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

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

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

Abstract. The construction of zkSNARKs involves designing a Polynomial IOP (PIOP) that matches with the constraint system for which it proves membership. Designing this PIOP 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 this mismatch, we propose a new methodology for designing PIOP, which first designs a middle layer protocol matching the constraint system, called the Vector Oracle protocol, and then compiles it into a PIOP. 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 than PIOPs. The Vector-Oracle-to-PIOP compilation procedure is protocol-independent, allowing us to present and optimize it as a standalone component, leading to a series of improvements.

We apply our methodology to construct three zkSNARKs, each targeting a constraint system: the Rank-1 Constraint System (R1CS), the Hadamard Product Relation (HPR), and the PLONK circuit. All three zkSNARKs achieve shorter proofs and smaller or identical verification costs compared to the state-of-the-art constructions targeting the same constraint systems. Specifically, VOR1CS defeats Marlin in proof size; VOHPR and VOPLONK outperform Sonic and PLONK, respectively, in both proof sizes and verification costs. In particular, the proof of VOPLONK has only two field elements and seven 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-

⋆ Partially supported by Cryptape.
⋆⋆ 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 costs 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

polynomial product. The ubiquity 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- and Hadamard-product checks, and then implements this protocol as a PIOP protocol. Figure 1 illustrates the proposed modification to the zkSNARK construction workflow.

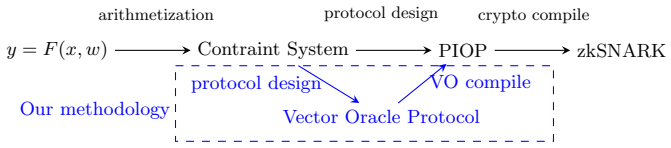

Fig. 1. The zkSNARK construction workflow. The Vector Oracle protocol is more intuitive to design than PIOP, and the VO-to-PIOP compilation is independent of the constraint system.

This new construction method, named *VOProof* for "Vector Oracle Proof", has the following two-fold advantage. First, a Vector Oracle (VO) 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 VO 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 the flexible combination of the VO protocol and the polynomial representation of the vector operations, thus opening a large design space for optimization. We formalize this method, explore the design space, and show that the new method 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 design by explicitly separating the constraint-system-specific protocol logic from the polynomial-related techniques. The protocol logic is described via a VO protocol (Sect. 3), where the parties have access to a vector oracle. The vector oracle stores vectors submitted from the prover and receives queries from the verifier to manipulate these vectors. The verifier may also query the oracle to check vector identities involving Hadamard products and inner products.

We compare designing zkSNARKs using PIOP directly and using the VO model as follows.

– *Equivalence with PIOP:* a VO protocol can be compiled to a PIOP protocol targeting the same constraint system and vice versa, i.e., VO protocols can prove the same set of constraint systems as PIOP.
– *Straightforward Design:* VO protocols are simpler to design than PIOPs, due to the intuitive interface provided by the vector oracle.
– *New Optimization Techniques:* zkSNARKs from VO protocols are easier to optimize due to the simplicity and the separation of the protocol logics from the compilation. In fact, the VO model uncovers the optimizations of vector concatenation and the batching of all the Hadamard- and inner-product checks into a single polynomial equation. In prior constructions, these optimizations are tricky to apply and thus left unnoticed.
– *Zero-Knowledge:* the zero-knowledge of VO-based SNARKs is ensured by the compilation and no longer a duty of the protocol designer.
– *Expressiveness:* the PIOPs compiled from VO protocols are only a subset of all zero-knowledge PIOPs, i.e., the VO model is *not* expressive enough to capture all PIOP protocols. Despite the limited expressiveness, VO protocols can prove all the constraint systems that PIOPs are capable of.

Implementing VO Protocols for Influential Circuit-based Constraint Systems. We present VOProof, a collection of three VO protocols targeting three circuit-based constraint systems, respectively. In Sect. 4, we construct protocols for R1CS (Rank-1 Constraint System) and HPR (Hadamard Product Relation). In Sect. 5, we provide a protocol for the constraint system of PLONK [17]. We name these protocols VOR1CS, VOHPR and VOPLONK. These protocols are constructed in a modular manner by composing building-block protocols. These building blocks are independently useful in constructing specialized-purpose SNARKs, e.g., for verifying the matrix-vector product.

