the column labeled with $\mathcal{I}_m\ell$ has the maximum non-zero entries. We then assign $P_a[\mathcal{I}_1]$ to $\mathcal{P}_1$, $P_a[\mathcal{I}_2]$ to $\mathcal{P}_2$, ..., and $P_a[\mathcal{I}_\ell]$ to $\mathcal{P}_\ell$. Then, we consecutively and reversly assign $P_a[\mathcal{I}_{\ell+1}]$ to $\mathcal{P}_\ell$, $P_a[\mathcal{I}_{\ell+2}]$ to $\mathcal{P}_{\ell-1}$, ..., and $P_a[\mathcal{I}_{2\ell}]$ to $\mathcal{P}_1$. Repeat the above procedures until each sub-prover holds m columns.

While the distribution algorithm above works effectively for most columns, it encounters a problem with the first columns of P_a, P_b, and P_c. These columns contain constraints for the prescribed and public witness one. The issue is that these columns are often extremely dense, containing a large number of non-zero entries. Assigning them entirely to the first sub-prover, $\mathcal{P}_1$, would create a significant imbalance, violating our "even" goal.

We solve this problem by splitting the constraints in the first columns across all ℓ sub-provers. First, a partial component of the one witness is added to the witness vector of every sub-prover. Concurrently, an additional column is inserted at the beginning of the R1CS matrices for all sub-provers, creating a dedicated location for the new constraint. Second, the non-zero entries in original first column is partitioned into ℓ subsets, and assigned to each sub-prover. Finally, for the i-th subset of non-zero entries, the corresponding constraints are assigned exclusively to the i-th sub-prover $\mathcal{P}_i$. In the new first column of all other sub-provers, the matrix entries for these specific constraints are set to zero, ensuring they are only active in $\mathcal{P}_i$'s matrices. As a result, the non-zero entries in the first columns of the three matrices are also made "even". Note that the above splitting method only depends on the location of witness instead of the values. Hence, it also works when there exists other extremely dense columns.

In Algorithm 1, it suffices to work over size-$O(m\ell)$ vectors where each entry describes the non-zero entry number of matrices. Hence, the algorithm costs a time complexity of at most $O(m\ell \log m\ell)$ for sorting. Despite this, it is concretely efficient in practice, *e.g.*, 0.5 s when $\tilde{m} = 2^{20}$, which is much smaller than the SNARK prover time.

E Supplementary Details of Section 5

E.1 Preliminaries of Pairing-based PCSs

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 $g_i \in \mathbb{G}_i$, where $i \in [2]$. We next give the batch-PCS definition extended from univariate polynomials as in [7] to bivariate ones.

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

Input: Public R1CS matrices $P_a, P_b, P_c \in \mathbb{F}^{m\ell \times m\ell}$. Number of sub-provers ℓ.
Output: The "evened" R1CS sub-matrices: $P_a^{(i)}, P_b^{(i)}, P_c^{(i)} \in \mathbb{F}^{m\ell \times (m+1)}$ for $i \in [\ell]$.

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 $w_1^{(i)} = \mathsf{one}$ 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 $i \in \mathcal{I}$ in an ascending order. That is to say, $\mathcal{I}_{m\ell}$ has the maximum non-zero entries.
6: Let $j \leftarrow 1$.
7: **for** $i = m\ell$ **downto** 1 **do**
8: $\quad k \leftarrow \mathcal{I}_i$
9: $\quad$ **if** $\lfloor \frac{m\ell - i}{\ell} \rfloor \mod 2 = 0$ **then**
10: $\quad\quad$ Give $P_a[k], P_b[k], P_c[k]$ and $\boldsymbol{w}_k$ to the sub-prover $\mathcal{P}_j$
11: $\quad\quad$ $j \leftarrow (j \mod \ell) + 1$
12: $\quad$ **else** $\lfloor \frac{m\ell - i}{\ell} \rfloor \mod 2 = 1$
13: $\quad\quad$ Give $P_a[k], P_b[k], P_c[k]$ and $\boldsymbol{w}_k$ to the sub-prover $\mathcal{P}_{\ell - j + 1}$
14: $\quad\quad$ $j \leftarrow (j \mod \ell) + 1$
15: $\quad$ **end if**
16: **end for**

Definition 3. *A batch bivariate PCS is a triplet* $(\mathsf{Gen}, \mathsf{Com}, \mathsf{Open})$ *such that*

- $\mathsf{Gen}(m, \ell)$ - *Given positive integers* m *and* ℓ*, output a structured reference string (SRS)* srs *of size* $O(m\ell)$*.*
- $\mathsf{Com}(f, \mathsf{srs})$ - *Given a polynomial* $f \in \mathbb{F}_{m,\ell}[X, Y]$ *and an* $\mathsf{srs} \leftarrow \mathsf{Gen}(m, \ell)$*, output a commitment* cm *to* $f(X, Y)$*.*
- 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* $t = \mathsf{poly}(\lambda)$*.*
  2. *Subsets* $A_1, \ldots, A_n$ *and* $B_1, \ldots B_n$*. For simplicity, assume* $|A_k| = |B_k| = t$ *for* $k \in [n]$*. Let* $A = \bigcup_{k\in[n]} A_k$ *and* $B = \bigcup_{k\in[n]} B_k$*.*
  3. $\mathsf{cm}_{f_1}, \ldots, \mathsf{cm}_{f_n}$*, which are the commitments to* $f_1(X, Y), \ldots, f_n(X, Y)$*.*
  4. *Claimed evaluations of* $\{f_k(\alpha, \beta)\}_{k\in[n]}$ *for* $(\alpha, \beta) \in A_k \times B_k$*.*
  5. $\{r_k \in \mathbb{F}_{t,t}[X, Y]\}_{k\in[n]}$*, which describe all the correct openings, i.e.,* $r_k(\alpha, \beta) = f_k(\alpha, \beta)$ *for each* $k \in [n]$*,* $\alpha \in A_k$*, and* $\beta \in B_k$*.*

  This interactive argument satisfies:

- ***Completeness:*** *Fix any* n, t*,* A*,* B*,* $f_1, \ldots, f_n \in \mathbb{F}_{m,\ell}[X, Y]$ *and* $r_1, \ldots, r_n \in \mathbb{F}_{t,,t}[X, Y]$*. Suppose for each* $k \in [n]$*,* $\mathsf{cm}_{f_k} = \mathsf{Com}(f_k(X, Y), \mathsf{srs})$*. Let* $f_k(X, Y) - r_k(X, Y) = P_k(X, Y) + Q_k(X, Y)$*. Suppose* $Z_{A_k}(X) \mid P_k(X, Y)$ *and* $Z_{B_k}(Y) \mid Q_k(X, Y)$*. 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* $m, \ell = \mathsf{poly}(\lambda)$*, the probability of* $\mathcal{A}$ *winning the following game is* $\mathsf{negl}(\lambda)$ *over randomness of* $\mathcal{A}$*,* $\mathcal{V}$*, and* Gen*.*
  1. *Given* $\mathsf{srs} = \mathsf{Gen}(m, \ell)$*,* $\mathcal{A}$ *outputs* $\mathsf{cm}_{f_1}, \ldots, \mathsf{cm}_{f_n} \in \mathbb{G}_1$*.*
  2. $\mathcal{E}$*, given access to the messages of* $\mathcal{A}$ *during the previous steps, outputs* $f_1, \ldots, f_n \in \mathbb{F}_{m,\ell}[X, Y]$*.*
  3. $\mathcal{A}$ *outputs* $A_1, \ldots, A_n$*,* $B_1, \ldots, B_n$*, and* $\{f_k(\alpha, \beta)\}_{k\in[n]}$ *for* $(\alpha, \beta) \in A_k \times B_k$*.*
  4. $\mathcal{A}$ *takes the part of* $\mathcal{P}$ *in* Open *with inputs* $\mathsf{cm}_{f_1}, \ldots, \mathsf{cm}_{f_n}$*,* $A_1, \ldots, A_n$*,* $B_1, \ldots, B_n$*, and* $\{f_k(\alpha, \beta)\}_{k\in[n]}$ *for* $(\alpha, \beta) \in A_k \times B_k$*.*
  5. $\mathcal{A}$ *wins if* $\mathcal{V}$ *outputs accept and, for some* $k \in [n]$*, there exist polynomials* $P_k(X, Y)$ *and* $Q_k(X, Y)$ *such that* $Z_{A_k}(X) \nmid P_k(X, Y)$ *or* $Z_{B_k}(Y) \nmid Q_k(X, Y)$*, or* $f_k(X, Y) \neq P_k(X, Y) + Q_k(X, Y)$*.*

E.2 Construction of Batch Bivariate KZG

We present the formal batch bivariate KZG PCS in Protocol 8. Consider n bivariate polynomials $\{f_k(X, Y)\}_{k\in[n]}$, where the claimed evaluations of $f_k(X, Y)$ are $\{f_k(\alpha, \beta)\}_{(\alpha,\beta)\in A_k\times B_k}$. For simplicity, assume for each $k \in [n]$, $|A_k| = |B_k| = t$, hence each polynomial is opened at t^2 points over $A_k \times B_k$. Denote $A = \bigcup_{k\in[n]} A_k$ and $B = \bigcup_{k\in[n]} B_k$. Below we describe the main idea.

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

Second (in Step 3), for $k \in [n]$, $\mathcal{P}$ splits $f_k(X,Y)$ as $P_k(X,Y) = f_k(X,Y) - s_k(X,Y)$ and $Q_k(X,Y) = s_k(X,Y) - r_k(X,Y)$ and handles these two polynomials separately like Equation (21). Given a random challenge γ, for the first half, $\mathcal{P}$ multiplies each $P_k(X,Y)$ with an auxiliary polynomial $Z_{A\setminus A_k}(X)$ to allow a random linear combination of all $P_k(X,Y)$. The combined polynomial is $P(X,Y) = \sum_{k\in[n]} \gamma^{k-1} \cdot Z_{A\setminus A_k}(X) \cdot P_k(X,Y)$. Similarly, $\mathcal{P}$ computes another combined polynomial $Q(X,Y) = \sum_{k\in[n]} \gamma^{k-1} \cdot Z_{B\setminus B_k}(Y) \cdot Q_k(X,Y)$. By definition, $Z_{A_k}(X) \mid P_k(X,Y)$, and hence $Z_A(X) \mid P(X,Y)$. Similarly, we have $Z_B(Y) \mid Q(X,Y)$. $\mathcal{P}$ sends to $\mathcal{V}$ the commitments to $p(X,Y) = P(X,Y)/Z_A(X)$ (*i.e.*, U_1) and $q(X,Y) = Q(X,Y)/Z_B(Y)$ (*i.e.*, U_2).

