VCProof: Constructing Shorter and Faster-to-Verify zkSNARKs with Vector Oracles

Yuncong Zhang[1⋆], Ren Zhang[2,3], Geng Wang[1], and Dawu Gu[1⋆⋆]

[1] Shanghai Jiao Tong University
{shjdzhangyuncong,wanggxx,dwgu}@sjtu.edu.cn
[2] Nervos
ren@nervos.org
[3] Shandong Institute of Blockchain

Abstract. The construction of zkSNARKs involves designing a Polynomial IOP that matches with the constraint system for which it proves membership. Designing this Polynomial IOP is a challenging task because the constraint system is typically not expressed in terms of polynomials but in terms of matrices and vectors. To mitigate mismatch, we propose a new methodology for the first step in SNARK construction, that first designs a matching Vector Oracle protocol before compiling it into a Polynomial IOP. The native first-class citizens of the Vector Oracle protocol are vectors; and by virtue of matching with the language of the arithmetic constraint system, Vector Oracle protocols are more intuitive to design and analyze. The Vector-Oracle-to-PIOP compilation procedure is protocol-independent, allowing us to present and optimize it as a standalone component, leading to the discovery of a series of acceleration techniques.

We apply our methodology to construct three zkSNARKs, each targeting a constraint system: the Rank-1 Constaint System (R1CS), the Hadamard Product Relation (HPR), and a modified PLONK circuit. All three zkSNARKs achieve shorter proofs and/or smaller verification costs compared to the state-of-the-art constructions targeting the same constraint systems. Specifically, VCProof/R1CS defeats Marlin in proof size, with a slightly higher verification cost; VCProof/HPR and VCProof/POV outperform Sonic and PLONK, respectively, in both proof sizes and verification costs. In particular, the proof of VCProof/POV has only two field elements and six group elements, thus becoming the shortest among all existing universal-setup zkSNARKs.

Keywords: Zero-Knowledge · SNARK · PIOP · Vector Oracle

1 Introduction

Zero-knowledge SNARKs (zkSNARKs), first introduced by Bitansky et al. in 2012 [1], allow a prover to generate a short proof π for a computa-

⋆ Supervised by Dawu Gu, and also partially supported by Nervos Foundation.
⋆⋆ Corresponding author.

tion output $y = F(x, w)$ of an arbitrary function F, such that a resource-restricted verifier can validate y with at most $O(\text{polylog}(|F|))$ computational and storage cost while learning nothing about the secret input w. Recent years witnessed a surge of zkSNARKs with various properties, e.g., constant verification time [2,3,4,5], universal setup [3,4,5,6,7,8,9], transparent setup [6,7,8,9], and post-quantum security [7,8,10]. New designs emerge rapidly with smaller construction and verification costs, shorter proofs, and fewer security assumptions. Despite its short history, zkSNARKs have already been deployed in many blockchain-based scenarios, e.g., Zcash [11], the first fully anonymous cryptocurrency, and Aztec [12] and zkSync [13], two projects boosting the scalability and privacy of Ethereum—the cryptocurrency with the second-largest market capitalization.

Albeit diverse in the underlying techniques, most zkSNARKs share the same construction workflow centered on a *Polynomial IOP (PIOP)* protocol, as pointed out by Bünz et al. [6]. In this workflow, the to-be-verified equation $y = F(x, w)$ is first transformed into a constraint system over a finite field $\mathbb{F}$. Second, a PIOP protocol is designed to verify that an instance-witness pair (x, w) satisfies the constraint system. The PIOP protocol involves two parties: a prover and a verifier. The prover sends to the verifier, potentially in multiple rounds, a set of polynomials, each encapsulated in a *polynomial oracle*—a kind of idealized functionality. The verifier may query these oracles for evaluations of these polynomials at arbitrary elements of $\mathbb{F}$. Finally, the PIOP protocol is compiled into a zkSNARK by a polynomial commitment scheme, e.g., KZG [14] or DARK [6]. Concurrent with [6], the notion PIOP is also proposed by Chiesa et al. with the name *Algebraic Holographic Proof (AHP)* [4] and Gabizon et al. as *Polynomial Protocols* [3]. As a middle layer between the constraint system and the zkSNARK, PIOP abstracts away the complexities in the latter's underlying cryptography. Moreover, this middle layer leaves the choice of the polynomial commitment scheme flexible, allowing practitioners to fine-tune the trade-off between higher efficiency and fewer security assumptions.

This paper focuses on the PIOP-designing step. We observe a common pattern among prior constructions: they all need to implement, with polynomial equations, one or both of two basic vector operations, which are (1) the inner product, and (2) the *Hadamard product*, i.e., entry-wise product. For example, in Libra [15], Marlin [4] and Fractal [8], the inner product is implemented using the sumcheck protocol [16] or its univariate variant [10], and the Hadamard product directly corresponds to the poly-

nomial product. The indispensability of these vector operations inspires us to propose a new zkSNARK construction methodology, which first designs a *Vector Oracle protocol* that translates the constraint system into inner-product and Hadamard-product checks, and then implements this protocol as a PIOP protocol.

This new construction method, named *VCProof* for "VeCtor Proof", has the following two-fold advantage. First, a Vector Oracle protocol is more intuitive to design than a PIOP protocol targeting the same constraint system, because the constraint system usually consists of equations of matrices and vectors, rather than polynomials. Second, by defining the Vector Oracle protocol as a standalone abstraction, we explicitly separate the polynomial representations of the aforementioned basic vector operations from the protocol-specific logic. This separation enables flexible combination of the Vector Oracle protocol and the polynomial representation of the vector operations, thus opening a large design space for optimization. We formalize this method and show that it is capable of producing zkSNARKs, for various constraint systems, with shorter proofs and verification time than the state of the art. Specifically, our contributions include:

The Vector Oracle Formalization. We simplify the PIOP protocol design by separating (1) the implementation of the Hadamard product and the inner product operations from (2) the constraint-system-specific protocol logic, which is described via a Vector Oracle protocol (Sect. 3). A Vector Oracle protocol is formalized as an interactive protocol in which the prover and the verifier have access to a *vector oracle*, which receives vector submissions from the prover, and receives queries from the verifier to manipulate these vectors or to check vector identities involving Hadamard products and inner products. We show that a Vector Oracle protocol can be compiled to a PIOP protocol targeting the same constraint system, and vice versa. This formalization also uncovers the optimization technique of batching all the Hadamard- and inner-product checks into a single polynomial equation. The resulting PIOP protocol is significantly optimized in the number of polynomial oracles (only one more than the number of prover-submitted vectors) and the number of evaluation queries (two to four distinct points).

Efficient Vector Oracle Protocols for Verifying Vector Permutation and Sparse Matrix Vector Multiplication. We present a protocol for verifying vector permutations with only one preprocessed vector and one prover-submitted vector, and a protocol for sparse matrix vector multiplications with only two preprocessed vectors and four prover-

submitted vectors (Sect. 4). These protocols serve as building blocks for protocols that verify the circuit-based constraint systems. These building blocks are also useful in building zkSNARKs for specialized tasks, e.g., cryptographic tools like vector commitments, matrix commitments, range proofs, without resorting to general-purpose zkSNARKs that usually have higher proving and verification costs.

Implementing Vector Oracle Protocols for Influential Circuit-based Constraint Systems. We present three Vector Oracle protocols targeting three circuit-based constraint systems, respectively: the Rank-1 Constraint System (R1CS), the Hadamard Product Relation (HPR), and a modified version of the PLONK circuit which we name the POV (Permutation Oecumenical Vector-oracle) relation (Sect. 5). We name these protocols VCProof/R1CS, VCProof/HPR and VCProof/POV. These protocols are constructed in a modular manner by composing the aforementioned building blocks, and further optimized by the vector concatenation technique to reduce the number of vectors submitted by the prover.

New zkSNARKs with Shorter Proofs and Smaller Verification Costs. Finally, we compile our Vector Oracle protocols to produce zkSNARKs with competitive efficiency compared to the state of the art (Sect. 6). We transform the above three Vector Oracle protocols to PIOP protocols, which are further compiled into zkSNARKs via both KZG [14] and DARK [6] polynomial commitment schemes. When compiled with KZG, these PIOP protocols produce preprocessing zkSNARKs with universal trusted setups. Compared to the state-of-the-art constructions for the same constraint systems, all three zkSNARKs reduce the proof size by roughly a half, while two of them also lower the verification cost. The efficiency improvements result from (1) the vector concatenation technique applied in the Vector Oracle protocols, (2) the batching technique in the Vector Oracle compiler, and (3) a concrete-level optimization inspired by PLONK [3]. Although the vector concatenation technique slightly sacrifices the efficiency in proof construction, the prover efficiency remains comparable to the concurrent constructions due to other optimizations. In more detail, the verification cost of VCProof/R1CS is slightly larger than Marlin by 20 group exponentiations. For HPR, compared to Sonic, the verifier of VCProof/HPR is faster by roughly six times. For the PLONK circuit, the verifier of VCProof/POV outperforms PLONK by six group exponentiations. In particular, the proof of VCProof/POV has only two field elements and six group elements, becoming the shortest among all known universal-setup zkSNARKs. If we compile the PIOP protocols using DARK instead, the resulting zkSNARKs have transparent setups at

the cost of a logarithmic blowup in proof sizes and verifier costs with respect to the circuit size.

1.1 Related Works

We classify zkSNARKs into three groups, based on how they achieve *succinctness*—the "S" in zkSNARK. The first group includes BulletProof [17] and Aurora [10], which achieve logarithmic proof sizes and linear verifier complexities. The second group target only *uniform* circuits, i.e., those with very short representations. Examples include Libra [15], which requires the circuit to be layered and *log-space uniform*, and STARK [7] and vRAM [18], which target Random-Access-Machines (RAMs) that are equivalent to circuits consisting of repetitions of the same sub-circuit.

Most zkSNARKs fall into the third group, which introduce preprocessing, allowing the verifier to read a short digest instead of the complete circuit. The hash-based design Fractal [8] does not require trusted setup and is presumably post-quantum secure. Pinocchio [19] and Groth16 [2] are pairing-based zkSNARKs that require per-circuit trusted setups. In comparison, Marlin [4], PLONK [3] and Sonic [5] only require a universal trusted setup. These pairing-based zkSNARKs have constant proof sizes and verifier complexities.

Supersonic [6] and Claymore [20] cross the group boundaries by focusing more on the methodology instead of standalone zkSNARKs. Specifically, Supersonic proposes the DARK polynomial commitment and the PIOP formalization, and Claymore focuses on the PIOP protocol. Our work is inspired by them as we also focus on improving the methodology and constructing abstract-level protocols.

2 Preliminaries

2.1 Notations

Let $\mathbb{Z}$ be the set of integers and $\mathbb{N}$ the set of nonnegative integers. For convenience, we abbreviate the set $\{i\}_{i=1}^{n}$ by $[n]$, and $\{i\}_{i=m}^{n}$ by $[m..n]$ for $m < n$. Throughout this paper, we use a unique finite field $\mathbb{F}$. Elements in $\mathbb{F}$ are also called scalars.

We denote the vectors by bold lowercase letters, e.g., $\boldsymbol{a} \in \mathbb{F}^n$ is a vector of size n over $\mathbb{F}$. We use $\boldsymbol{a}_{[i]}$ for the i-th element of the vector $\boldsymbol{a}$, where the indices start from 1. Let $\boldsymbol{a}_{[i..j]} := (\boldsymbol{a}_{[i]}, \cdots, \boldsymbol{a}_{[j]})$ for $i \leq j$. For $i > n$, we treat $\boldsymbol{a}_{[i]} = 0$ for convenience.

We use the following binary operations between vectors. For two vectors $\boldsymbol{a} \in \mathbb{F}^{n_1}$ and $\boldsymbol{b} \in \mathbb{F}^{n_2}$, their *concatenation* is $\boldsymbol{a} \| \boldsymbol{b} := (\boldsymbol{a}_{[1]}, \cdots, \boldsymbol{a}_{[n_1]}, \boldsymbol{b}_{[1]}, \cdots, \boldsymbol{b}_{[n_2]}) \in \mathbb{F}^{n_1+n_2}$. Their *sum* is $\boldsymbol{a} + \boldsymbol{b} := (\boldsymbol{a}_{[i]} + \boldsymbol{b}_{[i]})_{i=1}^{\max\{n_1,n_2\}} \in \mathbb{F}^{\max\{n_1,n_2\}}$. Their *inner product* is $\boldsymbol{a} \cdot \boldsymbol{b} := \sum_{i \in [\min\{n_1,n_2\}]} \boldsymbol{a}_{[i]} \cdot \boldsymbol{b}_{[i]}$. Their *Hadamard (entrywise) product* is $\boldsymbol{a} \circ \boldsymbol{b} := (\boldsymbol{a}_{[i]} \cdot \boldsymbol{b}_{[i]})_{i=1}^{\min\{n_1,n_2\}} \in \mathbb{F}^{\min\{n_1,n_2\}}$. We will use *power vectors*, i.e., vectors of the form $(1, \alpha, \alpha^2, \cdots, \alpha^{n-1})$, denoted by $\boldsymbol{\alpha}^n$. In particular, $\mathbf{1}^n$ and $\mathbf{0}^n$ are the all-one and all-zero vectors of size n.

We use bold capital uppercase letters for matrices, e.g., $\boldsymbol{M} \in \mathbb{F}^{m \times n}$ is a matrix of size $m \times n$ over $\mathbb{F}$. The element of $\boldsymbol{M}$ in the i-th row and j'th column of the matrix is $\boldsymbol{M}_{[i,j]}$. The i-th row is a vector of size n denoted by $\boldsymbol{M}_{[i,..]}$ and the j'th column is $\boldsymbol{M}_{[..,j]} \in \mathbb{F}^m$. The matrix-vector product is denoted by either by $\boldsymbol{M}\boldsymbol{v}$ for right multiplication of vectors or $\boldsymbol{v}^\mathsf{T}\boldsymbol{M}$ for left multiplication.

We write $f(X) \in \mathbb{F}^d[X]$ as a polynomial of degree at most d over field $\mathbb{F}$. When the context is clear, we use f_i to represent the coefficient for X^i. For a vector $\boldsymbol{v} \in \mathbb{F}^d$, let $f_{\boldsymbol{v}}(X)$ be the polynomial of degree less than d that uses the elements of $\boldsymbol{v}$ as coefficients, i.e., $f_{\boldsymbol{v}}(X) = \sum_{i=1}^{d} \boldsymbol{v}_{[i]} X^{i-1}$.

2.2 Polynomial IOP for Indexed Relations

This work focuses on protocols that admit a preprocessing procedure which, on input the index for the target relation, produces helpful information for the prover and the verifier in the protocol. The indexed relation is a convenient notion in this context. Formally, an indexed relation $\mathcal{R}$ is a set of triples $(\mathtt{i}, \mathtt{x}, \mathtt{w})$ where $\mathtt{i}$ is the index, $\mathtt{x}$ is the instance, and $\mathtt{w}$ is the witness. The indexed language induced by $\mathcal{R}$ is $\mathcal{L}(\mathcal{R}) := \{(\mathtt{i}, \mathtt{x}) : \exists \mathtt{w}, (\mathtt{i}, \mathtt{x}, \mathtt{w}) \in \mathcal{R}\}$.

A *PIOP (Polynomial Interactive Oracle Proof)* protocol [6] is an interactive protocol in which a prover shows a verifier that $(\mathtt{i}, \mathtt{x}) \in \mathcal{L}(\mathcal{R})$. The protocol may involve an indexer which preprocesses the index $\mathtt{i}$ and sends helpful information to the prover and the verifier. Unlike ordinary interactive protocols, the prover and the indexer in a PIOP protocol may send *polynomial oracles* to the verifier. Furthermore, the prover messages are restricted to polynomial oracles. A polynomial oracle is an idealized functionality that encapsulates a polynomial, say $f(X)$. The verifier can query any $z \in \mathbb{F}$ to the oracle and receive $f(z)$ as the reply. We denote the polynomial oracle for $f(X)$ by $[f(X)]$.

We focus on *public-coin* protocols where the verifier messages are all fresh uniformly random strings.

Definition 1 (Public-Coin Preprocessing PIOP). *An preprocessing PIOP protocol with degree bound d for an indexed relation $\mathcal{R}$ is a protocol consisting of three PPT algorithms $(\mathsf{I}, \mathsf{P}, \mathsf{V})$. The input to this protocol is a tuple $(\mathtt{i}, \mathtt{x}, \mathtt{w})$. The indexer I takes as input $\mathtt{i}$ and outputs $\mathtt{i}_P$, $\mathtt{i}_V$ and a list of polynomials $f^{(1)}(X), \cdots, f^{(m)}(X) \in \mathbb{F}^d[X]$. The prover P takes as input $\mathtt{i}_P, \mathtt{x}, \mathtt{w}$, and the verifier V takes as input $\mathtt{i}_V, \mathtt{x}$. Then P and V proceed in r rounds. In the i-th round, V may send to P uniformly random strings obtained by tossing a public coin, and P produces a polynomial $f^{(m+i)}(X)$ of degree at most d. After the last round, V produces the query points $\{(z_{i,1}, \cdots, z_{i,n_i})\}_{i=1}^{m+r}$, and receives $\{(y_{i,1}, \cdots, y_{i,n_i})\}_{i=1}^{m+r} := \{(f^{(i)}(z_{i,1}), \cdots, f^{(i)}(y_{i,n_i}))\}_{i=1}^{m+r}$. Finally, V outputs $b \in \{0,1\}$, which is regarded as the output of the entire protocol on input $(\mathtt{i}, \mathtt{x}, \mathtt{w})$, denoted by $b \leftarrow \langle \mathsf{I}, \mathsf{P}, \mathsf{V} \rangle (\mathtt{i}, \mathtt{x}, \mathtt{w})$. The* ***transcript*** *of the PIOP protocol consists of all the verifier messages, all the evaluation points $\{(z_{i1}, \cdots, z_{in_i})\}_{i=1}^{m+r}$, and all the evaluation results $\{(y_{i1}, \cdots, y_{in_i})\}_{i=1}^{m+r}$. The transcript is denoted by $\mathsf{tr}\langle \mathsf{I}, \mathsf{P}, \mathsf{V} \rangle (\mathtt{i}, \mathtt{x}, \mathtt{w})$.*

A PIOP protocol may have the following properties:

- *It has completeness error ε_c (or perfect completeness for $\varepsilon_c = 0$) if for any $(\mathtt{i}, \mathtt{x}, \mathtt{w}) \in \mathcal{R}$, we have*

$$\Pr\left[b = 0 \mid b \leftarrow \langle \mathsf{I}, \mathsf{P}, \mathsf{V} \rangle (\mathtt{i}, \mathtt{x}, \mathtt{w})\right] \leq \varepsilon_c.$$

- *It has soundness error ε_s (or perfect soundness for $\varepsilon_s = 0$) if for any $(\mathtt{i}, \mathtt{x}) \notin \mathcal{L}(\mathcal{R})$, and any unbounded algorithm P^*, we have*

$$\Pr\left[b = 1 \mid b \leftarrow \langle \mathsf{I}, \mathsf{P}^*, \mathsf{V} \rangle (\mathtt{i}, \mathtt{x}, \perp)\right] \leq \varepsilon_s.$$

The probabilities are over the randomnesses of the prover and the public coin tossed by the verifier.

The notion *zero-knowledge (ZK)* requires that any verifier cannot acquire any information by interacting with the honest prover. This security is formally defined by a simulator which, without learning the witness, outputs the transcript whose distribution is indistinguishable from that of a real execution of the protocol. In this paper, we focus on *honest-verifier statistical zero-knowledge*, which only requires that the simulator exists for the honest verifier. This version of zero-knowledge suffices in the context of public-coin protocols which can be transformed into zkSNARKs via the Fiat-Shamir heuristic.

Definition 2 (Honest-Verifier Statistical Zero-Knowledge). *The Polynomial IOP $(\mathsf{I}, \mathsf{P}, \mathsf{V})$ for the indexed relation $\mathcal{R}$ is statistical honest-*

verifier zero-knowledge if for any $(\mathbb{i}, \mathbb{x}, \mathbb{w}) \in \mathcal{R}$ *there exists a PPT algorithm* S *such that the statistical distance between the distributions of the following two random variables is bounded by a negligible value* ε*:*

$$\mathsf{tr} \leftarrow \mathsf{tr}\langle \mathsf{I}, \mathsf{P}, \mathsf{V} \rangle(\mathbb{i}, \mathbb{x}, \mathbb{w}) \quad \textit{and} \quad \mathsf{tr}' \leftarrow \mathsf{S}(\mathbb{i}, \mathbb{x})$$

where the distributions are over the randomnesses of the prover, the public coin, and the randomnesses of the simulator. We call S *the simulator that simulates the transcript of this protocol with statistical distance* ε*.*

3 Vector Oracle Model

We propose the *Vector Oracle* model where the verifier has oracle access to functionalities for checking the Hadamard- and inner-product relations between vectors. A protocol in the Vector Oracle model involves two parties, namely the prover and the verifier, together with a *vector oracle* $\mathcal{O}^{\boldsymbol{V}}$ that, intuitively, provides the following functionalities:

- $\mathcal{O}^{\boldsymbol{V}}$ maintains a set of vectors $\boldsymbol{V}$, each vector has size at most n.
- The prover may submit to $\mathcal{O}^{\boldsymbol{V}}$ arbitrary vectors of size at most n, and $\mathcal{O}^{\boldsymbol{V}}$ stores these vectors in $\boldsymbol{V}$.
- The verifier may also submit to $\mathcal{O}^{\boldsymbol{V}}$ vectors of size at most n, and $\mathcal{O}^{\boldsymbol{V}}$ stores these vectors in $\boldsymbol{V}$. However, the verifier vectors are limited to the following types:
  - sparse vector, i.e., the number of nonzero entries is $O(1)$;
  - power vector, i.e., of the form $\boldsymbol{\alpha}^n = (1, \alpha, \cdots, \alpha^{n-1})$ for $\alpha \in \mathbb{F}$;
  - a vector obtained by applying a constant number of operations to the vectors in $\boldsymbol{V}$, where the operations may include vector addition, scalar multiplication, right shifting, and concatenation.
- The verifier may ask $\mathcal{O}^{\boldsymbol{V}}$ to check Hadamard relations of the form $\boldsymbol{a} \circ \boldsymbol{b} = \boldsymbol{c} \circ \boldsymbol{d}$, or inner product relations of the form $\boldsymbol{a} \cdot \boldsymbol{b} = \boldsymbol{c} \cdot \boldsymbol{d}$, for arbitrary $(\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}, \boldsymbol{d}) \in \boldsymbol{V}^4$.

A Vector Oracle protocol is similar to an interactive protocol, with the following exceptions: (1) the prover does not send messages to the verifier directly but submits vectors to $\mathcal{O}^{\boldsymbol{V}}$ instead; (2) beside sending messages to the prover, the verifier may submit vectors to $\mathcal{O}^{\boldsymbol{V}}$, and query $\mathcal{O}^{\boldsymbol{V}}$ to check Hadamard- or inner-product relations; and (3) the output of this protocol is not decided by the verifier. Instead, the protocol outputs 1 if all the Hadamard- and inner-product checks are satisfied, and 0 otherwise.

Before we present a formalization of the Vector Oracle model, we introduce the notion *structured vectors*, which intuitively refer to vectors

with succinct representations. This notion covers both sparse vectors and power vectors. Formally, a vector $\boldsymbol{v}$ is structured if $f_{\boldsymbol{v}}(X)$ can be evaluated by a few field operations.

Definition 3 (Structured Polynomials and Vectors). *A polynomial $f(X)$ is structured if there exists an arithmetic circuit C over $\mathbb{F}$ that computes $f(X)$, where $|C| = O(\log(\deg(f(X))))$. A vector $\boldsymbol{v} \in \mathbb{F}^n$ is structured if $f_{\boldsymbol{v}}(X)$ is structured.*

In particular, the power vector $\boldsymbol{\gamma}^n$ is structured since the value of $f_{\boldsymbol{\gamma}^n}(X)$ can be efficiently evaluated by computing $\frac{(\gamma X)^n - 1}{\gamma X - 1}$ for $X \neq 1/\gamma$, or n for $X = 1/\gamma$. A sparse vector with a constant number of nonzero entries is also structured.

We observe that applying addition, scaling, and shifting operations to a vector $\boldsymbol{v}$ is equivalent to multiplying the polynomial $f_{\boldsymbol{v}}(X)$ by a sparse (thus structured) polynomial. This observation allows us to unify all the verifier submissions into a single type of query. Specifically, let $\boldsymbol{v}$ and $\boldsymbol{v}'$ be two vectors stored in $\boldsymbol{V}$, the verifier may specify two structured polynomials $\psi(X)$ and $\psi'(X)$, and submit to $\mathcal{O}^{\boldsymbol{V}}$ the coefficient vector of $f_{\boldsymbol{v}}(X) \cdot \psi(X) + f_{\boldsymbol{v}'}(X) \cdot \psi'(X)$. This query covers all the needed vector operations. Furthermore, the verifier may submit arbitrary structured vectors to $\mathcal{O}^{\boldsymbol{V}}$ by letting $\boldsymbol{v} = \mathbf{1}$[1] and $\psi'(X) = 0$.

We focus on public-coin protocols where the verifier messages are uniformly random. Formally, we define the verifier as a deterministic algorithm and let the verifier messages be read from a random tape shared by both parties.

Definition 4 (Public-Coin Preprocessing Vector Oracle Protocol). *A Vector Oracle protocol for an indexed relation $\mathcal{R}$ is a tuple $(\mathsf{I}, \mathsf{P}, \mathsf{V})$, where P is a probabilistic polynomial-time machine, I and V are two deterministic polynomial-time machines. Let R be a random tape consisting of uniformly random bits.*

Syntax. *For any $(\mathtt{i}, \mathtt{x}, \mathtt{w})$, this protocol decides whether $(\mathtt{i}, \mathtt{x}, \mathtt{w}) \in \mathcal{R}$ as follows. First, I takes input $\mathtt{i}$ and outputs vectors $\boldsymbol{v}_1, \cdots, \boldsymbol{v}_m$, together with some helping information $\mathtt{i}_P$ and $\mathtt{i}_V$ for P and V respectively, where $\mathtt{i}_V$ can be computed from $\mathtt{i}_P$. Let $r > 0$, $n_i \in [n], k_i \geq 0$ for $i \in [r]$. Let the initial state of P be $\mathsf{st}_0 := (\mathtt{i}_P, \mathtt{x}, \mathtt{w})$. Then, for i from 1 to r:*

- *Read a uniformly random string $\omega_i \in \{0,1\}^{k_i}$ from the tape R;*
- *P computes $(\boldsymbol{v}_{i+m}, \mathsf{st}_i) \leftarrow \mathsf{P}(\mathsf{st}_{i-1}, \omega_i)$ where $\boldsymbol{v}_{i+m} \in \mathbb{F}^{n_i}$.*

Next, V computes $(\mathcal{A}, \mathcal{H}, \mathcal{Q}) \leftarrow \mathsf{V}(\mathtt{i}_V, \omega_1, \cdots, \omega_r)$ where

– $\mathcal{A} = \{(\ell_i, \psi_i(X), \ell'_i, \psi'_i(X))\}_{i=1}^{s}$ *are the* Add *queries, where* $\ell_i, \ell'_i \in [m+r+i-1] \cup \{\perp\}$ *and* $\psi_i(X), \psi'_i(X) \in \mathbb{F}[X]$ *are structured polynomials such that* $\deg(\psi_i(X)) + |\boldsymbol{v}_{\ell_i}| \leq n$ *and* $\deg(\psi'_i(X)) + |\boldsymbol{v}_{\ell'_i}| \leq n$*, where* $\boldsymbol{v}_\perp$ *is defined to be* $\mathbf{1}$[1]*;*
– $\mathcal{H} = \{h_{ij}\}_{i\in[t_H], j\in[4]} \in [m+r+s]^{t_H \times 4}$ *are the* Hadamard *queries;*
– $\mathcal{I} = \{q_{ij}\}_{i\in[t_I], j\in[4]} \in [m+r+s]^{t_I \times 4}$ *are the* InnerProduct *queries.*

For $i \in [s]$*, let* $\boldsymbol{v}_{m+r+i}$ *be the coefficient vector of* $\psi_i(X) \cdot f_{\boldsymbol{v}_{\ell_i}}(X) + \psi'_i(X) \cdot f_{\boldsymbol{v}_{\ell'_i}}(X)$ *of size* $\max\{\deg(\psi_i(X)) + |\boldsymbol{v}_{\ell_i}|, \deg(\psi'_i(X)) + |\boldsymbol{v}_{\ell'_i}|\}$.

Finally, the protocol outputs 1 *if for every* $i \in [t_H]$, $\boldsymbol{v}_{h_{i1}} \circ \boldsymbol{v}_{h_{i2}} = \boldsymbol{v}_{h_{i3}} \circ \boldsymbol{v}_{h_{i4}}$*, and for every* $i \in [t_I]$, $\boldsymbol{v}_{q_{i1}} \cdot \boldsymbol{v}_{q_{i2}} = \boldsymbol{v}_{q_{i3}} \cdot \boldsymbol{v}_{q_{i4}}$*. Otherwise, this protocol outputs* 0*. The output of this protocol is denoted by* $\langle \mathsf{I}, \mathsf{P}, \mathsf{V}\rangle(\mathbb{i}, \mathbb{x}, \mathbb{w})$*, which is a random variable whose distribution is over the randomness of* P *and the random tape* R*.*

Efficiency*. We say this protocol uses* m *preprocessed vectors,* r *rounds (or prover vectors), vector size bound* n*,* s Add *queries (or verifier vectors),* t_H Hadamard *queries,* t_I InnerProduct *queries.*

Security*. The Vector Oracle protocol* $(\mathsf{I}, \mathsf{P}, \mathsf{V})$ *verifies a relation* $\mathcal{R}$ *with completeness error* ε_c *and soundness error* ε_s*, if for any* $(\mathbb{i}, \mathbb{x}, \mathbb{w}) \in \mathcal{R}$*,*

$$\Pr[\langle \mathsf{I}, \mathsf{P}, \mathsf{V}\rangle(\mathbb{i}, \mathbb{x}, \mathbb{w}) = 0] \leq \varepsilon_c$$

and for any $(\mathbb{i}, \mathbb{x}) \notin \mathcal{L}(\mathcal{R})$ *and any unbounded algorithm* P^**,*

$$\Pr[\langle \mathsf{I}, \mathsf{P}^*, \mathsf{V}\rangle(\mathbb{i}, \mathbb{x}, \perp) = 1] \leq \varepsilon_s.$$

The protocol has perfect completeness (soundness) if $\varepsilon_c = 0$ *(*$\varepsilon_s = 0$*).*

We do not define zero-knowledge for Vector Oracle protocols, as the syntax has already guaranteed that the verifier learns nothing except the validity of $(\mathbb{i}, \mathbb{x})$. However, to compile a Vector Oracle protocol to a zero-knowledge PIOP protocol, we need the protocol to have *q-wise independence* as in Definition 5.