New zkSNARKs with Shorter Proofs and Smaller Verification Costs. We present zkSNARKs with competitive efficiency compared to the state of the art (Sect. 6). We first compile VOProof to PIOPs using our VO-to-PIOP compiler, then transform the PIOPs to universal-setup preprocessing zkSNARKs via the KZG [14] polynomial commitment. Compared to the state-of-the-art constructions for the same constraint systems, all VOProof zkSNARKs reduce the proof sizes by half or more, while two of them also lower the verification costs. In particular, the proof of VOPLONK has only two field elements and seven group elements, becoming the shortest among all known universal-setup zkSNARKs.

The efficiency improvements result from (1) the vector concatenation technique applied in the VO 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 proving cost remains comparable (increased by two to three times) to the concurrent constructions due to other optimizations. For comparison, we also analyzed the efficiency of the VOProof zkSNARKs without using the vector concatenation. These zkSNARKs have the same or better performances in both proving efficiencies and proof sizes compared to prior works.

1.1 Related Works

We classify zkSNARKs into three groups, based on how they achieve *succinctness*—the "S" in zkSNARK. The first group includes BulletProofs [18] 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 [19], 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 introduces preprocessing, allowing the verifier to read a short digest instead of the complete circuit. Spartan [9] and Fractal [8] do not require trusted setups and Fractal is proved post-quantum secure [20]. Pinocchio [21] 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 [22] 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. 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 (entry-wise) 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 $\boldsymbol{\alpha}^n$ where $\alpha \in \mathbb{F}$. In particular, $\mathbf{1}^n$ and $\mathbf{0}^n$ are the all-one and all-zero vectors of size n. We also use *unit vectors* $\boldsymbol{e}_i$ that has a single one at position i and zeros anywhere else.

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 matrix-vector product is denoted by either $\boldsymbol{M}\boldsymbol{v}$ 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 less than 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 $f_{\boldsymbol{v}}(X) \in \mathbb{F}^{<d}[X]$ be the polynomial that uses the elements of $\boldsymbol{v}$ as coefficients, i.e., $f_{\boldsymbol{v}}(X) = \sum_{i=1}^{d} \boldsymbol{v}_{[i]} X^{i-1}$.

Finally, we introduce the indexed relation, a convenient notion for defining protocols with a preprocessing procedure. 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 $\mathcal{L}(\mathcal{R}) := \{(\mathtt{i}, \mathtt{x}) : \exists \mathtt{w}, (\mathtt{i}, \mathtt{x}, \mathtt{w}) \in \mathcal{R}\}$.

2.2 Interactive Proof for Indexed Relations

This work focuses on protocols that admit a preprocessing procedure which, on inputting an index of the relation, produces helpful information for the prover and the verifier in the protocol.

Definition 1 (Preprocessing Interactive Proof). *A* preprocessing interactive proof *is a tuple of PPT algorithms* $(\mathsf{I}, \mathsf{P}, \mathsf{V})$ *named the indexer, the prover, and the verifier, respectively:*

1. I *takes input* $\mathtt{i}$ *and outputs helpful information* $\mathtt{i}_P$ *and* $\mathtt{i}_V$ *for the prover and the verifier, respectively;*
2. P *takes inputs* $\mathtt{i}_P, \mathtt{i}_V, \mathtt{x}, \mathtt{w}$ *and* V *takes inputs* $\mathtt{i}_V, \mathtt{x}$*; they interact with each other; in the end, the verifier outputs one bit* b*.*

The above procedure is denoted by $b \leftarrow \langle \mathsf{I}(\mathtt{i}), \mathsf{P}(\mathtt{x}, \mathtt{w}), \mathsf{V}(\mathtt{x}) \rangle$*.*

We say $(\mathsf{I}, \mathsf{P}, \mathsf{V})$ *verifies the relation* $\mathcal{R}$ *with completeness error* ε_c *if for any* $(\mathtt{i}, \mathtt{x}, \mathtt{w}) \in \mathcal{R}$*,*

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

and soundness error ε_s *if for any* $(\mathtt{i}, \mathtt{x}) \notin \mathcal{L}(\mathcal{R})$ *and any unbounded algorithm* P^**,*

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

We say $(\mathsf{I}, \mathsf{P}, \mathsf{V})$ *is* public-coin *if all the randomnesses used by* V *are public coins, i.e., they are sent to the prover immediately after they are read by the verifier from the random tape.*

The transcript *of an execution of* $(\mathsf{I}, \mathsf{P}, \mathsf{V})$ *is the string consisting of* $\mathtt{i}_V$ *and all the messages exchanged between* P *and* V*. The transcript is denoted by* $\mathsf{tr}\langle \mathsf{I}(\mathtt{i}), \mathsf{P}(\mathtt{x}, \mathtt{w}), \mathsf{V}(\mathtt{x}) \rangle$*.*

For convenience, whenever we mention "interactive proof", we are referring to "preprocessing interactive proof", unless otherwise stated. Similarly, by "PIOP" and "zkSNARK", we refer to the preprocessing versions by default.