Third (in Step 5), $\mathcal{P}$ proves to $\mathcal{V}$ the validity of $P(X,Y)$ and $Q(X,Y)$ relying on random challenges z_1 and z_2. $\mathcal{P}$ builds a polynomial $P'(X,Y)$ similar to $P(X,Y)$ but modifies $Z_{A\setminus A_k}(X)$ as $Z_{A\setminus A_k}(z_1)$ such that $P'(X,Y) = \sum_{k\in[n]} \gamma^{k-1} \cdot Z_{A\setminus A_k}(z_1) \cdot P_k(X,Y)$. Let $W_1(X,Y) = P'(X,Y) - Z_A(z_1) \cdot p(X,Y)$. As $W_1(z_1,Y) = 0$, it holds that $(X - z_1) \mid W_1(X,Y)$. Similarly, $\mathcal{P}$ can build $Q'(X,Y)$ and $W_2(X,Y)$. Then, it holds that $(Y - z_2) \mid W_2(X,Y)$. $\mathcal{P}$ sends commitments of $W_1(X,Y)/(X - z_1)$ and $W_2(X,Y)/(Y - z_2)$ to $\mathcal{V}$.

Finally (in Step 6), $\mathcal{V}$ constructs the commitments to $W_1(X,Y)$ and $W_2(X,Y)$, and checks if $W_1(X,Y)/(X - z_1) \cdot (X - z_1) = W_1(X,Y)$ and $W_2(X,Y)/(Y - z_2) \cdot (Y - z_1) = W_2(X,Y)$ over commitments via bilinear pairings.

Proof (Proof of Theorem 6). Completeness holds by design. We argue the knowledge soundness in AGM, *i.e.*, the existence of an efficient extractor $\mathcal{E}$ such that an algebraic adversary $\mathcal{A}$ can only win the game with a probability of $\mathsf{negl}(\lambda)$. We kindly refer to [7] for background about ideal and real pairing checks.

We will use the following lemma for soundness proof.

Lemma 5 ([7]). *Fix a subset $S \subset T \subset \mathbb{F}$, and a polynomial $g(X)$. 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 as described in Definition 3. $\mathcal{A}$ begins by outputting commitments $\mathsf{cm}_{f_1}, \ldots, \mathsf{cm}_{f_n} \in \mathbb{G}_1$. In the AGM, each cm_{f_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 $f_k(X,Y) = \sum_{i=0}^{m-1} \sum_{j=0}^{\ell-1} f_{k,i,j} X^i Y^j$. $\mathcal{A}$ also outputs $A_1, \ldots, A_n$, $B_1, \ldots B_n$, and $\{f_k(\alpha,\beta)\}_{\alpha\in A_k, \beta\in B_k}$, which determine the public polynomials $r_1(X,Y), \ldots, r_n(X,Y)$.

Assume for some $k^* \in [n]$, we have $Z_{A_{k^*}}(X) \nmid (f_{k^*}(X,Y) - s_{k^*}(X,Y))$ or $Z_{B_{k^*}}(Y) \nmid (s_{k^*}(X,Y) - r_{k^*}(X,Y))$. The assumption is feasible as otherwise we have for all $k \in [n]$, there exist $p_k(X,Y)$ and $q_k(X,Y)$ s.t. $f_k(X,Y) - r_k(X,Y) = p_k(X,Y) \cdot Z_{A_k}(X) + q_k(X,Y) \cdot Z_{B_k}(Y)$. Then, there exist $P_k(X,Y) = p_k(X,Y) \cdot Z_{A_k}(X)$ and $Q_k(X,Y) = q_k(X,Y) \cdot Z_{B_k}(Y)$ such that $Z_{A_k}(X) \mid P_k(X,Y)$, $Z_{B_k}(Y) \mid Q_k(X,Y)$, and $f_k(X,Y) = P_k(X,Y) + Q_k(X,Y)$. This violates the winning requirements for $\mathcal{A}$ in Definition 3, which means that all the evaluations $\{f_k(\alpha,\beta)\}_{(\alpha,\beta)\in A_k\times B_k}$ are valid.

WLOG, assume $Z_{A_{k^*}}(X) \nmid (f_{k^*}(X,Y) - s_{k^*}(X,Y))$ and $Z_{B_{k^*}}(Y) \mid (s_{k^*}(X,Y) - r_{k^*}(X,Y))$. Other cases are similar and we omit them. Then, we know from

Protocol 8 (Batch bivariate KZG).

1. $\mathsf{Gen}(m,\ell)$: *for* $\sigma,\tau \in \mathbb{F}$ *probably generated by a ceremony [24], 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. $\mathsf{Com}(\mathsf{srs}, \{f_k(X,Y)\})$: *For all* $k \in [n]$, *given* $f_k \in \mathbb{F}_{m,\ell}[X,Y]$, *the prover computes the commitment* $\mathsf{cm}_{f_k} = [f_k(\sigma,\tau)]_1$ *to* $f_k(X,Y)$.

$\underline{\mathsf{Open}(\{\mathsf{cm}_k, A_k, B_k, f_k(\alpha,\beta)_{\alpha\in A_k,\beta\in B_k}\}_{k\in[n]})}$:

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

$$W_1 = \sum_{k\in[n]} \left(\gamma^{k-1} \cdot Z_{A\setminus A_k}(z_1) \cdot (f_k(X,Y) - s_k(X,Y))\right) - Z_A(z_1) \cdot p(X,Y),$$

$$W_2 = \sum_{k\in[n]} \left(\gamma^{k-1} \cdot Z_{B\setminus B_k}(z_2) \cdot (s_k(X,Y) - r_k(z_1,z_2))\right) - Z_B(z_2) \cdot q(X,Y).$$

   By definition, we have $(X - z_1) \mid W_1(X,Y)$ *and* $(Y - z_2) \mid W_2(X,Y)$.
   (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* $V_1 = [w_1(\sigma,\tau)]_1$ *and* $V_2 = [w_2(\sigma,\tau)]_1$.
6. $\mathcal{V}$: *computes* $r_k(X,Y)$ *for* $k \in [n]$. *Compute* F_1 *and* F_2 *s.t.*

$$F_1 = \sum_{k\in[n]} \gamma^{k-1} \cdot Z_{A\setminus A_k}(z_1) \cdot (\mathsf{cm}_{f_k} - \mathsf{cm}_{s_k}) - Z_A(z_1) \cdot U_1,$$

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

which are indeed commitments to $W_1(X,Y)$ *and* $W_2(X,Y)$. *Accept iff*

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

Lemma 5 that $Z_{A\setminus A_{k^*}}(X)\cdot(f_{k^*}(X,Y)-s_{k^*}(X,Y))$ is not divisible by $Z_A(X)$. Let

$$f(X,Y)=\sum_{k\in[n]\setminus k^*}\gamma^{k-1}\cdot Z_{A\setminus A_k}(X)\cdot(f_k(X,Y)-s_k(X,Y))+$$
$$\gamma^{k^*-1}\cdot Z_{A\setminus A_{k^*}}(X)\cdot(f_{k^*}(X,Y)-s_{k^*}(X,Y)).$$

Using Lemma 4, we know that except with probability $n/|\mathbb{F}|$ over γ, $f(X,Y)$ is not divisible by $Z_A(X)$. Assume we are not in this case. 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 challenges $z_1,z_2\in\mathbb{F}$. Since $f(X,Y)$ is not divisible by $Z_A(X)$, for any $p(X,Y)$, there are at most $2m$ values of $z_1\in\mathbb{F}$ s.t. $f(z_1,Y)=p(z_1,Y)\cdot Z_A(z_1)$; thus z_1 of this form chosen by $\mathcal{V}$ is with probability of $\mathsf{negl}(\lambda)$ when the finite field is sufficiently large.

Assume z_1 is not in this error form. $\mathcal{P}$ now outputs $w_1=[W_1(\sigma,\tau)]_1$ and w_2. According to [7, Lemma 2.2], it suffices to restrict the upper bound of the probability that the ideal check passes corresponding to the probability that the real pairing check passes in Step 6. Denote $W_1^*(X,Y)=$

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

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 $W_1^*(z_1,Y)=f(z_1,Y)-p(z_1,Y)\cdot Z_A(z_1)=0$. This a contradiction.

The proof for $Z_{B_{k^*}}(Y)\nmid(s_{k^*}(X,Y)-r_{k^*}(X,Y))$ is similar and we omit it here. In summary, the ideal check can only pass with $\mathsf{negl}(\lambda)$ probability over the randomness of $\mathcal{V}$, which implies the same thing for the real pairing check.

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

Opening complexity. We argue that given the polynomial $f_k(X,Y)$ for each $k\in[n]$, the computation of $s_k(X,Y)$ costs at most $O(t\cdot m\ell+t^2\ell)$ field operations. We explain this as below. WLOG, we can express $f_k(X,Y)$ as $\sum_{j\in[\ell]}f_j(X)\cdot Y^{j-1}$, and express $s_k(X,Y)$ as $\sum_{j\in[\ell]}s_j(X)\cdot Y^{j-1}$. Recall that $s_k(X,Y)$ satisfies $s_k(\alpha,Y)=f_k(\alpha,Y)$ for all $\alpha\in A_k$, and thus its X-degree is less than t. Hence, we have for any $j\in[\ell]$, the degree of $s_j(X)$ is less than t; and further, for all $\alpha\in A_k$, $s_j(\alpha)=f_j(\alpha)$. Thus, to compute $s_k(X,Y)$, it suffices to first compute $f_j(\alpha)$ for all $j\in[\ell]$ and $\alpha\in A_k$, and then interpolate $s_j(X)$ for $j\in[\ell]$. The computation of $f_j(\alpha)$ costs $O(tm\ell)$, and the interpolation costs $O(\ell\cdot t^2)$ (without FFTs). Hence, the computation of $s_k(X,Y)$ for all $k\in[n]$ costs $O(nt\cdot m\ell+nt^2\ell)$ field operations. Given the polynomials $\{s_k(X,Y)\}_{k\in[n]}$ where each $s_k\in\mathbb{F}_{t,\ell}[X,Y]$, it costs $O(n\cdot t\ell)$ group operations to generate all the commitments $[s_k(\sigma,\tau)]_1$.