Definition 5 (q-wise Independence). *Let* $(\mathsf{I}, \mathsf{P}, \mathsf{V})$ *be an* r*-round Vector Oracle protocol that uses* m *preprocessed vectors and* r *prover vectors* $\boldsymbol{v}_{m+1}, \boldsymbol{v}_{m+2}, \cdots, \boldsymbol{v}_{m+r}$*. This protocol is said to be* q*-wise independent, if for any* $\mathbb{i}, \mathbb{x} \in \mathcal{L}(\mathcal{R})$ *and public coin randomnesses* $\omega_1, \cdots, \omega_r$*, and for each* $i \in [m+1..m+r]$*,* $\boldsymbol{v}_i$ *either:*

– *contains at least* q *independent uniformly random elements over* $\mathbb{F}$*; or*

– *is deterministically and efficiently computable from the public information, namely* $\mathtt{i}$, $\mathtt{x}$ *and* $\omega_1, \cdots, \omega_r$.

The distribution is over the random coins of P.

Theorem 1 presents a compiler from Vector Oracle protocols to PIOP protocols. If the Vector Oracle protocol is q-wise independent for $q \geq 3$, then the resulting PIOP protocol is honest-verifier zero-knowledge.

Theorem 1. *Let the preprocessing Vector Oracle protocol* (VO.I, VO.P, VO.V) *verify the relation* $\mathcal{R}$ *with vector size bound* n, m *preprocessed vectors,* r *rounds,* s Add *queries,* t_H Hadamard *queries,* t_I InnerProduct *queries, completeness error* ε_c, *and soundness error* ε_s. *There exists a preprocessing PIOP protocol* $(\mathsf{I}, \mathsf{P}, \mathsf{V})$ *that verifies* $\mathcal{R}$ *with completeness error* ε_c *and soundness error* $\varepsilon_s + \frac{2n \cdot r + 3n + t_H + t_I - 2}{|\mathbb{F}| - 1}$. *This PIOP protocol has* m *preprocessed polynomial oracles,* $r + 1$ *rounds (online polynomial oracles), degree bound* $2n - 1$, *and at most* $3(m + r) + 2$ *evaluation queries at* 4 *distinct evaluation points.*

The protocol $(\mathsf{I}, \mathsf{P}, \mathsf{V})$ *is honest-verifier zero-knowledge if the Vector Oracle protocol* (VO.I, VO.P, VO.V) *is* q*-wise independent for* $q \geq 3$. *Specifically, there exists a PPT algorithm* S *that simulates the transcript of* $\langle \mathsf{I}, \mathsf{P}, \mathsf{V} \rangle(\mathtt{i}, \mathtt{x}, \mathtt{w})$ *with statistical distance* $\frac{4}{|\mathbb{F}| - 1}$ *for any* $(\mathtt{i}, \mathtt{x}, \mathtt{w}) \in \mathcal{R}$.

If either $t_H = 0$, *i.e., the Vector Oracle protocol does not use any* Hadamard *queries, or alternatively* $t_I = 0$, *i.e., not any* InnerProduct *queries, the honest-verifier zero-knowledge is achieved with* $q \geq 2$ *and statistical distance* $\frac{2}{|\mathbb{F}| - 1}$, *and the protocol uses at most* $2(m + r) + 2$ *evaluation queries at* 3 *distinct evaluation points.*

We provide a proof sketch as an overview of how this Vector Oracle compiler works and leave the complete proof in Appendix A due to the space limit. One of the key insights of the proof is inspired by (and modifies) the batched InnerProduct protocol of Claymore [20].

Proof (Sketch). We construct a PIOP protocol $(\mathsf{I}, \mathsf{P}, \mathsf{V})$ that verifies, with degree bound $2n - 1$, the same relation $\mathcal{R}$, and show that this PIOP protocol has the desired properties. The PIOP protocol works in the same way as $(\mathsf{VO.I}, \mathsf{VO.P}, \mathsf{VO.V})$ except that we replace the vectors with the corresponding polynomials. More specifically, for every vector $\boldsymbol{v}_i$ submitted to $\mathcal{O}^{\boldsymbol{V}}$ by either VO.I or VO.P, in the PIOP protocol I or P correspondingly sends the polynomial oracle $[f_{\boldsymbol{v}_i}(X)]$ to V.

When VO.V submits a vector by an Add query $(\ell_i, \psi_i(X), \ell'_i, \psi'_i(X))$, V simulates the corresponding polynomial oracle $[f_{\boldsymbol{v}_{m+r+i}}(X)]$ by querying

$[f_{\boldsymbol{v}_{\ell_i}}(X)]$ and $[f_{\boldsymbol{v}'_{\ell_i}}(X)]$ and evaluating $\psi_i(X) \cdot f_{\boldsymbol{v}_{\ell_i}}(X) + \psi'_i(X) \cdot f_{\boldsymbol{v}'_{\ell_i}}(X)$ locally. Recall that $\psi(X)$ and $\psi'(X)$ are structured polynomials, therefore V can evaluate them efficiently.

As a result, V has oracle access to all the polynomials $f_{\boldsymbol{v}_i}(X)$ for $i \in [m + r + s]$. To implement the restriction on the vector sizes, the $\mathsf{Hadamard}$ queries, and the $\mathsf{InnerProduct}$ queries, V needs to check the following types of statements:

- **The prover polynomials have the expected degrees.** To show that $\deg(f_{\boldsymbol{v}_{m+i}}(X)) < n_i$, P sends the polynomial $h(X) = X^{2n-n_i} \cdot f_{\boldsymbol{v}_{m+i}}(X)$ to V. The degree bound of the PIOP model guarantees that P only send polynomials of degree less than $2n$, thus ensuring $\deg(f_{\boldsymbol{v}_i}(X)) < n_i$.
- **The Hadamard relations.** To check that $\boldsymbol{v}_i \circ \boldsymbol{v}_j = \boldsymbol{v}_k \circ \boldsymbol{v}_\ell$, V sends a uniformly random $\alpha \in \mathbb{F}^*$ to P, and P replies with a polynomial $h(X) = X^{n-1} \cdot (f_{\boldsymbol{v}_i}(\alpha X^{-1}) \cdot f_{\boldsymbol{v}_j}(X) - f_{\boldsymbol{v}_k}(\alpha X^{-1}) \cdot f_{\boldsymbol{v}_\ell}(X))$. Note that if $\boldsymbol{v}_i \circ \boldsymbol{v}_j \neq \boldsymbol{v}_k \circ \boldsymbol{v}_\ell$ then the coefficient for X^{n-1} in $h(X)$ is nonzero with overwhelming probability for uniformly random α.
- **The inner product relations.** To check that $\boldsymbol{v}_i \cdot \boldsymbol{v}_j = \boldsymbol{v}_k \cdot \boldsymbol{v}_\ell$, P sends $h(X) = X^{n-1} \cdot (f_{\boldsymbol{v}_i}(X^{-1}) \cdot f_{\boldsymbol{v}_j}(X) - f_{\boldsymbol{v}_k}(X^{-1}) \cdot f_{\boldsymbol{v}_\ell}(X))$ to V. Note that $\boldsymbol{v}_i \cdot \boldsymbol{v}_j = \boldsymbol{v}_k \cdot \boldsymbol{v}_\ell$ if and only if the coefficient for X^{n-1} in $h(X)$ is zero.

Therefore, P needs to send $r + t_H + t_I$ polynomial oracles to V, denoted by $\{[h_i(X)]\}_{i=1}^{r+t_H+t_I}$ respectively, and show that for all $i \in [r + t_H + t_I]$, $h_i(X)$ is computed as specified above, and for all $i \in [r + 1..r + t_H + t_I]$, the coefficient for X^{n-1} in $h_i(X)$ is zero.

The key insight to boost the efficiency of the compiled PIOP protocol is to batch all these checks together. Specifically, V sends a random $\beta \in \mathbb{F}^*$ to P. Instead of sending the bunch of $h_i(X)$, P computes $h(X) := \sum_{i=1}^{r+t_H+t_I} \beta^{i-1} h_i(X)$ and sends $[h(X)]$ to V. V checks that $[h(X)]$ is computed as expected by querying $[h(X)]$ at a random point z, and compares the response with the result of evaluating $h(z)$ locally using α, β and querying the polynomial oracles $[f_{\boldsymbol{v}_i}(X)]$. After validating $[h(X)]$, V is convinced that for $i \in [r]$, $\deg(h_i(X))$ is bounded by $2n - 1$, because if not, $\deg(h(X))$ will be at least $2n$ with overwhelming probability.

It remains to show that for all $i \in [r + 1..r + t_H + t_I]$, the coefficient for X^{n-1} in $h_i(X)$ is zero. Since $n_i \leq n$ for $i \in [r]$, the coefficient for X^{n-1} in $h_i(X) := X^{2n-n_i} \cdot f_{\boldsymbol{v}_{m+i}}(X)$ are already guaranteed to be zero. Therefore, it suffices for V to check that the coefficient of X^{n-1} in $h(X)$

is zero, which fails with overwhelming probability if any $h_i(X)$ does not satisfy the condition.

To show that $h(X)$ has a zero X^{n-1} term, P does not send $h(X)$ to V directly, but instead sends a polynomial $\bar{h}(X)$ to V such that $h(X) = \bar{h}(\gamma \cdot X) - \gamma^{n-1} \cdot \bar{h}(X)$ where γ is an element with multiplicative order larger than $2n$. Note that such $\bar{h}(X)$ exists if and only if the coefficient for X^{n-1} in $h(X)$ is zero. Now V checks $h(X)$ is computed as expected as before, but by querying $[\bar{h}(X)]$ and computing $\bar{h}(\gamma \cdot X) - \gamma^{n-1} \cdot \bar{h}(X)$ instead of directly querying $[h(X)]$.

We have completed the description of the PIOP protocol. We refer to Appendix A for the strict security proof of this above protocol. Here we give an intuitive explanation for why it is honest-verifier zero-knowledge. Note that in the PIOP protocol, every polynomial $f_{\boldsymbol{v}_{m+i}}(X)$ is queried at most three times at z, z^{-1} and αz^{-1} respectively, and the $q \geq 3$ independent and uniform coefficients in $\boldsymbol{v}_i$ suffice to randomize the query responses, and the simulator samples them uniformly over $\mathbb{F}$. For the two queries to $\bar{h}(X)$ at $\gamma \cdot z$ and z, the simulator may sample $\bar{h}(z)$ uniformly over $\mathbb{F}$, and computes $\bar{h}(\gamma \cdot z)$ as $h(z) + \gamma^{n-1} \cdot \bar{h}(z)$, where $h(z)$ is evaluated according to the definition of $h(X)$ and using the sampled query responses from $f_{\boldsymbol{v}_{m+i}}(X)$. The uniformity of $\bar{h}(z)$ follows from the fact that P may set the coefficient for X^{n-1} in $\bar{h}(X)$ arbitrarily without affecting the identity $h(X) = \bar{h}(\gamma \cdot X) - \gamma^{n-1} \cdot \bar{h}(X)$. Setting this coefficient to a uniformly random element over $\mathbb{F}$ justifies the uniformity of $\bar{h}(z)$.

We conclude with the efficiency of the PIOP protocol. The protocol uses m preprocessed and $r+1$ online polynomial oracles, i.e., $[f_{\boldsymbol{v}_i}(X)]$ for $i \in [1..m+r]$ and $[\bar{h}(X)]$. Each $[f_{\boldsymbol{v}_i}(X)]$ is queried at most three times and $[\bar{h}(X)]$ is queried twice, so the total number of evaluation queries is at most $3(r+m)+2$, at four distinct points, i.e., $z, \alpha z^{-1}, z^{-1}, \gamma \cdot z$. If $\mathsf{Hadamard}$ (or $\mathsf{InnerProduct}$) query is never issued is the Vector Oracle protocol, the evaluation queries at αz^{-1} (or z^{-1}) will be unnecessary, and we are left with $2(r+m)+2$ evaluation queries at three distinct points. □

We remark that the number of evaluation queries in Theorem 1 is only a worst-case upper bound, assuming that every prover vector depends on the witness and is involved in all the $\mathsf{Hadamard}$ or $\mathsf{InnerProduct}$ queries. In practice, this number can be much smaller. We reduce the evaluation queries in our protocols by several optimizations introduced in Section 6.

Theorem 2 shows that PIOP protocols (not necessarily zero-knowledge) can also be compiled to q-wise independent Vector Oracle protocols. This

implies that the Vector Oracle protocols are sufficiently expressive to capture the same relations as PIOP protocols.

Theorem 2. *Let the preprocessing PIOP protocol* (PIOP.I, PIOP.P, PIOP.V) *verify the relation* $\mathcal{R}$ *with* m *preprocessed polynomial oracles,* r *rounds, polynomial degree bound* n*,* s *evaluation queries, completeness error* ε_c*, and soundness error* ε_s*. Assume that the* PIOP.V *is computed by an arithmetic circuit of size* V*. For any* $q \geq 0$*, there exists a* q*-wise independent Vector Oracle protocol* $(\mathsf{I}, \mathsf{P}, \mathsf{V})$ *that verifies* $\mathcal{R}$ *with* $r + 2$ *rounds (prover vectors),* m *preprocessed vectors,* s InnerProduct *queries,* 4 Hadamard *queries, completeness error* $\varepsilon_c + O((q + V)/|\mathbb{F}|)$ *and soundness error* $\varepsilon_c + O((q + V)/|\mathbb{F}|)$*.*

We leave the proof of Theorem 2 in Appendix B.

4 Verify Permutations and Sparse Linear Algebra

Now we present building blocks for the Vector Oracle protocols verifying the circuit-based constraint systems in the next section. These building blocks function as part of a complete protocol and are refered to as *subprotocols*. The subprotocols do not strictly follow the syntax of Definition 4, as they are designed to prove certain relationships between vectors already stored in the vector set $\boldsymbol{V}$ maintained by the vector oracle. However, the definition of completeness, soundness, and q-wise independence still apply to the subprotocols, by viewing all the executions before this subprotocol as the preprocessing procedure, the indices of the input vectors in $\boldsymbol{V}$ as the instance, and the concrete values of the input vectors as the witness.

At the core of these subprotocols is the Permute subprotocol, which checks two vectors are reordering of each other. The Permute subprotocol is partially inspired by the shuffle argument of Bayer, et al. [21]. The same insights also appear in the permutation check of PLONK [3] and the signature of correct computation by Sonic [5]. Permute leads to the CopyCheck subprotocol which checks particular elements in a vector are identical to each other. Based on CopyCheck, we develop the SparseMVP subprotocol for checking sparse matrix vector multiplications. Due to the space limit, we leave the proofs for all lemmas and theorems in Appendix C.

We start with a formal definition of the (batched) permutation.

Definition 6 (Permutation). *We say a bijection* $\sigma : [n] \to [n]$ *is a permutation over* $[n]$ *and denote the set of all permutations over* $[n]$ *by* $\Sigma([n])$*. The* ***permutation*** *of* $\boldsymbol{v} \in \mathbb{F}^n$ *by* σ *is* $\sigma(\boldsymbol{v}) := (\boldsymbol{v}_{[\sigma(i)]})_{i=1}^n$*, and the*

batched permutation *of the matrix* $\boldsymbol{V} = (\boldsymbol{v}_1, \cdots, \boldsymbol{v}_m) \in \mathbb{F}^{n \times m}$ *by* σ *is* $\sigma(\boldsymbol{V}) := (\sigma(\boldsymbol{v}_1), \cdots, \sigma(\boldsymbol{v}_m))$.

The following lemma characterizes the insights behind the Permute protocol.

Lemma 1. *Let* $\boldsymbol{A} = (\boldsymbol{a}_1, \cdots, \boldsymbol{a}_m), \boldsymbol{B} = (\boldsymbol{b}_1, \cdots, \boldsymbol{b}_m) \in \mathbb{F}^{n \times m}$. *If* $\boldsymbol{A}$ *and* $\boldsymbol{B}$ *are not batched permutations of each other, then for uniformly random* $\alpha_0, \alpha_1, \cdots, \alpha_m \in \mathbb{F}$, *the probability that*

$$\prod_{i=1}^{n} \left(\alpha_0 + \sum_{j=1}^{m} \alpha_j \boldsymbol{a}_{\boldsymbol{j}[i]} \right) = \prod_{i=1}^{n} \left(\alpha_0 + \sum_{j=1}^{m} \alpha_j \boldsymbol{b}_{\boldsymbol{j}[i]} \right)$$

is bounded by $\frac{n}{|\mathbb{F}|}$.

Before we apply Lemma 1 to construct the Permute protocol, we first introduce a tool to check that the product of elements in vector $\boldsymbol{a}$ is the same as the product of elements in vector $\boldsymbol{b}$.

4.1 Product Equality

Given inputs $\boldsymbol{a}$ and $\boldsymbol{b}$ of size m and n respectively, the ProductEq protocol checks that $\prod_{i=1}^{\ell} \boldsymbol{a}_{[i]} = \prod_{i=1}^{\ell} \boldsymbol{b}_{[i]}$ for some $\ell \leq \min\{m, n\}$. This protocol assumes that $\boldsymbol{a}$ and $\boldsymbol{b}$ contain no zero in their first ℓ elements.

The prover computes and submits the vector $\boldsymbol{r}$ to $\mathcal{O}^{\boldsymbol{V}}$ given by:

$$\boldsymbol{r} := (\boldsymbol{r}_{[i]})_{i=1}^{\ell} \quad \text{where} \quad \boldsymbol{r}_{[i]} := \prod_{j=1}^{i} \frac{\boldsymbol{a}_{[j]}}{\boldsymbol{b}_{[j]}}.$$

The verifier checks that: 1) the vector $\boldsymbol{r}$ is indeed computed by the above equation; 2) $\boldsymbol{r}_{[\ell]} = 1$ which is equivalent to $\prod_{i=1}^{\ell} \boldsymbol{a}_{[i]} = \prod_{i=1}^{\ell} \boldsymbol{b}_{[i]}$.

The first condition is equivalent to the following identities:

$$1 \cdot \boldsymbol{a}_{[1]} = \boldsymbol{r}_{[1]} \cdot \boldsymbol{b}_{[1]}, \quad \boldsymbol{r}_{[1]} \cdot \boldsymbol{a}_{[2]} = \boldsymbol{r}_{[2]} \cdot \boldsymbol{b}_{[2]}, \quad \cdots \quad \boldsymbol{r}_{[\ell-1]} \cdot \boldsymbol{a}_{[\ell]} = \boldsymbol{r}_{[\ell]} \cdot \boldsymbol{b}_{[\ell]}.$$

The verifier appends a dummy identity $(\boldsymbol{r}_{[\ell]} - 1) \cdot \boldsymbol{a}_{[\ell+1]} = 0 \cdot \boldsymbol{b}_{[\ell+1]}$ to these identities and summarize them in the Hadamard query

$$((1 \| \boldsymbol{r}) - (\boldsymbol{0}^{\ell} \| 1)) \circ \boldsymbol{a} = \boldsymbol{r} \circ \boldsymbol{b}. \tag{1}$$

The verifier checks the second condition, i.e., $\boldsymbol{r}_{[\ell]} = 1$, by another Hadamard query

$$(\boldsymbol{r} - (\boldsymbol{0}^{\ell-1} \| 1)) \circ (\boldsymbol{0}^{\ell-1} \| 1) = \boldsymbol{0} \tag{2}$$

Recall that we require the protocols to have q-wise independence, which means every prover vector contains q uniformly random elements. Therefore, the prover uniformly samples the randomizer $\boldsymbol{\delta} \xleftarrow{\$} \mathbb{F}^q$ and appends $\boldsymbol{\delta}$ to $\boldsymbol{r}$. Note that the identity (2) is not affected by appending $\boldsymbol{\delta}$. To avoid breaking the identity (1), the prover pads $\max\{m,n\} - \ell$ zeros to $\boldsymbol{r}$ before appending $\boldsymbol{\delta}$ to it.

This protocol is formalized in Algorithm 1. For clarity and convenience, we do not bother to formalize the protocol in rounds or expand the verifier queries explicitly.

Algorithm 1 ProductEq Protocol

Input: $\boldsymbol{a} \in \mathbb{F}^m, \boldsymbol{b} \in \mathbb{F}^n, \ell \in [\min\{m,n\}]$ where $\boldsymbol{a}_{[i]}, \boldsymbol{b}_{[i]} \neq 0$ for any $i \in [\ell]$
Check: $\prod_{i=1}^{\ell} \boldsymbol{a}_{[i]} = \prod_{i=1}^{\ell} \boldsymbol{b}_{[i]}$

1: P samples $\boldsymbol{\delta} \xleftarrow{\$} \mathbb{F}^q$;
2: P computes $\boldsymbol{r} = \left(\prod_{j=1}^{i} (\boldsymbol{a}_{[j]}/\boldsymbol{b}_{[j]})\right)_{i=1}^{\ell} \| \mathbf{0}^{\max\{m,n\}-\ell} \| \boldsymbol{\delta}$;
3: P submits $\boldsymbol{r}$ of size $\max\{m,n\} + q$ to $\mathcal{O}^{\boldsymbol{V}}$;
4: V queries $\mathcal{O}^{\boldsymbol{V}}$ to check that $((1\|\boldsymbol{r}) - (\mathbf{0}^{\ell}\|1)) \circ \boldsymbol{a} = \boldsymbol{r} \circ \boldsymbol{b}$;
5: V queries $\mathcal{O}^{\boldsymbol{V}}$ to check that $(\boldsymbol{r} - (\mathbf{0}^{\ell-1}\|1)) \circ (\mathbf{0}^{\ell-1}\|1) = \mathbf{0}$.

Theorem 3. *The* ProductEq *protocol in Algorithm 1 is perfectly complete, perfectly sound, and q-wise independent.*

4.2 Batched Permutation

Given input vectors $\{\boldsymbol{a}^{(i)}\}_{i=1}^m$, $\{\boldsymbol{b}^{(i)}\}_{i=1}^m$ and an integer ℓ where $\ell \leq |\boldsymbol{a}^{(i)}|$ and $\ell \leq |\boldsymbol{b}^{(i)}|$ for every $i \in [m]$, the subprotocol Permute checks that the matrices $\boldsymbol{A} = (\boldsymbol{a}^{(1)}_{[1..\ell]}, \cdots, \boldsymbol{a}^{(m)}_{[1..\ell]})$ and $\boldsymbol{B} = (\boldsymbol{b}^{(1)}_{[1..\ell]}, \cdots, \boldsymbol{b}^{(m)}_{[1..\ell]})$ are batched permutations of each other.

The verifier uniformly samples $\alpha_0, \alpha_1, \cdots, \alpha_m \xleftarrow{\$} \mathbb{F}$ and sends them to the prover. Let $\boldsymbol{a} = \alpha_0 \cdot \mathbf{1}^{\ell} + \sum_{i=1}^m \alpha_i \cdot \boldsymbol{a}^{(i)}$ and $\boldsymbol{b} = \alpha_0 \cdot \mathbf{1}^{\ell} + \sum_{i=1}^m \alpha_i \cdot \boldsymbol{b}^{(i)}$. By Lemma 1, if $\boldsymbol{A}$ and $\boldsymbol{B}$ are indeed batched permutations of each other, then we have $\prod_{i=1}^{\ell} \boldsymbol{a}_{[i]} = \prod_{i=1}^{\ell} \boldsymbol{b}_{[i]}$. Otherwise, the identity holds with negligible probability. With an overwhelming probability, $\boldsymbol{a}$ and $\boldsymbol{b}$ do not contain any zeros in their first ℓ elements. Therefore, it suffices that the prover and the verifier run the ProductEq protocol on inputs $\boldsymbol{a}$, $\boldsymbol{b}$ and ℓ. This protocol is formalized in Algorithm 2.

Algorithm 2 Permute Protocol

Input: $\{\boldsymbol{a}^{(i)}\}_{i=1}^m, \{\boldsymbol{b}^{(i)}\}_{i=1}^m \in \mathbb{F}^n, \ell \leq n$
Check: There exists $\sigma \in \Sigma([\ell])$ such that $\boldsymbol{a}^{(i)}_{[1..\ell]} = \sigma(\boldsymbol{b}^{(i)}_{[1..\ell]})$ for every $i \in [m]$.

1: V samples $\alpha_0, \alpha_1, \cdots, \alpha_m \xleftarrow{\$} \mathbb{F}^*$ and sends $\alpha_0, \alpha_1, \cdots, \alpha_m$ to P;
2: V submits $\boldsymbol{a} = \alpha_0 \cdot \mathbf{1} + \sum_{i=1}^m \alpha_i \cdot \boldsymbol{a}^{(i)}$ of size n;
3: V submits $\boldsymbol{b} = \alpha_0 \cdot \mathbf{1} + \sum_{i=1}^m \alpha_i \cdot \boldsymbol{b}^{(i)}$ of size n;
4: P and V run the protocol ProductEq with inputs $\boldsymbol{a}, \boldsymbol{b}, \ell$.

Theorem 4. *The* Permute *protocol in Algorithm 2 has completeness error* $\frac{\ell}{|\mathbb{F}|}$ *(or has perfect completeness in the case of* ProductEq*), soundness error* $\frac{2\ell}{|\mathbb{F}|-1}$ *(or* $\frac{\ell}{|\mathbb{F}|-1}$ *in the case of* ProductEq*), and q-wise independence.*

4.3 Copy Check

The above Permute protocol allows us to check that some elements in a vector are identical. Formally,

Definition 7 (Copy Condition). *Let n be an integer, $\Pi = \{S_1, \cdots, S_t\}$ be a partition over $[n]$. We say $\boldsymbol{v} \in \mathbb{F}^n$ satisfies the* ***copy condition*** *under partition Π, if Π partitions $\boldsymbol{v}$ into groups of identical elements, i.e., for any $k \in [t]$ and $i, j \in S_k$, $\boldsymbol{v}_{[i]} = \boldsymbol{v}_{[j]}$.*

Given a partition Π over $[n]$, the subprotocol CopyCheck verifies that the input vector $\boldsymbol{v} \in \mathbb{F}^n$ satisfies the copy condition of Π. First, the indexer finds a permutation σ over $[n]$ such that for any $i \neq j \in [n]$, i and j are partitioned in the same group of Π if and only if $\sigma^t(i) = j$ for some $t \in \mathbb{Z}$. In another word, the cycles of σ induces a partition, denoted by Π_σ, that is the same as Π. By the group theory, such σ exist for every partition and can be found in $O(n)$ time.

The indexer applies σ to an *identity* vector whose elements are different from each other, e.g., the power vector $\boldsymbol{\gamma}^n = (1, \gamma, \cdots, \gamma^{n-1})$ for some generator γ of the multiplicative group $\mathbb{F}^*$. Let the permuted vector be $\boldsymbol{\sigma} := (\gamma^{\sigma(i)-1})_{i=1}^n$. The indexer submits $\boldsymbol{\sigma}$ to $\mathcal{O}^{\boldsymbol{V}}$.

Next, the prover and the verifier execute the Permute protocol to check that $\{\boldsymbol{v}, \boldsymbol{\gamma}^n\}$ and $\{\boldsymbol{v}, \boldsymbol{\sigma}\}$ are batched permutations of each other, which is equivalent to $\sigma(\boldsymbol{v}) = \boldsymbol{v}$. Therefore, for any $i \neq j \in [n]$ partitioned in the same group of Π, $\sigma^k(i) = j$ implies $\boldsymbol{v}_{[i]} = \boldsymbol{v}_{[\sigma(i)]} = \cdots = \boldsymbol{v}_{[\sigma^k(i)]} = \boldsymbol{v}_{[j]}$, and $\boldsymbol{v}$ satisfies the copy condition of Π. This is formalized as the CopyCheck protocol in Algorithm 3.

Theorem 5. *The* CopyCheck *protocol in Algorithm 3 has completeness error* $\frac{\ell}{|\mathbb{F}|}$*, soundness error* $\frac{2\ell}{|\mathbb{F}|-1}$*, and q-wise independence.*

Algorithm 3 CopyCheck Protocol

Index: $\ell \in \mathbb{Z}^+$ and a partition Π over $[\ell]$
Input: $\boldsymbol{v} \in \mathbb{F}^n$ with $n \geq \ell$
Check: $\boldsymbol{v}_{[1..\ell]}$ satisfies the copy condition of Π.
Preprocessing:

1: I finds $\sigma \in \Sigma([\ell])$ such that $\Pi_\sigma = \Pi$;
2: Let γ be a generator of the multiplicative group $\mathbb{F}^*$;
3: I computes $\boldsymbol{\sigma} = (\gamma^{\sigma(i)-1})_{i=1}^{\ell}$;
4: I sends $\boldsymbol{\sigma}, \ell$ to P, sends γ, ℓ to V, and submits $\boldsymbol{\sigma}$ of size ℓ to $\mathcal{O}^{\boldsymbol{V}}$.

Online:

1: V submits $\boldsymbol{\gamma} = (\gamma^{i-1})_{i=1}^{\ell}$ of size ℓ;
2: P and V run the protocol Permute with inputs $\{\boldsymbol{v}, \boldsymbol{\gamma}\}, \{\boldsymbol{v}, \boldsymbol{\sigma}\}, \ell$.