The notion *zero-knowledge (ZK)* requires that any verifier cannot acquire any information by interacting with the honest prover. This notion is formalized by a simulator that samples the transcript indistinguishably from those of honest executions. In this paper, we focus on *honest-verifier statistical zero-knowledge (HVSZK)*, which only requires that the simulator exists for the honest verifier. This version of ZK 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 preprocessing interactive proof* $(\mathsf{I}, \mathsf{P}, \mathsf{V})$ *for the indexed relation* $\mathcal{R}$ *is statistical honest-verifier zero-knowledge (HVSZK) if there exists a PPT algorithm* S *such that for any* $(\mathtt{i}, \mathtt{x}, \mathtt{w}) \in \mathcal{R}$*, the statistical distance between the*

distributions of the following two random variables tr *and* tr' *is bounded by a negligible* ε*:*

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

where the distributions are over all random coins. We say S *simulates the transcript of this protocol with statistical distance* ε*.*

2.3 Polynomial IOP

A *PIOP (Polynomial Interactive Oracle Proof)* protocol [6] is an interactive proof where, unlike in ordinary interactive proofs, the prover and the indexer may send *polynomial oracles* to the verifier. A polynomial oracle encapsulates a polynomial, say $f(X)$, and replies $f(z)$ when queried with any $z \in \mathbb{F}$. We denote the polynomial oracle for $f(X)$ by $[f(X)]$.

Definition 3 (Preprocessing PIOP). *A preprocessing PIOP with degree bound* D *is a public-coin interactive proof* $(\mathsf{I}, \mathsf{P}, \mathsf{V})$*, except:*

- I *and* P *do not send messages to* V*; instead, they output polynomials of degree less than* D*;*
- *for every polynomial* $f(X)$ *output by* I *or* P*,* V *has access to the* evaluation oracle *of* $f(X)$*, denoted by* $[f(X)]$*. Specifically, the verifier may query* $[f(X)]$ *with arbitrary* $z \in \mathbb{F}$ *and receives the reply* $y = f(z)$*.*

The notions *completeness* and *soundness* follow directly from those of interactive proofs. However, the definition of HVSZK should be handled carefully, as the verifier no longer reads the entire polynomials but only their evaluations at a few points. Moreover, the randomness for computing the evaluation queries should also appear in the transcript, although they do not appear in the verifier messages. Therefore, we redefine the transcript of a PIOP execution as the string consisting of 1) $\mathbb{i}_V$, 2) all the verifier randomnesses, and 3) the replies from the polynomial oracles. With this new definition of the transcript, the definition of HVSZK of PIOP is the same as in Definition 2.

2.4 The zkSNARK Construction Workflow

Here we illustrate the zkSNARK construction workflow (Fig. 1) in more detail. The starting point of this workflow is a computation, usually represented by an *arithmetic circuit* C that computes a function over $\mathbb{F}$.

The first step of this workflow, called *arithmetization*, transforms the arithmetic circuit into a *constraint system* formalized as indexed relations.

Several constraint systems are available for this step. In this paper, we build zkSNARKs for the following popular constraint systems:

- R1CS (Rank-1 Constraint System) is used in Pinocchio [21], Groth16 [2], Aurora [10], Fractal [8], Marlin [4], Spartan [9].
- HPR (Hadamard Product Relation) was proposed by Bootle et al. [23]. Its variations are used in BulletProofs [18], Sonic [5], Claymore [22]. A recent lattice-based zkSNARK [24] also chooses this relation.
- The constraint system of PLONK is used solely in PLONK [3], one of the most efficient and popular zkSNARKs in the literature.

The second step in the workflow produces a PIOP for verifying the indexed relation specified by the constraint system. This step is where most zkSNARK constructions vary and is the focus of this work.

Finally, the PIOP is compiled into a zkSNARK via two cryptographic tools. First, the polynomial oracles are replaced by polynomial commitments, and the evaluation queries are replaced by an interactive protocol between the prover and the verifier. This step turns the PIOP into an interactive proof in the standard model. Next, this interactive proof is turned into a zkSNARK by the Fiat-Shamir transformation [25] that replaces the verifier messages by query replies from a random oracle approximated by a collision-resistant hash function.