We next analyze the complexity of computing the public polynomial $r_k(X,Y)$. Given the polynomial $s_k(X,Y)=\sum_{j\in[\ell]}s_j(X)\cdot Y^{j-1}$, it takes at most $O(t\ell)$ to rewrite it as $s_k(X,Y)=\sum_{i\in[t]}s_i(Y)\cdot X^{i-1}$, where each $s_i\in\mathbb{F}_\ell[Y]$. WLOG, we

can also express $r_k(X,Y)$ as $\sum_{i\in[t]} r_i(Y)\cdot X^{i-1}$, where each $r_i \in \mathbb{F}_t[Y]$. Recall that $r_k(X,Y)$ satisfies $r_k(X,\beta) = s_k(X,\beta)$ for all $\beta \in B_k$, and hence for any i and all $\beta \in B_k$, $s_i(\beta) = r_i(\beta)$. Thus, to compute $r_k(X,Y)$, it suffices to first compute $s_i(\beta)$ for all $i \in [t]$ and $\beta \in B_k$, and then interpolate $r_i(Y)$ for $i \in [t]$. The former costs $O(t^2\ell)$, and the latter costs $O(t\cdot t^2)$. So, the computation of $r_k(X,Y)$ for all $k \in [n]$ costs $O(nt^2\ell + nt^3)$ field 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 $P(X,Y)$ is a random linear combination of n size-(m,ℓ) polynomials, and $Q(X,Y)$ is a random linear combination of n size-(t,ℓ) 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 opening complexity is $O(ntm\ell + nt^2\ell + nt^3)$ field operations plus $O(m\ell + nt\ell)$ group operations. In contrast, the trivial prover complexity needs $O(t^2n\cdot m\ell)$ group operations. We also note that n and t are usually small constants in SNARKs [13,14] and also in our PIOP.

Proof size. The proof size includes: 1) sending commitments of all $s_k(X,Y)$, which is n group elements; 2) sending U_1, U_2, V_1, and V_2, which is 4 group elements. Hence, the total proof size is $n+4$ group elements. In contrast, the trivial proof size is $2t^2n$ group elements.

Verifier complexity. The verifier complexity includes: 1) computing the public polynomials $\{r_k(X,Y)\}_{k\in[n]}$, which is $O(nt^2\ell + nt^3)$ field operations; 2) $O(n)$ exponentiations due to computing all $[r_k(z_1,z_2)]_1$, $O(n)$ exponentiations for computing F_1 and F_2, and 4 pairings for the validity check. Hence, the total verifier complexity is $O(nt^2\ell + nt^3)$ field operations, $O(n)$ exponentiations, and 4 pairings. In contrast, the trivial verifier complexity is $2t^2n$ pairings. □

E.3 Batch Bivariate KZG over Non-Cartesian-Product Points

Our batch bivariate KZG in Section E.2 assumes that each polynomial is evaluated at t^2 diverse points over a Cartesian product of two sets. We generalize it to a more common setting to handle n polynomials $\{f_k(X,Y)\}_{k\in[n]}$, where the evaluation points of each polynomial $f_k(X,Y)$ are $\{(\alpha_{j,k},\beta_{j,k})\}_{j\in[t]}$ for $\alpha_{j,k} \in A_k$ and $\beta_{j,k} \in B_k$. Here we assume that the Y-dimension points are different from each other, *i.e.*, for all $i,j \in [t]$, $\alpha_{i,k} \neq \alpha_{j,k}$. Extending this to a more general case (arbitrary $\alpha_{j,k}$ and $\beta_{j,k}$ for any j and k) is left for a further work.

We show how to reduce the quotient relation of non-Cartesian-product cases into a relation of Cartesian-product cases as in Equation (20). For simplicity, we first handle a single polynomial $f(X,Y)$ evaluated at the point set $\{(\alpha_j,\beta_j)\}_{j\in[\ell]}$.

The challenge is that with these conditions, we cannot directly write the relation using two terms with *split* quotient polynomials over the dimensions X and Y. For example, given t claimed evaluations $\{f(\alpha_j,\beta_j)\}_{j\in[t]}$ and a public size-(t,t) polynomial $r'(X,Y)$ corresponding to these evaluations, there seems a unique method to write the relation, *i.e.*, there exist polynomials $\{q_j(X,Y)\}_{j\in[t]}$

such that $f(X,Y) - r'(X,Y) = \sum_{j\in[t]} q_j(X,Y)\cdot(X-\alpha_j)\cdot(Y-\beta_j)$. Hence, the proof size would be linear to the number of opened points, as the prover would send commitments to $\{q_j(X,Y)\}_{j\in[t]}$ to the verifier.

A direct approach is to let the prover extend the public evaluation points into a Cartesian product, send the extended points and evaluations to the verifier, and prove their validity via Equation (20). However, this makes the verifier know additional evaluations, which increases the computation overhead and may leak knowledge when considering zero knowledge.

To address this, we instead see the originally public polynomial $r(X,Y)$ from Cartesian product as a secret polynomial. The prover computes and commits it, while the verifier checks its validity without computing it. Specifically, define the sets $A = \{\alpha_j\}_{j\in[t]}$ and $B = \{\beta_j\}_{j\in[t]}$, define a public size-$t$ polynomial $v(X)$ interpolated from $\{(\alpha_j, f(\alpha_j,\beta_j))\}_{j\in[t]}$, and define a public size-t polynomial $u(X)$ interpolated from $\{(\alpha_j,\beta_j)\}_{j\in[t]}$. Then, the committed $r(X,Y)$ is valid iff $v(\alpha) = r(\alpha,\beta)$ for $\alpha\in A$ and $\beta\in B$. We find that to prove its validity, it suffices to prove the existence of $p(X,Y)$ and $q(X)$ s.t.

$$r(X,Y) - v(X) = (Y-u(X))\cdot p(X,Y) + q(X)\cdot Z_A(X). \tag{45}$$

This holds as when $Y = u(X)$, the equation is irrelevant to Y and we have $r(\alpha,u(\alpha)) = v(\alpha)$, so the second term in the right had can be written as $q(X)\cdot Z_A(X)$. Given a valid $r(X,Y)$, if Equation (20) holds, then $\{f(\alpha_j,\beta_j)\}_{j\in[t]}$ are correct with a high probability as each $f(\alpha_j,\beta_j) - r(\alpha_j,\beta_j)$ is zero. Now, the proof size is $O(1)$ regardless of the number of evaluation points.

We note that in Equation (45), the X-degrees of $p(X,Y)$ and $q(X)$ are at least $t-2$. That is to say, Equation (45) cannot be directly seen as an Euclidean polynomial division over $Y-u(X)$. To see this, we introduce an auxiliary univariate polynomial $R(X) = r(X,u(X))$ of degree $2t-2$. Then, Equation (45) can be split as the following two equations via the Euclidean polynomial division

$$\begin{aligned} r(X,Y) - R(X) &= (Y-u(X))\cdot p(X,Y),\\ R(X) - v(X) &= q(X)\cdot Z_A(X). \end{aligned} \tag{46}$$

The first equation has a quotient polynomial $p(X,Y)$ as when $Y = u(X)$, it always equals to zero. The second equation holds as it is only related to X and equals to zero for all $\alpha\in A$. As the degree of $R(X)$ is $2t-2$, the X-degree of $p(X,Y)$ is $t-1$, and the degree of $q(X)$ is $t-2$.

Generalization to multiple polynomials. We next handle the batch case for n polynomials $\{f_k(X,Y)\}_{k\in[n]}$, where each polynomial is evaluated at a point set $\{(\alpha_{k,j},\beta_{k,j})\}_{j\in[t]}$ for $\alpha_{k,j}\in A_k$ and $\beta_{k,j}\in B_k$.

For each $k\in[n]$, define the polynomials $r_k(X,Y), u_k(X)$ and $v_k(X)$ corresponding to $f_k(X,Y)$. Define the set $A = \bigcup_{k\in[n]} A_k$. To enable efficient batching, for each $k\in[n]$ we let the prover additionally commit the auxiliary polynomial $R_k(X) = r_k(X,u(X))$ to split the two quotient polynomials $p(X,Y)$ and $q(X)$. Then, given a random challenge γ, the prover can combine multiple quotient

relations as in Equation (46) into a single one

$$\sum_{k\in[n]} \gamma^{k-1}\cdot\big(r_k(X,Y)-R_k(X)\big)\cdot\prod_{i\in[n]\setminus k}(Y-u_i(X))=\prod_{i\in[n]}(Y-u_i(X))\cdot p(X,Y),$$
$$\sum_{k\in[n]} \gamma^{k-1}\cdot\big(R_k(X)-v_k(X)\big)\cdot Z_{A\setminus A_k}(X)=q(X)\cdot Z_A(X). \tag{47}$$

If Equation (47) holds, then by Lemma 4, with a high probability that $(Y-u_k(X)) \mid \big(r_k(X,Y)-R_k(X)\big)$ and $Z_A(X)\mid\big(R_k(X)-v_k(X)\big)$ both hold for each $k\in[n]$. This further means $r_k(X,Y)$ is valid with a high probability.

With these modifications, we can reduce the non-Cartesian-product case into a Cartesian-product case. The proof size would increase from $n+4$ group elements into $3n+6$ group elements for additionally sending commitments to $\{r_k(X,Y),R_k(X)\}_{k\in[n]}, p(X,Y)$ and $q(X)$.

E.4 Batch Bivariate KZG with Shared Points over the Y-dimension

Protocol 9 gives the variant of batch bivariate KZG when the evaluation points over Y-dimension are all the same.

Theorem 9. *Protocol 9 is a batch scheme for bivariate PCS. Committing n bivariate polynomials belonging to $\mathbb{F}_{m,\ell}[X,Y]$ costs $O(nm\ell)\,\mathbb{G}$. When opening t points (with the same Y-dimension points) on each of the n bivariate polynomials, the opening complexity is $O(m\ell)\,\mathbb{G}+O(nm\ell+nt)\,\mathbb{F}$. The proof size is $4\,|\mathbb{G}|+(n+1)\,|\mathbb{F}|$. The verifier complexity is $O(n)\,\mathbb{G}+O(nt)\,\mathbb{F}+5\,P$.*

Proof. Completeness holds by design. We argue the knowledge soundness in the AGM below. Let $\mathcal{A}$ be such an algebraic adversary. $\mathcal{A}$ begins by outputting $\mathsf{cm}_{f_1},\ldots,\mathsf{cm}_{f_n}$, and each cm_{f_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 $R_1,\ldots,R_n,\beta,A$.