4.4 Sparse Matrix Vector Product

Finally, we develop a protocol that verifies the matrix-vector product for sparse matrices. Formally, let $\boldsymbol{M} \in \mathbb{F}^{m \times n}$ be a sparse matrix that contains at most ℓ nonzero elements. Given the input vectors $\boldsymbol{a} \in \mathbb{F}^n$ and $\boldsymbol{b} \in \mathbb{F}^m$, the protocol SparseMVP checks that $\boldsymbol{Ma} = \boldsymbol{b}$. We first explain the insights behind this protocol.

Random Linear Combination. For uniformly random $\alpha \xleftarrow{\$} \mathbb{F}^*$, if we have $(\boldsymbol{\alpha}^m)^\mathsf{T}\boldsymbol{Ma} = \boldsymbol{\alpha}^m \cdot \boldsymbol{b}$, which is equivalent to $f_{\boldsymbol{Ma}}(\alpha) = f_{\boldsymbol{b}}(\alpha)$, we are confident that $\boldsymbol{Ma} = \boldsymbol{b}$ due to the Schwartz-Zippel lemma. Therefore, the verifier sends a uniformly random α to the prover, the prover computes $\boldsymbol{c} = (\boldsymbol{\alpha}^m)^\mathsf{T}\boldsymbol{M}$ and submits $\boldsymbol{c}$ to $\mathcal{O}^{\boldsymbol{V}}$, and the verifier checks that $\boldsymbol{c} \cdot \boldsymbol{a} = \boldsymbol{\alpha}^m \cdot \boldsymbol{b}$. If $\boldsymbol{c}$ is indeed computed as expected, the verifier is convinced that $\boldsymbol{Ma} = \boldsymbol{b}$.

It remains to show that $\boldsymbol{c}$ is indeed $(\boldsymbol{\alpha}^m)^\mathsf{T}\boldsymbol{M}$. By applying the idea of random linear combination again, the verifier sends another random element $\beta \xleftarrow{\$} \mathbb{F}^*$ to the prover. If $\boldsymbol{c} \cdot \boldsymbol{\beta}^n = (\boldsymbol{\alpha}^m)^\mathsf{T}\boldsymbol{M}\boldsymbol{\beta}^n$, by applying the Schwartz-Zippel lemma again, the verifier is confident that $\boldsymbol{c} = (\boldsymbol{\alpha}^m)^\mathsf{T}\boldsymbol{M}$.

Therefore, the problem boils down to showing that $(\boldsymbol{\alpha}^m)^\mathsf{T}\boldsymbol{M}\boldsymbol{\beta}^n$ is equal to the inner product $\boldsymbol{c} \cdot \boldsymbol{\beta}^n$.

Sparse Representation of Matrix. Since the matrix $\boldsymbol{M}$ has only ℓ nonzero entries, we represent $\boldsymbol{M}$ by the row indices $(\mathsf{row}_1, \cdots, \mathsf{row}_\ell) \in [m]^\ell$, the column indices $(\mathsf{col}_1, \cdots, \mathsf{col}_\ell) \in [n]^\ell$ and the nonzero entries

$\boldsymbol{v} \in \mathbb{F}^\ell$, such that for each $i \in [\ell]$, $\boldsymbol{M}_{[\mathsf{row}_i, \mathsf{col}_i]} = \boldsymbol{v}_{[i]}$. Note that

$$(\boldsymbol{\alpha}^m)^\mathsf{T} \boldsymbol{M} \boldsymbol{\beta}^n = \sum_{i,j=1}^{m,n} \boldsymbol{M}_{[i,j]} \cdot \alpha^i \cdot \beta^j = \sum_{i \in [\ell]} \alpha^{\mathsf{row}_i} \cdot \beta^{\mathsf{col}_i} \cdot \boldsymbol{v}_{[i]}.$$

This value can be viewed as the inner product between $\boldsymbol{v}$ and $(\alpha^{\mathsf{row}_i} \cdot \beta^{\mathsf{col}_i})_{i=1}^\ell$. The latter is the Hadamard product of $(\alpha^{\mathsf{row}_i})_{i=1}^\ell$ and $(\beta^{\mathsf{col}_i})_{i=1}^\ell$. The vector $\boldsymbol{v}$, depending only on the matrix, is submitted by the indexer. The prover submits the other two vectors. Specifically, the prover computes $\boldsymbol{r} = (\alpha^{\mathsf{row}_i})_{i=1}^\ell$, $\boldsymbol{s} = (\beta^{\mathsf{col}_i})_{i=1}^\ell$ and $\boldsymbol{t} = \boldsymbol{s} \circ \boldsymbol{t}$, and submits them to $\mathcal{O}^{\boldsymbol{V}}$. The verifier checks that $\boldsymbol{t} = \boldsymbol{r} \circ \boldsymbol{s}$ and $\boldsymbol{t} \cdot \boldsymbol{v} = \boldsymbol{c} \cdot \boldsymbol{\beta}^n$.

It remains to show that $\boldsymbol{r}$ and $\boldsymbol{s}$ are indeed the expected vectors.

Copy Check. To check that $\boldsymbol{r} = (\alpha^{\mathsf{row}_i})_{i=1}^\ell$, the verifier concatenates $\boldsymbol{r}$ with $\boldsymbol{\alpha}^m$ to obtain $\boldsymbol{u} = \boldsymbol{r} \| \boldsymbol{\alpha}^m \in \mathbb{F}^{\ell+m}$, and asks the prover to show that $\boldsymbol{u}_{[i]} = \boldsymbol{u}_{[\ell+\mathsf{row}_i]}$ for every $i \in [\ell]$. This is exactly checking that $\boldsymbol{u}$ satisfies the copy condition for the partition $\Pi = \{S_i = \{j : j \in [\ell], \mathsf{row}_j = i\} \cup \{\ell + i\}\}_{i \in [m]}$ over $[\ell + m]$. This partition depends only on the public matrix $\boldsymbol{M}$, so the indexer may preprocess it as in the $\mathsf{CopyCheck}$ protocol.

Combine Two Copy Checks. The vector $\boldsymbol{s}$ can be verified in exactly the same way as $\boldsymbol{r}$. However, we may combine these two copy checks into a single one. In more detail, let $\boldsymbol{u} = \boldsymbol{r} \| \boldsymbol{s} \| \boldsymbol{\alpha}^m \| \boldsymbol{\beta}^n$ and the partition be

$$\begin{aligned} \Pi = &\{\{j : j \in [\ell], \mathsf{row}_j = i\} \cup \{2\ell + i\}\}_{i \in [m]} \cup \\ &\{\{j + \ell : j \in [\ell], \mathsf{col}_j = i\} \cup \{2\ell + m + i\}\}_{i \in [n]}. \end{aligned}$$

Then $\boldsymbol{u}$ satisfying the copy condition of Π implies that $\boldsymbol{r} = (\alpha^{\mathsf{row}_i})_{i=1}^\ell$ and $\boldsymbol{s} = (\beta^{\mathsf{col}_i})_{i=1}^\ell$.

As another optimization, the prover combines $\boldsymbol{r}$ and $\boldsymbol{s}$ into $\boldsymbol{w} = \boldsymbol{r} \| \boldsymbol{s}$ and submits $\boldsymbol{w}$ instead of the original two vectors. The condition that $\boldsymbol{r} \circ \boldsymbol{s} = \boldsymbol{t}$ is instead checked by $\boldsymbol{0}^\ell \| \boldsymbol{t} = \boldsymbol{w} \circ (\boldsymbol{0}^\ell \| \boldsymbol{w})$.

The above insights are formalized as the $\mathsf{SparseMVP}$ protocol in Algorithm 4. The formal presentation restricts the matrix-vector product to the prefixes of $\boldsymbol{a}$ and $\boldsymbol{b}$ to allow them to contain randomizers. Note that the vectors $\boldsymbol{c}, \boldsymbol{r}, \boldsymbol{s}, \boldsymbol{t}$ are not randomized because they depend only on the public information $\boldsymbol{M}$ and α, β.

Theorem 6. *The* $\mathsf{SparseMVP}$ *protocol in Algorithm 4 has completeness error* $\frac{2\ell+m+n}{|\mathbb{F}|}$*, soundness error* $\frac{3n+3m+4\ell}{|\mathbb{F}|-1}$*, and* q*-wise independence.*

Algorithm 4 SparseMVP Protocol

Index: a sparse representation of a matrix $\boldsymbol{M} \in \mathbb{F}^{m \times n}$: $\{\mathsf{row}_i, \mathsf{col}_i\}_{i=1}^{\ell}$ and $\boldsymbol{v} \in \mathbb{F}^{\ell}$
Input: $\boldsymbol{a} \in \mathbb{F}^{n'}, \boldsymbol{b} \in \mathbb{F}^{m'}$ where $n' \geq n, m' \geq m$
Check: $\boldsymbol{M}\boldsymbol{a}_{[1..n]} = \boldsymbol{b}_{[1..m]}$
Preprocessing:

1: I invokes $\mathsf{CopyCheck.I}$ with input $2\ell + m + n$ and

$$\begin{aligned} \Pi = \{S_i = \{j : j \in [\ell], \mathsf{row}_j = i\} \cup \{2\ell + i\}\}_{i \in [m]} \cup \\ \{S_i = \{j + \ell : j \in [\ell], \mathsf{col}_j = i\} \cup \{2\ell + m + i\}\}_{i \in [n]}. \end{aligned}$$

2: I sends $m, n, \ell, \boldsymbol{v}$ to P, and m, n, ℓ to V, and submits $\boldsymbol{v}$ of size ℓ to $\mathcal{O}^{\boldsymbol{V}}$.

Online:

1: V samples $\alpha \xleftarrow{\$} \mathbb{F}^*$ and sends α to P;
2: P computes $\boldsymbol{c} = (\boldsymbol{\alpha}^m)^{\mathsf{T}}\boldsymbol{M}$ and submits $\boldsymbol{c}$ of size n to $\mathcal{O}^{\boldsymbol{V}}$;
3: V queries $\mathcal{O}^{\boldsymbol{V}}$ to check that $\boldsymbol{c} \cdot \boldsymbol{a} = \boldsymbol{\alpha}^m \cdot \boldsymbol{b}$;
4: V samples $\beta \xleftarrow{\$} \mathbb{F}^*$ and sends β to P;
5: P computes $\boldsymbol{w} = (\alpha^{\mathsf{row}_i})_{i=1}^{\ell} \,\|\, \left(\beta^{\mathsf{col}_i}\right)_{i=1}^{\ell}$, $\boldsymbol{t} = \boldsymbol{w}_{[1..\ell]} \circ \boldsymbol{w}_{[\ell+1..2\ell]}$;
6: P submits $\boldsymbol{w}$ of size 2ℓ and $\boldsymbol{t}$ of size ℓ to $\mathcal{O}^{\boldsymbol{V}}$;
7: P and V run the protocol $\mathsf{CopyCheck}$ with inputs $\boldsymbol{w} \| \boldsymbol{\alpha}^m \| \boldsymbol{\beta}^n$ and index Π;
8: V queries $\mathcal{O}^{\boldsymbol{V}}$ to check that $\boldsymbol{w} \circ (\boldsymbol{0}^{\ell} \| \boldsymbol{w}) = \boldsymbol{1}^{2\ell} \circ (\boldsymbol{0}^{\ell} \| \boldsymbol{t})$ and $\boldsymbol{v} \cdot \boldsymbol{t} = \boldsymbol{c} \cdot \boldsymbol{\beta}^n$.

5 Vector Oracle Protocols for Constraint Systems

Now we apply the building blocks in the previous sections to construct Vector Oracle protocols for verifying circuit-based constraint systems. We focus on three influential indexed relations:

- The R1CS (Rank-1 Constrait System). The zkSNARKs targeting this relation include Pinocchio [19], Groth16 [2], Aurora [10], Fractal [8], Marlin [4], Spartan [9].
- The HPR (Hadamard Product Relation). This relation was first proposed by Bootle, et al. [22]. The zkSNARKs for variations of this relation include BulletProof [17], Sonic [5], Claymore [20]. A recent lattice-based zkSNARK [23] also chooses this relation.
- The PLONK relation for fan-in-2 circuits. This relation is used solely in the PLONK [3] scheme. We propose a Vector Oracle protocol for a modified version of the PLONK relation, which we call POV (Permutations for Oecumenical Vector-oracle) following the same naming strategy of PLONK.

5.1 Vector Oracle Protocols for R1CS and HPR

We first present the Vector Oracle protocols for verifying the R1CS and the HPR relations defined in Equation (3) and (4). Since these relations

involve only the Matrix-vector product and the Hadamard product, their verifying protocols are straightforward compositions of the SparseMVP subprotocol and Hadamard queries. Note that the matrices in these relations are usually sparse in practice. However, our protocol also works even if they are dense.

$$\mathcal{R}_{\mathsf{R1CS}} = \left\{ \left(\begin{pmatrix} m, n, \ell \\ \boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C} \end{pmatrix}, \begin{pmatrix} \boldsymbol{x}, \\ \boldsymbol{w} \end{pmatrix} \right) \middle| \begin{array}{l} \boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C} \in \mathbb{F}^{m \times n} \\ \boldsymbol{x} \in \mathbb{F}^{\ell}, \boldsymbol{w} \in \mathbb{F}^{n-\ell-1} \\ (\boldsymbol{A}(1\|\boldsymbol{z})) \circ (\boldsymbol{B}(1\|\boldsymbol{z})) = \boldsymbol{C}(1\|\boldsymbol{z}) \\ \quad \text{where } \boldsymbol{z} = \boldsymbol{x}\|\boldsymbol{w} \end{array} \right\} \tag{3}$$

$$\mathcal{R}_{\mathsf{HPR}} = \left\{ \left(\begin{pmatrix} m, n, \ell \\ \boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}, \boldsymbol{d} \end{pmatrix}, \begin{pmatrix} \boldsymbol{x}, \\ \boldsymbol{w}_1, \boldsymbol{w}_2, \boldsymbol{w}_3 \end{pmatrix} \right) \middle| \begin{array}{l} \boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C} \in \mathbb{F}^{m \times n}, \boldsymbol{d} \in \mathbb{F}^{m} \\ \boldsymbol{x} \in \mathbb{F}^{\ell}, \boldsymbol{w}_1, \boldsymbol{w}_2, \boldsymbol{w}_3 \in \mathbb{F}^{n} \\ \boldsymbol{w}_1 \circ \boldsymbol{w}_2 = \boldsymbol{w}_3 \\ \boldsymbol{A}\boldsymbol{w}_1 + \boldsymbol{B}\boldsymbol{w}_2 + \boldsymbol{C}\boldsymbol{w}_3 + \boldsymbol{d} \\ \quad = \boldsymbol{x}\|\boldsymbol{0}^{m-\ell} \end{array} \right\}. \tag{4}$$

R1CS. Given an index of the R1CS relation, the indexer concatenates the matrices into $\boldsymbol{M} = (\boldsymbol{A}^{\mathsf{T}}\|\boldsymbol{0}^{n\times q}\|\boldsymbol{C}^{\mathsf{T}}\|\boldsymbol{B}^{\mathsf{T}})^{\mathsf{T}} \in \mathbb{F}^{(3m+q)\times n}$ and invokes the indexer of SparseMVP with $\boldsymbol{M}$. The prover samples $\boldsymbol{\delta}_1, \boldsymbol{\delta}_2 \xleftarrow{\$} \mathbb{F}^q$, computes $\boldsymbol{z} = \boldsymbol{x}\|\boldsymbol{w}\|\boldsymbol{\delta}_1$ and $\boldsymbol{y} = (\boldsymbol{M}(1\|\boldsymbol{z}_{[1..n-1]}))\|\boldsymbol{\delta}_2$, and submits $\boldsymbol{w}, \boldsymbol{y}$ to $\mathcal{O}^{\boldsymbol{V}}$. Note that $\boldsymbol{y}$ is the concatenation of $\boldsymbol{A}(1\|\boldsymbol{z})$, $\boldsymbol{0}^q$, $\boldsymbol{C}(1\|\boldsymbol{z})$, $\boldsymbol{B}(1\|\boldsymbol{z})$ and $\boldsymbol{\delta}_2$. As a result, the Hadamard relation required in R1CS becomes

$$\boldsymbol{y}_{[1..m]} \circ \boldsymbol{y}_{[2m+q+1..3m+q]} = \boldsymbol{y}_{[m+q+1..2m+q]}.$$

The verifier checks this relation by Hadamard-multiplying the shifted $\boldsymbol{y}$ with itself. Formally

$$\boldsymbol{y} \circ (\boldsymbol{0}^{2m+q}\|\boldsymbol{y}) = (\boldsymbol{0}^{m}\|\boldsymbol{y}) \circ (\boldsymbol{0}^{2m+q}\|\boldsymbol{1}^{m}).$$

The verifier then checks $\boldsymbol{y}_{[1..3m+q]} = \boldsymbol{M}\boldsymbol{z}_{[1..n-1]}$ by the SparseMVP protocol. The above procedure is formalized in the VCProof/R1CS protocol in Algorithm 5.

Theorem 7. *The* VCProof/R1CS *protocol in Algorithm 5 is a VIOP protocol for the relation* $\mathcal{R}_{\mathsf{R1CS}}$ *with completeness error* $\frac{2s+3m+n+q}{|\mathbb{F}|}$*, soundness error* $\frac{4s+9m+3n+3q}{|\mathbb{F}|-1}$*, and* q*-wise independence.*

HPR. Similarly, let $\boldsymbol{M} = (\boldsymbol{d}\|\boldsymbol{A}\|\boldsymbol{0}^{n\times q}\|\boldsymbol{C}\|\boldsymbol{B}) \in \mathbb{F}^{m\times(3n+q+1)}$, and the indexer preprocessed $\boldsymbol{M}$ as in SparseMVP. The prover samples $\boldsymbol{\delta} \xleftarrow{\$} \mathbb{F}^q$

Algorithm 5 VCProof/R1CS Protocol

Index: $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C} \in \mathbb{F}^{m \times n}, m, n, \ell$ where $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}$ have s nonzero entries in total
Input: $\boldsymbol{x} \in \mathbb{F}^{\ell}$
Witness: $\boldsymbol{w} \in \mathbb{F}^{n-\ell-1}$
Check: $(\boldsymbol{A}(1\|\boldsymbol{x}\|\boldsymbol{w})) \circ (\boldsymbol{B}(1\|\boldsymbol{x}\|\boldsymbol{w})) = \boldsymbol{C}(1\|\boldsymbol{x}\|\boldsymbol{w})$
Preprocessing:

1: I computes the sparse representation of $\boldsymbol{M} = (\boldsymbol{A}^{\mathsf{T}}\|\boldsymbol{0}^{n\times q}\|\boldsymbol{C}^{\mathsf{T}}\|\boldsymbol{B}^{\mathsf{T}})^{\mathsf{T}} \in \mathbb{F}^{(3m+q)\times n}$ of length s and invokes SparseMVP.I with this representation.

Online:

1: P samples $\boldsymbol{\delta}_1 \xleftarrow{\$} \mathbb{F}^q$;
2: P computes $\boldsymbol{u} = \boldsymbol{w}\|\boldsymbol{\delta}_1$, $\boldsymbol{y} = (\boldsymbol{M}(\boldsymbol{x}\|\boldsymbol{w}))\|\boldsymbol{\delta}_2$;
3: P submits $\boldsymbol{u}$ of size $n-\ell+q-1$ and $\boldsymbol{y}$ of size $3m+2q$ to $\mathcal{O}^{\boldsymbol{V}}$;
4: P and V run the protocol SparseMVP with inputs $1\|\boldsymbol{x}\|\boldsymbol{u}, \boldsymbol{y}$ and with the sparse representation of $\boldsymbol{M}$ as the index;
5: V queries $\mathcal{O}^{\boldsymbol{V}}$ to check that $\boldsymbol{y} \circ (\boldsymbol{0}^{2m+q}\|\boldsymbol{y}) = (\boldsymbol{0}^{m}\|\boldsymbol{y}) \circ (\boldsymbol{0}^{2m+q}\|\boldsymbol{1}^{m})$.

and computes $\boldsymbol{w} = (\boldsymbol{w}_1\|\boldsymbol{\delta}\|\boldsymbol{w}_3\|\boldsymbol{w}_2)$. Then the HPR relation is equivalent to $\boldsymbol{M}(1\|\boldsymbol{w}) = \boldsymbol{x}\|\boldsymbol{0}^{m-\ell}$ and $\boldsymbol{w}_{[1..n]} \circ \boldsymbol{w}_{[2n+q+1..3n+q]} = \boldsymbol{w}_{[n+q+1..2n+q]}$. The first is checked by the SparseMVP protocol, and the second by

$$\boldsymbol{w} \circ (\boldsymbol{0}^{2n+q}\|\boldsymbol{w}) = (\boldsymbol{0}^{n}\|\boldsymbol{w}) \circ (\boldsymbol{0}^{2n+q}\|\boldsymbol{1}^{n}).$$

The above procedure is formalized in the VCProof/HPR protocol in Algorithm 6.

Algorithm 6 VCProof/HPR Protocol

Index: $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C} \in \mathbb{F}^{m \times n}, \boldsymbol{d} \in \mathbb{F}^{m}, m, n, \ell$ where $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}, \boldsymbol{d}$ have s nonzero entries in total
Input: $\boldsymbol{x} \in \mathbb{F}^{\ell}$
Witness: $\boldsymbol{w}_1, \boldsymbol{w}_2, \boldsymbol{w}_3 \in \mathbb{F}^{n}$
Check: $\boldsymbol{A}\boldsymbol{w}_1 + \boldsymbol{B}\boldsymbol{w}_2 + \boldsymbol{C}\boldsymbol{w}_3 + \boldsymbol{d} = \boldsymbol{x}\|\boldsymbol{0}^{m-\ell}$ and $\boldsymbol{w}_1 \circ \boldsymbol{w}_2 = \boldsymbol{w}_3$
Preprocessing:

1: I computes the sparse representation of $\boldsymbol{M} = (\boldsymbol{d}\|\boldsymbol{A}\|\boldsymbol{0}^{n\times q}\|\boldsymbol{C}\|\boldsymbol{B}) \in \mathbb{F}^{m\times(3n+q+1)}$ of length s and invokes SparseMVP.I with this representation

Online:

1: P samples $\boldsymbol{\delta} \xleftarrow{\$} \mathbb{F}^q$;
2: P computes $\boldsymbol{w} = \boldsymbol{w}_1\|\boldsymbol{\delta}\|\boldsymbol{w}_3\|\boldsymbol{w}_2$;
3: P submits $\boldsymbol{w}$ of size $3n+q$ to $\mathcal{O}^{\boldsymbol{V}}$;
4: P and V run the protocol SparseMVP with inputs $1\|\boldsymbol{w}, \boldsymbol{x}\|\boldsymbol{0}^{m-\ell}$ and with the sparse representation of $\boldsymbol{M}$ as the index;
5: V queries $\mathcal{O}^{\boldsymbol{V}}$ to check that $\boldsymbol{w} \circ (\boldsymbol{0}^{2n+q}\|\boldsymbol{w}) = (\boldsymbol{0}^{n}\|\boldsymbol{w}) \circ (\boldsymbol{0}^{2n+q}\|\boldsymbol{1}^{n})$.

Theorem 8. *The* VCProof/HPR *protocol in Algorithm 6 is a VIOP protocol for the relation* $\mathcal{R}_{\mathsf{HPR}}$ *with completeness error* $\frac{2s+m+3n+q+1}{|\mathbb{F}|}$*, soundness error* $\frac{4s+3m+9n+3q+3}{|\mathbb{F}|-1}$*, and* q*-wise independence.*

5.2 The POV Protocol

The PLONK circuit is dramatically different from the previous two constraint systems. We introduce the POV relation that optimizes the PLONK relation for the Vector Oracle model, and refer interested readers to [3] for the details of the original PLONK.

We start with an intuitive explanation of how does a POV relation characterize the computations of a fan-in-two circuit. Let circuit C contain n gates, including n_a addition gates and n_m multiplication gates where $n = n_a + n_m$. Then this circuit C can be viewed as a set of constraints over the following variables:

- $\{x_i\}_{i=1}^{\ell}$: public inputs and outputs;
- $\{a_i\}_{i=1}^{3n_a}$: left and right inputs, and outputs of the addition gates;
- $\{m_i\}_{i=1}^{3n_m}$: left and right inputs, and outputs of the multiplication gates;
- $\{z_i\}_{i=1}^{k}$: the distinct constants involved in this circuit.

The circuit exerts the following types of constraints over these variables:

- addition constraint, i.e., $a_i + a_{i+n_a} = a_{i+2n_a}$ for $i \in [n_a]$, corresponding to the addition gates;
- multiplication constraint, i.e., $m_i \cdot m_{i+n_m} = m_{i+2n_m}$ for $i \in [n_m]$, corresponding to the multiplication gates;
- copy constraint, i.e., a collection of identities each requiring one variable to equal another variable, corresponding to the wires.

We collect the variables involved in this circuit into four vectors: $\boldsymbol{x} \in \mathbb{F}^{\ell}, \boldsymbol{a} \in \mathbb{F}^{3n_a}, \boldsymbol{m} \in \mathbb{F}^{3n_m}$ and $\boldsymbol{z} \in \mathbb{F}^{k}$. Therefore, the addition constraint becomes $\boldsymbol{a}_{[1..n_a]} + \boldsymbol{a}_{[n_a+1..2n_a]} = \boldsymbol{a}_{[2n_a+1..3n_a]}$ and the multiplication constraint becomes $\boldsymbol{m}_{[1..n_m]} \circ \boldsymbol{m}_{[n_m+1..2n_m]} = \boldsymbol{m}_{[2n_m+1..3n_m]}$.

For the copy constraint, we concatenate all the above vectors into:

$$\boldsymbol{u} = \boldsymbol{m}_{[1..n_m]} \| \boldsymbol{0}^q \| \boldsymbol{a} \| \boldsymbol{m}_{[2n_m+1..3n_m]} \| \boldsymbol{m}_{[n_m+1..2n_m]} \| \boldsymbol{x} \| \boldsymbol{z}$$

and let Π be the partition over $[3n+k+\ell+q]$ such that any two variables in $\boldsymbol{u}$ are connected by wire (directly or indirectly) if and only if they are in the same partition of Π. Moreover, for $i \in [n_m+1..n_m+q]$, i.e., positions corresponding to the q zeros, i is included in a single partition. Therefore,

the copy constraint is equivalent to $\boldsymbol{u}$ satisfying the copy condition of Π. The ordering of variables in $\boldsymbol{u}$ can be arbitrary, and we pick this particular ordering to permit the shift-and-multiply technique in our protocol.

Formally,

$$\mathcal{R}_{\mathsf{POV}} := \left\{ \left(\begin{pmatrix} n_a, n_m, \\ k, \ell \\ \Pi, \boldsymbol{z} \end{pmatrix} \right) \middle| \begin{array}{l} \Pi \text{ is a partition over } [3n + k + \ell + q] \\ \text{ where } n = n_a + n_m \\ \boldsymbol{z} \in \mathbb{F}^k, \boldsymbol{a} \in \mathbb{F}^{3n_a}, \boldsymbol{m} \in \mathbb{F}^{3n_m}, \boldsymbol{x} \in \mathbb{F}^\ell \\ \boldsymbol{a}_{[1..n_a]} + \boldsymbol{a}_{[n_a+1..2n_a]} = \boldsymbol{a}_{[2n_a+1..3n_a]} \\ \boldsymbol{m}_{[1..n_m]} \circ \boldsymbol{m}_{[n_m+1..2n_m]} = \boldsymbol{m}_{[2n_m+1..3n_m]} \\ \boldsymbol{m}_{[1..n_m]} \| \boldsymbol{0}^q \| \boldsymbol{a} \| \boldsymbol{m}_{[2n_m+1..3n_m]} \| \boldsymbol{m}_{[n_m+1..2n_m]} \| \boldsymbol{x} \| \boldsymbol{z} \\ \quad \text{satisfies the copy condition of } \Pi \end{array} \right\} \tag{5}$$

Now we explain the VCProof/POV protocol that verifies the above relation. First, the indexer preprocesses Π as in the CopyCheck protocol. Then the indexer submits $\boldsymbol{z}$ to $\mathcal{O}^{\boldsymbol{V}}$. Next, the prover samples $\boldsymbol{\delta} \xleftarrow{\$} \mathbb{F}^q$ and assembles the vector:

$$\boldsymbol{v} = \boldsymbol{m}_{[1..n_m]} \| \boldsymbol{\delta} \| \boldsymbol{a} \| \boldsymbol{m}_{[2n_m+1..3n_m]} \| \boldsymbol{m}_{[n_m+1..2n_m]}.$$

Here we replace the q zeros in $\boldsymbol{v}$ by $\boldsymbol{\delta}$, which does not affect $\boldsymbol{v} \| \boldsymbol{x} \| \boldsymbol{z}$ satisfying the copy condition since Π places each of these q positions in a distinct group.

If we right shift $\boldsymbol{v}$ by $3n_a + 2n_m + q$ positions and multiply with $\boldsymbol{v}$ itself, we are effectively multiplying $\boldsymbol{m}_{[1..n_m]}$ and $\boldsymbol{m}_{[n_m+1..2n_m]}$. Then we check the multiplication constraints by comparing the result against $\boldsymbol{v}_{[3n_a+n_m+q+1..3n_a+2n_m+q]}$. Formally, the multiplication constraints are equivalent to

$$(\boldsymbol{0}^{3n_a+2n_m+q} \| \boldsymbol{v}) \circ \boldsymbol{v} = (\boldsymbol{0}^{3n_a+2n_m+q} \| \boldsymbol{1}^{n_m}) \circ (\boldsymbol{0}^{n_m} \| \boldsymbol{v}).$$

By a similar technique, the verifier checks the addition constraints by

$$((\boldsymbol{0}^{2n_a} \| \boldsymbol{v}) + (\boldsymbol{0}^{n_a} \| \boldsymbol{v}) - \boldsymbol{v}) \circ (\boldsymbol{0}^{n_m+2n_a+q} \| \boldsymbol{1}^{n_a}) = \boldsymbol{0}.$$

Finally, the prover and the verifier execute the CopyCheck subprotocol to check that $\boldsymbol{u} := \boldsymbol{v} \| \boldsymbol{x} \| \boldsymbol{z}$ satisfies the copy condition of Π. The above procedure is formalized in Algorithm 7.