By the protocol, $\mathcal{A}$ wins if $\mathcal{V}$ accepts and for some $k^*\in[n]$ and $j\in[t]$, the claimed $f_{k^*}(\alpha_j,\beta)$ is invalid. This means $q_{k^*}(X)$ does not exist, *i.e.*, $Z_{A_{k^*}}(X)\nmid f_{k^*}(X,\beta)-r_{k^*}(X)$. Further, we know from Lemma 5 that $Z_{A\setminus A_{k^*}}(X)\cdot(f_{k^*}(X,Y)-r_{k^*}(X,Y))$ is not divisible by $Z_A(X)$. Let

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

By Lemma 4, except with probability $n/|\mathbb{F}|$ over γ, $f(X)$ is not divisible by $Z_A(X)$. Now $\mathcal{A}$ outputs $Q=[q'(\sigma)]$ for some $q'\in\mathbb{F}_m[X]$. Given a random $\eta\in\mathbb{F}$, for any $q'(X)$ there are at most $2m$ values of $\eta\in\mathbb{F}$ s.t. $f(\eta)=q(\eta)\cdot Z_A(\eta)$. Thus, picking a η of this form for $\mathcal{V}$ is of probability $\mathsf{negl}(\lambda)$.

Assume we are not in this case. Then $\mathcal{A}$ must send a malicious $f_{k^*}(\eta,\beta)$ for some $k^*\in[n]$. By the randomness of θ, with a probability of $n/|\mathbb{F}|$, $\sum_{k\in[n]\setminus k^*}\theta^{k-1}$.

Protocol 9 (Batch bivariate KZG evaluated at a single $Y = \beta$).

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

$\underline{\mathsf{Open}(\{\mathsf{cm}_k\}_{k\in[n]}, \{A_k\}_{k\in[n]}, \beta; \{f_k(\alpha, \beta)_{\alpha\in A_k}\}_{k\in[n]}):}$

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* $\mathcal{V}$*, where* $q \in \mathbb{F}_{m-t}[X]$ *satisfies (WLOG, we assume* $m > t$*)*

$$\sum\nolimits_{k\in[n]} \gamma^{k-1} \cdot (f_k(X, \beta) - r_k(X)) \cdot Z_{A\setminus A_k}(X) = q(X) \cdot Z_A(X). \tag{48}$$

3. $\mathcal{V}$*: sends to* $\mathcal{P}$ *a random challenge* $\eta \in \mathbb{F}$.

4.(a) $\mathcal{P}$*: computes and 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* $Q_1 = [q_1(\sigma, \tau)]_1, Q_2 = [q_2(\sigma, \tau)]_1$ *and* $Q_3 = [q_3(\sigma)]_1$*, where*

$$\sum_{k\in[n]} \theta^{k-1} \cdot (f_k(X, Y) - f_k(\eta, \beta)) = q_1(X, Y) \cdot (X - \eta) + q_2(X, Y) \cdot (Y - \beta),$$
$$q(X) - q(\eta) = q_3(X) \cdot (X - \eta).$$

(d) $\mathcal{V}$*: computes univariate polynomial* $r_k \in \mathbb{F}_t[X]$ *for each* $k \in [n]$ *such that* $r_k(\alpha) = f_k(\alpha, \beta)$ *for all* $\alpha \in A_k$*. Accept iff:*

i. *Equation (48) holds when* $X = \eta$*;*

ii. $e(Q_3, [\sigma - \eta]_2) = e(Q - [q(\eta)]_1, [1]_2)$*;*

iii. $e(Q_1, [\sigma - \eta]_2) \cdot e(Q_2, [\tau - \beta]_2) = e(\sum_{k\in[n]} \theta^{i-1} \cdot (\mathsf{cm}_k - [f_k(\eta, \beta)]_1), [1]_2)$.

$(f_k(\eta, \beta) - f_k(\eta, \beta)) + \theta^{k^*-1} \cdot (f_k(\eta, \beta) - f_{k^*}(\eta, \beta)) \neq 0$. Denote $f^*(X, Y) = \sum_{k\in[n]\setminus k^*} \theta^{k-1} \cdot (f_k(X, Y) - f_k(\eta, \beta)) + \theta^{k^*-1} \cdot (f_k(X, Y) - f_{k^*}(\eta, \beta))$. The ideal check has the form $f^*(X, Y) = q_g(X, Y) \cdot (X - \eta) + q_h(X, Y) \cdot (Y - \beta)$ for the existence of $q_g(X, Y)$ and $q_h(X, Y)$. It can pass iff $f^*(\eta, \beta) \neq 0$, which further means that $\sum_{k\in[n]\setminus k^*} \theta^{k-1} \cdot (f_k(\eta, \beta) - f_k(\eta, \beta)) + \theta^{k^*-1} \cdot (f_k(\eta, \beta) - f_{k^*}(\eta, \beta)) = 0$. This is a contradiction. Thus, the ideal check can only pass with probability $\mathsf{negl}(\lambda)$, which implies the same thing for the real check.

Complexity. The srs and commitment generation are the same as Protocol 8.

Opening complexity. By definition, computing each $r_k(X)$ costs $O(t)$ field operations, and $O(nt)$ field operations in total for all $k \in [n]$. Evaluating $f_k(X, Y)$ at $Y = \beta$ for all $k \in [n]$ costs $O(n\ell)$ field operations. Next, computing $q(X)$ costs at most $O(nm)$ field operations as $f_k(X, \beta)$ is a univariate polynomial with degree less than m. Computing $q_1(X, Y)$ and $q_2(X, Y)$ costs an overhead of at most $O(nm\ell)$ field operations. Finally, computing Q, Q_1, Q_2 and Q_3 costs at most $O(m\ell)$ group operations due to the degree bounds of $q(X), q_1(X, Y), q_2(X, Y)$ and $q_3(X)$. Hence, the total opening complexity is $O(m\ell)\,\mathbb{G} + O(nm\ell + nt)\,\mathbb{F}$.

Proof size and verification. The proof size includes: 1) sending $(n+1)$ polynomial evaluations of $f_1(\eta, \beta), \ldots, f_n(\eta, \beta)$ and $q(\eta)$, which costs $(n + 1)$ field elements;

2) sending Q, Q_1, Q_2 and Q_3, which is 4 group elements. Thus, the total proof size is $4\,|\mathbb{G}| + (n+1)\,|\mathbb{F}|$, in contrast to $(n+4)\,|\mathbb{G}|$ in Protocol 8.

The verifier's computation includes: 1) computing $r_k(X)$, which costs $O(t)$ for each and totally $O(nt)$ field operations; 2) checking Equation (48), which costs $O(n)$ field operations; 3) checking the pairing-based equations, which costs $O(n)$ group exponentiations and 5 pairings. Hence, the total verifier complexity is $O(n)\,\mathbb{G} + O(nt)\,\mathbb{F} + 5\,P$. □

E.5 Construction of Distributed Batch Bivariate KZG

We present the distributed batch bivariate KZG in Protocol 10. To adapt to our distributed PIOP for R1CS, we modify the $\{[\tau^{j-1}]_1\}_{j\in[\ell]}$ in srs as $\{[L_j(\tau)]_1\}_{j\in[\ell]}$.

Proof. The security of Protocol 10 follows directly from Protocol 8 assuming each sub-prover is honest. Looking ahead, we would discuss the case where some sub-provers are malicious in Section F.

Complexity. The total SRS size is $O(m\ell)$, while each sub-prover only needs to store an SRS of size $O(m)$. The proof size is $n+4$ group elements, the same as the non-distributed Protocol 8. The amortized communication overhead for $\mathcal{P}_i$ 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 $O(nt^2\ell + nt^3)$ field operations, $O(n)$ group operations, and 4 pairings, which are the same as the non-distributed Protocol 8.

Opening complexity of sub-provers. We next analyze the opening complexity of sub-provers. Assume $|A_k| = |B_k| = t$ for all $k \in [n]$. For each sub-prover $\mathcal{P}_i$ and a fixed k, the computation of $\{f_{k,i}(\alpha)\}_{\alpha\in A_k}$ costs at most $O(tm)$ field operations, and then computing $s_{k,i}(X)$ takes an $O(t^2)$ field operations as each $N_\alpha(X)$ is a size-t polynomial. For n polynomials, this costs $O(ntm+nt^2)$ field operations totally. 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 total opening complexity of each sub-prover is $O(ntm + nt^2)$ field operations plus $O(m + nt)$ group operations.

Opening complexity of the master prover. For the master prover $\mathcal{P}_0$, we argue that computing $r_k(\alpha, \beta)$ for all k from $\{s_{k,i}(\alpha)\}_{i\in[\ell],\alpha\in A_k}$ costs $O(t^2\ell + t^3)$ field operations. To see this, $\mathcal{P}_0$ can first build $s_k(X, Y)$, and then follow Protocol 8 to compute $r_k(X, Y)$ (*cf.*, the complexity analysis in Appendix E.2). By its complexity analysis, the latter term costs $O(t^2\ell + t^3)$ field operations. For the former term, interpolating $s_{k,i}(X)$ for all $i \in [\ell]$ costs an overhead of $O(t^2\ell)$ field operations, and then $\mathcal{P}_0$ can compute $s_k(X, Y)$. Thus, computing all $r_k(X, Y)$ for $k \in [n]$ takes $O(nt^2\ell + nt^3)$ field operations.

Computing $q(X, Y)$ costs $O(nt\ell)$ field operations, and computing U_2 costs $O(t\ell)$ group operations. Computing $w_2(X, Y)$ need $O(nt\ell)$ field operations, and

Protocol 10 (Distributed batch bivariate KZG). *Suppose ℓ sub-provers $\mathcal{P}_1, \ldots, \mathcal{P}_\ell$ and the master prover $\mathcal{P}_0$. Given $\{f_k\}_{k\in[n]} \in \mathbb{F}_{m,\ell}[X,Y]$ s.t. $f_k(X,Y) = \sum_{i\in[\ell]} f_{k,i}(X)L_i(Y)$, assume that $\mathcal{P}_i$ is assigned with $\{f_{k,i}(X)\}_{k\in[n]}$.*

1. $\mathsf{Gen}(m,\ell)$*: choose uniform* $\sigma,\tau \in \mathbb{F}$*. Output* $\mathsf{srs} = (\{[\sigma^{h-1}L_i(\tau)]_1\}_{h\in[m],i\in[\ell]}, [1]_2, [\sigma]_2, [\tau]_2)$*. Assume each* $\mathcal{P}_i$ *is assigned with* $(\{[\sigma^{h-1}L_i(\tau)]_1\}_{h\in[m]}$*.*
2. $\mathsf{DKZG.Com}(\{f_k(X,Y)\}_{k\in[n]}, \mathsf{srs})$*: For each* $k \in [n]$*,* $\mathcal{P}_i$ *computes and sends to* $\mathcal{P}_0$ $\mathsf{cm}_{f_{k,i}} = [f_{k,i}(\sigma)L_i(\tau)]_1$*, who then computes the commitment to* $f_k(X,Y)$*,i.e.,* $\mathsf{cm}_{f_k} = \prod_{i\in[\ell]} \mathsf{cm}_{f_{k,i}}$*.*