Theorem 9. *The* VCProof/POV *protocol in Algorithm 7 is a VIOP protocol that validates the relation* $\mathcal{R}_{\mathsf{POV}}$ *with completeness error* $\frac{3n+k+\ell}{|\mathbb{F}|}$*, soundness error* $\frac{2(3n+k+\ell)}{|\mathbb{F}|-1}$*, and* q*-wise independence.*

Algorithm 7 VCProof/POV Protocol

Index: $\boldsymbol{z} \in \mathbb{F}^k$, n_a, n_m, ℓ, k, and partition Π over $[3n+k+\ell+q]$ where $n = n_a + n_m$
Instance: $\boldsymbol{x} \in \mathbb{F}^\ell$
Witness: $\boldsymbol{a} \in \mathbb{F}^{3n_a}, \boldsymbol{m} \in \mathbb{F}^{3n_m}$
Check: $\boldsymbol{a}_{[1..n_a]} + \boldsymbol{a}_{[n_a+1..2n_a]} = \boldsymbol{a}_{[2n_a+1..3n_a]}$,
$\boldsymbol{m}_{[1..n_m]} \circ \boldsymbol{m}_{[n_m+1..2n_m]} = \boldsymbol{m}_{[2n_m+1..3n_m]}$, and
$\boldsymbol{a}\|\boldsymbol{m}\|\boldsymbol{x}\|\boldsymbol{z}$ satisfies the copy condition of Π

Preprocessing:

1: I runs CopyCheck.I with index Π and $3n + m + \ell + q$;
2: I submits $\boldsymbol{z}$ to $\mathcal{O}^{\boldsymbol{V}}$, sends $\boldsymbol{z}, k, n_a, n_m, \ell$ to P, and k, n_a, n_m, ℓ to V.

Online:

1: P samples $\boldsymbol{\delta}_1, \boldsymbol{\delta}_2 \xleftarrow{\$} \mathbb{F}^q$;
2: P computes $\boldsymbol{v} = \boldsymbol{m}_{[1..n_m]}\|\boldsymbol{\delta}\|\boldsymbol{a}\|\boldsymbol{m}_{[2n_m+1..3n_m]}\|\boldsymbol{m}_{[n_m+1..2n_m]}$;
3: P submits $\boldsymbol{v}$ of size $3n + q$ to $\mathcal{O}^{\boldsymbol{V}}$;
4: V queries $\mathcal{O}^{\boldsymbol{V}}$ to check that $(\mathbf{0}^{n_m+2n_a+q}\|\mathbf{1}^{n_a}) \circ ((\mathbf{0}^{2n_a}\|\boldsymbol{v}) + (\mathbf{0}^{n_a}\|\boldsymbol{v}) - \boldsymbol{v}) = \mathbf{0}$;
5: V queries $\mathcal{O}^{\boldsymbol{V}}$ to check that $(\mathbf{0}^{3n_a+2n_m+q}\|\boldsymbol{v}) \circ \boldsymbol{v} = (\mathbf{0}^{3n_a+2n_m+q}\|\mathbf{1}^{n_m}) \circ (\mathbf{0}^{n_m}\|\boldsymbol{v})$;
6: P and V run the protocol CopyCheck with inputs $\boldsymbol{v}\|\boldsymbol{x}\|\boldsymbol{z}$ and the index Π and $3n + k + \ell + q$.

6 Efficiency Analysis

We compile the Vector Oracle protocols in the last section into PIOP protocols and compare them with other PIOP-based works in the number of polynomial oracles and evaluation queries. Then we transform these protocols into zkSNARKs and compare them with concurrent works in concrete efficiency. The comparison shows that the PIOP protocols and zkSNARKs in our work have significant advantages, which are attributed to various optimization techniques summarized below.

First, we extensively apply the vector concatenation technique to reduce the number of prover vectors, thus the number of polynomial commitments in the ultimate zkSNARK. This technique is embodied in combining the copy checks in SparseMVP protocol, and concatenating the witness vectors in VCProof/R1CS, VCProof/HPR and VCProof/POV. However, this technique also increases the maximal polynomial degrees, thus undermining the proving efficiency.

Another significant optimization is inspired by a technique in PLONK (attributed to Mary Maller in Section 4 of [3]) to reduce the number of evaluation queries in the PIOP protocol. This optimization exploits the fact that if the underlying polynomial commitment scheme is homomorphic, the verifier may linearly combine polynomial oracles into a new oracle and query this oracle directly. See Appendix A.1 for more details about this optimization. As a showcase, consider checking the identity

$f_1(\alpha X^{-1}) \cdot g_1(X) - f_2(\alpha X^{-1}) \cdot g_2(X) = h(X)$. The verifier first queries $f_1(X)$ and $f_2(X)$ with $\alpha \cdot x^{-1}$ and receives y_1 and y_2, then computes the polynomial oracle for $y_1 \cdot g_1(X) - y_2 \cdot g_2(X) - h(X)$ and checks that this polynomial evaluates to 0 at x. In this example, the optimization saves 2 queries compared to naively querying each of $g_1(X), g_2(X)$ and $h(X)$ and linearly combining the responses.

The above technique inspires another optimization in the Vector Oracle protocol level. Specifically, as a result of the polynomial merging, the right operands in all the Hadamard and InnerProduct queries will never be evaluated after compiled into a PIOP protocol. Therefore, we have carefully ordered the operands in the Hadamard and InnerProduct queries to place most of the prover vectors in the second operand, eliminating more than half of the evaluation queries and a distinct query point.

6.1 Comparison

We first compare our work with other PIOP protocols. Table 1 shows that the VCProof protocols have significant advantages in the number of polynomial oracles and evaluations, whereas the maximal polynomial degrees are almost doubled compared to Marlin and PLONK, which is expected as the result of the vector concatenation technique.

Table 1. Comparison between the VCProof protocols and other PIOPs. M, N are the numbers of rows and columns of the matrices in R1CS and HPR, C is the number of gates, C_a is the number of addition gates, ℓ is the number of public input/output values. Regarding Sonic, the total number of nonzero elements in each row of the three matrices is bounded by $m = 3$. The columns are split if the matrix is too dense, and the increase to the column number depends on the shape of the circuit, thus the big-O notation.

relation	protocol	# polynomials offline/online	# evaluations	# distinct points	# max degree
R1CS	Marlin [4]	9/12	18	3	$6S + 6$
	VCProof/R1CS	2/8	6	3	$6S + 3M + N + 7$
HPR	Sonic [5]	6/16	16	4	$O(N)$
	VCProof/HPR	2/7	6	3	$6S + M + 3N + 7$
Fan-in-2 Circuit	PLONK [3]	8/7	7	2	$3C$
	VCProof/POV	2/4	3	2	$5C + C_a + 4$

Next, we apply polynomial commitment schemes and the Fiat-Shamir heuristic to compile the VCProof protocols into zkSNARKs. We only consider polynomial commitment schemes that support homomorphic addition of commitments, in particular, the batched version of KZG [14] and

DARK [6], to apply the aforementioned concrete-level optimization. For reference, we present the KZG version of our zkSNARKs in Appendix D. Table 2 is a summary of the proving costs, verification costs and proof sizes of the KZG version of VCProof with other constant-verifier zkSNARKs. See Appendix E for a more comprehensive comparison that includes the DARK version of VCProof and more concurrent zkSNARKs.

Table 2. Concrete comparison of verifier costs and proof sizes between the constant-verifier zkSNARKs. S is the number of matrix nonzero entries, N is the number of matrix columns, C is the number of gates, C_a is the number of addition gates, and ℓ is the size of public input/output. For clarity, we neglected some small terms in the numbers for $\mathbb{G}_{1/2}$-exp.

relation	zkSNARK	proving cost		verification cost		proof size	
		$\mathbb{G}_{1/2}$-exp	$\mathbb{F}$-mul	Pairings	$\mathbb{G}_1$-exp	$\mathbb{G}_{1/2}$	$\mathbb{F}$
R1CS	Groth16 [2]	$3S+2N$	$O(S\log(S))$	3	ℓ	3	-
	Marlin [4]	$21S$	$O(S\log(S))$	2	-	13	21
	VCProof/R1CS	$48S+29N$	$O(S\log S)$	2	20	11	5
HPR	Sonic [5]	$273N$	$O(N\log(N))$	13	-	20	16
	VCProof/HPR	$46S+30N$	$O(S\log(S))$	2	19	10	5
Fan-in-2 Circuit	PLONK[1] [3]	$11C$	$O(C_m\log(C_m))$	2	16	7	7
	PLONK[2] [3]	$9C$	$O(C_m\log(C_m))$	2	18	9	7
	VCProof/POV	$27C+3C_a$	$O(C\log(C))$	2	12	6	2

The proving cost is dominated by the group exponentiations and the finite field operations in Fast-Fourier-Transforms (FFT). We only collect enough information to compare the number of group exponentiations. Although VCProof/HPR outperforms Sonic in this metric, the others increase this number by a factor of approximately 2-3 compared to the concurrent works, due to the doubled maximal polynomial degree. Regarding the verification cost, the VCProof/R1CS is slightly more expensive than Marlin, while VCProof/POV outperforms PLONK. Compared to Sonic, VCProof/HPR reduces the number of pairings, at the cost of 19 group exponentiations. Since the pairings are typically much more expensive than group exponentiations, the efficiency of VCProof/HPR is considerably better. Finally, the proof sizes of the VCProof zkSNARKs outperform the others in each relation (ignoring Groth16 which relies on per-circuit setups). In particular, the KZG version of VCProof/POV sets a new record of proof size with two field elements and six group elements.

7 Conclusion

We introduced the Vector Oracle model that assisted us in constructing zkSNARKs for various constraint systems. These zkSNARKs achieve shorter proofs and faster verifications than the state of the art, thanks to the techniques of vector concatenation and linear combination of polynomial commitments. Although it is possible to construct our protocols and to apply the optimizations directly in the language of PIOP, the simplicity brought by the Vector Oracle model plays an indispensable role in uncovering these constructions and optimizations.

The zkSNARKs proposed in this work prioritize shorter proofs and lower verification costs over the proving efficiency. We choose this priority because in typical scenarios, in particular blockchains, the prover is executed only once for each instance, while the proof is stored and verified a potentially infinite number of times. However, it is straightforward to modify our protocols to prioritize the prover by splitting the vectors, at the cost of several more group elements in the proof and more group exponentiations in the verifier. This flexibility in choosing different tradeoffs is another benefit brought by the simplicity and high modularity of VCProof.

Acknowledgement. We thank Alan Szepieniec for his valuable comments both to the writing and the technical part of this paper.

References

1. Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In *Innovations in Theoretical Computer Science 2012*, pages 326–349. ACM, 2012.
2. Jens Groth. On the size of pairing-based non-interactive arguments. In *Advances in Cryptology - EUROCRYPT 2016*, volume 9666 of *LNCS*, pages 305–326. Springer, 2016.
3. Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru. PLONK: permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. *IACR Cryptol. ePrint Arch.*, 2019:953, 2019.
4. Alessandro Chiesa, Yuncong Hu, Mary Maller, Pratyush Mishra, Noah Vesely, and Nicholas P. Ward. Marlin: Preprocessing zksnarks with universal and updatable SRS. In *Advances in Cryptology - EUROCRYPT 2020*, volume 12105 of *LNCS*, pages 738–768. Springer, 2020.
5. Mary Maller, Sean Bowe, Markulf Kohlweiss, and Sarah Meiklejohn. Sonic: Zero-knowledge snarks from linear-size universal and updateable structured reference strings. *IACR Cryptol. ePrint Arch.*, 2019:99, 2019.
6. Benedikt Bünz, Ben Fisch, and Alan Szepieniec. Transparent snarks from DARK compilers. In *Advances in Cryptology - EUROCRYPT 2020*, volume 12105 of *LNCS*, pages 677–706. Springer, 2020.

7. Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Scalable, transparent, and post-quantum secure computational integrity. *IACR Cryptol. ePrint Arch.*, 2018:46, 2018.
8. Alessandro Chiesa, Dev Ojha, and Nicholas Spooner. Fractal: Post-quantum and transparent recursive proofs from holography. In *Advances in Cryptology - EUROCRYPT 2020*, volume 12105 of *LNCS*, pages 769–793. Springer, 2020.
9. Srinath Setty. Spartan: Efficient and general-purpose zksnarks without trusted setup. In *Advances in Cryptology - CRYPTO 2020*, volume 12172 of *LNCS*, pages 704–737. Springer, 2020.
10. Eli Ben-Sasson, Alessandro Chiesa, Michael Riabzev, Nicholas Spooner, Madars Virza, and Nicholas P. Ward. Aurora: Transparent succinct arguments for R1CS. In *Advances in Cryptology - EUROCRYPT 2019*, volume 11476 of *LNCS*, pages 103–128. Springer, 2019.
11. Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *2014 IEEE Symposium on Security and Privacy, SP 2014*, pages 459–474. IEEE Computer Society, 2014.
12. Aztec, 2021. https://zk.money.
13. zksync, 2021. https://zksync.io.
14. Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. Constant-size commitments to polynomials and their applications. In *Advances in Cryptology - ASIACRYPT 2010*, volume 6477 of *LNCS*, pages 177–194. Springer, 2010.
15. Tiancheng Xie, Jiaheng Zhang, Yupeng Zhang, Charalampos Papamanthou, and Dawn Song. Libra: Succinct zero-knowledge proofs with optimal prover computation. In *Advances in Cryptology - CRYPTO 2019*, volume 11694 of *LNCS*, pages 733–764. Springer, 2019.
16. Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. Delegating computation: interactive proofs for muggles. In *Proceedings of the 40th Annual ACM Symposium on Theory of Computing*, pages 113–122. ACM, 2008.
17. Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Gregory Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *2018 IEEE Symposium on Security and Privacy, SP 2018*, pages 315–334. IEEE Computer Society, 2018.
18. Yupeng Zhang, Daniel Genkin, Jonathan Katz, Dimitrios Papadopoulos, and Charalampos Papamanthou. vram: Faster verifiable RAM with program-independent preprocessing. In *2018 IEEE Symposium on Security and Privacy, SP 2018*, pages 908–925. IEEE Computer Society, 2018.
19. Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. Pinocchio: Nearly practical verifiable computation. In *2013 IEEE Symposium on Security and Privacy, SP 2013*, pages 238–252. IEEE Computer Society, 2013.
20. Alan Szepieniec and Yuncong Zhang. Polynomial iops for linear algebra relations. *IACR Cryptol. ePrint Arch.*, 2020:1022, 2020.
21. Stephanie Bayer and Jens Groth. Efficient zero-knowledge argument for correctness of a shuffle. In *Advances in Cryptology - EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 263–280. Springer, 2012.
22. Jonathan Bootle, Andrea Cerulli, Pyrros Chaidos, Jens Groth, and Christophe Petit. Efficient zero-knowledge arguments for arithmetic circuits in the discrete log setting. In *Advances in Cryptology - EUROCRYPT 2016*, volume 9666 of *LNCS*, pages 327–357. Springer, 2016.

23. Carsten Baum, Jonathan Bootle, Andrea Cerulli, Rafaël del Pino, Jens Groth, and Vadim Lyubashevsky. Sub-linear lattice-based zero-knowledge arguments for arithmetic circuits. In *Advances in Cryptology - CRYPTO 2018*, volume 10992 of *LNCS*, pages 669–699. Springer, 2018.

A Proof of Theorem 1

First, we introduce the Schwartz-Zippel lemma, which is used intensively in all the proofs thereafter.

Lemma 2 (Schwartz-Zippel). *For a u-variate polynomial $f(X_1, \cdots, X_u)$ of total degree d over $\mathbb{F}$, let S be a finite subset of $\mathbb{F}$ and $z_1, \cdots, z_u$ be selected at random independently and uniformly from S. Then*

$$\Pr[f(z_1, \cdots, z_u) = 0] \leq \frac{d}{|S|}.$$

Now we present the proof of Theorem 1. We first construct the PIOP protocol $(\mathsf{I}, \mathsf{P}, \mathsf{V})$. Then we show that it has the claimed security and efficiency.

Construct the PIOP Protocol. The algorithm I computes $(\boldsymbol{v}_1, \cdots, \boldsymbol{v}_m, \mathbb{i}_P, \mathbb{i}_V) \leftarrow \mathsf{VO.I}(\mathbb{i})$. Then I computes $f_{\boldsymbol{v_i}}(X) = \sum_{j=1}^{|\boldsymbol{v}_i|} \boldsymbol{v_i}_{[j]} X^{j-1}$ for every $i \in [m]$. Finally, I sends $\mathbb{i}_P$ to P, and sends $\mathbb{i}_V$ to V together with the polynomial oracles $[f_{\boldsymbol{v_i}}(X)]$ for every $i \in [m]$.

The algorithms P and V work in the same way as $\mathsf{VO.P}$ and $\mathsf{VO.V}$, except that P produces polynomials instead of vectors. In more detail, after receiving $\mathbb{i}_P$, P computes $\mathsf{st}_0 = (\mathbb{i}_P, \mathbb{x}, \mathbb{w})$. Then for each i from 1 to r:

- If $k_i > 0$, samples $\omega_i \xleftarrow{\$} \{0,1\}^k_i$ uniformly randomly and sends them to P;
- P computes $(\boldsymbol{v}_{m+i}, \mathsf{st}_i) \leftarrow \mathsf{VO.P}(\mathsf{st}_{i-1}, \omega_i)$, and produces the polynomial $f_{\boldsymbol{v}_{m+i}}(X) \leftarrow \sum_{j=1}^{n_i} \boldsymbol{v_{m+i}}_{[j]} X^{j-1}$.

Next, V computes the queries $(\mathcal{A}, \mathcal{H}, \mathcal{Q}) \leftarrow \mathsf{VO.V}(\mathbb{i}_V, \omega_1, \cdots, \omega_r)$. Recall that we required $\mathbb{i}_V$ to be computable from $\mathbb{i}_P$, therefore P can also produce $(\mathcal{A}, \mathcal{H}, \mathcal{Q})$ by executing $\mathsf{VO.V}$.

Let the queries be $\mathcal{A} = \{(\ell_i, \psi_i(X), \ell'_i, \psi'_i(X))\}_{i=1}^s$, $\mathcal{H} = \{h_{ij}\}_{i\in[t_H], j\in[4]}$, and $\mathcal{I} = \{q_{ij}\}_{i\in[t_I], j\in[4]}$, exactly as in Definition 4. For each $i \in [s]$, P computes $f_{\boldsymbol{v}_{m+r+i}}(X) = \psi_i(X) \cdot f_{\boldsymbol{v}_{\ell_i}}(X) + \psi'_i(X) \cdot f_{\boldsymbol{v}_{\ell'_i}}(X)$. By the limit on the degree of $\psi(X)$ and $\psi'(X)$, we have $\deg(f_{\boldsymbol{v}_{m+r+i}}(X)) < n$. Note that since

$\psi_i(X)$ and $\psi'_i(X)$ are structured, P can compute the polynomial multiplications in $O(n)$ time, i.e., by applying the vector operations represented by $\psi(X)$ and $\psi'(X)$.

Now P and V check the Hadamard and inner product relations, i.e., $\boldsymbol{v}_{h_{i1}} \circ \boldsymbol{v}_{h_{i2}} = \boldsymbol{v}_{h_{i3}} \circ \boldsymbol{v}_{h_{i4}}$ for $i \in [t_H]$, and $\boldsymbol{v}_{q_{i1}} \cdot \boldsymbol{v}_{q_{i2}} = \boldsymbol{v}_{q_{i3}} \cdot \boldsymbol{v}_{q_{i4}}$ for $i \in [t_I]$. P and V also check the degree of the prover polynomial $f_{\boldsymbol{v}_{m+i}}(X)$ is less than n_i for every $i \in [r]$. In more detail, P and V proceed as follows.

1. Let $\gamma \in \mathbb{F}^*$ be a generator of the multiplication group $\mathbb{F}^*$.
2. V samples $\alpha, \beta \leftarrow \mathbb{F}^*$ and sends α, β to P.
3. P computes

$$\begin{aligned} h(X) = {} & X^{n-1} \sum_{i \in [t_1]} \beta^{i-1} \left(f_{\boldsymbol{v}_{h_{i1}}}(\alpha X^{-1}) f_{\boldsymbol{v}_{h_{i2}}}(X) - f_{\boldsymbol{v}_{h_{i3}}}(\alpha X^{-1}) f_{\boldsymbol{v}_{h_{i4}}}(X) \right) \\ & + X^{n-1} \sum_{i \in [t_2]} \beta^{t_1+i-1} \left(f_{\boldsymbol{v}_{q_{i1}}}(X^{-1}) f_{\boldsymbol{v}_{q_{i2}}}(X) - f_{\boldsymbol{v}_{q_{i3}}}(X^{-1}) f_{\boldsymbol{v}_{q_{i4}}}(X) \right) \\ & + \sum_{i \in [r]} \beta^{t_1+t_2+i-1} X^{2n-n_i} \cdot f_{\boldsymbol{v}_{m+i}}(X) \end{aligned} \tag{6}$$

4. P samples $\delta \xleftarrow{\$} \mathbb{F}$ and computes

$$\bar{h}(X) = \sum_{i=0}^{2n-1} \bar{h}_i X^i \text{ where } \bar{h}_i = \begin{cases} h_i \cdot (\gamma^i - \gamma^{n-1})^{-1}, & i \neq n-1 \\ \delta, & i = n-1 \end{cases} \tag{7}$$

5. P sends the polynomial oracle $[\bar{h}(X)]$ to V. Now the interaction between P and V finishes. V completes the rest of the work.
6. V samples $x \xleftarrow{\$} \mathbb{F}^*$ and let $z = \alpha \cdot x^{-1}$.
7. For $i \in [m+r]$, V evaluates $y_i^{(1)} = f_{\boldsymbol{v}_i}(z)$ by querying $[f_{\boldsymbol{v}_i}(X)]$.
   For $i \in [s]$, V evaluates $y_{m+r+i}^{(1)} = f_{\boldsymbol{v}_{m+r+i}}(z)$ by computing $\psi_i(z) \cdot y_{\ell_i}^{(1)} + \psi'_i(z) \cdot y_{\ell'_i}^{(1)}$ locally.
8. For $i \in [m+r]$, V evaluates $y_i^{(2)} = f_{\boldsymbol{v}_i}(x^{-1})$ by querying $[f_{\boldsymbol{v}_i}(X)]$.
   For $i \in [s]$, V evaluates $y_{m+r+i}^{(2)} = f_{\boldsymbol{v}_{m+r+i}}(x^{-1})$ by computing $\psi_i(x^{-1}) \cdot y_{\ell_i}^{(2)} + \psi'_i(x^{-1}) \cdot y_{\ell'_i}^{(2)}$ locally.

9. Let

$$g(X) = \gamma^{n-1} \cdot \bar{h}(X) + \sum_{i\in[t_1]} x^{n-1} \cdot \beta^{i-1} \left(y^{(1)}_{h_{i1}} \cdot f_{\boldsymbol{v}_{h_{i2}}}(X) - y^{(1)}_{h_{i3}} \cdot f_{\boldsymbol{v}_{h_{i4}}}(X) \right) + \sum_{i\in[t_2]} x^{n-1} \cdot \beta^{t_1+i-1} \left(y^{(2)}_{h_{i1}} \cdot f_{\boldsymbol{v}_{q_{i2}}}(X) - y^{(2)}_{h_{i3}} \cdot f_{\boldsymbol{v}_{q_{i4}}}(X) \right) + \sum_{i\in[r]} x^{2n-n_i} \cdot \beta^{t_1+t_2+i-1} \cdot f_{\boldsymbol{v}_{m+i}}(X). \tag{8}$$

   Note that $g(x) = \gamma^{n-1} \cdot \bar{h}(x) + h(x)$ because $g(X)$ is simply replacing part of $h(X)$, i.e., the $f_{\boldsymbol{v}_{h_{i1}}}(\alpha \cdot X^{-1})$, $f_{\boldsymbol{v}_{h_{i3}}}(\alpha \cdot X^{-1})$, $f_{\boldsymbol{v}_{h_{i1}}}(X^{-1})$, $f_{\boldsymbol{v}_{h_{i3}}}(X^{-1})$ and X^{2n-n_i}, by their evaluations at x.
10. For $i \in [m+r]$, V evaluates $y^{(3)}_i = f_{\boldsymbol{v}_i}(x)$ by querying $[f_{\boldsymbol{v}_i}(X)]$.
    For $i \in [s]$, V evaluates $y^{(3)}_{m+r+i} = f_{\boldsymbol{v}_{m+r+i}}(x)$ by computing $\psi_i(x) \cdot y^{(3)}_{\ell_i} + \psi'_i(x) \cdot y^{(3)}_{\ell'_i}$ locally.
11. V queries $(x, \gamma \cdot x)$ to $[\bar{h}(X)]$ and receives (h_1, h_2) respectively. Then V computes

$$g = g(x) = \gamma^{n-1} \cdot h_1 + \sum_{i\in[t_1]} x^{n-1} \cdot \beta^{i-1} \left(y^{(1)}_{h_{i1}} \cdot y^{(3)}_{h_{i2}} - y^{(1)}_{h_{i3}} \cdot y^{(3)}_{h_{i4}} \right) + \sum_{i\in[t_2]} x^{n-1} \cdot \beta^{t_1+i-1} \left(y^{(2)}_{h_{i1}} \cdot y^{(3)}_{q_{i2}} - y^{(2)}_{h_{i3}} \cdot y^{(3)}_{q_{i4}} \right) + \sum_{i\in[r]} x^{2n-n_i} \cdot \beta^{t_1+t_2+i-1} \cdot y^{(3)}_{m+i} \tag{9}$$

    If $h_2 = g$, V outputs 1. Otherwise, V outputs 0.

Completeness and Soundness. We show that the above defined PIOP protocol has the claimed completeness error and soundness error.

Consider the following sequence of statements:

$$(\mathbb{i}, \mathbb{x}) \in \mathcal{L}(\mathcal{R}) \tag{10}$$

$$\boldsymbol{v}_{h_{i1}} \circ \boldsymbol{v}_{h_{i2}} = \boldsymbol{v}_{h_{i3}} \circ \boldsymbol{v}_{h_{i4}} \quad \forall i \in [t_H] \tag{11}$$

$$\boldsymbol{v}_{q_{i1}} \cdot \boldsymbol{v}_{q_{i2}} = \boldsymbol{v}_{q_{i3}} \cdot \boldsymbol{v}_{q_{i4}} \quad \forall i \in [t_I] \tag{12}$$

$$\deg(f_{\boldsymbol{v}_{m+i}}(X)) < n_i \quad \forall i \in [r] \tag{13}$$

$$f_{\boldsymbol{v}_{h_{i1}} \circ \boldsymbol{v}_{h_{i2}}}(\alpha) - f_{\boldsymbol{v}_{h_{i3}} \circ \boldsymbol{v}_{h_{i4}}}(\alpha) = 0 \text{ for all } i \in [t_1] \tag{14}$$

The coefficient for X^{n-1} in the following polynomial is zero $\forall i \in [t_H]$

$$X^{n-1} \cdot (f_{\boldsymbol{v}_{h_{i1}}}(\alpha X^{-1}) \cdot f_{\boldsymbol{v}_{h_{i2}}}(X) - f_{\boldsymbol{v}_{h_{i3}}}(\alpha X^{-1}) \cdot f_{\boldsymbol{v}_{h_{i4}}}(X)) \quad (15)$$

The coefficient for X^{n-1} in the following polynomial is zero $\forall i \in [t_I]$

$$X^{n-1} \cdot (f_{\boldsymbol{v}_{q_{i1}}}(X^{-1}) \cdot f_{\boldsymbol{v}_{q_{i2}}}(X) - f_{\boldsymbol{v}_{q_{i3}}}(X^{-1}) \cdot f_{\boldsymbol{v}_{q_{i4}}}(X)) \quad (16)$$

$$\deg(h(X)) \leq 2n - 1 \quad (17)$$

$$\text{The coefficient for } X^{n-1} \text{ in } h(X) \text{ is zero} \quad (18)$$

$$\bar{h}_i = h_i \cdot (\gamma^i - \gamma^{d-1})^{-1} \qquad \forall i \in [0..2n-2] \backslash \{n-1\} \quad (19)$$

$$h_i = \bar{h}_i \cdot (\gamma^i - \gamma^{n-1}) \qquad \forall i \in [0..2n-2] \quad (20)$$

$$h(X) = \bar{h}(\gamma \cdot X) - \gamma^{n-1} \cdot \bar{h}(X) \quad (21)$$

$$\bar{h}(\gamma \cdot x) = \gamma^{n-1} \cdot \bar{h}(x) + h(x) \quad (22)$$

$$h_2 = g \quad (23)$$

Completeness follows from the fact that (10)(13) and (19) hold when the prover is honest, and the sequences of implications

- (10) $\Rightarrow$ (11)(12) that follows the completeness of $(\mathsf{VO.I}, \mathsf{VO.P}, \mathsf{VO.V})$ and fails with probability at most ε_c;
- (11) $\Rightarrow$ (14) $\Rightarrow$ (15) where the second implication follows from the fact that the constant term of

  $$f_{\boldsymbol{v}_{h_{i1}}}(\alpha X^{-1}) \cdot f_{\boldsymbol{v}_{h_{i2}}}(X) - f_{\boldsymbol{v}_{h_{i3}}}(\alpha X^{-1}) \cdot f_{\boldsymbol{v}_{h_{i4}}}(X)$$

  is $\sum_{j=1}^{n}(\boldsymbol{v}_{\boldsymbol{h_{i1}}[j]} \cdot \boldsymbol{v}_{\boldsymbol{h_{i2}}[j]} - \boldsymbol{v}_{\boldsymbol{h_{i3}}[j]} \cdot \boldsymbol{v}_{\boldsymbol{h_{i4}}[j]}) \cdot \alpha^{j-1}$ which is exactly the left hand side of (14);
- (12) $\Rightarrow$ (16) that follows from the fact that the constant term of

  $$f_{\boldsymbol{v}_{q_{i1}}}(X^{-1}) \cdot f_{\boldsymbol{v}_{q_{i2}}}(X) - f_{\boldsymbol{v}_{q_{i3}}}(X^{-1}) \cdot f_{\boldsymbol{v}_{q_{i4}}}(X)$$

  is $\sum_{j=1}^{n}(\boldsymbol{v}_{\boldsymbol{q_{i1}}[j]} \cdot \boldsymbol{v}_{\boldsymbol{q_{i2}}[j]} - \boldsymbol{v}_{\boldsymbol{q_{i3}}[j]} \cdot \boldsymbol{v}_{\boldsymbol{q_{i4}}[j]})$ which is exactly $\boldsymbol{v}_{q_{i1}} \cdot \boldsymbol{v}_{q_{i2}} - \boldsymbol{v}_{q_{i3}} \cdot \boldsymbol{v}_{q_{i4}}$;
- (13) $\Rightarrow$ (17) that follows from the definition of $h(X)$;
- (15)(16) $\Rightarrow$ (18) that follows from the definition of $h(X)$ and the fact that $n_i \leq n$ for every $i \in [r]$;
- (18)(19) $\Rightarrow$ (20) $\Rightarrow$ (21) $\Rightarrow$ (22) $\Rightarrow$ (23) where the last implication follows from $g(x) = \gamma^{n-1} \cdot \bar{h}(x) + h(x)$.

Therefore, the completeness error of the protocol is ε_c.

For the soundness error, note that the verifier accepts if and only if (17) and (23) holds. The soundness follows from the following sequences of implications:

- (23) $\Rightarrow$ (22) $\Rightarrow$ (21) where the second implication follows from the Schwartz-Zippel Lemma and fails with probability at most $\frac{2(n-1)}{|\mathbb{F}|-1}$;
- (21) $\Rightarrow$ (20) $\Rightarrow$ (18) $\Rightarrow$ (16)(15) where the last implication follows from the fact that $n_i \leq n$ for $i \in [r]$ and the Schwartz-Zippel Lemma, and fails with probability at most $\frac{t_1+t_2}{|\mathbb{F}|-1}$;
- (17) $\Rightarrow$ (13) that follows from the Schwartz-Zippel lemma and fails with probability at most $\frac{2n\cdot r}{|\mathbb{F}|-1}$;
- (16)(15)(13) $\Rightarrow$ (14)(12);
- (14) $\Rightarrow$ (11) that follows from the Schwartz-Zippel Lemma and fails with probability at most $\frac{n}{|\mathbb{F}|-1}$
- (12)(11) $\Rightarrow$ (10) that follows from the soundness of ($\mathsf{VO.I}$, $\mathsf{VO.P}$, $\mathsf{VO.V}$) and fails with probability at most ε_s.

Therefore, the event that (23) $\not\Rightarrow$ (10) happens with probability $\varepsilon_s + \frac{2n\cdot r+3n+t_1+t_2-2}{|\mathbb{F}|-1}$, which is the soundness error of $(\mathsf{I}, \mathsf{P}, \mathsf{V})$.

Honest-Verifier Statistical Zero-Knowledge. Assuming that the VIOP protocol ($\mathsf{VO.I}$, $\mathsf{VO.P}$, $\mathsf{VO.V}$) is q-wise independent for $q \geq 3$, we show that the PIOP protocol $(\mathsf{I}, \mathsf{P}, \mathsf{V})$ is honest-verifier zero-knowledge as follows. Recall that q-wise independence requires for every $i \in [r]$, $\boldsymbol{v}_{m+i}$ contains at least q uniformly random independent elements or is deterministically computable from the public information.

We construct a simulator S that given $\mathbb{i}, \mathbb{x}$ samples the verifier view with negligible statistical distance. The verifier view contains the following values: the verifier messages, i.e., $\alpha_{i1}, \cdots, \alpha_{in_i}$ for i from 1 to r and α, x, β, and the responses from the evaluation queries, i.e., $y_i^{(1)} := f_{\boldsymbol{v}_i}(\alpha \cdot x^{-1})$, $y_i^{(2)} := f_{\boldsymbol{v}_i}(x^{-1})$, $y_i^{(3)} := f_{\boldsymbol{v}_i}(x)$ for every $i \in [m+r]$, and $h_1 = \bar{h}(x), h_2 = \bar{h}(\gamma \cdot x)$.

The simulator S executes I and V with the inputs $\mathbb{i}$ and $\mathbb{x}$ to simulate a run of the protocol without the prover and computes everything it can without the witness. The rest of the verifier view that cannot be directly computed from the public information are h_1, h_2 and $y_i^{(1)}, y_i^{(2)}, y_i^{(3)}$ for $i \in \mathcal{I}$ where $\mathcal{I} \subseteq [m+1..m+r]$ is the set of indices for prover vectors that depend on the witness. S samples $y_i^{(1)}, y_i^{(2)}, y_i^{(3)}$ for $i \in \mathcal{I}$ and h_1 uniformly randomly from $\mathbb{F}$, and computes $h_2 = g$ by Equation (9).

We show that the output of the above-defined S only has a negligible statistical difference from the verifier view. It suffices to show that $y_i^{(1)}, y_i^{(2)}, y_i^{(3)}$ for $i \in \mathcal{I}$ and h_1 in the real execution are uniformly random over $\mathbb{F}^{|\mathcal{I}|+1}$ independent of the rest of the verifier view. Consider the following Vandermonde matrix of size $3 \times n$.

$$\boldsymbol{X} = \begin{pmatrix} 1 & 1 & 1 \\ \alpha \cdot x^{-1} & x & x^{-1} \\ (\alpha \cdot x^{-1})^2 & x^2 & x^{-2} \\ \vdots & \vdots & \vdots \\ (\alpha \cdot x^{-1})^{n-1} & x^{(n-1)} & x^{-(n-1)} \end{pmatrix}$$

Every 3 rows of this matrix form an invertible sub-matrix except when $\alpha \cdot x, x$ and x^{-1} contain duplicates, which happens if $x^2 = 1$ or $\alpha \in \{1, x^{-2}\}$, whose probability is bounded by $\frac{4}{|\mathbb{F}|-1}$.

For any $i \in [r]$, $(y_i^{(1)}, y_i^{(2)}, y_i^{(3)}) = \boldsymbol{v}_{m+i}^{\mathsf{T}} \boldsymbol{X}$. By the q-wise independence, $\boldsymbol{v}_{m+i}$ contains $q \geq 3$ uniformly random elements, $(y_i^{(1)}, y_i^{(2)}, y_i^{(3)})$ is also uniformly random over $\mathbb{F}^3$, except with probability $\frac{4}{|\mathbb{F}|-1}$. Moreover, h_1 is uniform over $\mathbb{F}$ because $\bar{h}(X)$ contains the uniform and independent δ as its coefficient for X^{n-1}. In conclusion, the statistical difference between the output of S and the verifier view is at most $\frac{4}{|\mathbb{F}|-1}$.

Efficiency. The number of online polynomial oracles involved in this protocol is $r + 1$, i.e., a polynomial oracle for each round of the VIOP protocol plus $[\bar{h}(X)]$. The number of preprocessed polynomial oracles is the same as the number of preprocessed vectors, which is m. There are 4 distinct evaluation points, $\alpha \cdot x^{-1}, x, x^{-1}$ and $\gamma \cdot x$, and for each of the $r + m$ polynomial oracles sent by the prover or the indexer, the oracle is queried at most three times with $\alpha \cdot x^{-1}$, x and/or x^{-1}. Therefore, the total number of evaluation queries is at most $3(r+m)$ plus the two queries to $[\bar{h}(X)]$.

Restricted to Hadamard-Only or Inner-Product-Only. If the verifier in VIOP protocol $(\mathsf{VO.I}, \mathsf{VO.P}, \mathsf{VO.P})$ does not issue any $\mathsf{InnerProduct}$ queries (or alternatively any $\mathsf{Hadamard}$ queries), then in the compiled PIOP protocol $(\mathsf{I}, \mathsf{P}, \mathsf{V})$ the evaluation point x^{-1} (or $\alpha \cdot x^{-1}$) needs not be queried. Therefore, the total number of evaluation queries becomes $2(r + m) + 2$, and the number of distinct evaluation points becomes 3. The above analysis of zero-knowledge still holds except that the Vandermonde matrix $\boldsymbol{X}$ has only two columns. Therefore, each of the prover

vectors needs to contain only $q \geq 2$ randomizers to make $y_i^{(1)}, y_i^{(2)}$ uniform. The probability that $\boldsymbol{X}$ contains singular submatrix becomes $\frac{2}{|\mathbb{F}|-1}$.

A.1 Optimizations

The above-defined compiler may benefit from several optimizations. The most significant optimization exploits the additive homomorphism of the underlying polynomial commitment scheme that instantiates the polynomial oracles. The additive homomorphism allows the verifier to linearly combine existing polynomial oracles into a new oracle, and queries this oracle at the cost of a single query instead of simulating it using the original oracles. The PIOP model does not characterize the homomorphism of the underlying polynomial commitment scheme. Therefore, we did not apply this optimization in the formal proof.

We optimize the compiler as follows: the verifier computes the polynomial oracle of $g(X)$ in Equation (8) and evaluates $g = g(x)$ with a single query. In more detail, for $i \in [s]$, V computes the polynomial oracle for $\bar{f}_{\boldsymbol{v}_{m+r+i}}(X) := \psi_i(x) \cdot f_{\boldsymbol{v}_{\ell_i}}(X) + \psi_i'(x) \cdot f_{\boldsymbol{v}_{\ell_i'}}(X)$ by locally computing $\psi_i(x), \psi_i'(x)$ and linearly combining the oracles for $f_{\boldsymbol{v}_{\ell_i}}(X)$ and $f_{\boldsymbol{v}_{\ell_i'}}(X)$. Finally, V computes the oracle for $\bar{g}(X)$ with Equation (8) by linearly combining the oracles for every $f_{\boldsymbol{v}_{h_{ij}}}(X)$, $f_{\boldsymbol{v}_{q_{ij}}}(X)$ and $\bar{h}(X)$.

We remark that the resulting polynomial oracle is not an oracle for the original $g(X)$, but for another polynomial $\bar{g}(X)$ that satisfies $\bar{g}(x) = g(x)$, since the $\psi_i(X)$ and $\psi_i'(X)$ are replaced by their evaluations at x. However, this is not a problem since the goal of V is to evaluate $g(x)$ only.

The ability to linearly combine the polynomial oracles also allows another strategy that eliminates the evaluation point $\gamma \cdot x$ at the cost of one more polynomial oracle. This trace-off is worthwhile because it results in smaller zkSNARK proofs and also reduces the degree bound from $2n-1$ to $n-1$.

To understand how this strategy works, recall that we introduced the polynomial $\bar{h}(X)$ for showing that $h(X)$ has coefficient 0 for the X^{n-1} term. This purpose can be accomplished alternatively by splitting $h(X)$ into two polynomials $\bar{h}_1(X)$ and $\bar{h}_2(X)$ of degrees at most $n-2$ and $n-1$ respectively, such that $h(X) = \bar{h}_1(X) + X^n \cdot \bar{h}_2(X)$. P sends the polynomials $\bar{h}_1'(X) := X \cdot \bar{h}_1(X)$ and $\bar{h}_2(X)$ to V, who computes the polynomial oracle for $\bar{g}(X)$ as before, but this time the $\gamma^{n-1} \cdot \bar{h}(X)$ term in $\bar{g}(X)$ is replaced by $-x^{-1} \cdot \bar{h}_1'(X) - x^n \cdot \bar{h}_2(X)$. If $\bar{h}_1(X)$ and $\bar{h}_2(X)$ are correct, $g(x)$ should be 0, and V checks this by a single query. This optimization eliminates two query responses (i.e., $\bar{g}(x)$ and $\bar{h}(\gamma \cdot x)$) from

the zkSNARK proof together with one distinct evaluation point (i.e., $\gamma \cdot x$), and reduces the degree bound to $n-1$, at the cost of one more polynomial oracle.

B Proof of Theorem 2

Here we present the proof of Theorem 2. The idea is to do the opposite of the proof of Theorem 1, i.e., replace the polynomials with vectors.

The algorithm I computes $f^{(1)}(X), \cdots, f^{(m)}(X) \leftarrow \mathsf{PIOP.I}$ and submits their coefficient vectors $\boldsymbol{v}_1, \cdots, \boldsymbol{v}_m$ to $\mathcal{O}^{\boldsymbol{V}}$ for every $i \in [m]$. Finally, I sends $\mathbb{i}_P$, $\mathbb{i}_V$ and $\boldsymbol{v}_1, \cdots, \boldsymbol{v}_m$ to P, and $\mathbb{i}_V$ to V.

The algorithms P and V work in the same way as $\mathsf{PIOP.P}$ and $\mathsf{PIOP.V}$, except that whenever $\mathsf{PIOP.P}$ sends a polynomial oracle $[f^{(m+i)}(X)]$ of degree at most $n-1$ to $\mathsf{PIOP.V}$, P samples $\boldsymbol{\delta}_{m+i} \xleftarrow{\$} \mathbb{F}^q$ and computes $\boldsymbol{v}_{m+i}$ which is the concatenation of the coefficient vector of $f^{(m+i)}(X)$ of size n and $\boldsymbol{\delta}_{m+i}$.

Finally, V sends to P all the randomnesses that are needed to compute the evaluation queries, and they both compute $\{(z_{i1}, \cdots, z_{in_i})\}_{i=1}^{m+r}$. Then P computes $\{(y_{i1}, \cdots, y_{in_i})\}_{i=1}^{m+r} \leftarrow \{(f^{(i)}(z_{i1}), \cdots, f^{(i)}(y_{in_i}))\}_{i=1}^{m+r}$.

Since P knows all the inputs to $\mathsf{PIOP.V}$ (i.e., $\mathbb{i}_V$, $\mathbb{x}$) and all the verifier messages, P learns the state of $\mathsf{PIOP.V}$, denoted by st_V, when $\mathsf{PIOP.V}$ computes the final decision bit. Therefore, P and V can execute the $\mathsf{VCProof/POV}$ protocol in Algorithm 7 to show that

$$\mathsf{PIOP.V}(\mathsf{st}_V, \{(y_{i1}, \cdots, y_{in_i})\}_{i=1}^{m+r}) = 1 \tag{24}$$

using st_V as the public input $\boldsymbol{x}$, with the following differences: the preprocessed vectors are hard-coded into P and V, and submitted to $\mathcal{O}^{\boldsymbol{V}}$ by V instead of by I. This is possible since the preprocessed vectors depend only on the circuit of $\mathsf{PIOP.V}$. This incurs an overhead on V that is linear to the computation time of $\mathsf{PIOP.V}$.

During the execution of Algorithm 7, P sends to V a vector $\boldsymbol{v}$ that contains all the input/output of the gates in the circuit, including all of $\{(y_{i1}, \cdots, y_{in_i})\}_{i=1}^{m+r}$. Assume the positions of these values in $\boldsymbol{v}$ are $\{(\ell_{i1}, \cdots, \ell_{in_i})\}_{i=1}^{m+r}$. Note that these positions are determined by the circuit of $\mathsf{PIOP.V}$ and can be hard-coded into P and V. Then, for each $i \in [m+r]$ and $j \in [n_i]$, V checks $f^{(i)}(z_{ij}) = y_{ij}$ by an $\mathsf{InnerProduct}$ query for

$$\boldsymbol{v}_i \cdot (1, z_{ij}, \cdots, z_{ij}^{n-1}) = \boldsymbol{v} \cdot \boldsymbol{e}_{\ell_{ij}}$$

where $\boldsymbol{e}_{\ell_{ij}}$ is the unit vector consisting of all zeros except a single 1 at position ℓ_{ij}.

The completeness of the above protocol follows from those of the PIOP protocol and the VCProof/POV protocol by design. Therefore, the completeness error is $\varepsilon_c + O(V/|\mathbb{F}|)$. For soundness, note that if V accepts then $f^{(i)}(z_{ij}) = \boldsymbol{v}_i \cdot (1, z_{ij}, \cdots, z_{ij}^{n-1}) = y_{ij}$ for every $i \in [m+r]$ and $j \in [n_i]$. Therefore, by the soundness of the PIOP protocol, the probability that Equation (24) holds when $(\mathbb{i}, \mathbb{x})$ is invalid is bounded by ε_s. Finally, by the soundness of the VCProof/POV protocol, the probability that V accepts while (24) does not hold is bounded by $O(V/|\mathbb{F}|)$. By union bound, the soundness error of the VIOP protocol is $\varepsilon_s + O(V/|\mathbb{F}|)$.

The number of prover vectors is $r+2$, i.e., one vector for each polynomial oracle, and two vectors induced by the VCProof/POV protocol. The number of Hadamard queries is 4 brought by the VCProof/POV protocol. The number of InnerProduct queries is s, i.e., one for each evaluation query.

C Security Proofs for Vector Oracle Protocols

C.1 Proof of Lemma 1

We first introduce the following lemma.

Lemma 3. *Let $\boldsymbol{A}, \boldsymbol{B} \in \mathbb{F}^{n \times m}$ where $n < |\mathbb{F}|$. If for any $\boldsymbol{x} \in \mathbb{F}^m$, $\boldsymbol{Ax}$ and $\boldsymbol{Bx}$ are permutations of each other, then $\boldsymbol{A}, \boldsymbol{B}$ are batched permutations of each other.*

Proof. **Claim 1.** *For any nonzero vector $\boldsymbol{a} \in \mathbb{F}^m$, there are exactly $|\mathbb{F}|^{m-1}$ vectors $\boldsymbol{x} \in \mathbb{F}^m$ such that $\boldsymbol{x} \cdot \boldsymbol{a} = 0$.* To see this, note that if $\boldsymbol{a}_{[k]} \neq 0$, then for any $\boldsymbol{x}_{[1]}, \cdots, \boldsymbol{x}_{[k-1]}, \boldsymbol{x}_{[k+1]}, \cdots, \boldsymbol{x}_{[m]} \in \mathbb{F}$, $\boldsymbol{x}_{[k]} = -\boldsymbol{a}_{[k]}^{-1} \cdot \sum_{i \neq k} (\boldsymbol{a}_{[i]} \cdot \boldsymbol{x}_{[i]})$ is the only choice such that $\boldsymbol{x} \cdot \boldsymbol{a} = 0$. As a corollary, for any $\boldsymbol{a} \neq \boldsymbol{b} \in \mathbb{F}^m$, there are exactly $|\mathbb{F}|^{m-1}$ choices of $\boldsymbol{x}$ such that $\boldsymbol{a} \cdot \boldsymbol{x} = \boldsymbol{b} \cdot \boldsymbol{x}$.

We prove Lemma 3 by induction on n. Obviously, for $n = 1$, $\boldsymbol{A}$ and $\boldsymbol{B}$ are vectors, and $\boldsymbol{Ax}$ and $\boldsymbol{Bx}$ are permutations of each other if and only if they are identical. By Claim 1, if $\boldsymbol{Ax} = \boldsymbol{Bx}$ for all $\boldsymbol{x} \in \mathbb{F}^m$, we immediately have $\boldsymbol{A} = \boldsymbol{B}$.

Now we assume that Lemma 3 holds for $1, \cdots, n-1$, and prove the case for n. We claim that the first row of $\boldsymbol{A}$ is contained in the rows of $\boldsymbol{B}$. Otherwise, by Claim 1, there are at most $n \cdot |\mathbb{F}|^{m-1}$ vectors $\boldsymbol{x}$ such that $(\boldsymbol{Ax})_{[1]}$ is contained in $\boldsymbol{Bx}$. By $n < |\mathbb{F}|$, there exists at least one $\boldsymbol{x}$ such that $(\boldsymbol{Ax})_{[1]}$ is not contained in $\boldsymbol{Bx}$, contradiction to the assumption that $\boldsymbol{Ax}$ and $\boldsymbol{Bx}$ are permutations of each other for all $\boldsymbol{x}$. Therefore, the first row of $\boldsymbol{A}$ is contained in the rows of $\boldsymbol{B}$. If we remove the first row of $\boldsymbol{A}$

and the identical row in $\boldsymbol{B}$ and let the result matrices be $\boldsymbol{A}'$ and $\boldsymbol{B}'$, then $\boldsymbol{A}'\boldsymbol{x}$ and $\boldsymbol{B}'\boldsymbol{x}$ are still permutations of each other for every $\boldsymbol{x}$. By the induction assumption of $n-1$, $\boldsymbol{A}'$ and $\boldsymbol{B}'$ are batched permutations of each other, therefore $\boldsymbol{A}$ and $\boldsymbol{B}$ are batched permutations of each other. □

We recall Lemma 1 as follows.

Lemma 1. *Let $\boldsymbol{A} = (\boldsymbol{a}_1, \cdots, \boldsymbol{a}_m), \boldsymbol{B} = (\boldsymbol{b}_1, \cdots, \boldsymbol{b}_m) \in \mathbb{F}^{n\times m}$. If $\boldsymbol{A}$ and $\boldsymbol{B}$ are not batched permutations of each other, then for uniformly random $\alpha_0, \alpha_1, \cdots, \alpha_m \in \mathbb{F}$, the probability that*

$$\prod_{i=1}^{n}\left(\alpha_0 + \sum_{j=1}^{m}\alpha_j \boldsymbol{a}_{\boldsymbol{j}[i]}\right) = \prod_{i=1}^{n}\left(\alpha_0 + \sum_{j=1}^{m}\alpha_j \boldsymbol{b}_{\boldsymbol{j}[i]}\right)$$

is bounded by $\frac{n}{|\mathbb{F}|}$.

Proof. Assume that $\boldsymbol{A}, \boldsymbol{B}$ are not batched permutations of each other. Consider the multivariate polynomials

$$f(X_0, X_1, \cdots, X_m) = \prod_{i=1}^{n}\left(X_0 + \sum_{j=1}^{m}\boldsymbol{a}_{\boldsymbol{j}[i]} \cdot X_j\right)$$
$$g(X_0, X_1, \cdots, X_m) = \prod_{i=1}^{n}\left(X_0 + \sum_{j=1}^{m}\boldsymbol{b}_{\boldsymbol{j}[i]} \cdot X_j\right)$$

both of total degree n.

We claim that $f(X_0, X_1, \cdots, X_m) \neq g(X_0, X_1, \cdots, X_m)$. Otherwise, for any $\boldsymbol{x} \in \mathbb{F}^m$, $f(X_0, \boldsymbol{x})$ and $g(X_0, \boldsymbol{x})$ are the same polynomial with respect to X_0 and have the same roots. Note that the roots of $f(X_0, \boldsymbol{x})$ are exactly the elements in $\boldsymbol{A}\boldsymbol{x}$, and the roots of $g(X_0, \boldsymbol{x})$ are exactly the elements in $\boldsymbol{B}\boldsymbol{x}$. Therefore, the vectors $\boldsymbol{A}\boldsymbol{x}$ and $\boldsymbol{B}\boldsymbol{x}$ are permutations of each other. By Lemma 3, $\boldsymbol{A}$ and $\boldsymbol{B}$ are batched permutations of each other, contradiction to the assumption.

Therefore, $f(X_0, X_1, \cdots, X_m) \neq g(X_0, X_1, \cdots, X_m)$. By Schwartz-Zippel Lemma, for uniformly random $\alpha_0, \alpha_1, \cdots, \alpha_m \in \mathbb{F}$, the probability that $f(\alpha_0, \alpha_1, \cdots, \alpha_m) = g(\alpha_0, \alpha_1, \cdots, \alpha_m)$ is no more than $\frac{n}{|\mathbb{F}|}$. □

C.2 Security Proofs for the Protocols

Theorem 3. *The* ProductEq *protocol in Algorithm 1 is perfectly complete, perfectly sound, and q-wise independent.*

Proof. Consider the following statements:

$$\prod_{i=1}^{\ell} \boldsymbol{a}_{[i]} = \prod_{i=1}^{\ell} \boldsymbol{b}_{[i]} \tag{25}$$

$$\boldsymbol{r} = \left(\prod_{j=1}^{i} \boldsymbol{a}_{[i]} / \boldsymbol{b}_{[i]} \right)_{i=1}^{\ell} \| \mathbf{0}^{\max\{m,n\}-\ell} \| \boldsymbol{\delta} \tag{26}$$

$$\boldsymbol{r}_{[\ell]} = 1 \tag{27}$$

$$(\boldsymbol{r} - (\mathbf{0}^{\ell-1} \| 1)) \circ (\mathbf{0}^{\ell-1} \| 1) = \mathbf{0} \tag{28}$$

$$\boldsymbol{a}_{[1]} \cdot 1 = \boldsymbol{r}_{[1]} \cdot \boldsymbol{b}_{[1]} \text{ and } \forall i \in [2..\ell], \boldsymbol{a}_{[i]} \cdot \boldsymbol{r}_{[i-1]} = \boldsymbol{r}_{[i]} \cdot \boldsymbol{b}_{[i]}, \tag{29}$$

$$\boldsymbol{a} \circ ((1 \| \boldsymbol{r}) - (\mathbf{0}^{\ell} \| 1)) = \boldsymbol{r} \circ \boldsymbol{b} \tag{30}$$

The completeness follows from the fact that (25) and (26) hold when the prover is honest, and the sequences of implications (25)(26) $\Rightarrow$ (27) $\Rightarrow$ (28), (26) $\Rightarrow$ (29), and (26)(27)(29) $\Rightarrow$ (30).

The soundness follows from the fact that (28) and (30) hold when the verifier accepts, and the sequences of implications (28) $\Rightarrow$ (27), (30) $\Rightarrow$ (29), and (29)(27) $\Rightarrow$ (25).

The q-wise independence follows from the fact that $\boldsymbol{\delta}$ is uniformly random over $\mathbb{F}^q$ independent of all the other elements. □

Theorem 4. *The* Permute *protocol in Algorithm 2 has completeness error* $\frac{\ell}{|\mathbb{F}|}$ *(or has perfect completeness in the case of* ProductEq*), soundness error* $\frac{2\ell}{|\mathbb{F}|-1}$ *(or* $\frac{\ell}{|\mathbb{F}|-1}$ *in the case of* ProductEq*), and q-wise independence.*

Proof. Consider the following statements:

$$\boldsymbol{a}^{(i)}_{[1..\ell]} = \sigma(\boldsymbol{b}^{(i)}_{[1..\ell]}) \quad \forall i \in \{1, 2, \cdots, m\} \text{ for some } \sigma \in \Sigma([\ell]) \tag{31}$$

$$\prod_{i=1}^{\ell} \boldsymbol{a}_{[i]} = \prod_{i=1}^{\ell} \boldsymbol{b}_{[i]} \tag{32}$$

$$\mathsf{V} \text{ accepts in the subprotocol } \mathsf{ProductEq} \tag{33}$$

The completeness follows from the implications (31) $\Rightarrow$ (32) due to the linearity of σ, and (32) $\Rightarrow$ (33) that follows from the completeness of protocol ProductEq, which fails if $\boldsymbol{a}, \boldsymbol{b}$ contain zero. Since for any index i, for any randomizers $\alpha_1, \cdots, \alpha_m$, there exists at most one α_0 such that

$\alpha_0 + \sum_{j=1}^{m} \alpha_j \cdot \boldsymbol{a}_{[i]}^{(j)} = 0$, so the number of α_0's such that $\boldsymbol{a}, \boldsymbol{b}$ contains 0 is at most ℓ. Therefore, the completeness error is bounded by $\frac{\ell}{|\mathbb{F}|}$.

The soundness follows from the implications (33) $\Rightarrow$ (32) due to the soundness of protocol ProductEq, which fails if $\boldsymbol{a}$ or $\boldsymbol{b}$ contains zero, and (32) $\Rightarrow$ (31) which fails with probability at most $\frac{\ell}{|\mathbb{F}|-1}$ due to Lemma 1. Therefore, the soundness error of the Permute protocol is bounded by $\frac{2\ell}{|\mathbb{F}|-1}$.

The q-wise independence follows from the subprotocol ProductEq. □

Theorem 5. *The* CopyCheck *protocol in Algorithm 3 has completeness error* $\frac{\ell}{|\mathbb{F}|}$*, soundness error* $\frac{2\ell}{|\mathbb{F}|-1}$*, and* q*-wise independence.*

Proof. Consider the following statements:

$$\boldsymbol{v} \text{ satisfies the copy condition of } \Pi \tag{34}$$

$$\sigma(\boldsymbol{v}_{[1..\ell]}) = \boldsymbol{v}_{[1..\ell]} \tag{35}$$

$$\{\boldsymbol{v}_{[1..\ell]}, \boldsymbol{\gamma}\}, \{\boldsymbol{v}_{[1..\ell]}, \boldsymbol{\sigma}\} \text{ are batched permutations of each other} \tag{36}$$

$$\mathsf{V} \text{ accepts in the subprotocol Permute} \tag{37}$$

The completeness follows from the sequence of implications (34) $\Rightarrow$ (35) $\Rightarrow$ (36) $\Rightarrow$ (37) where the last implication follows from the completeness of the subprotocol Permute which fails with probability $\frac{\ell}{|\mathbb{F}|}$. Therefore, the completeness error of CopyCheck is $\frac{\ell}{|\mathbb{F}|}$.

The soundness follows from the implications (37) $\Rightarrow$ (36) due to the soundness of protocol Permute and fails with probability $\frac{2\ell}{|\mathbb{F}|-1}$, and (36) $\Rightarrow$ (35) because σ is the unique permutation such that $\sigma(\boldsymbol{\gamma}) = \boldsymbol{\sigma}$, and (35) $\Rightarrow$ (34). Therefore, the soundness error of the CopyCheck protocol is bounded by $\frac{2\ell}{|\mathbb{F}|-1}$.