$\mathsf{DKZG.Open}(\{\mathsf{cm}_k, A_k, B_k, \{f_k(\alpha,\beta)\}_{(\alpha,\beta)\in A_k\times B_k}\}_{k\in[n]})$*:*

1. *Provers compute and send the commitments to* $s_k(X,Y)$*, i.e.,* $\{\mathsf{cm}_{s_k}\}_{k\in[n]}$ *to* $\mathcal{V}$*.*
   (a) $\mathcal{P}_i$*: for all* $k \in [n]$*, computes* $f_{k,i}(\alpha)$ *for all* $\alpha \in A_k$ *and computes* $s_{k,i} \in \mathbb{F}_t[X]$*. By Equation (22), we have* $s_{k,i}(X) = \sum_{j\in[t]} f_{k,i}(\alpha)\cdot N_\alpha(X)$ *for* $\alpha \in A_k$ *as for any* $\alpha \in A_k$*, we have* $s_{k,i}(\alpha) = f_{k,i}(\alpha)$*. Send* $s_{k,i}(\alpha)$ *for all* $k \in [n]$ *and* $\alpha \in A_k$ *to* $\mathcal{P}_0$*. Compute* $\mathsf{cm}_{s_{k,i}} = [s_{k,i}(\sigma)L_i(\tau)]_1$ *and send it to* $\mathcal{P}_0$*.*
   (b) $\mathcal{P}_0$*: Compute* $\mathsf{cm}_{s_k} = [s_k(\sigma,\tau)]_1 = \prod_{i\in[\ell]} \mathsf{cm}_{s_{k,i}}$*. This is feasible as by definition, we have* $s_k(X,Y) = \sum_{i\in[\ell]} s_{k,i}(X)L_i(Y)$*. For each* $k \in [n]$*, compute* $r_k \in \mathbb{F}_{t,t}[X,Y]$ *such that* $r_k(X,\beta) = s_k(X,\beta)$ *for all* $\beta \in B_k$*.*
2. $\mathcal{V}$*: sends a random challenge* $\gamma \in \mathbb{F}$ *to* $\mathcal{P}_0$*, who transfers it to all* $\mathcal{P}_i$*.*
3. *Provers compute* U_1 *and* U_2 *as defined in Step 3 of Protocol 8.*
   (a) $\mathcal{P}_i$*: computes* $p_i(X) = P_i(X)/Z_A(X)$ *where*

$$P_i(X) = \sum_{k\in[n]} \gamma^{k-1}\cdot Z_{A\setminus A_k}(X)\cdot(f_{k,i}(X) - s_{k,i}(X)).$$

   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 *locally the same as in Step 3 of Protocol 8.*
4. $\mathcal{V}$*: sends challenges* $z_1, z_2 \in \mathbb{F}$ *to* $\mathcal{P}_0$*, who transfers them to all* $\mathcal{P}_i$*.*
5. *Provers compute and send* V_1 *and* V_2 *to* $\mathcal{V}$ *as defined in Step 5 of Protocol 8.*
   (a) $\mathcal{P}_i$*: computes* $w_{1,i}(X) = W_{1,i}(X)/(X - z_1)$ *where*

$$W_{1,i}(X) = \sum_{k\in[n]} \gamma^{k-1}\cdot Z_{A\setminus A_k}(z_1)\cdot\big(f_{k,i}(X) - s_{k,i}(X)\big) - Z_A(z_1)\cdot P_i(X).$$

   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 *locally according to Step 5 of Protocol 8.*
6. $\mathcal{V}$*: acts the same as in Step 6 of Protocol 8 for verification.*

computing V_2 costs $O(t\ell)$ group operations. Hence, the master prover complexity is $O(nt^2\ell + nt^3)$ field operations plus $O(t\ell)$ group operations.

These complexities show that our distributed batch KZG exhibits a full linear speedup, as the sub-prover complexity is irrelevant to ℓ, the master prover complexity is irrelevant to m, and the proof size and verifier complexity remain with the non-distributed scheme as in Protocol 8. □

We note that Protocol 9 can be extended into a distributed scheme similar to how Protocol 10 generalizes Protocol 8. We omit the details here.

F Accountability

This section argues the accountability of Soloist, which shows its ability of an honest master prover to identify a malicious sub-prover who tries to forge a proof by sending malicious messages. On a high level, the accountability follows from the fact that, with at most a slight modification, the message each sub-prover sends to the master prover can be seen as an independent sub-proof for some public relation. The master prover can then act as a verifier to check its validity. Indeed, the final proof generated by the master prover is the aggregation of these sub-proofs relying on the additive homomorphism of commitment.

We modify the definitions of accountability in Pianist [36] as below.

Definition 4 (Distributed SNARK with accountability). *Given ℓ sub-provers $\mathcal{P}_1, \ldots, \mathcal{P}_\ell$ and a master prover $\mathcal{P}_0$, assume each sub-prover $\mathcal{P}_i$ is assigned with its public parameters pp_i, a sub-statement $\mathtt{x}_i$, its corresponding sub-witness $\mathtt{w}_i$ and a binary relation $\mathcal{R}_i$. These are generated by the whole statement $\mathtt{x}$ and witness $\mathtt{w}$ running a Split algorithm, i.e., $(\mathtt{x}_i, \mathtt{w}_i, \mathcal{R}_i) \leftarrow \mathsf{Split}(\mathtt{x}, \mathtt{w}, \mathcal{R})$. Assume that $\mathcal{P}_0$ holds its public parameter pp_0, and $\mathcal{V}$ has its public parameter $\mathsf{pp}_\mathcal{V}$. The core algorithms for related to accountability are as follows.*

- $\pi_i \leftarrow \mathsf{DisProve}_i(\mathsf{pp}_i, \mathtt{x}_i, \mathtt{w}_i)$. *Given the sub-witness $\mathtt{w}_i$, $\mathcal{P}_i$ generates a sub-proof π_i for the sub-statement $\mathtt{x}_i$.*
- $\{0,1\} \leftarrow \mathsf{Test}_i(\mathsf{pp}_0, \mathtt{x}_i, \pi_i)$. *$\mathcal{P}_0$ checks the sub-proof π_i of $\mathcal{P}_i$. If the algorithm outputs 1, $\mathcal{P}_i$ can use it as a valid contribution to the final proof. If the algorithm outputs 0, $\mathcal{P}_0$ would mark $\mathcal{P}_i$ as a malicious party.*
- $\pi \leftarrow \mathsf{Merge}(\mathsf{pp}_0, \{\pi_i\}_{i\in[\ell]})$. *$\mathcal{P}_0$ aggregates all the valid sub-proofs $\pi_1, \ldots, \pi_\ell$ and generates the final proof π.*
- $\{0,1\} \leftarrow \mathsf{Verify}(\mathsf{pp}_\mathcal{V}, \mathtt{x}, \pi)$. *$\mathcal{V}$ checks the validity of π, and outputs accept or reject.*

An accountable distributed SNARK should satisfy the following security property named as "partial knowledge soundness".

Given valid public parameters $\mathsf{pp}_0, \mathsf{pp}_1, \ldots, \mathsf{pp}_\ell$ and $\mathsf{pp}_\mathcal{V}$, valid $(\{\mathtt{x}_i, \mathtt{w}_i, \mathcal{R}_i\}_{i\in[\ell]}) \leftarrow \mathsf{Split}(\mathtt{x}, \mathtt{w}, \mathcal{R})$, for each $i \in [\ell]$ and any PPT adversary $\mathcal{P}_i^*$ with auxiliary information $\mathtt{z} \leftarrow \mathcal{P}_i^*(\mathsf{pp}_i, \mathtt{x}_i)$, there exists an expected polynomial time extractor $\mathcal{E}$ such that the following probability is $\mathsf{negl}(\lambda)$:

$$\Pr\left[\begin{array}{l|l} \pi_j \leftarrow \mathsf{DisProve}_j(\mathsf{pp}_j, \mathtt{x}_j, \mathtt{w}_j), \forall j \in [\ell] \setminus i, & (\mathtt{x}_i, \mathtt{w}_i) \notin \mathcal{R}_i, \\ \pi_i \leftarrow \mathsf{DisProve}_{\mathcal{P}_i^*}(\mathsf{pp}_i, \mathtt{x}_i, \mathtt{z}), 1 \leftarrow \mathsf{Test}_i(\mathsf{pp}_0, \mathtt{x}_i, \pi_i), & \mathtt{w}_i \leftarrow \mathcal{E}^{\mathcal{P}_i^*(\mathtt{z})}(\mathsf{pp}_i, \mathtt{x}_i) \\ \pi \leftarrow \mathsf{Merge}(\mathsf{pp}_0, \{\pi_i\}_{i\in[\ell]}), \mathsf{Verify}(\mathsf{pp}_\mathcal{V}, \mathtt{x}, \pi) = 1 & \end{array}\right].$$

This property argues that, whenever a malicious $\mathcal{P}_i^*$ generates a sub-proof π_i, if it passes the Test_i algorithm and leads to a final proof π accepted by $\mathcal{V}$, then with a high probability that the extractor could output a valid $\mathtt{w}_i$ such that $(\mathtt{x}_i, \mathtt{w}_i) \in \mathcal{R}_i$.[19]

We next show the accountability of Soloist by showing that both our distributed PIOP and distributed PCS satisfy this property.

Accountability of the distributed PIOPs. We show the accountability of our distributed PIOP with slight modifications (after instantiated with the distributed bivariate KZG and the univariate KZG) in Protocol 4, Protocol 5, and Protocol 7 seperately.

Modifications for Protocol 4. In Step 2, apart from the oracles of $g_1(X)$ and $h_1(X)$, $\mathcal{P}_i$ is additionally required to send a polynomial oracle of $f_i(X)$ to $\mathcal{P}_0$.[20] By the construction of our distributed batch KZG, we assume that the SRS held by $\mathcal{P}_i$ is $[L_i(\tau)]_1, [\sigma L_i(\tau)]_1, \ldots, [\sigma^{m-1} L_i(\tau)]_1$. Let the coefficient of $f_i(X)$ be $(f_{i,1}, \ldots, f_{i,m})$. Then, the oracle of $f_i(X)$ is concretely $[\sum_{j\in[m]} f_{i,j} \cdot \sigma^{j-1} \cdot L_i(\tau)]$ with a generator g_1. We expect this oracle to be a standard KZG commitment (*e.g.*, $[\sum_{j\in[m]} f_{i,j} \cdot \sigma^{j-1}]$) for direct polynomial proof and verification, but it has an additional term $[L_i(\tau)]$. Below we show that with slight modifications, this polynomial oracle can still be seen as a KZG commitment to $f_i(X)$ in the real world. As groups we use in practice are typically prime-order, $L_i(\tau) \cdot g_1$ is also a generator of the group, and then the SRS can be equivalently written as $[1]_{1_i}, [\sigma]_{1_i}, \ldots, [\sigma^{m-1}]_{1_i}$ with a generator $L_i(\tau) \cdot g_1$. Hence, if needed, $\mathcal{P}_i$ can still generate the KZG evaluation proof for $f_i(X)$ using this SRS, and the verifier can check its validity using the verification SRS $[1]_{2_i} = L_i(\tau) \cdot g_2$.