The q-wise independence follows from that of the subprotocol Permute. □

Theorem 6. *The* SparseMVP *protocol in Algorithm 4 has completeness error* $\frac{2\ell+m+n}{|\mathbb{F}|}$*, soundness error* $\frac{3n+3m+4\ell}{|\mathbb{F}|-1}$*, and* q*-wise independence.*

Proof. Consider the following statements:

$$\boldsymbol{M}\boldsymbol{a}_{[1..n]} = \boldsymbol{b}_{[1..m]} \tag{38}$$

$$\boldsymbol{c} = (\boldsymbol{\alpha}^m)^{\mathsf{T}} \boldsymbol{M} \tag{39}$$

$$\boldsymbol{c} \cdot \boldsymbol{a} = \boldsymbol{\alpha}^m \cdot \boldsymbol{b} \tag{40}$$

$$\boldsymbol{w} = (\alpha^{\mathsf{row}_i})_{i=1}^{\ell} \| (\beta^{\mathsf{col}_i})_{i=1}^{\ell} \tag{41}$$

$$\mathsf{V} \text{ accepts in the subprotocol } \mathsf{CopyCheck} \tag{42}$$

$$\boldsymbol{t} = \boldsymbol{w}_{[1..\ell]} \circ \boldsymbol{w}_{[\ell+1..2\ell]} \tag{43}$$

$$\boldsymbol{w} \circ (\boldsymbol{0}^{\ell} \| \boldsymbol{w}) = \boldsymbol{0}^{\ell} \| \boldsymbol{t} \tag{44}$$

$$(\boldsymbol{\alpha}^m)^{\mathsf{T}} \boldsymbol{M} \boldsymbol{\beta}^n = \boldsymbol{t} \cdot \boldsymbol{v} \tag{45}$$

$$(\boldsymbol{\alpha}^m)^{\mathsf{T}} \boldsymbol{M} \boldsymbol{\beta}^n = \boldsymbol{c} \cdot \boldsymbol{\beta}^n \tag{46}$$

$$\boldsymbol{t} \cdot \boldsymbol{v} = \boldsymbol{c} \cdot \boldsymbol{\beta}^n \tag{47}$$

The completeness follows from the fact that if the prover is honest then all of (38)(39)(41) and (43) are true, and the following implications:

- (38)(39) ⇒ (40)
- (39) ⇒ (46)
- (41) ⇒ (42) that follows from the completeness of $\mathsf{CopyCheck}$ that fails with probability $\frac{2\ell+m+n}{|\mathbb{F}|}$
- (43) ⇒ (44)
- (41)(43) ⇒ (45)
- (45)(46) ⇒ (47)

Therefore, the completeness error is $\frac{2\ell+m+n}{|\mathbb{F}|}$.

The soundness follows from the fact that (40)(42)(44) and (47) hold if the verifier accepts, and the following implications:

- (42) ⇒ (41) that follows from the soundness of $\mathsf{CopyCheck}$ that fails with probability $\frac{2(2\ell+m+n)}{|\mathbb{F}|-1}$
- (44) ⇒ (43)
- (41)(43) ⇒ (45)
- (45)(47) ⇒ (46) ⇒ (39) where the second implication follows from the Schwartz-Zippel lemma and fails with probability $\frac{n}{|\mathbb{F}|-1}$
- (39)(40) ⇒ (38) that follows from the Schwartz-Zippel lemma and fails with probability $\frac{m}{|\mathbb{F}|}$

Therefore, the soundness error of the $\mathsf{SparseMVP}$ protocol is $\frac{3n+3m+4\ell}{|\mathbb{F}|-1}$.

The q-wise independence follows from the fact that all of $\boldsymbol{c}, \boldsymbol{w}, \boldsymbol{t}$ can be deterministically computed from the index and the verifier randomnesses, i.e., $\{\mathsf{row}_i, \mathsf{col}_i\}_{i=1}^{\ell}$ and α, β, and that the subprotocol $\mathsf{CopyCheck}$ is q-wise independent. □

Theorem 7. *The* VCProof/R1CS *protocol in Algorithm 5 is a VIOP protocol for the relation* $\mathcal{R}_{\mathsf{R1CS}}$ *with completeness error* $\frac{2s+3m+n+q}{|\mathbb{F}|}$*, soundness error* $\frac{4s+9m+3n+3q}{|\mathbb{F}|-1}$*, and* q*-wise independence.*

Proof. Consider the following statements:

$$(\boldsymbol{A}(1\|\boldsymbol{x}\|\boldsymbol{w})) \circ (\boldsymbol{B}(1\|\boldsymbol{x}\|\boldsymbol{w})) = \boldsymbol{C}(1\|\boldsymbol{x}\|\boldsymbol{w}) \tag{48}$$

$$\boldsymbol{y}_{[1..3m+q]} = \boldsymbol{M}(1\|\boldsymbol{x}\|\boldsymbol{w}) \tag{49}$$

$$\mathsf{V} \text{ accepts in the subprotocol } \mathsf{SparseMVP} \tag{50}$$

$$\boldsymbol{y}_{[1..m]} = \boldsymbol{A}(1\|\boldsymbol{x}\|\boldsymbol{w}), \quad \boldsymbol{y}_{[m+q+1..2m+q]} = \boldsymbol{C}(1\|\boldsymbol{x}\|\boldsymbol{w}), \text{ and } \boldsymbol{y}_{[2m+q+1..3m+q]} = \boldsymbol{B}(1\|\boldsymbol{x}\|\boldsymbol{w}) \tag{51}$$

$$\boldsymbol{y}_{[1..m]} \circ \boldsymbol{y}_{[2m+q+1..3m+q]} = \boldsymbol{y}_{[m+q+1..2m+q]} \tag{52}$$

$$\boldsymbol{y} \circ (\boldsymbol{0}^{2m+q}\|\boldsymbol{y}) = (\boldsymbol{0}^{m}\|\boldsymbol{y}) \circ (\boldsymbol{0}^{2m+q}\|\boldsymbol{1}^{m}) \tag{53}$$

The completeness follows from the fact that (48) and (49) hold if the prover is honest, and the sequence of implications (49) $\Rightarrow$ (50) that follows from the completeness of SparseMVP which fails with probability $\frac{2s+3m+n+q}{|\mathbb{F}|}$, and (49) $\Rightarrow$ (51), (48)(51) $\Rightarrow$ (52) $\Rightarrow$ (53). Therefore, the completeness error of VCProof/R1CS is $\frac{2s+3m+n+q}{|\mathbb{F}|}$.

The soundness follows from the fact that (50) and (53) hold if the verifier accepts, and the sequence of implications (50) $\Rightarrow$ (49) due to the soundness of protocol SpaseMVP and fails with probability $\frac{4s+9m+3n+3q}{|\mathbb{F}|-1}$, and (53) $\Rightarrow$ (52), (49) $\Rightarrow$ (51), and (51)(52) $\Rightarrow$ (48). Therefore, the soundness error of the VCProof/R1CS protocol is bounded by $\frac{4s+9m+3n+3q}{|\mathbb{F}|-1}$.

The q-wise independence follows from the fact that $\boldsymbol{\delta}_1$ and $\boldsymbol{\delta}_2$ are uniformly random and the q-wise independence of the subprotocol SparseMVP. □

Theorem 8. *The* VCProof/HPR *protocol in Algorithm 6 is a VIOP protocol for the relation* $\mathcal{R}_{\mathsf{HPR}}$ *with completeness error* $\frac{2s+m+3n+q+1}{|\mathbb{F}|}$*, soundness error* $\frac{4s+3m+9n+3q+3}{|\mathbb{F}|-1}$*, and* q*-wise independence.*

Proof. Consider the following statements:

$$\boldsymbol{A}\boldsymbol{w}_1 + \boldsymbol{B}\boldsymbol{w}_2 + \boldsymbol{C}\boldsymbol{w}_3 + \boldsymbol{d} = \boldsymbol{x}\|\boldsymbol{0}^{m-\ell} \tag{54}$$

$$\boldsymbol{w}_1 \circ \boldsymbol{w}_2 = \boldsymbol{w}_3 \tag{55}$$

$$\boldsymbol{M}(1\|\boldsymbol{w}) = \boldsymbol{x}\|\mathbf{0}^{m-\ell} \tag{56}$$

$$\mathsf{V} \text{ accepts in the subprotocol } \mathsf{SparseMVP} \tag{57}$$

$$\boldsymbol{w}_{[1..n]} \circ \boldsymbol{w}_{[2n+q+1..3n+q]} = \boldsymbol{w}_{[n+q+1..2n+q]} \tag{58}$$

$$\boldsymbol{w} \circ (\mathbf{0}^{2n+q}\|\boldsymbol{w}) = (\mathbf{0}^{n}\|\boldsymbol{w}) \circ (\mathbf{0}^{2n+q}\|\mathbf{1}^{n}) \tag{59}$$

The completeness follows from the fact that (54) and (55) hold if the prover is honest, and the sequence of implications (54) $\Rightarrow$ (56) $\Rightarrow$ (57) where the second implication follows from the completeness of $\mathsf{SparseMVP}$ which fails with probability $\frac{2s+m+3n+q+1}{|\mathbb{F}|}$, and (55) $\Rightarrow$ (58) $\Rightarrow$ (59). Therefore, the completeness error of $\mathsf{VCProof/HPR}$ is $\frac{2s+m+3n+q+1}{|\mathbb{F}|}$.

The soundness follows from the fact that (57) and (59) hold if the verifier accepts, and the sequence of implications (57) $\Rightarrow$ (56) $\Rightarrow$ (54) where the first implication follows from the soundness of protocol $\mathsf{SpaseMVP}$ and fails with probability $\frac{4s+3m+9n+3q+3}{|\mathbb{F}|-1}$, and (59) $\Rightarrow$ (58) $\Rightarrow$ (55). Therefore, the soundness error of the $\mathsf{VCProof/R1CS}$ protocol is bounded by $\frac{4s+3m+9n+3q+3}{|\mathbb{F}|-1}$.

The q-wise independence follows from the fact that $\boldsymbol{\delta}$ is uniformly random and the q-wise independence of the subprotocol $\mathsf{SparseMVP}$. □

Theorem 9. *The* $\mathsf{VCProof/POV}$ *protocol in Algorithm 7 is a VIOP protocol that validates the relation* $\mathcal{R}_{\mathsf{POV}}$ *with completeness error* $\frac{3n+k+\ell}{|\mathbb{F}|}$*, soundness error* $\frac{2(3n+k+\ell)}{|\mathbb{F}|-1}$*, and* q*-wise independence.*

Proof. Consider the following statements:

$$\boldsymbol{a}_{[1..n_a]} + \boldsymbol{a}_{[n_a+1..2n_a]} = \boldsymbol{a}_{[2n_a+1..3n_a]} \tag{60}$$

$$\boldsymbol{m}_{[1..n_m]} \circ \boldsymbol{m}_{[n_m+1..2n_m]} = \boldsymbol{m}_{[2n_m+1..3n_m]} \tag{61}$$

$$\boldsymbol{a}\|\boldsymbol{m}\|\boldsymbol{x}\|\boldsymbol{z} \text{ satisfies the copy condition of } \Pi \tag{62}$$

$$\boldsymbol{v} = \boldsymbol{m}_{[1..n_m]}\|\boldsymbol{\delta}\|\boldsymbol{a}\|\boldsymbol{m}_{[2n_m+1..3n_m]}\|\boldsymbol{m}_{[n_m+1..2n_m]} \tag{63}$$

$$((\mathbf{0}^{2n_a}\|\boldsymbol{v}) + (\mathbf{0}^{n_a}\|\boldsymbol{v}) - \boldsymbol{v}) \circ (\mathbf{0}^{n_m+2n_a+q}\|\mathbf{1}^{n_a}) = \mathbf{0} \tag{64}$$

$$(\mathbf{0}^{3n_a+2n_m+q}\|\boldsymbol{v}) \circ \boldsymbol{v} = (\mathbf{0}^{3n_a+2n_m+q}\|\mathbf{1}^{n_m}) \circ (\mathbf{0}^{n_m}\|\boldsymbol{v}) \tag{65}$$

$$\mathsf{V} \text{ accepts in the subprotocol } \mathsf{CopyCheck} \tag{66}$$

Note that (63) always holds since $\boldsymbol{v}$ consists of nothing beyond prover witnesses and randomnesses, and has no statement to check for. In another word, whatever P provides in $\boldsymbol{v}$ are just regarded as what P intended to use as the witnesses and randomnesses.

The completeness follows from the fact that if the prover is honest then all of (60)(61) and (62) are true, and the following implications:

– (60)(63) ⇒ (64)
– (61)(63) ⇒ (65)
– (62)(63) ⇒ (66) that follows from the completeness of the subprotocol CopyCheck which fails with probability $\frac{3n+k+\ell+q}{|\mathbb{F}|}$.

The soundness follows from the fact that (64)(65) and (66) hold if the verifier accepts, and the following implications:

– (63)(66) ⇒ (62) that follows from the soundness of the subprotocol CopyCheck which fails with probability $\frac{2(3n+k+\ell+q)}{|\mathbb{F}|-1}$;
– (65)(63) ⇒ (61)
– (64)(63)⇒ (60)

Therefore, the soundness error of the VCProof/POV protocol is $\frac{2(3n+k+\ell+q)}{|\mathbb{F}|-1}$.

The q-wise independence follows from the fact that $\boldsymbol{\delta}$ is uniformly random independent of all the other elements, and that the subprotocol CopyCheck is q-wise independent. □

D The Complete zkSNARKs

For reference, we present the complete zkSNARKs compiled from our protocols using the optimized compiler and the KZG polynomial commitment scheme.

D.1 Cryptographic Primitives

Prime Field and Bilinear Pairing. We choose $\mathbb{F} = \mathbb{F}_p$ for a prime $p \approx 2^\lambda$ where λ is the security parameter. Let γ be a generator of the multiplicative group $\mathbb{F}^*$.

A bilinear pairing scheme consists of a tuple $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, G_1, G_2, G_T)$ where $\mathbb{G}_1, \mathbb{G}_2$ and $\mathbb{G}_T$ are groups of size p, $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is an efficiently computable bilinear non-degenerate bilinear map, G_1, G_2 are uniformly random chosen from $\mathbb{G}_1, \mathbb{G}_2$ respectively, and $G_T = e(G_1, G_2)$. We use the notations $[x]_1 := x \cdot G_1$ and $[x]_2 := x \cdot G_2$.

Hash to Field. Let $H : \{0,1\}^* \to \mathbb{F}$ be a hash function modeled as a random oracle. In cases where multiple random elements are needed in the same round, let $H_i(x)$ be a convenient alias of $H(i\|x)$ where i is encoded by its fixed-length binary representation.

The KZG Polynomial Commitment. For completeness, we present the KZG scheme here. The description is copied almost verbatim from PLONK [3], except that we present the non-interactive version by applying the Fiat-Shamir heuristic using H. The open and vrfy algorithms validates the evaluations of $t = t_1 + t_2 + t_3$ polynomials at 3 distinct evaluation points, which is the maximal number that our algorithm uses.

- $\mathsf{gen}(D)$ - choose uniform $x \in \mathbb{F}$. Output $\mathsf{srs} = ([1]_1, [x]_1, \cdots, [x^{D-1}]_1, [1]_2, [x]_2)$.
- $\mathsf{com}(f(X), \mathsf{srs}) := [f(x)]_1$.
- $\mathsf{open}(\{\mathsf{cm}_i, y_i, f_i(X)\}_{i=1}^{t_1}, \{\mathsf{cm}'_i, y'_i, f'_i(X)\}_{i=1}^{t_2}, \{\mathsf{cm}''_i, y''_i, f''_i(X)\}_{i=1}^{t_3}, \{z, z', z''\})$
  1. $\xi := H_1(\{\mathsf{cm}_i, y_i\}_{i=1}^{t_1}, \{\mathsf{cm}'_i, y'_i\}_{i=1}^{t_2}, \{\mathsf{cm}''_i, y''_i\}_{i=1}^{t_3}, \{z, z', z''\})$ and ξ', ξ'' are computed with the same inputs by using H_2 and H_3 instead.
  2. Let
  $$q(X) := \sum_{i=1}^{t_1} \xi^{i-1} \cdot \frac{f_i(X) - y_i}{X - z}$$
  and $q'(X), q''(X)$ similarly by using $f'_i(X), \xi', z', y'_i$ and $f''_i(X), \xi'', z'', y''_i$ instead respectively.
  3. Compute $W := [q(x)]_1$, $W' := [q'(x)]_1$, $W'' := [q''(x)]_1$ using srs.
  4. Output (W, W', W'')
- $\mathsf{vrfy}(\{\mathsf{cm}_i, y_i\}_{i=1}^{t_1}, \{\mathsf{cm}'_i, y'_i\}_{i=1}^{t_2}, \{\mathsf{cm}''_i, y''_i\}_{i=1}^{t_3}, \{z, z', z''\}, \{W, W', W''\}, [x]_2)$
  1. Computes ξ, ξ', ξ'' in the same way as in open.
  2. $r', r'' \xleftarrow{\$} \mathbb{F}$
  3. Let
  $$Q := \sum_{i=1}^{t_1} \xi^{i-1} \cdot \mathsf{cm}_i - \left[\sum_{i=1}^{t_1} \xi^{i-1} \cdot y_i\right]_1$$
  and Q', Q'' are computed similarly by using $\mathsf{cm}'_i, \xi', y'_i$ and $\mathsf{cm}''_i, \xi'', y''_i$ instead respectively.
  4. Let $F := Q + r'Q' + r''Q''$.
  5. Outputs accept if and only if
  $$e(F + z \cdot W + r'z' \cdot W' + r''z'' \cdot W'', [1]_2) \cdot e(-W - r' \cdot W' - r'' \cdot W'', [x]_2).$$

Remark 1. Let t be the number of evaluation points, t^* the number of distinct points, d_i be the maximal degree of polynomials evaluated at the i-th distinct point z_i, then the efficiency of the above scheme is summarized as follows.

- The proof size is t^* elements in $\mathbb{G}_1$.
- For $d < D$ and $f(X) \in \mathbb{F}^d[X]$, the computation of com is dominated by d $\mathbb{G}_1$ exponentiations.

– The computation of open is dominated by $\sum_{i\in[t^*]} d_i$ $\mathbb{G}_1$ exponentiations.
– The computation of vrfy is dominated by 2 pairings and $t + 2t^* - 2$ $\mathbb{G}_1$ exponentiations.

D.2 The zkSNARK for R1CS

Let $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C} \in \mathbb{F}^{M\times N}$ be matrices each has at most S nonzero entries. For an instance $\boldsymbol{x} \in \mathbb{F}^{\ell}$, the zkSNARK $\mathsf{VCProof/R1CS}$ generates a proof for the existence of $\boldsymbol{w} \in \mathbb{F}^{N-\ell-1}$ such that $\boldsymbol{A}(1\|\boldsymbol{x}\|\boldsymbol{w}) \circ \boldsymbol{B}(1\|\boldsymbol{x}\|\boldsymbol{w}) = \boldsymbol{C}(1\|\boldsymbol{x}\|\boldsymbol{w})$.

Setup. Output $\mathsf{srs} := \mathsf{gen}(D)$ for $D \geq \max\{9S + 1, 6S + 3M + N + 4\}$.

Preprocessing. The inputs are the matrix sparse representations, i.e., $\{\mathsf{row}_i^A, \mathsf{col}_i^A, \mathsf{val}_i^A\}_{i=1}^S$, $\{\mathsf{row}_i^B, \mathsf{col}_i^B, \mathsf{val}_i^B\}_{i=1}^S$ and $\{\mathsf{row}_i^C, \mathsf{col}_i^C, \mathsf{val}_i^C\}_{i=1}^S$, where $\mathsf{row}_i \in [M], \mathsf{col}_i \in [N]$ and $\mathsf{val}_i \in \mathbb{F}$.

1. $\boldsymbol{row} := (\mathsf{row}_i^A)_{i=1}^S \| (\mathsf{row}_i^B + 2M + 3)_{i=1}^S \| (\mathsf{row}_i^C + M + 3)_{i=1}^S$
2. $\boldsymbol{col} := (\mathsf{col}_i^A)_{i=1}^S \| (\mathsf{col}_i^B)_{i=1}^S \| (\mathsf{col}_i^C)_{i=1}^S$
3. $\boldsymbol{val} := (\mathsf{val}_i^A)_{i=1}^S \| (\mathsf{val}_i^B)_{i=1}^S \| (\mathsf{val}_i^C)_{i=1}^S$
4. For $i \in [3S]$,
   – let $\mathsf{prevrow}_i$ be the maximal $j \in [i-1]$ such that $\boldsymbol{row}_{[j]} = \boldsymbol{row}_{[i]}$, or $6S + \boldsymbol{row}_{[i]}$ if such j does not exist;
   – let $\mathsf{prevcol}_i$ be the maximal $j \in [i-1]$ such that $\boldsymbol{col}_{[j]} = \boldsymbol{col}_{[i]}$, or $6S + 3M + 3 + \boldsymbol{col}_{[i]}$ if such j does not exist.
5. For $i \in [3M + 3]$,
   – let maxrow_i be the maximal $j \in [3S]$ such that $\boldsymbol{row}_{[j]} = i$, or $6S + i$ if such j does not exist;
6. For $i \in [N]$,
   – let maxcol_i be the maximal $j \in [3S]$ such that $\boldsymbol{col}_{[j]} = i$, or $3S + 3M + 3 + i$ if such j does not exist.
7. $\boldsymbol{\sigma} := (\gamma^{\sigma(i)-1})_{i=1}^{6S+3M+N+3}$ where

$$\sigma(i) := \begin{cases} \mathsf{prevrow}_i, & 1 \leq i \leq 3S \\ 3S + \mathsf{prevcol}_{i-3S}, & 3S+1 \leq i \leq 6S \\ \mathsf{maxrow}_{i-6S}, & 6S+1 \leq i \leq 6S+3M+3 \\ 3S + \mathsf{maxcol}_{i-6S-3M-3}, & 6S+3M+4 \leq i \leq 6S+3M+N+3 \end{cases}$$

8. $\mathsf{cm}_v := \mathsf{com}(f_{\boldsymbol{val}}(X), \mathsf{srs})$, $\mathsf{cm}_\sigma := \mathsf{com}(f_{\boldsymbol{\sigma}}(X), \mathsf{srs})$
9. Output
   – $\mathsf{vk} = (\mathsf{cm}_v, \mathsf{cm}_\sigma)$
   – $\mathsf{pk} = (\boldsymbol{row}, \boldsymbol{col}, \boldsymbol{val}, \boldsymbol{\sigma}, \mathsf{cm}_v, \mathsf{cm}_\sigma)$

Prover. The inputs are srs, pk, instance $\boldsymbol{x} \in \mathbb{F}^{\ell}$ and witness $\boldsymbol{w} \in \mathbb{F}^{N-\ell-1}$.

1. $\boldsymbol{\delta}_1, \boldsymbol{\delta}_2 \xleftarrow{\$} \mathbb{F}^3$, $\boldsymbol{u} := \boldsymbol{w} \| \boldsymbol{\delta}_1 \in \mathbb{F}^{N-\ell+3}$, $\mathsf{cm}_u := \mathsf{com}(f_{\boldsymbol{u}}(X), \mathsf{srs})$.
2. $\boldsymbol{y} := (\boldsymbol{M}(1 \| \boldsymbol{x} \| \boldsymbol{w})) \| \boldsymbol{\delta}_2 \in \mathbb{F}^{3M+6}$ where $\boldsymbol{M} \in \mathbb{F}^{(3M+3) \times N}$ is the matrix with sparse representation $\boldsymbol{row}, \boldsymbol{col}, \boldsymbol{val}$
3. $\mathsf{cm}_y := \mathsf{com}(f_{\boldsymbol{y}}(X), \mathsf{srs})$.
4. $\alpha := H_1(\mathsf{cm}_v, \mathsf{cm}_\sigma, \boldsymbol{x}, \mathsf{cm}_u, \mathsf{cm}_y)$.
5. $\boldsymbol{c} := (\boldsymbol{\alpha}^{3M+3})^{\mathsf{T}} \boldsymbol{M} \in \mathbb{F}^N$, $\mathsf{cm}_c := \mathsf{com}(f_{\boldsymbol{c}}(X), \mathsf{srs})$.
6. $\beta := H_1(\mathsf{cm}_v, \mathsf{cm}_\sigma, \boldsymbol{x}, \mathsf{cm}_u, \mathsf{cm}_y, \mathsf{cm}_c)$.
7. $\boldsymbol{z} := (\alpha^{\boldsymbol{row}_{[i]}})_{i=1}^{3S} \| (\beta^{\boldsymbol{col}_{[i]}})_{i=1}^{3S} \in \mathbb{F}^{6S}$, $\mathsf{cm}_z := \mathsf{com}(f_{\boldsymbol{z}}(X), \mathsf{srs})$.
8. $\boldsymbol{t} := (\alpha^{\boldsymbol{row}_{[i]}} \cdot \beta^{\boldsymbol{col}_{[i]}})_{i=1}^{3S} \in \mathbb{F}^{3S}$, $\mathsf{cm}_t := \mathsf{com}(f_{\boldsymbol{t}}(X), \mathsf{srs})$.
9. $\zeta := H_1(\mathsf{cm}_v, \mathsf{cm}_\sigma, \boldsymbol{x}, \mathsf{cm}_u, \mathsf{cm}_y, \mathsf{cm}_c, \mathsf{cm}_z, \mathsf{cm}_t)$, compute ζ' and ζ'' with the same inputs by using H_2 and H_3 instead.
10. $\boldsymbol{a} := \zeta \cdot \boldsymbol{1}^{6S+3M+N+3} + \zeta' \cdot (\boldsymbol{z} \| \boldsymbol{\alpha}^{3M+3} \| \boldsymbol{\beta}^N) + \zeta'' \cdot \boldsymbol{\gamma}^{6S+3M+N+3}$.
11. $\boldsymbol{b} := \zeta \cdot \boldsymbol{1}^{6S+3M+N+3} + \zeta' \cdot (\boldsymbol{z} \| \boldsymbol{\alpha}^{3M+3} \| \boldsymbol{\beta}^N) + \zeta'' \cdot \boldsymbol{\sigma}$.
12. $\boldsymbol{r} := \left(\prod_{j=1}^{i} \boldsymbol{a}_{[j]} / \boldsymbol{b}_{[j]} \right)_{i=1}^{6S+3M+N+3} \in \mathbb{F}^{6S+3M+N+3}$, $\mathsf{cm}_r := \mathsf{com}(f_{\boldsymbol{r}}(X), \mathsf{srs})$.
13. $\nu := H_1(\mathsf{cm}_v, \mathsf{cm}_\sigma, \boldsymbol{x}, \mathsf{cm}_u, \mathsf{cm}_y, \mathsf{cm}_c, \mathsf{cm}_z, \mathsf{cm}_t, \mathsf{cm}_r)$, compute ω with the same inputs by using H_2 instead.
14. Compute
    - $h_1(X) := f_{\boldsymbol{y}}(\nu \cdot X^{-1}) \cdot f_{\boldsymbol{y}}(X) \cdot X^{6M+9}$
    - $h_2(X) := f_{\boldsymbol{y}}(\nu \cdot X^{-1}) \cdot (\nu X^{-1})^{3M+3} \cdot \left(\sum_{i=1}^{3M+3} X^{6M+8+i} \right)$
    - $h_3(X) := f_{\boldsymbol{z}}(\nu \cdot X^{-1}) \cdot X^{3S} \cdot f_{\boldsymbol{z}}(X)$
    - $h_4(X) := \left(\sum_{i=1}^{6S} (\nu X^{-1})^{i-1} \right) \cdot f_{\boldsymbol{t}}(X) \cdot X^{3S}$
    - $h_5(X) := \left(f_{\boldsymbol{r}}(\nu \cdot X^{-1}) \cdot \nu X^{-1} + 1 - (\nu X^{-1})^{6S+3M+N+3} \right) \cdot f_{\boldsymbol{a}}(X)$
    - $h_6(X) := f_{\boldsymbol{r}}(\nu \cdot X^{-1}) \cdot f_{\boldsymbol{b}}(X)$
    - $h_7(X) := \left(f_{\boldsymbol{r}}(\nu \cdot X^{-1}) - (\nu \cdot X^{-1})^{6S+3M+N+2} \right) \cdot X^{6S+3M+N+2}$
    - $h_8(X) := f_{\boldsymbol{c}}(X^{-1}) \cdot (1 + f_{\boldsymbol{x}}(X) \cdot X + f_{\boldsymbol{u}}(X) \cdot X^{\ell+1})$
    - $h_9(X) := \left(\sum_{i=1}^{3M+3} (\alpha \cdot X^{-1})^{i-1} \right) \cdot f_{\boldsymbol{y}}(X)$
    - $h_{10}(X) := f_{\boldsymbol{val}}(X^{-1}) \cdot f_{\boldsymbol{t}}(X)$
    - $h_{11}(X) := f_{\boldsymbol{c}}(X^{-1}) \cdot \left(\sum_{i=1}^{N} (\beta \cdot X)^{i-1} \right)$
15. Compute

$$
\begin{aligned}
h(X) := {} & (h_1(X) - h_2(X)) + \omega \cdot (h_3(X) - h_4(X)) + \omega^2 \cdot (h_5(X) - h_6(X)) + \\
& \omega^3 \cdot h_7(X) + \omega^4 \cdot (h_8(X) - h_9(X)) + \omega^5 \cdot (h_{10}(X) - h_{11}(X)) + \\
& \omega^6 \cdot f_{\boldsymbol{u}}(X) \cdot X^{D-N-3} + \omega^7 \cdot f_{\boldsymbol{y}}(X) \cdot X^{D-3M-6} + \\
& \omega^8 \cdot f_{\boldsymbol{c}}(X) \cdot X^{D-N} + \omega^9 \cdot f_{\boldsymbol{z}}(X) \cdot X^{D-6S} + \\
& \omega^{10} \cdot f_{\boldsymbol{t}}(X) \cdot X^{D-3S} + \omega^{11} \cdot f_{\boldsymbol{r}}(X) \cdot X^{D-6S-3M-N-3}.
\end{aligned}
$$

16. Compute $\bar{h}_1(X) := (X^D \cdot h(X)) \bmod X^D$, $\bar{h}_2(X) := \lfloor h(X) \cdot X^{-1} \rfloor$ where $\lfloor \cdot \rfloor$ means removing all the monomials with negative powers.
17. $\mathsf{cm}_{h_1} := \mathsf{com}(\bar{h}_1(X), \mathsf{srs})$, $\mathsf{cm}_{h_2} := \mathsf{com}(\bar{h}_2(X), \mathsf{srs})$
18. $z := H_1(\mathsf{cm}_v, \mathsf{cm}_\sigma, \boldsymbol{x}, \mathsf{cm}_u, \mathsf{cm}_y, \mathsf{cm}_c, \mathsf{cm}_z, \mathsf{cm}_t, \mathsf{cm}_r, \mathsf{cm}_{h_1}, \mathsf{cm}_{h_2})$
19. $y_y := f_{\boldsymbol{y}}(\nu \cdot z^{-1})$, $y_z := f_{\boldsymbol{z}}(\nu \cdot z^{-1})$, $y_r := f_{\boldsymbol{r}}(\nu \cdot z^{-1})$, $y_c := f_{\boldsymbol{c}}(z^{-1})$, $y_{val} := f_{\boldsymbol{val}}(z^{-1})$
20. Compute

$$
\begin{aligned}
\bar{g}(X) := & \left(y_y \cdot z^{6M+9} + \omega^7 \cdot z^{D-3M-6} - \omega^4 \cdot \frac{(\alpha \cdot z^{-1})^{3M+3} - 1}{\alpha \cdot z^{-1} - 1} \right) \cdot f_{\boldsymbol{y}}(X) \\
& + (\omega \cdot y_z \cdot z^{3S} + \zeta' \cdot \omega^2 \cdot (y_r \cdot \nu \cdot z^{-1} \\
& \quad - (\nu \cdot z^{-1})^{6S+3M+N+3} - y_r + 1) + \omega^9 \cdot z^{D-6S}) \cdot f_{\boldsymbol{z}}(X) \\
& + \left(\omega^{10} \cdot z^{D-3S} - \omega \cdot \frac{(\nu \cdot z^{-1})^{6S} - 1}{\nu \cdot z^{-1} - 1} \cdot z^{3S} + \omega^5 \cdot y_{val} \right) \cdot f_{\boldsymbol{t}}(X) \\
& + (\omega^4 \cdot y_c \cdot z^{\ell+1} + \omega^6 \cdot z^{D-N-3}) \cdot f_{\boldsymbol{u}}(X) + \omega^8 \cdot z^{D-N} \cdot f_{\boldsymbol{c}}(X) + \\
& + \omega^{11} \cdot z^{D-6S-3M-N-3} \cdot f_{\boldsymbol{r}}(X) - \omega^2 \cdot y_r \cdot \zeta'' \cdot f_{\boldsymbol{\sigma}}(X) - z^{-D} \cdot \bar{h}_1(X) - z \cdot \bar{h}_2(X) \\
& + \Bigg(-y_y \cdot \nu^{3M+3} \cdot \frac{z^{6M+9} - z^{3M+6}}{z - 1} \\
& \quad + \omega^2 \cdot (y_r \cdot \nu \cdot z^{-1} + 1 - (\nu \cdot z^{-1})^{6S+3M+N+3}) \cdot \zeta'' \cdot \frac{(\gamma \cdot z)^{6S+3M+N+3} - 1}{\gamma \cdot z - 1} \\
& \quad + \omega^2 \cdot (-y_r + y_r \cdot \nu \cdot z^{-1} + 1 - (\nu \cdot z^{-1})^{6S+3M+N+3}) \cdot \\
& \quad \left(\zeta \cdot \frac{z^{6S+3M+N+3} - 1}{z - 1} + \zeta' \cdot z^{6S} \cdot \left(\frac{(\alpha \cdot z)^{3M+3} - 1}{\alpha \cdot z - 1} + z^{3M+3} \cdot \frac{(\beta \cdot z)^N - 1}{\beta \cdot z - 1} \right) \right) \\
& \quad + \omega^3 \cdot (y_r \cdot z^{6S+3M+N+2} - \nu^{6S+3M+N+2}) \\
& \quad + \omega^4 \cdot y_c \cdot (f_{\boldsymbol{x}}(z) \cdot z + 1) - \omega^5 \cdot y_c \cdot \frac{(\beta \cdot z)^N - 1}{\beta \cdot z - 1} \Bigg)
\end{aligned}
$$

21. $\mathsf{cm}_g := \mathsf{com}(\bar{g}(X), \mathsf{srs})$ computed by linearly combining cm_y, cm_z, cm_t, cm_c, cm_r, cm_σ, cm_{h_1}, cm_{h_2} and $[1]_1$.
22. Compute $(W, W', W'') := \mathsf{open} \begin{pmatrix} \{(\mathsf{cm}_y, y_y, f_{\boldsymbol{y}}(X)), (\mathsf{cm}_z, y_z, f_{\boldsymbol{z}}(X)), \\ \quad (\mathsf{cm}_r, y_r, f_{\boldsymbol{r}}(X))\}, \\ \{(\mathsf{cm}_c, y_c, f_{\boldsymbol{c}}(X)), (\mathsf{cm}_{val}, y_{val}, f_{\boldsymbol{val}}(X))\}, \\ \{\mathsf{cm}_g, 0, \bar{g}(X)\}, \\ \{\nu \cdot z^{-1}, z^{-1}, z\} \end{pmatrix}$
23. Output $\pi := (\mathsf{cm}_u, \mathsf{cm}_y, \mathsf{cm}_c, \mathsf{cm}_z, \mathsf{cm}_t, \mathsf{cm}_r, \mathsf{cm}_{h_1}, \mathsf{cm}_{h_2}, y_y, y_z, y_r, y_c, y_{val}, W, W', W'')$

Verifier. The inputs are $[x]_2, \mathsf{vk}, \boldsymbol{x}$ and π.

1. Compute

- $\alpha := H_1(\mathsf{cm}_v, \mathsf{cm}_\sigma, \boldsymbol{x}, \mathsf{cm}_u, \mathsf{cm}_y)$
- $\beta := H_1(\mathsf{cm}_v, \mathsf{cm}_\sigma, \boldsymbol{x}, \mathsf{cm}_u, \mathsf{cm}_y, \mathsf{cm}_c)$
- $\zeta := H_1(\mathsf{cm}_v, \mathsf{cm}_\sigma, \boldsymbol{x}, \mathsf{cm}_u, \mathsf{cm}_y, \mathsf{cm}_c, \mathsf{cm}_z, \mathsf{cm}_t)$
- $\zeta' := H_2(\mathsf{cm}_v, \mathsf{cm}_\sigma, \boldsymbol{x}, \mathsf{cm}_u, \mathsf{cm}_y, \mathsf{cm}_c, \mathsf{cm}_z, \mathsf{cm}_t)$
- $\zeta'' := H_3(\mathsf{cm}_v, \mathsf{cm}_\sigma, \boldsymbol{x}, \mathsf{cm}_u, \mathsf{cm}_y, \mathsf{cm}_c, \mathsf{cm}_z, \mathsf{cm}_t)$
- $\nu := H_1(\mathsf{cm}_v, \mathsf{cm}_\sigma, \boldsymbol{x}, \mathsf{cm}_u, \mathsf{cm}_y, \mathsf{cm}_c, \mathsf{cm}_z, \mathsf{cm}_t, \mathsf{cm}_r)$
- $\omega := H_2(\mathsf{cm}_v, \mathsf{cm}_\sigma, \boldsymbol{x}, \mathsf{cm}_u, \mathsf{cm}_y, \mathsf{cm}_c, \mathsf{cm}_z, \mathsf{cm}_t, \mathsf{cm}_r)$
- $z := H_1(\mathsf{cm}_v, \mathsf{cm}_\sigma, \boldsymbol{x}, \mathsf{cm}_u, \mathsf{cm}_y, \mathsf{cm}_c, \mathsf{cm}_z, \mathsf{cm}_t, \mathsf{cm}_r, \mathsf{cm}_{h_1}, \mathsf{cm}_{h_2})$

2. Compute

$$
\begin{aligned}
\mathsf{cm}_g := & \left(y_y \cdot z^{6M+9} + \omega^7 \cdot z^{D-3M-6} - \omega^4 \cdot \frac{(\alpha \cdot z^{-1})^{3M+3} - 1}{\alpha \cdot z^{-1} - 1} \right) \cdot \mathsf{cm}_y \\
& + (\omega \cdot y_z \cdot z^{3S} + \zeta' \cdot \omega^2 \cdot (y_r \cdot \nu \cdot z^{-1} \\
& \quad - (\nu \cdot z^{-1})^{6S+3M+N+3} - y_r + 1) + \omega^9 \cdot z^{D-6S}) \cdot \mathsf{cm}_z \\
& + \left(\omega^{10} \cdot z^{D-3S} - \omega \cdot \frac{(\nu \cdot z^{-1})^{6S} - 1}{\nu \cdot z^{-1} - 1} \cdot z^{3S} + \omega^5 \cdot y_{val} \right) \cdot \mathsf{cm}_t \\
& + (\omega^4 \cdot y_c \cdot z^{\ell+1} + \omega^6 \cdot z^{D-N-3}) \cdot \mathsf{cm}_u + \omega^8 \cdot z^{D-N} \cdot \mathsf{cm}_c + \\
& + \omega^{11} \cdot z^{D-6S-3M-N-3} \cdot \mathsf{cm}_r - \omega^2 \cdot y_r \cdot \zeta'' \cdot \mathsf{cm}_\sigma - z^{-D} \cdot \mathsf{cm}_{h_1} - z \cdot \mathsf{cm}_{h_2} \\
& + \Bigg[- y_y \cdot \nu^{3M+3} \cdot \frac{z^{6M+9} - z^{3M+6}}{z - 1} \\
& \quad + \omega^2 \cdot (y_r \cdot \nu \cdot z^{-1} + 1 - (\nu \cdot z^{-1})^{6S+3M+N+3}) \cdot \zeta'' \cdot \frac{(\gamma \cdot z)^{6S+3M+N+3} - 1}{\gamma \cdot z - 1} \\
& \quad + \omega^2 \cdot (-y_r + y_r \cdot \nu \cdot z^{-1} + 1 - (\nu \cdot z^{-1})^{6S+3M+N+3}) \cdot \\
& \quad \left(\zeta \cdot \frac{z^{6S+3M+N+3} - 1}{z - 1} + \zeta' \cdot z^{6S} \cdot \left(\frac{(\alpha \cdot z)^{3M+3} - 1}{\alpha \cdot z - 1} + z^{3M+3} \cdot \frac{(\beta \cdot z)^N - 1}{\beta \cdot z - 1} \right) \right) \\
& \quad + \omega^3 \cdot (y_r \cdot z^{6S+3M+N+2} - \nu^{6S+3M+N+2}) \\
& \quad + \omega^4 \cdot y_c \cdot (f_{\boldsymbol{x}}(z) \cdot z + 1) - \omega^5 \cdot y_c \cdot \frac{(\beta \cdot z)^N - 1}{\beta \cdot z - 1} \Bigg]_1
\end{aligned}
$$

3. Output $\mathsf{vrfy} \left(\begin{array}{l} \{(\mathsf{cm}_w, y_w), (\mathsf{cm}_z, y_z), (\mathsf{cm}_r, y_r)\}, \{(\mathsf{cm}_c, y_c), (\mathsf{cm}_{val}, y_{val})\}, \\ \{(\mathsf{cm}_g, 0)\}, \{\nu \cdot z^{-1}, z^{-1}, z\}, \{W, W', W''\}, [x]_2 \end{array} \right)$.

D.3 The zkSNARK for HPR

Let $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C} \in \mathbb{F}^{M \times N}$ be matrices each has at most S nonzero entries, and $\boldsymbol{d} \in \mathbb{F}^M$ has at most S' nonzero entries. For an instance $\boldsymbol{x} \in \mathbb{F}^\ell$, the zk-SNARK $\mathsf{VCProof}/\mathsf{HPR}$ generates a proof for the existence of $\boldsymbol{w}_1, \boldsymbol{w}_2, \boldsymbol{w}_3 \in \mathbb{F}^N$ such that $\boldsymbol{A}\boldsymbol{w}_1 + \boldsymbol{B}\boldsymbol{w}_2 + \boldsymbol{C}\boldsymbol{w}_3 + \boldsymbol{d} = \boldsymbol{x} \| \boldsymbol{0}^{M-\ell}$ and $\boldsymbol{w}_1 \circ \boldsymbol{w}_2 = \boldsymbol{w}_3$.

Setup. Output $\mathsf{srs} := \mathsf{gen}(D)$ for $D \geq \max\{9S+3S'+1, 6S+2S'+M+3N+4\}$.

Preprocessing The inputs are the matrix sparse representations, i.e., $\{\mathsf{row}_i^A, \mathsf{col}_i^A, \mathsf{val}_i^A\}_{i=1}^S$, $\{\mathsf{row}_i^B, \mathsf{col}_i^B, \mathsf{val}_i^B\}_{i=1}^S$ and $\{\mathsf{row}_i^C, \mathsf{col}_i^C, \mathsf{val}_i^C\}_{i=1}^S$, where $\mathsf{row}_i \in [M], \mathsf{col}_i \in [N]$ and $\mathsf{val}_i \in \mathbb{F}$, and the sparse representation of $\boldsymbol{d}$, i.e., $\{(\mathsf{rowd}_i, \mathsf{vald}_i)\}_{i=1}^{S'}$.

1. $\boldsymbol{row} := (\mathsf{rowd}_i)_{i=1}^{S'} \| (\mathsf{row}_i^A)_{i=1}^S \| (\mathsf{row}_i^B)_{i=1}^S \| (\mathsf{row}_i^C)_{i=1}^S$
2. $\boldsymbol{col} := \mathbf{1}^{S'} \| (\mathsf{col}_i^A + 1)_{i=1}^S \| (\mathsf{col}_i^B + 2N + 4)_{i=1}^S \| (\mathsf{col}_i^C + N + 4)_{i=1}^S$
3. $\boldsymbol{val} := (\mathsf{vald}_i)_{i=1}^{S'} \| (\mathsf{val}_i^A)_{i=1}^S \| (\mathsf{val}_i^B)_{i=1}^S \| (\mathsf{val}_i^C)_{i=1}^S$
4. For $i \in [3S+S']$,
   - let $\mathsf{prevrow}_i$ be the maximal $j \in [i-1]$ such that $\boldsymbol{row}_{[j]} = \boldsymbol{row}_{[i]}$, or $6S+2S'+\boldsymbol{row}_{[i]}$ if such j does not exist;
   - let $\mathsf{prevcol}_i$ be the maximal $j \in [i-1]$ such that $\boldsymbol{col}_{[j]} = \boldsymbol{col}_{[i]}$, or $6S+2S'+M+\boldsymbol{col}_{[i]}$ if such j does not exist.
5. For $i \in [M]$,
   - let maxrow_i be the maximal $j \in [3S+S']$ such that $\boldsymbol{row}_{[j]} = i$, or $6S+2S'+i$ if such j does not exist;
6. For $i \in [3N+4]$,
   - let maxcol_i be the maximal $j \in [3S+S']$ such that $\boldsymbol{col}_{[j]} = i$, or $3S+S'+M+i$ if such j does not exist.
7. $\boldsymbol{\sigma} := (\gamma^{\sigma(i)-1})_{i=1}^{6S+M+3N+4}$ where

$$\sigma(i) := \begin{cases} \mathsf{prevrow}_i, & 1 \leq i \leq 3S+S' \\ 3S+S'+\mathsf{prevcol}_{i-3S-S'}, & 3S+1 \leq i \leq 6S+2S' \\ \mathsf{maxrow}_{i-6S-2S'}, & 6S+2S'+1 \leq i \leq 6S+2S'+M \\ 3S+\mathsf{maxcol}_{i-6S-2S'-M}, & 6S+2S'+M+1 \leq i \leq 6S+2S'+M+3N+3 \end{cases}$$

8. $\mathsf{cm}_v := \mathsf{com}(f_{\boldsymbol{val}}(X), \mathsf{srs})$, $\mathsf{cm}_\sigma := \mathsf{com}(f_{\boldsymbol{\sigma}}(X), \mathsf{srs})$
9. Output
   - $\mathsf{vk} = (\mathsf{cm}_v, \mathsf{cm}_\sigma)$
   - $\mathsf{pk} = (\boldsymbol{val}, \boldsymbol{\sigma}, \mathsf{cm}_v, \mathsf{cm}_\sigma)$

Prover. The inputs are srs, pk, instance $\boldsymbol{x} \in \mathbb{F}^\ell$ and witnesses $\boldsymbol{w}_1, \boldsymbol{w}_2, \boldsymbol{w}_3 \in \mathbb{F}^N$.

1. $\boldsymbol{\delta} \xleftarrow{\$} \mathbb{F}^3$, $\boldsymbol{w} := \boldsymbol{w}_1 \| \boldsymbol{\delta} \| \boldsymbol{w}_3 \| \boldsymbol{w}_2 \in \mathbb{F}^{3N+3}$, $\mathsf{cm}_w := \mathsf{com}(f_{\boldsymbol{w}}(X), \mathsf{srs})$.
2. $\alpha := H_1(\mathsf{cm}_v, \mathsf{cm}_\sigma, \boldsymbol{x}, \mathsf{cm}_w)$.
3. $\boldsymbol{c} := (\boldsymbol{\alpha}^M)^\mathsf{T} \boldsymbol{M} \in \mathbb{F}^{3N+4}$, $\mathsf{cm}_c := \mathsf{com}(f_{\boldsymbol{c}}(X), \mathsf{srs})$.
4. $\beta := H_1(\mathsf{cm}_v, \mathsf{cm}_\sigma, \boldsymbol{x}, \mathsf{cm}_w, \mathsf{cm}_c)$.

5. $\boldsymbol{z} := (\alpha^{\boldsymbol{row}_{[i]}})_{i=1}^{3S+S'} \| (\beta^{\boldsymbol{col}_{[i]}})_{i=1}^{3S+S'} \in \mathbb{F}^{6S+2S'}$, $\mathsf{cm}_z := \mathsf{com}(f_{\boldsymbol{z}}(X), \mathsf{srs})$.
6. $\boldsymbol{t} := (\alpha^{\boldsymbol{row}_{[i]}} \cdot \beta^{\boldsymbol{col}_{[i]}})_{i=1}^{3S+S'} \in \mathbb{F}^{3S+S'}$, $\mathsf{cm}_t := \mathsf{com}(f_{\boldsymbol{t}}(X), \mathsf{srs})$.
7. $\zeta := H_1(\mathsf{cm}_v, \mathsf{cm}_\sigma, \boldsymbol{x}, \mathsf{cm}_w, \mathsf{cm}_c, \mathsf{cm}_z, \mathsf{cm}_t)$, compute ζ' and ζ'' with the same inputs by using H_2 and H_3 instead.
8. $\boldsymbol{a} := \zeta \cdot \mathbf{1}^{6S+2S'+M+3N+4} + \zeta' \cdot (\boldsymbol{z} \| \boldsymbol{\alpha}^M \| \boldsymbol{\beta}^{3N+4}) + \zeta'' \cdot \boldsymbol{\gamma}^{6S+2S'+M+3N+4}$.
9. $\boldsymbol{b} := \zeta \cdot \mathbf{1}^{6S+2S'+M+3N+4} + \zeta' \cdot (\boldsymbol{z} \| \boldsymbol{\alpha}^M \| \boldsymbol{\beta}^{3N+4}) + \zeta'' \cdot \boldsymbol{\sigma}$.
10. $\boldsymbol{r} := \left(\prod_{j=1}^{i} \boldsymbol{a}_{[j]} / \boldsymbol{b}_{[j]} \right)_{i=1}^{6S+2S'+M+3N+4} \in \mathbb{F}^{6S+2S'+M+3N+4}$, $\mathsf{cm}_r := \mathsf{com}(f_{\boldsymbol{r}}(X), \mathsf{srs})$.
11. $\nu := H_1(\mathsf{cm}_v, \mathsf{cm}_\sigma, \boldsymbol{x}, \mathsf{cm}_w, \mathsf{cm}_c, \mathsf{cm}_z, \mathsf{cm}_t, \mathsf{cm}_r)$, compute ω with the same inputs by using H_2 instead.
12. Compute
    - $h_1(X) := f_{\boldsymbol{w}}(\nu \cdot X^{-1}) \cdot f_{\boldsymbol{w}}(X) \cdot X^{6N+9}$
    - $h_2(X) := f_{\boldsymbol{w}}(\nu \cdot X^{-1}) \cdot (\nu X^{-1})^{3N+3} \cdot \left(\sum_{i=1}^{3N+3} X^{6N+8+i} \right)$
    - $h_3(X) := f_{\boldsymbol{z}}(\nu \cdot X^{-1}) \cdot X^{3S+S'} \cdot f_{\boldsymbol{z}}(X)$
    - $h_4(X) := \left(\sum_{i=1}^{6S+2S'} (\nu X^{-1})^{i-1} \right) \cdot f_{\boldsymbol{t}}(X) \cdot X^{3S+S'}$
    - $h_5(X) := \left(f_{\boldsymbol{r}}(\nu \cdot X^{-1}) \cdot \nu X^{-1} + 1 - (\nu X^{-1})^{6S+2S'+M+3N+4} \right) \cdot f_{\boldsymbol{a}}(X)$
    - $h_6(X) := f_{\boldsymbol{r}}(\nu \cdot X^{-1}) \cdot f_{\boldsymbol{b}}(X)$
    - $h_7(X) := \left(f_{\boldsymbol{r}}(\nu \cdot X^{-1}) - (\nu \cdot X^{-1})^{6S+2S'+M+3N+3} \right) \cdot X^{6S+2S'+M+3N+3}$
    - $h_8(X) := f_{\boldsymbol{c}}(X^{-1}) \cdot (1 + f_{\boldsymbol{w}}(X) \cdot X)$
    - $h_9(X) := \left(\sum_{i=1}^{M} (\alpha \cdot X^{-1})^{i-1} \right) \cdot f_{\boldsymbol{x}}(X)$
    - $h_{10}(X) := f_{\boldsymbol{val}}(X^{-1}) \cdot f_{\boldsymbol{t}}(X)$
    - $h_{11}(X) := f_{\boldsymbol{c}}(X^{-1}) \cdot \left(\sum_{i=1}^{3N+4} (\beta \cdot X)^{i-1} \right)$
13. Compute

$$
\begin{aligned}
h(X) := {} & (h_1(X) - h_2(X)) + \omega \cdot (h_3(X) - h_4(X)) + \omega^2 \cdot (h_5(X) - h_6(X)) + \\
& \omega^3 \cdot h_7(X) + \omega^4 \cdot (h_8(X) - h_9(X)) + \omega^5 \cdot (h_{10}(X) - h_{11}(X)) + \\
& \omega^6 \cdot f_{\boldsymbol{w}}(X) \cdot X^{D-3N-3} + \omega^7 \cdot f_{\boldsymbol{c}}(X) \cdot X^{D-3N-4} + \omega^8 \cdot f_{\boldsymbol{z}}(X) \cdot X^{D-6S-2S'} + \\
& \omega^9 \cdot f_{\boldsymbol{t}}(X) \cdot X^{D-3S-S'} + \omega^{10} \cdot f_{\boldsymbol{r}}(X) \cdot X^{D-6S-2S'-M-3N-4}.
\end{aligned}
$$

14. Compute $\bar{h}_1(X) := (X^D \cdot h(X)) \bmod X^D$, $\bar{h}_2(X) := \lfloor h(X) \cdot X^{-1} \rfloor$ where $\lfloor \cdot \rfloor$ means removing all the monomials with negative powers.
15. $\mathsf{cm}_{h_1} := \mathsf{com}(\bar{h}_1(X), \mathsf{srs})$, $\mathsf{cm}_{h_2} := \mathsf{com}(\bar{h}_2(X), \mathsf{srs})$
16. $z := H_1(\mathsf{cm}_v, \mathsf{cm}_\sigma, \boldsymbol{x}, \mathsf{cm}_w, \mathsf{cm}_c, \mathsf{cm}_z, \mathsf{cm}_t, \mathsf{cm}_r, \mathsf{cm}_{h_1}, \mathsf{cm}_{h_2})$
17. $y_w := f_{\boldsymbol{w}}(\nu \cdot z^{-1})$, $y_z := f_{\boldsymbol{z}}(\nu \cdot z^{-1})$, $y_r := f_{\boldsymbol{r}}(\nu \cdot z^{-1})$, $y_c := f_{\boldsymbol{c}}(z^{-1})$, $y_{val} := f_{\boldsymbol{val}}(z^{-1})$

18. Compute

$$\begin{aligned}
\bar{g}(X) := \big(&y_w \cdot z^{6N+9} + \omega^6 \cdot z^{D-3N-3} - \omega^4 \cdot y_c \cdot z\big) \cdot f_{\boldsymbol{w}}(X) \\
&+(\omega \cdot y_z \cdot z^{3S+S'} + \zeta' \cdot \omega^2 \cdot (y_r \cdot \nu \cdot z^{-1} \\
&\quad -(\nu \cdot z^{-1})^{6S+2S'+M+3N+3} - y_r + 1) + \omega^8 \cdot z^{D-6S-2S'}) \cdot f_{\boldsymbol{z}}(X) \\
&+\left(\omega^9 \cdot z^{D-3S-S'} - \omega \cdot \frac{(\nu \cdot z^{-1})^{6S+2S'} - 1}{\nu \cdot z^{-1} - 1} \cdot z^{3S+S'} + \omega^5 \cdot y_{val}\right) \cdot f_{\boldsymbol{t}}(X) \\
&+\omega^7 \cdot z^{D-3N-4} \cdot f_{\boldsymbol{c}}(X) + \omega^{10} \cdot z^{D-6S-2S'-M-3N-4} \cdot f_{\boldsymbol{r}}(X) - \omega^2 \cdot y_r \cdot \zeta'' \cdot f_{\boldsymbol{\sigma}}(X) \\
&-z^{-D} \cdot \bar{h}_1(X) - z \cdot \bar{h}_2(X) \\
&+\Bigg(-y_w \cdot \nu^{3N+3} \cdot \frac{z^{6N+9} - z^{3N+6}}{z-1} \\
&\quad +\omega^2 \cdot (y_r \cdot \nu \cdot z^{-1} + 1 - (\nu \cdot z^{-1})^{6S+M+3N+3}) \cdot \zeta'' \cdot \frac{(\gamma \cdot z)^{6S+M+3N+3} - 1}{\gamma \cdot z - 1} \\
&\quad +\omega^2 \cdot (-y_r + y_r \cdot \nu \cdot z^{-1} + 1 - (\nu \cdot z^{-1})^{6S+2S'+M+3N+4}) \cdot \\
&\quad \left(\zeta \cdot \frac{z^{6S+2S'+M+3N+4} - 1}{z-1} + \zeta' \cdot z^{6S+2S'} \cdot \left(\frac{(\alpha \cdot z)^M - 1}{\alpha \cdot z - 1} + z^M \cdot \frac{(\beta \cdot z)^{3N+4} - 1}{\beta \cdot z - 1}\right)\right) \\
&\quad +\omega^3 \cdot (y_r \cdot z^{6S+2S'+M+3N+3} - \nu^{6S+2S'+M+3N+3}) - \omega^4 \cdot \frac{(\alpha \cdot z^{-1})^M - 1}{\alpha \cdot z^{-1} - 1} \cdot f_{\boldsymbol{x}}(z) \\
&\quad +\omega^4 \cdot y_c - \omega^5 \cdot y_c \cdot \frac{(\beta \cdot z)^{3N+4} - 1}{\beta \cdot z - 1}\Bigg)
\end{aligned}$$

19. $\mathsf{cm}_g := \mathsf{com}(\bar{g}(X), \mathsf{srs})$ computed by linearly combining cm_w, cm_z, cm_t, cm_c, cm_r, cm_σ, cm_{h_1}, cm_{h_2} and $[1]_1$.
20. Compute $(W, W', W'') := \mathsf{open}\left(\begin{array}{l}\{(\mathsf{cm}_w, y_w, f_{\boldsymbol{w}}(X)), (\mathsf{cm}_z, y_z, f_{\boldsymbol{z}}(X)), \\ \quad (\mathsf{cm}_r, y_r, f_{\boldsymbol{r}}(X))\}, \\ \{(\mathsf{cm}_c, y_c, f_{\boldsymbol{c}}(X)), (\mathsf{cm}_{val}, y_{val}, f_{\boldsymbol{val}}(X))\}, \\ \{\mathsf{cm}_g, 0, \bar{g}(X)\}, \\ \{\nu \cdot z^{-1}, z^{-1}, z\}\end{array}\right)$
21. Output $\pi := (\mathsf{cm}_w, \mathsf{cm}_c, \mathsf{cm}_z, \mathsf{cm}_t, \mathsf{cm}_r, \mathsf{cm}_{h_1}, \mathsf{cm}_{h_2}, y_w, y_z, y_r, y_c, y_{val}, W, W', W'')$

Verifier. The inputs are $[x]_2, \mathsf{vk}, \boldsymbol{x}$ and π.