In Step 2, when $\mathcal{P}_i$ sends oracles $g_{1,i}(X)$ and $h_{1,i}(X)$ to $\mathcal{P}_0$, $\mathcal{P}_i$ is now required to additionally send the claimed sum $T_{1,i} = \sum_{x\in\mathbb{H}} f_i(x)$ to $\mathcal{P}_0$, who further transfers it to the verifier.[21] According to our construction, these polynomial oracles are directly KZG commitments with a normal SRS $[1]_1, [\sigma]_1 \ldots, [\sigma^{m-1}]_1$.

In Step 4a, when $\mathcal{P}_i$ sends $f_i(\alpha), g_{1,i}(\alpha)$ and $h_{1,i}(\alpha)$ to $\mathcal{P}_0$, $\mathcal{P}_i$ is now required to additionally send KZG evaluation proofs of them. Then, $\mathcal{P}_0$ acts as a verifier to check these KZG evaluation proofs, and checks if

$$(\alpha - u_1) \cdot f_i(\alpha) = \alpha \cdot g_{1,i}(\alpha) + (\alpha - u_1) \cdot T_{1,i} + (\alpha - u_1) \cdot Z_{\mathbb{H}}(\alpha) \cdot h_{1,i}(\alpha).$$

If any one does not pass, $\mathcal{P}_0$ would mark $\mathcal{P}_i$ as a malicious party.

The sub-relation $\mathcal{R}_i$ here is:

$$\{(T_{1,i}, \mathbb{H}; f_i(X)) : \sum_{x\in\mathbb{H}} f_i(x) = T_{1,i}\},$$

[19] This property is similar to the partial witness-extended emulation as defined in Pianist [36]. A slight modification is that we require the extractor to extract the witness in a condition that the verifier accepts the final proof π.

[20] Specifically, it is a virtual polynomial constructable by $f_{p_i}(X), f'_{w_i}(X), R_i(r^m)$, $f_{v_i}(X), f_{a_i}(X), f'_{b_i}(X)$, and $f_{c_i}(X)$. For simplicity, we use $f_i(X)$ here.

[21] According to Definition 4, $T_{1,i}$ should satisfy that $\sum_{i\in[\ell]} T_{1,i} = 0$. Otherwise, the verifier would directly reject.

i.e., $\mathcal{P}_i$ indeed holds a secret polynomial $f_i \in \mathbb{F}_m[X]$ such that $\sum_{x\in\mathbb{H}} f_i(x) = T_{1,i}$. By the perfect soundness of univariate sum-check [5], the Schwartz-Zippel lemma, and the knowledge soundness of the KZG PCS, if all these checks pass, then the sub-relation follows with a high probability. This further shows that the partial knowledge soundness also follows.

Modifications for Protocol 5. In this protocol, sub-provers and the master prover show that $g(X,Y) = \sum_{i\in[\ell]} g_i(X)L_i(Y)$ is consistent with $b(X,Y) = \sum_{i\in[\ell]} b_i(X)L_i(Y)$ and $B(X,Y) = \sum_{i\in[\ell]} B_i(X)L_i(Y)$, *i.e.*, $g(x,y) = \frac{1}{\gamma+\xi\cdot b(x,y)+B(x,y)}$ for $x \in \mathbb{M}$ and $y \in \mathbb{L}$. This is proved by each sub-prover $\mathcal{P}_i$ proving that the polynomials $b_i(X), B_i(X)$ and $g_i(X)$ satisfy $g_i(x) = \frac{1}{\gamma+\xi\cdot b_i(x)+B_i(x)}$ for $x \in \mathbb{M}$. As these polynomials are of size $\tilde{m} = |\mathbb{M}|$ defined by evaluations over $\mathbb{M}$, this can be further reduced to that, given commitments to $b_i(X), B_i(X)$ and $g_i(X)$, each sub-prover proves that it holds a quotient polynomial $q_{g,i}(X)$ s.t.

$$g_i(X)\cdot(\gamma+\beta\cdot b_i(X)+B_i(X)) - 1 = q_{g,i}(X)\cdot Z_{\mathbb{M}}(X). \tag{49}$$

This is the sub-relation.

We next modify the protocol to achieve accountability. In this protocol, $\mathcal{P}_i$ would originally send the oracles of $b_i(X), B_i(X), g_i(X)$ and $q_{g,i}(X)$ to $\mathcal{P}_0$. After receiving the random challenge δ, $\mathcal{P}_i$ sends $b_i(\delta), B_i(\delta), g_i(\delta), q_{g,i}(\delta)$ and now is required to send their KZG evaluation proofs to $\mathcal{P}_0$. Then, $\mathcal{P}_0$ acts as a verifier to check the KZG proofs, and checks if

$$g_i(\delta)\cdot(\gamma+\beta\cdot b_i(\delta)+B_i(\delta)) - 1 = q_{g,i}(\delta)\cdot Z_{\mathbb{M}}(\delta).$$

If any one does not pass, $\mathcal{P}_0$ would mark $\mathcal{P}_i$ as a malicious party.

If all these checks pass, then by the Schwartz-Zippel lemma and the knowledge soundness of the KZG PCS, it holds with a high probability that the polynomials $g_i(X), b_i(X), B_i(X)$ held by $\mathcal{P}_i$ satisfies Equation (49), which further shows that $g_i(X)$ is consistent with $b_i(X)$ and $B_i(X)$. This further shows that the partial knowledge soundness also follows.

Modifications for Protocol 7. In Step 3, $\mathcal{P}_i$ would show the validity of the polynomials $A_{\mathsf{high},i}(X), A_{\mathsf{low},i}(X)$ and $B_i(X)$, which are proved by the lookup PIOP in Protocol 5 as described above.

In Step 3, $\mathcal{P}_i$ computes $f_{p_i}(\alpha)$ and sends it to $\mathcal{P}_0$. By the construction of $f_{p_i}(X)$, we have

$$f_{p_i}(\alpha) = \sum_{x\in\mathbb{M}} \mathsf{val}_i(x)\cdot r^{\mathsf{row}_i(x)}\cdot\alpha^{\mathsf{col}_i(x)} = \sum_{x\in\mathbb{M}} \mathsf{val}_i(x)\cdot A_{\mathsf{high},i}(x)\cdot A_{\mathsf{row},i}(x)\cdot B_i(x). \tag{50}$$

The sub-relation here is: given commitments to $\mathsf{val}_i(X), A_{\mathsf{high},i}(X), A_{\mathsf{row},i}(X)$ and $B_i(X)$, the sub-prover $\mathcal{P}_i$ shows the validity of $f_{p_i}(\alpha)$ by showing Equation (50).

By Protocol 7, $\mathcal{P}_i$ would invoke a univariate sum-check and send the commitments to $g_{3,i}(X)$ and $\{h_{3,j,i}(X)\}_{j\in[3]}$ to the verifier. To achieve accountability, when $\mathcal{P}_i$ sends $\mathsf{val}_i(\delta), A_{\mathsf{high},i}(\delta), A_{\mathsf{low},i}(\delta), B_i(\delta), g_3(\delta)$ and $\{h_{3,j,i}(\delta)\}_{j\in[3]}$ to $\mathcal{P}_0$,

it is additionally required to send the KZG evaluation proofs for these polynomials. $\mathcal{P}_0$ then checks if the following equation holds

$$\begin{aligned}&(\delta - u_3) \cdot \mathsf{val}_i(\delta) \cdot A_{\mathsf{high},i}(\delta) \cdot A_{\mathsf{low},i}(\delta) \cdot B_i(\delta)\\ =&X \cdot g_{3,i}(\delta) + (\delta - u_3) \cdot \big(f_{p_i}(\alpha)/\tilde{m} + \cdot Z_{\mathbb{M}}(X) \cdot \sum_{j\in[3]} \delta^{(j-1)\tilde{m}} \cdot h_{3,j,i}(\delta)\big).\end{aligned}$$

If it does not pass, $\mathcal{P}_0$ would mark $\mathcal{P}_i$ as a malicious party.

If all these checks pass, then by the perfect soundness of univariate sumcheck, the Schwartz-Zippel lemma, and the knowledge soundness of the KZG PCS, the value of $f_{p_i}(\alpha)$ is valid with a high probability. This further shows that the partial knowledge soundness also follows.

Accountability of the distributed PCS. We next show the accountability of our distributed batch KZG in Protocol 10 with slight modifications. The sub-relation here is that: 1) the evaluations of all $\{s_{k,i}(\alpha)\}_{k\in[n]}$ are valid, and 2) $f_{k,i}(X) - s_{k,i}(X)$ is divisible by $Z_{A_k}(X)$ for $k \in [n]$.

For 1), when $\mathcal{P}_i$ sends $s_{k,i}(\alpha)$ for all $k \in [n]$ and $\alpha \in A_k$, $\mathcal{P}_i$ also generates a KZG evaluation proof for these evaluations and sends them to $\mathcal{P}_0$. $\mathcal{P}_0$ verifies these proofs, and marks $\mathcal{P}_i$ as a malicious party if any proof does not pass. If all these proofs pass, this means that $\mathcal{P}_i$ holds polynomials $s_{k,i}(X)$ with consistent evaluations with a high probability.

For 2), let $U_{1,i} = [p_{1,i}(\sigma)L_i(\tau)]_1$ as in Step 3 s.t. $p_{1,i}(X) = P_i(X)/Z_A(X)$ and

$$P_i(X) = \sum_{k\in[n]} \gamma^{k-1} \cdot Z_{A\setminus A_k}(X) \cdot (f_{k,i}(X) - s_{k,i}(X)).$$

Let $V_{1,i} = [w_{1,i}(\sigma)L_i(\tau)]_1$ as in Step 5 s.t. $w_{1,i}(X) = W_{1,i}(X)/(X - z_1)$ and

$$W_{1,i}(X) = \sum_{k\in[n]} \gamma^{k-1} \cdot Z_{A\setminus A_k}(z_1) \cdot \big(f_{k,i}(X) - s_{k,i}(X)\big) - Z_A(z_1) \cdot P_i(X).$$

After receiving $U_{1,i}$ and $w_{1,i}$, $\mathcal{P}_0$ verifies its validity by checking if $W_{1,i}(X) = w_{1,i}(X) \cdot (X - z_1)$ over the commitment dimension. Specifically, $\mathcal{P}_0$ computes $g_i = \sum_{k\in[n]} \gamma^{k-1} \cdot Z_{A\setminus A_k}(z_1) \cdot (\mathsf{cm}_{f_{k,i}} - \mathsf{cm}_{s_{k,i}}) - Z_A(z_1) \cdot U_{1,i}$, and checks if

$$e(g_i, [1]_{2_i}) = e(V_{1,i}, [\sigma - z_1]_{2_i}),$$

where $[1]_{2_i}, [\sigma]_{2_i}, \ldots, [\sigma^{m-1}]_{2_i}$ are the SRS of $\mathcal{P}_i$ with generator $L_i(\tau) \cdot g$, and $\mathcal{P}_0$ holds an SRS of $[1]_{2_i}$ for KZG verification.

By the knowledge soundness of KZG, if all these checks pass, then with a high probability that the sub-relation holds. This further shows that the partial knowledge soundness also follows.

Complexity analysis. The overhead of the modifications above all comes from generating constant-number KZG evaluation proofs on constant-number size-$O(m)$ polynomials. Thus, the additional overheads of each $\mathcal{P}_i$ and $\mathcal{P}_0$ are $O(m)$ and $O(1)$, respectively. The additional proof size and amortized communication are $O(1)$. The additional verification overhead is $O(1)$ for KZG verification. We then conclude that achieving accountability does not introduce any additional asymptotic overhead.

G Adding Zero Knowledge

In this section, we show how to achieve zero knowledge for Soloist. Below, we first present several key facts to simplify the construction.

- Zk-SNARKs could be constructed by zero-knowledge PIOPs and PCSs with hiding Com and zero-knowledge Eval algorithms [11,13,14] (*cf.*, Definition 2).
- Our preprocessing PIOPs and corresponding PCSs, proving properties of *public* R1CS matrices, does not need to achieve zero knowledge.
- When considering zero knowledge, we can treat all provers as a whole from the view of the verifier, thereby adapting existing non-distributed techniques.

Based on these facts, we conclude that to build zero-knowledge Soloist, it suffices to modify the PIOP in Protocol 4 and the PCS in Protocol 9.

G.1 Zero-Knowledge PIOPs

We recall the definition of perfect zero-knowledge PIOPs in [13,14].

Definition 5. *A PIOP $\langle \mathcal{P}, \mathcal{V} \rangle$ has* perfect zero-knowledge *with query bound t if there exists a PPT simulator $\mathcal{S}$ such that for every statement $\mathbf{x}$, witness $\mathbf{w}$, and every t-admissible verifier $\mathcal{V}^*$, the transcripts below are identically distributed:*

$$\mathsf{View}\left(\mathcal{P}(\mathbf{x}; \mathbf{w}), \mathcal{V}^*\right) \approx \mathcal{S}^{\mathcal{V}^*}(\mathbf{x}) .$$

Here, the view consists of $\mathcal{V}^*$'s randomness, the non-oracle messages sent by $\mathcal{P}$, and the list of answers to $\mathcal{V}^*$'s oracle queries. A verifier is t-admissible *if it makes no more than t queries. We say that $\langle \mathcal{P}, \mathcal{V} \rangle$ is zero-knoweldge if there exists a simulator for $\mathcal{V}$.*

Zero-knowledge univariate sum-check PIOP. Aurora [5] describes a zero-knowledge method for univariate sum-check PIOP, which we recall below.

Zero-knowledge univariate sum-check PIOP $\langle \mathcal{P}, \mathcal{V} \rangle$:

- Input: a polynomial $f \in \mathbb{F}_m[X]$ and a claimed sum T.

1. $\mathcal{P}$ samples a random polynomial $u \in \mathbb{F}_m[X]$ and computes $U = \sum_{x \in \mathbb{H}} u(x)$. Send the oracle $u(X)$ and value U to $\mathcal{V}$.
2. $\mathcal{V}$ sends a random challenge $r \in \mathbb{F}$ to $\mathcal{P}$.
3. $\mathcal{P}$ and $\mathcal{V}$ run the univariate sum-check PIOP (*cf.*, Section 2) over the polynomial $f(X) + r \cdot u(X)$ and the claimed sum $T + r \cdot U$.

Our construction. Recall that Protocol 4 is an amortized version of a distributed univariate sum-check PIOP to prove Equation (7). As the amortized univariate sum-check does not affect zero knowledge [5], it suffices to construct an HVZK distributed PIOP for $\sum_{x \in \mathbb{H}} \sum_{i \in [\ell]} f_i(x) = T$, where $f_i(X) = f_i^{(1)}(X) \cdot f_i^{(2)}(X) \cdot R_i(z_1) - f_i^{(3)}(r) \cdot R_i(z_2)$ and $T = 0$ in Equation (7).

Zero-knowledge PIOP for $\sum_{x \in \mathbb{H}} \sum_{i \in [\ell]} f_i(x) = T$ $\langle \mathcal{P}, \mathcal{V} \rangle$:

– Input: polynomials $f_i \in \mathbb{F}_{2m-1}[X]$ for $i \in [\ell]$ and a claimed sum T.

1. $\mathcal{P}_i$ picks a random polynomial $u_i \in \mathbb{F}_{2m-1}[X]$ and computes $U_i = \sum_{x\in\mathbb{H}} u_i(x)$. Send the oracle $u_i(X)$ and value U_i to the master prover $\mathcal{P}_0$.
2. $\mathcal{P}_0$ computes and sends the oracle $u(X,Y) = \sum_{i\in[\ell]} u_i(X)L_i(Y)$ and the value $U = \sum_{i\in[\ell]} U_i$ to $\mathcal{V}$.
3. $\mathcal{V}$ sends a random challenge $r \in \mathbb{F}$ to $\mathcal{P}$.
4. $\mathcal{P}$ and $\mathcal{V}$ run the two-dimension sum-check PIOP (*cf.*, Section 3.2) to prove $\sum_{x\in\mathbb{H}}\sum_{i\in[\ell]}\big(f_i(x) + r\cdot u_i(x)\big) = T + r\cdot U$.[22] Specifically,
   (a) $\mathcal{P}_i$ computes $g_i(X), h_i(X)$ by dividing $Z_{\mathbb{H}}(X)$ over $f_i(X) + r\cdot u_i(X)$ s.t.
   $$f_i(X) + r\cdot u_i(X) = X\cdot g_{1,i}(X) + T_{1,i}/m + Z_{\mathbb{H}}(X)\cdot h_{1,i}(X).$$
   $\mathcal{P}_i$ sends the oracles $g_{1,i}(X), h_{1,i}(X)$ to $\mathcal{P}_0$.
   (b) $\mathcal{P}_0$ computes and sends the oracles $g_1(X) = \sum_{i\in[\ell]} g_{1,i}(X)$ and $h_1(X) = \sum_{i\in[\ell]} h_{1,i}(X)$ to $\mathcal{V}$.
   (c) $\mathcal{V}$ sends a random challenge $\alpha \in \mathbb{F}$ to $\mathcal{P}_0$, who transferrs to $\mathcal{P}_i$.
   (d) $\mathcal{P}_i$ computes and sends $f_i(\alpha), u_i(\alpha), g_{1,i}(\alpha), h_{1,i}(\alpha)$ to $\mathcal{P}_0$.
   (e) $\mathcal{P}_0$ computes $f(\alpha,Y) = \sum_{i\in[\ell]} f_i(\alpha)L_i(Y)$ and $u(\alpha,Y) = \sum_{i\in[\ell]} u_i(\alpha)L_i(Y)$. Compute $g_1(\alpha) = \sum_{i\in[\ell]} g_{1,i}(\alpha)$ and $h_1(\alpha) = \sum_{i\in[\ell]} h_{1,i}(\alpha)$. Compute $g_2(Y)$, $h_2(Y)$ such that by dividing $Z_{\mathbb{L}}(Y)$ over $f(\alpha,Y) + r\cdot u(\alpha,Y)$ such that
   $$f(\alpha,Y) + r\cdot u(\alpha,Y) = Y\cdot g_2(Y) + T_2/\ell + Z_{\mathbb{L}}(Y)\cdot h_2(Y). \tag{51}$$
   (f) $\mathcal{V}$ queries $g_1(\alpha), h_1(\alpha)$, picks a random $\beta \in \mathbb{F}$ and queries $f(\alpha,\beta), u(\alpha,\beta), g_2(\beta)$, $h_2(\beta)$. Accept iff Equation (51) holds at $Y = \beta$, where $T_2 = \alpha\cdot g_1(\alpha) + T + r\cdot U + Z_{\mathbb{H}}(\alpha)\cdot h_1(\alpha)$.

In the above protocol, completeness holds directly. As for soundness, for any sets of $u_i(X), U$ and $f_i(X)$, if $\sum_{x\in\mathbb{H}}\sum_{i\in[\ell]} f_i(x) \neq T$, then the probability of $\sum_{x\in\mathbb{H}}\sum_{i\in[\ell]} f_i(x) + r\cdot u_i(x) = T + r\cdot U$ is $1/|\mathbb{F}|$ due to the randomness of r. The above modifications for zero knowledge do not affect the complexities. We next describe a simulator $\mathcal{S}$ that, given access to a (malicious) verifier $\mathcal{V}^*$ and oracle access to $f(X,Y)$, perfectly simulates $\mathcal{V}^*$'s view in the real protocol.

1. $\mathcal{S}$ samples random polynomials $u_{\mathsf{sim},i} \in \mathbb{F}_{2m-1}[X]$ for $i \in [\ell]$.
2. $\mathcal{S}$ sends $U_{\mathsf{sim}} = \sum_{i\in[\ell]}\sum_{x\in\mathbb{H}} u_{\mathsf{sim},i}(x)$ to $\mathcal{V}^*$.
3. Receive r_{sim} from $\mathcal{V}^*$.
4. Randomly sample $p_{\mathsf{sim}} \in \mathbb{F}_{2m-1}[X]$ for $i \in [\ell]$ such that, $\sum_{x\in\mathbb{H}}\sum_{i\in[\ell]} p_{\mathsf{sim}}(x) = T + r\cdot U_{\mathsf{sim}}$.
5. Answer any query to $f(X,Y)$ by querying $f(X,Y)$, and answer any query to $u(X,Y)$ by querying $\sum_{i\in[\ell]} p_{\mathsf{sim}}(X)L_i(Y) - r\cdot f(X,Y)$.
6. Run the following procedures as an honest prover.