1. Compute
   - $\alpha := H_1(\mathsf{cm}_v, \mathsf{cm}_\sigma, \boldsymbol{x}, \mathsf{cm}_w)$
   - $\beta := H_1(\mathsf{cm}_v, \mathsf{cm}_\sigma, \boldsymbol{x}, \mathsf{cm}_w, \mathsf{cm}_c)$
   - $\zeta := H_1(\mathsf{cm}_v, \mathsf{cm}_\sigma, \boldsymbol{x}, \mathsf{cm}_w, \mathsf{cm}_c, \mathsf{cm}_z, \mathsf{cm}_t)$
   - $\zeta' := H_2(\mathsf{cm}_v, \mathsf{cm}_\sigma, \boldsymbol{x}, \mathsf{cm}_w, \mathsf{cm}_c, \mathsf{cm}_z, \mathsf{cm}_t)$

- $\zeta'' := H_3(\mathsf{cm}_v, \mathsf{cm}_\sigma, \boldsymbol{x}, \mathsf{cm}_w, \mathsf{cm}_c, \mathsf{cm}_z, \mathsf{cm}_t)$
- $\nu := H_1(\mathsf{cm}_v, \mathsf{cm}_\sigma, \boldsymbol{x}, \mathsf{cm}_w, \mathsf{cm}_c, \mathsf{cm}_z, \mathsf{cm}_t, \mathsf{cm}_r)$
- $\omega := H_2(\mathsf{cm}_v, \mathsf{cm}_\sigma, \boldsymbol{x}, \mathsf{cm}_w, \mathsf{cm}_c, \mathsf{cm}_z, \mathsf{cm}_t, \mathsf{cm}_r)$
- $z := H_1(\mathsf{cm}_v, \mathsf{cm}_\sigma, \boldsymbol{x}, \mathsf{cm}_w, \mathsf{cm}_c, \mathsf{cm}_z, \mathsf{cm}_t, \mathsf{cm}_r, \mathsf{cm}_{h_1}, \mathsf{cm}_{h_2})$

2. Compute

$$
\begin{aligned}
\mathsf{cm}_g := \big(& y_w \cdot z^{6N+9} + \omega^6 \cdot z^{D-3N-3} - \omega^4 \cdot y_c \cdot z\big) \cdot \mathsf{cm}_w \\
&+(\omega \cdot y_z \cdot z^{3S+S'} + \zeta' \cdot \omega^2 \cdot (y_r \cdot \nu \cdot z^{-1} \\
&\quad -(\nu \cdot z^{-1})^{6S+2S'+M+3N+3} - y_r + 1) + \omega^8 \cdot z^{D-6S-2S'}) \cdot \mathsf{cm}_z \\
&+ \left(\omega^9 \cdot z^{D-3S-S'} - \omega \cdot \frac{(\nu \cdot z^{-1})^{6S+2S'} - 1}{\nu \cdot z^{-1} - 1} \cdot z^{3S+S'} + \omega^5 \cdot y_{val}\right) \cdot \mathsf{cm}_t \\
&+\omega^7 \cdot z^{D-3N-4} \cdot \mathsf{cm}_c + \omega^{10} \cdot z^{D-6S-2S'-M-3N-4} \cdot \mathsf{cm}_r - \omega^2 \cdot y_r \cdot \zeta'' \cdot \mathsf{cm}_\sigma \\
&-z^{-D} \cdot \mathsf{cm}_{h_1} - z \cdot \mathsf{cm}_{h_2} \\
&+ \Bigg(-y_w \cdot \nu^{3N+3} \cdot \frac{z^{6N+9} - z^{3N+6}}{z-1} \\
&\quad +\omega^2 \cdot (y_r \cdot \nu \cdot z^{-1} + 1 - (\nu \cdot z^{-1})^{6S+M+3N+3}) \cdot \zeta'' \cdot \frac{(\gamma \cdot z)^{6S+M+3N+3} - 1}{\gamma \cdot z - 1} \\
&\quad +\omega^2 \cdot (-y_r + y_r \cdot \nu \cdot z^{-1} + 1 - (\nu \cdot z^{-1})^{6S+2S'+M+3N+4}) \cdot \\
&\quad \left(\zeta \cdot \frac{z^{6S+2S'+M+3N+4} - 1}{z-1} + \zeta' \cdot z^{6S+2S'} \cdot \left(\frac{(\alpha \cdot z)^M - 1}{\alpha \cdot z - 1} + z^M \cdot \frac{(\beta \cdot z)^{3N+4} - 1}{\beta \cdot z - 1}\right)\right) \\
&\quad +\omega^3 \cdot (y_r \cdot z^{6S+2S'+M+3N+3} - \nu^{6S+2S'+M+3N+3}) - \omega^4 \cdot \frac{(\alpha \cdot z^{-1})^M - 1}{\alpha \cdot z^{-1} - 1} \cdot f_{\boldsymbol{x}}(z) \\
&\quad +\omega^4 \cdot y_c - \omega^5 \cdot y_c \cdot \frac{(\beta \cdot z)^{3N+4} - 1}{\beta \cdot z - 1}\Bigg)
\end{aligned}
$$

3. Output $\mathsf{vrfy}\left(\begin{array}{l}\{(\mathsf{cm}_y, y_y), (\mathsf{cm}_z, y_z), (\mathsf{cm}_r, y_r)\}, \{(\mathsf{cm}_c, y_c), (\mathsf{cm}_{val}, y_{val})\}, \\ \{(\mathsf{cm}_g, 0)\}, \{\nu \cdot z^{-1}, z^{-1}, z\}, \{W, W', W''\}, [x]_2\end{array}\right)$.

D.4 The zkSNARK for POV

The zkSNARK $\mathsf{VCProof/POV}$ generates a proof for the satisfiability of a fan-in-2 circuit with C_a addition gates, C_m multiplication gates, L wires, ℓ public inputs/outputs and K constants. Let $C = C_a + C_m$. The circuit C is represented in the following format:

- a vector $\boldsymbol{z} \in \mathbb{F}^K$ representing all the constants;
- an array of index pairs $\{(V_i, V_i')\}_{i=1}^L$ where each V_i or V_i' is an integer in $[3C + K + \ell + 2] \backslash \{C_m + 1, C_m + 2\}$. Each pair (V_i, V_i') represents a wire that connects two variables. More specifically:

- the left/right input and the output of the k'th addition gate are indexed by $k + C_m + 2, k + C_a + C_m + 2$ and $k + 2C_a + C_m + 2$ respectively;
- the left/right input and the output of the k'th multiplication gate are indexed by $k, k + 3C_a + 2C_m + 2$ and $k + 3C_a + C_m + 2$ respectively;
- the k'th public input/output is indexed by $k + 3C + 2$;
- the k'th constant is indexed by $k + 3C + \ell + 2$.

Given an instance $\boldsymbol{x} \in \mathbb{F}^\ell$, we say $\boldsymbol{x}$ satisfies the circuit $\mathsf{C} = (\boldsymbol{z}, \{(V_i, V_i')\}_{i=1}^L)$ if there exists $\boldsymbol{a} \in \mathbb{F}^{3C_a}, \boldsymbol{m} \in \mathbb{F}^{3C_m}$ such that

- $\boldsymbol{a}_{[1..C_a]} + \boldsymbol{a}_{[C_a+1..2C_a]} = \boldsymbol{a}_{[2C_a+1..3C_a]}$
- $\boldsymbol{m}_{[1..C_m]} \circ \boldsymbol{m}_{[C_m+1..2C_m]} = \boldsymbol{m}_{[2C_m+1..3C_m]}$
- the vector $\boldsymbol{u} = \boldsymbol{m}_{[1..C_m]} \| \mathbf{0}^2 \| \boldsymbol{a} \| \boldsymbol{m}_{[2C_m+1..3C_m]} \| \boldsymbol{m}_{[C_m+1..2C_m]} \| \boldsymbol{x} \| \boldsymbol{z}$ satisfies that for every $i \in [L]$, $\boldsymbol{u}_{[V_i]} = \boldsymbol{u}_{[V_i']}$.

Setup. Output $\mathsf{srs} := \mathsf{gen}(D)$ for $D \geq 5C + C_a + 5$.

Preprocessing. On input a circuit $\mathsf{C} = (\boldsymbol{z}, \{(V_i, V_i')\}_{i=1}^L)$, compute the vector $\boldsymbol{\sigma} \in \mathbb{F}^{3C+K+\ell+2}$ as follows:

1. Initialize $\boldsymbol{\sigma} := \boldsymbol{\gamma}^{3C+K+\ell+2}$.
2. For every $i \in [L]$, swap the values of $\boldsymbol{\sigma}_{[V_i]}$ and $\boldsymbol{\sigma}_{[V_i']}$.

We remark that the above algorithm works because the wires of a valid circuit does not contain any cycles.

Compute $\mathsf{cm}_z := \mathsf{com}(f_{\boldsymbol{z}}(X), \mathsf{srs})$ and $\mathsf{cm}_\sigma := \mathsf{com}(f_{\boldsymbol{\sigma}}(X), \mathsf{srs})$. Finally, output $\mathsf{pk} := (\boldsymbol{\sigma}, \boldsymbol{z}, \mathsf{cm}_\sigma, \mathsf{cm}_z)$ and $\mathsf{vk} := (\mathsf{cm}_\sigma, \mathsf{cm}_z)$.

Prover. The inputs are $\mathsf{srs}, \mathsf{pk}$, instance $\boldsymbol{x}$ and witnesses $\boldsymbol{a} \in \mathbb{F}^{3C_a}, \boldsymbol{m} \in \mathbb{F}^{3C_m}$.

1. $\boldsymbol{\delta}_1, \boldsymbol{\delta}_2 \xleftarrow{\$} \mathbb{F}^2$.
2. $\boldsymbol{v} := \boldsymbol{m}_{[1..C_m]} \| \boldsymbol{\delta}_1 \| \boldsymbol{a} \| \boldsymbol{m}_{[2C_m+1..3C_m]} \| \boldsymbol{m}_{[C_m+1..2C_m]} \in \mathbb{F}^{3C+2}$.
3. $\mathsf{cm}_v := \mathsf{com}(f_{\boldsymbol{v}}(X), \mathsf{srs})$.
4. $\zeta := H_1(\mathsf{cm}_\sigma, \mathsf{cm}_z, \boldsymbol{x}, \mathsf{cm}_v)$, compute ζ' and ζ'' with the same inputs by using H_2 and H_3 instead.
5. $\boldsymbol{c} := \zeta \cdot \mathbf{1}^{3C+K+\ell+2} + \zeta' \cdot (\boldsymbol{v} \| \boldsymbol{x} \| \boldsymbol{z}) + \zeta'' \cdot \boldsymbol{\gamma}^{3C+K+\ell+2}$.
6. $\boldsymbol{b} := \zeta \cdot \mathbf{1}^{3C+K+\ell+2} + \zeta' \cdot (\boldsymbol{v} \| \boldsymbol{x} \| \boldsymbol{z}) + \zeta'' \cdot \boldsymbol{\sigma}$.
7. $\boldsymbol{r} := \left(\prod_{j=1}^i \boldsymbol{c}_{[j]} / \boldsymbol{b}_{[j]} \right)_{i=1}^{3C+K+\ell+2} \| \boldsymbol{\delta}_2 \in \mathbb{F}^{3C+K+\ell+4}$, $\mathsf{cm}_r := \mathsf{com}(f_{\boldsymbol{r}}(X), \mathsf{srs})$.

8. $\nu := H_1(\mathsf{cm}_\sigma, \mathsf{cm}_z, \boldsymbol{x}, \mathsf{cm}_v, \mathsf{cm}_r)$, $\omega := H_2(\mathsf{cm}_\sigma, \mathsf{cm}_z, \boldsymbol{x}, \mathsf{cm}_v, \mathsf{cm}_r)$
9. Compute
   - $h_1(X) := \left(\sum_{i=1}^{C_a} (\nu X^{-1})^{C_m+2C_a+1+i}\right) \cdot f_{\boldsymbol{v}}(X) \cdot (X^{2C_a} + X^{C_a} - 1)$
   - $h_2(X) := f_{\boldsymbol{v}}(\nu X^{-1}) \cdot (\nu X^{-1})^{3C_a+2C_m+2} \cdot f_{\boldsymbol{v}}(X)$
   - $h_3(X) := \left(\sum_{i=1}^{C_m} (\nu X^{-1})^{2C_m+3C_a+1+i}\right) \cdot f_{\boldsymbol{v}}(X) \cdot X^{C_m}$
   - $h_4(X) := \left(f_{\boldsymbol{r}}(\nu \cdot X^{-1}) \cdot \nu X^{-1} + 1 - (\nu X^{-1})^{3C+K+\ell+2}\right) \cdot f_{\boldsymbol{c}}(X)$
   - $h_5(X) := f_{\boldsymbol{r}}(\nu \cdot X^{-1}) \cdot f_{\boldsymbol{b}}(X)$
   - $h_6(X) := \left(f_{\boldsymbol{r}}(\nu \cdot X^{-1}) - (\nu \cdot X^{-1})^{3C+K+\ell+1}\right) \cdot X^{3C+K+\ell+1}$
10. Compute

$$\begin{aligned} h(X) := h_1(X) &+ \omega \cdot (h_2(X) - h_3(X)) + \omega^2 \cdot (h_4(X) - h_5(X)) + \omega^3 \cdot h_6(X) \\ &+ \omega^4 \cdot f_{\boldsymbol{v}}(X) \cdot X^{D-3C-2} + \omega^5 \cdot f_{\boldsymbol{r}}(X) \cdot X^{D-3C-K-\ell-4}. \end{aligned}$$

11. Compute $\bar{h}_1(X) := (X^D \cdot h(X)) \bmod X^D$, $\bar{h}_2(X) := \lfloor h(X) \cdot X^{-1} \rfloor$ where $\lfloor \cdot \rfloor$ means removing all the monomials with negative powers.
12. $\mathsf{cm}_{h_1} := \mathsf{com}(\bar{h}_1(X), \mathsf{srs})$, $\mathsf{cm}_{h_2} := \mathsf{com}(\bar{h}_2(X), \mathsf{srs})$.
13. $z := H_1(\mathsf{cm}_\sigma, \mathsf{cm}_z, \boldsymbol{x}, \mathsf{cm}_v, \mathsf{cm}_r, \mathsf{cm}_{h_1}, \mathsf{cm}_{h_2})$.
14. $y_v := f_{\boldsymbol{v}}(\nu \cdot z^{-1})$, $y_r := f_{\boldsymbol{r}}(\nu \cdot z^{-1})$.
15. Compute

$$\begin{aligned} \bar{g}(X) := \Bigg(& \omega^4 \cdot z^{D-3C-2} + (\nu z^{-1})^{C_m+2C_a+2} \cdot \frac{(\nu z^{-1})^{C_a} - 1}{\nu z^{-1} - 1} \cdot ((\nu \cdot z^{-1})^{2C_a} + (\nu \cdot z^{-1})^{C_a} - 1) \\ & + \omega \cdot \left(y_v \cdot (\nu z^{-1})^{3C_a+2C_m+2} - \frac{(\nu z^{-1})^{3C+2} - (\nu z^{-1})^{2C_m+3C_a+2}}{\nu z^{-1} - 1} \cdot z^{C_m} \right) + \\ & + \omega^2 \cdot \zeta' \cdot \left(\left(y_r \cdot \nu \cdot z^{-1} + 1 - (\nu z^{-1})^{3C+K+\ell+2} \right) \cdot z^{C_m} - y_r \right) \Big) \cdot f_{\boldsymbol{v}}(X) \\ & + \omega^2 \cdot \zeta' \cdot z^{3C+\ell+2} \cdot \left(\left(y_r \cdot \nu \cdot z^{-1} + 1 - (\nu z^{-1})^{3C+K+\ell+2} \right) \cdot z^{C_m} - y_r \right) \cdot f_{\boldsymbol{z}}(X) \\ & + \omega^2 \cdot \zeta'' \cdot f_{\boldsymbol{\sigma}}(X) + \omega^5 \cdot z^{D-3C-K-\ell-4} \cdot f_{\boldsymbol{r}}(X) - z^{-D} \cdot \bar{h}_1(X) - z \cdot \bar{h}_2(X) \\ & + \left(\omega^2 \cdot \left(\zeta'' \cdot \frac{(\gamma \cdot z)^{3C+K+\ell+2} - 1}{\gamma \cdot z - 1} + \zeta \cdot \left(\frac{(\nu z^{-1})^{3C+K+\ell+2} - 1}{\nu z^{-1} - 1} - \frac{z^{3C+K+\ell+2} - 1}{z - 1} \right) \right) \right. \\ & \left. + \omega^3 \cdot (y_r - (\nu z^{-1})^{3C+K+\ell+1}) \cdot z^{3C+K+\ell+1} \right) \end{aligned}$$

16. $\mathsf{cm}_g := \mathsf{com}(\bar{g}(X), \mathsf{srs})$ computed by linearly combining cm_v, cm_z, cm_r, cm_σ, cm_{h_1}, cm_{h_2} and $[1]_1$.
17. Compute $(W, W') := \mathsf{open}\left(\begin{array}{l} \{(\mathsf{cm}_v, y_v, f_{\boldsymbol{v}}(X)), (\mathsf{cm}_r, y_r, f_{\boldsymbol{r}}(X))\}, \\ \{\mathsf{cm}_g, 0, \bar{g}(X)\}, \\ \{\nu \cdot z^{-1}, z\} \end{array} \right)$
18. Output $\pi := (\mathsf{cm}_v, \mathsf{cm}_r, \mathsf{cm}_{h_1}, \mathsf{cm}_{h_2}, y_v, y_r, W, W')$

Verifier. The inputs are $[x]_2$, instance $\boldsymbol{x} \in \mathbb{F}^\ell$ and the proof π.

1. Compute
   - $\zeta := H_1(\mathsf{cm}_\sigma, \mathsf{cm}_z, \boldsymbol{x}, \mathsf{cm}_v)$
   - $\zeta' := H_2(\mathsf{cm}_\sigma, \mathsf{cm}_z, \boldsymbol{x}, \mathsf{cm}_v)$
   - $\zeta'' := H_3(\mathsf{cm}_\sigma, \mathsf{cm}_z, \boldsymbol{x}, \mathsf{cm}_v)$
   - $\nu := H_1(\mathsf{cm}_\sigma, \mathsf{cm}_z, \boldsymbol{x}, \mathsf{cm}_v, \mathsf{cm}_r)$
   - $\omega := H_2(\mathsf{cm}_\sigma, \mathsf{cm}_z, \boldsymbol{x}, \mathsf{cm}_v, \mathsf{cm}_r)$
   - $z := H_1(\mathsf{cm}_\sigma, \mathsf{cm}_z, \boldsymbol{x}, \mathsf{cm}_v, \mathsf{cm}_r, \mathsf{cm}_{h_1}, \mathsf{cm}_{h_2})$
2. Compute

$$
\begin{aligned}
\mathsf{cm}_g := \Bigg(& \omega^4 \cdot z^{D-3C-2} + (\nu z^{-1})^{C_m+2C_a+2} \cdot \frac{(\nu z^{-1})^{C_a} - 1}{\nu z^{-1} - 1} \cdot ((\nu \cdot z^{-1})^{2C_a} + (\nu \cdot z^{-1})^{C_a} - 1) \\
& + \omega \cdot \left(y_v \cdot (\nu z^{-1})^{3C_a+2C_m+2} - \frac{(\nu z^{-1})^{3C+2} - (\nu z^{-1})^{2C_m+3C_a+2}}{\nu z^{-1} - 1} \cdot z^{C_m} \right) + \\
& + \omega^2 \cdot \zeta' \cdot \left(\left(y_r \cdot \nu \cdot z^{-1} + 1 - (\nu z^{-1})^{3C+K+\ell+2} \right) \cdot z^{C_m} - y_r \right) \Bigg) \cdot \mathsf{cm}_v \\
& + \omega^2 \cdot \zeta' \cdot z^{3C+\ell+2} \cdot \left(\left(y_r \cdot \nu \cdot z^{-1} + 1 - (\nu z^{-1})^{3C+K+\ell+2} \right) \cdot z^{C_m} - y_r \right) \cdot \mathsf{cm}_z \\
& + \omega^2 \cdot \zeta'' \cdot \mathsf{cm}_\sigma + \omega^5 \cdot z^{D-3C-K-\ell-4} \cdot \mathsf{cm}_r - z^{-D} \cdot \mathsf{cm}_{h_1} - z \cdot \mathsf{cm}_{h_2} \\
& + \Bigg[\omega^2 \cdot \left(\zeta'' \cdot \frac{(\gamma \cdot z)^{3C+K+\ell+2} - 1}{\gamma \cdot z - 1} + \zeta \cdot \left(\frac{(\nu z^{-1})^{3C+K+\ell+2} - 1}{\nu z^{-1} - 1} - \frac{z^{3C+K+\ell+2} - 1}{z - 1} \right) \right) \\
& + \omega^3 \cdot (y_r - (\nu z^{-1})^{3C+K+\ell+1}) \cdot z^{3C+K+\ell+1} \Bigg]_1
\end{aligned}
$$

3. Output $\mathsf{vrfy}\left(\begin{matrix} \{(\mathsf{cm}_v, y_v), (\mathsf{cm}_r, y_r)\}, \{(\mathsf{cm}_g, 0)\}, \\ \{\nu \cdot z^{-1}, z\}, \{W, W'\}, [x]_2 \end{matrix} \right)$.

E Concrete Efficiency

Here we present a more thorough comparison between our works and other zkSNARKs. We first compare the communication costs, including the proof sizes and public parameter sizes, and the asymptotic complexities, between both versions of VCProof and a wide range of concurrent works. After that, we compare the KZG version of VCProof and the other constant-verifier zkSNARKs with respect to the concrete cost.

Table 3 and 4 compare the costs of our work with a wide range of zkSNARKs with respect to the communication costs and asymptotic computation costs. For Aurora, Fractal and Spartan, their proof sizes are an order of magnitude larger than the others, and we ignore the concrete values and only leave an asymptotic estimate.

Note also that the indexers of the VCProof zkSNARKs are all linear, as opposed to the quasi-linear indexer of other zkSNARKs. This discrepancy stems from using the monomial base, and as a consequence, the indexer does not need the FFT to transform the target vector into a polynomial.

Table 3. Comparison of communication costs between the VCProof schemes and other zkSNARKs. The superscript 1 for VCProof protocols refers to the KZG version, and the superscript 2 refers to the DARK version. The two versions of PLONK are respectively the short-proof version and the fast-prover version.

relation	zkSNARK	proof size	pk size	vk size	prover setup	verifier setup
R1CS	Marlin [4]	$21\mathbb{F}+13\mathbb{G}_1$	$9S\mathbb{F}$	$9\mathbb{G}_1$	$(6S+6)\mathbb{G}_1$	$1\mathbb{G}_2$
	Aurora [10]	$O(\log^2(S))$	-	-	-	-
	Fractal [8]	$O(\log^2(S))$	$9S\mathbb{F}$	$9\mathbb{F}$	-	-
	Groth16 [2]	$2\mathbb{G}_1+1\mathbb{G}_2$	$(3+M+2N)\mathbb{G}_1$ $+(3+N)\mathbb{G}_2$	$(\ell+1)\mathbb{G}_1$ $+3\mathbb{G}_2$	-	-
	Spartan [9]	$O(\log^2(S))$	$O(S)$	$O(1)$	$O(S)$	$O(1)$
	VCProof/R1CS[1]	$5\mathbb{F}+11\mathbb{G}_1$	$(3M+N$ $+9S+3)\mathbb{F}$	$2\mathbb{G}_1$	$(6S+3M+$ $N+7)\mathbb{G}_1$	$1\mathbb{G}_2$
	VCProof/R1CS[2]	$(8+2\log(N))\mathbb{G}_U$ $+(5+4\log(N))\mathbb{F}$	$(3M+N$ $+9S+3)\mathbb{F}$	$2\mathbb{G}_U$	$(6S+3M+$ $N+7)\mathbb{G}_U$	$1\mathbb{G}_U$
HPR	Sonic [5]	$16\mathbb{F}+20\mathbb{G}_1$	$O(N)$	$6\mathbb{G}_1$	$O(N)$	$1\mathbb{G}_2$
	Supersonic [6]	$(7+2\log(C))\mathbb{G}_U$ $+(2+3\log(C))\mathbb{F}$	$8C\mathbb{F}$	$8\mathbb{G}_U$	$8C\mathbb{G}_U$	$1\mathbb{G}_U$
	BulletProof [17]	$2\log(N)\mathbb{G}$	-	-	-	-
	VCProof/HPR[1]	$5\mathbb{F}+10\mathbb{G}_1$	$(3N+M$ $+9S+3)\mathbb{F}$	$2\mathbb{G}_1$	$(6S+M$ $+3N+7)\mathbb{G}_1$	$1\mathbb{G}_2$
	VCProof/HPR[2]	$(7+2\log(N))\mathbb{G}_U$ $+(5+4\log(N))\mathbb{F}$	$(3N+M$ $+9S+3)\mathbb{F}$	$2\mathbb{G}_U$	$(6S+M+$ $3N+7)\mathbb{G}_U$	$1\mathbb{G}_U$
Fan-in-2 Circuit	PLONK[1] [3]	$7\mathbb{F}+7\mathbb{G}_1$	$6C\mathbb{F}$	$6\mathbb{G}_1$	$3C\mathbb{G}_1$	$1\mathbb{G}_2$
	PLONK[2] [3]	$7\mathbb{F}+9\mathbb{G}_1$	$8C\mathbb{F}$	$8\mathbb{G}_1$	$C\mathbb{G}_1$	$1\mathbb{G}_2$
	VCProof/POV[1]	$2\mathbb{F}+6\mathbb{G}_1$	$(\ell+2K+3C$ $+2)\mathbb{F}$	$2\mathbb{G}_1$	$(5C+C_a$ $+4)\mathbb{G}_1$	$1\mathbb{G}_2$
	VCProof/POV[2]	$(5+2\log(N))\mathbb{G}_U$ $(2+3\log(N))\mathbb{F}$	$(\ell+2K+3C$ $+2)\mathbb{F}$	$2\mathbb{G}_U$	$(5C+C_a$ $+4)\mathbb{G}_U$	$1\mathbb{G}_U$

Table 5 compares the concrete prover and verifier computation costs of the KZG version of the VCProof with other constant-verifier zkSNARKs, namely PLONK, Sonic, Marlin and Groth16. The works of Marlin, Groth16 and Sonic do not provide the concrete number of FFTs, so we only present an asymptotic estimate for the number of field operations. Although the verifiers also need field operations, we neglect them as their costs are negligible compared to pairings and group exponentiations.

Table 4. Comparison of computational costs between the VCProof schemes and other zkSNARKs. The superscript 1 for VCProof protocols refers to the KZG version, and the superscript 2 refers to the DARK version. The two versions of PLONK are respectively the short-proof version and the fast-prover version.

relation	zkSNARK	prove	verify	index	setup	univ.	transp.
R1CS	Marlin [4]	$O(S\log(S))$	$O(1)$	$O(S\log(S))$	$O(S)$	✓	
	Aurora [10]	$O(S\log(S))$	$O(S)$	-	-	✓	✓
	Fractal [8]	$O(S\log(S))$	$O(\log(S))$	$O(S\log(S))$	-	✓	✓
	Groth16 [2]	$O(N\log(N))$	$O(1)$	$O(S\log(S))$	-		
	Spartan [9]	$O(S)$	$O(\log^2(S))$	$O(S\log(S))$	$O(S)$	✓	✓
	VCProof/R1CS[1]	$O(S\log(S))$	$O(1)$	$O(S)$	$O(S)$	✓	
	VCProof/R1CS[2]	$O(S\log(S)$	$O(\log(S))$	$O(S)$	$O(S)$	✓	✓
HPR	Sonic [5]	$O(N\log(N))$	$O(1)$	$O(N\log(N))$	$O(N)$	✓	
	Supersonic [6]	$O(N\log(N))$	$O(\log(N))$	$O(N\log(N))$	$O(N)$	✓	✓
	BulletProof [17]	$O(N\log(N))$	$O(N)$	-	-	✓	✓
	VCProof/HPR[1]	$O(S\log(S))$	$O(1)$	$O(S)$	$O(S)$	✓	
	VCProof/HPR[2]	$O(S\log(S))$	$O(\log(S))$	$O(S)$	$O(S)$	✓	✓
Fan-in-2 Circuit	PLONK[1] [3]	$O(C\log(C))$	$O(1)$	$O(C\log(C))$	$O(C)$	✓	
	PLONK[2] [3]	$O(C\log(C))$	$O(1)$	$O(C\log(C))$	$O(C)$	✓	
	VCProof/POV[1]	$O(C\log(C))$	$O(1)$	$O(C)$	$O(C)$	✓	
	VCProof/POV[2]	$O(C\log(C))$	$O(\log(C))$	$O(C)$	$O(C)$	✓	✓

Table 5. Concrete comparison of prover and verifier computation costs between the constant-verifier zkSNARKs, i.e., VCProof (KZG), PLONK, Sonic, Marlin and Groth16.

relation	zkSNARK	prove	verify
R1CS	Groth16 [2]	$(3S+N-\ell)\mathbb{G}_1$-exp + $N\mathbb{G}_2$-exp + $O(S\log(S))\mathbb{F}$-mul	3 Pairings + $\ell\mathbb{G}_1$-exp
	Marlin [4]	$21S\mathbb{G}_1$-exp + $O(S\log(S))\mathbb{F}$-mul	2 Pairings
	VCProof/R1CS	$(48S+29N-\ell+36)\mathbb{G}_1$-exp 4 FFTs of size $12S+8N$ 2 FFTs of size $12S$ 4 FFTs of size $6S$ or $6N$ 2 FFTs of size $2N$	2 Pairings + $20\mathbb{G}_1$-exp
HPR	Sonic [5]	$273N\mathbb{G}_1$-exp + $O(N\log(N))\mathbb{F}$-mul	13 Pairings
	VCProof/HPR	$(46S+30N+33)\mathbb{G}_1$-exp 4 FFTs of size $12S+8N$ 2 FFTs of size $12S$ 6 FFTs of size $6S$ or $6N$	2 Pairings + $19\mathbb{G}_1$-exp
Fan-in-2 Circuit	PLONK[1] [3]	$11C\mathbb{G}_1$-exp 8 FFTs of size $4C_m$ 5 FFTs of size $2C_m$ 12 FFTs of size C_m	2 Pairings + $16\mathbb{G}_1$-exp
	PLONK[2] [3]	$9C\mathbb{G}_1$-exp + same FFTs as above	2 Pairings + $18\mathbb{G}_1$-exp
	VCProof/POV	$(27C+3C_a+3K+3\ell+25)\mathbb{G}_1$-exp 6 FFTs of size $6C$	2 Pairings + $12\mathbb{G}_1$-exp

Regarding the prover cost, VCProof/HPR outperforms Sonic while the number of group exponentiations in VCProof/R1CS and VCProof/POV are 2-5 times larger than the other works.

We remark that the FFTs in VCProof are involved only in polynomial multiplications. Apart from the polynomial multiplications, our prover costs are linear to the problem size. In contrast, in other zkSNARKs, the FFTs are pervasive as they are needed every time the prover or the indexer commits to a polynomial. The usage of FFTs also restricts the choice of finite fields to those with smooth multiplicative subgroups, which is the case for almost all existing zkSNARKs.