To see that $\mathcal{V}^*$'s view is perfectly simulated, we adapt the techniques from Aurora [5] and consider a hybrid experiment in which the "hybrid prover" reads

[22] For simplicity, we omit to use the LDT elimination technique as in Protocol 2.

all of $f(X,Y)$ (like the honest prover in the real world) but can modify messages after they are sent (like the simulator in the ideal world). It runs basically the same as $\mathcal{S}$ but modifies $u(X,Y)$ as $\sum_{i\in[\ell]} p_{\mathsf{sim}}(X)L_i(Y) - r\cdot f(X,Y)$ in Step 4. The distribution of $\mathcal{V}^*$'s view in the real protocol is identical to the distribution of $\mathcal{V}^*$'s view in the above experiment, which is also idential to $\mathcal{S}$'s output.

G.2 Zero-Knowledge PCSs

Libra [58] presents a zero-knowledge multilinear KZG with hiding Com and zero-knowledge Eval algorithms secure under the knowledge of exponentiation assumption. It can be naturally modified as a bivariate or univariate version secure in the algebraic group model.

Given a polynomial $f(X,Y) = \sum_{i\in[\ell]} f_i(X)L_i(Y)$, we adapt the techniques in Libra to modify it as $f'(X,Y) = \sum_{i\in[\ell]}(f_i(X)+v_i(X)\cdot Z_{\mathbb{H}}(X))\cdot L_i(Y)$, where $v_i(X)$ is a degree-$(t{+}1)$ random polynomial and t is the query number to $f(X,Y)$. This guarantees that $\sum_{x\in\mathbb{H}}\sum_{i\in[\ell]} f_i(x) = \sum_{x\in\mathbb{H}}\sum_{i\in[\ell]} f_i(X) + v_i(X)\cdot Z_{\mathbb{H}}(x)$. Further, each sub-prover $\mathcal{P}_i$ could modify the i-th part locally, which does not affect the sub-prover complexity.

Given the modifications to $f_k(X,Y)$ for all $k\in[n]$ as in Protocol 9, the prover invokes the Com algorithm of Libra to obtain cm_k, acts as an honest prover to commit and open intermediate polynomials like $q(X), q_1(X), q_2(X)$, $q_3(X)$, and runs the zero-knowledge Eval of Libra to open $f'_k(X,Y)$. Any query to $f_k(X,Y)$ is modified into $f'_k(X,Y)$, with an additional requirement that $\alpha,\eta\notin\mathbb{H}$. Then, evaluations on $f'_k(X,Y)$ would not leak any information. Also, the intermediate polynomials including $q(X), q_1(X), q_2(X), q_3(X)$ are masked by $\sum_{i\in[\ell]} v_i(X)L_i(Y)$, which do not leak additional information. We omit the formal protocols and proofs here and kindly refer to [58] for details..

H Additional Comparison and Evaluation

H.1 Evaluation of IPA and Batch Bivariate KZG

Performance of IPAs. We implement IPAs in Marlin [14], Dark [11], and our scheme by combining the corresponding PIOPs with the batch KZG of Boneh *et al.* [7]. Fig. 7 shows their performances. As shown, our IPA has a 20% faster prover time, $5\% \sim 80\%$ faster verifier time, and $30\% \sim 200\%$ smaller proof size. This benefits from the superior complexities for fewer FFTs in prover, fewer PCS commitments and openings, and fewer proof size (*cf.*, Table 1).

There are also other potential IPAs from the "PIOP + PCS" paradiagm with constant proof size, such as Count [35], SZ22 [52], and Dew [2]. Due to the lack of open-sourced implementations, we compare them in theory. Count proposes an improved univariate sum-check, which can be used to build 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

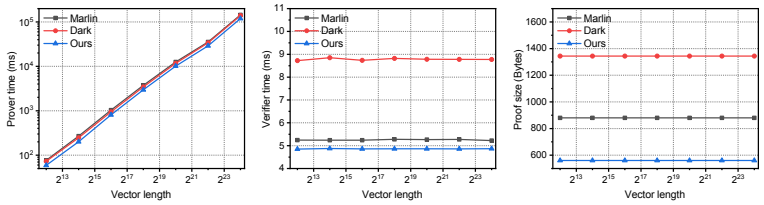

Fig. 7. Performance of IPAs from "PIOP + PCS" with constant proof size

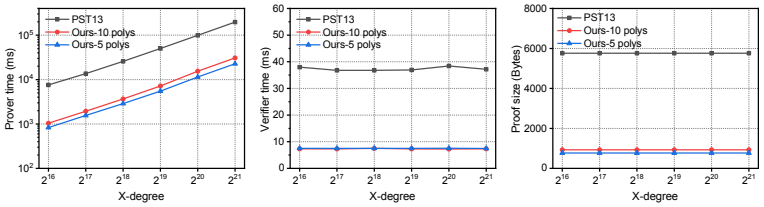

Fig. 8. Performance of batch bivariate KZG with the same Y-dimension point

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 more abstract than the algebraic group model [17] in our IPA. Also, many recent SNARKs [14,19,13], if using the KZG PCS and its variant, are secure in the algebraic group model.

Performance of batch bivariate KZG. As the implementation of the batch scheme of Boomy [31] is not released, and the code of HyperPlonk's batch scheme lacks support for high-degree multivariate polynomials, we compare our batch scheme with the baseline bivariate KZG PST13 [42], where we directly open all polynomials and points without batching. We evaluate Protocol 9 where the evaluation points share the same Y-dimension, as it is exactly the case in Soloist. We test to open 20 random points in two cases: 2 on each of 10 polynomials; or 4 on each of 5 polynomials, both with the Y-degree set to 7 (emulating the case where the number of sub-provers is 8 when considering the distributed setting). These parameters are chosen or generalized from the practical test cases of Soloist. Compared with PST13 [42] for directly opening all points, ours has a $7 \sim 10\times$ faster prover time, $5\times$ faster verifier time, and a $6 \sim 7\times$ smaller proof size. When opening more points on one polynomial, our PCS would feature a smaller proof size, which is only linear in the polynomial number and irrelevant to the point number on each polynomial or the polynomial size.

Although we do not compare the evaluations of Boomy and HyperPlonk, we argue that they are either infeasible or inefficient compared with our batch bivariate KZG. Specifically, Boomy only supports to open multiple points over one single polynomial. When forced to open n polynomials (by repeating the scheme n times), its concrete proof size would be $2n\,|\mathbb{G}|$, which is concretely $> 2\times$ larger than our proof size $4\,|\mathbb{G}| + (n+1)\,|\mathbb{F}|$ (*cf.*, Table 2). Further, the

Table 6. Performance of the "making-even" Algorithm 1 over zkRollup circuit

Prover number	original $\tilde{m}$	evened $\tilde{m}$	Prover number	original $\tilde{m}$	evened $\tilde{m}$
2	2^{20}	2^{19}	4	2^{19}	2^{18}
8	2^{18}	2^{17}	16	2^{18}	2^{16}

Table 7. Performance of the "making-even" Algorithm 1 over ECDSA circuit

Prover number	original $\tilde{m}$	evened $\tilde{m}$	Prover number	original $\tilde{m}$	evened $\tilde{m}$
2	2^{20}	2^{19}	4	2^{20}	2^{18}
8	2^{20}	2^{17}	16	2^{20}	2^{16}

opening complexity (over group) of Boomy is linear to the polynomial number n, while our PCS is irrelevant to n. The proof size of HyperPlonk is linear to the sum of the degrees over the X and Y dimensions. Hence, if proving large-size polynomials (*e.g.*, linear in the statement size) as in Soloist, its proof size would be much larger than ours. Further, its prover in the opening phase needs to run FFTs due to the high-degree multivariate sum-check, while our PCS does not requre any FFTs.

H.2 Comparison with Non-Group-based Distributed SNARKs

DeVirgo [57] is a distributed SNARK from a distributed PIOP for data-parallel circuits and a code-based distributed PCS [57]. Such PCSs do not require group operations and can be concretely faster than group-based ones. Hence, DeVirgo can be faster than Soloist in proving. According to its figures, proving signatures with 2^{25} constraints utilized with 32 machines costs 5s, which is $6\times$ faster than ours when proving random R1CS with constraints of the same number.

However, DeVirgo only supports data-parallel circuits instead of general ones supported by Soloist. DeVirgo's proof size is about 1.9MB according to the reported figures, while ours is only 10KB and $190\times$ smaller. Also, DeVirgo's communication complexity is linear in the statement size, and is up to several GBs, which is $10^5\times$ larger than ours, limiting its practical use in specific applications.

H.3 Performance of Our "Making-Even" Algorithm

To demonstrate the performance of Algorithm 1, we run Soloist over the circuit with 1 batch of transactions, where $\tilde{m} = 2^{20}$ in the non-distributed case. As shown in Table 6, after running Algorithm 1, the size of $\tilde{m}$ is optimally reduced by $\ell\times$ given ℓ provers, and is $2 \sim 4\times$ smaller than not running the algorithm. In addition, running Algorithm 1 is efficient, which costs only 0.5s.

Apart from the zkRollup circuit, we also implement Soloist over the ECDSA verification circuit from Circom[23], which proves knowledge of a private key corresponding to an Ethereum address. It has $247,380$ constraints, and the largest

[23] https://github.com/0xPARC/circom-ecdsa

number of non-zero entries in R1CS matrices (*i.e.*, $\tilde{m}$) is close to 2^{21}. Table 7 presents the size of $\tilde{m}$ before and after running the "making-even" algorithm. As shown, the size of $\tilde{m}$ is also optimally minimized by $\ell\times$, and is $2 \sim 16\times$ smaller than without running Algorithm 1.

I Our Circuit Division Algorithm for Hekaton

This section proposes the circuit division algorithm to implement Hekaton over general circuits. Our goal is to make the number of shared wires among sub-circuits minimized. First, see the circuit to be proved as a hypergraph, where each edge can connect more than two nodes. Then, treat circuit constraints as nodes of hypergraph and treat circuit variables as edges. Each edge would link all constraints using this variable. Finally, run the KaHyPar algorithm [47] of *hypergraph partitioning with minimum cut* to divide circuits, *i.e.*, making the number of edges spanning multiple partitions minimized. By definition, the number of these edges means exactly the number of shared wires.

Apart from using the KaHyPar algorithm based on hypergraph, there may exist other valid circuit division algorithms, which is left for a further work. Despite this, we note that some compilers from a SNARK for data-parallel circuits to a SNARK for general circuits (*e.g.*, [8]) are not applicable. This is because Hekaon originally supports general circuits instead of aiming specifically data-parallel circuits. The performance degradation with the increase of prover number is due to the characteristics of Hekaton's aggregation scheme.