3 Vector Oracle Model

The Vector Oracle model introduces a *vector oracle* to interactive proofs. A vector oracle $\mathcal{O}$ maintains a dictionary of vectors. Each vector is indexed by a bit-string referred to as the vector's name. The indexer and the prover may submit arbitrary vectors in $\mathbb{F}^n$ to $\mathcal{O}$. The verifier, however, can only submit a restricted class of vectors, including sparse vectors and power vectors of the form $\boldsymbol{\alpha}^k := (1, \alpha, \alpha^2, \cdots, \alpha^{k-1})$. Although the oracle does not limit the density of the sparse vector, submitting a dense vector in the place of a sparse vector will not lead to any attacks, but only affects the computational workload of the verifier. The verifier can also manipulate existing vectors by linear combinations and right-shiftings. Finally, the verifier may query $\mathcal{O}$ to check Hadamard relations like $\boldsymbol{a} \circ \boldsymbol{b} \stackrel{?}{=} \boldsymbol{c} \circ \boldsymbol{d}$ or inner-product relations like $\boldsymbol{a} \cdot \boldsymbol{b} \stackrel{?}{=} \boldsymbol{c} \cdot \boldsymbol{d}$.

For convenience, we introduce the following notations. If we right-shift $\boldsymbol{v} \in \mathbb{F}^n$ by k positions, we get $\boldsymbol{v}^{\to k} := \mathbf{0}^{n-k} \| \boldsymbol{v}_{[1..n-k]} \in \mathbb{F}^n$, i.e., remove the right-most k elements and prefix the vector with k zeros. Denote the

dictionary of vectors be $\boldsymbol{V}$. For any $\mathsf{name} \in \{0,1\}^*$, we write $\mathsf{name} \in \boldsymbol{V}$ if $\boldsymbol{V}$ stores a vector under the key name, and refer to this vector by $\boldsymbol{V}[\mathsf{name}]$.

Definition 4 (Vector Oracle). *A vector oracle of vector size n internally maintains a dictionary $\boldsymbol{V} : \{0,1\}^* \to \mathbb{F}^n$ which starts out to be empty. The oracle accepts the following types of queries:*

- $\mathsf{VEC}(\mathsf{name} \in \{0,1\}^*, \boldsymbol{v} \in \mathbb{F}^n)$*: sets* $\boldsymbol{V}[\mathsf{name}] \leftarrow \boldsymbol{v}$*;*
- $\mathsf{POW}(\mathsf{name} \in \{0,1\}^*, \alpha \in \mathbb{F}, k \in [n])$*: sets* $\boldsymbol{V}[\mathsf{name}] \leftarrow \boldsymbol{\alpha}^k \| \boldsymbol{0}^{n-k}$*;*
- $\mathsf{SPA}(\mathsf{name} \in \{0,1\}^*, \boldsymbol{v} \in \mathbb{F}^n)$ *where* $\boldsymbol{v}$ *is sparse: sets* $\boldsymbol{V}[\mathsf{name}] \leftarrow \boldsymbol{v}$*;*
- $\mathsf{ADD}(\mathsf{name}, \mathsf{left}, \mathsf{right} \in \{0,1\}^*)$*: sets* $\boldsymbol{V}[\mathsf{name}] \leftarrow \boldsymbol{V}[\mathsf{left}] + \boldsymbol{V}[\mathsf{right}]$*;*
- $\mathsf{MUL}(\mathsf{name}, \mathsf{src} \in \{0,1\}^*, \alpha \in \mathbb{F})$*: sets* $\boldsymbol{V}[\mathsf{name}] \leftarrow \alpha \cdot \boldsymbol{V}[\mathsf{src}]$*;*
- $\mathsf{SHR}(\mathsf{name}, \mathsf{src} \in \{0,1\}^*, k \in [n-1])$*: sets* $\boldsymbol{V}[\mathsf{name}] \leftarrow \boldsymbol{V}[\mathsf{src}]^{\to k}$*;*
- $\mathsf{HAD}(\mathsf{a}, \mathsf{b}, \mathsf{c}, \mathsf{d} \in \{0,1\}^*)$*: aborts if* $\boldsymbol{V}[\mathsf{a}] \circ \boldsymbol{V}[\mathsf{b}] \neq \boldsymbol{V}[\mathsf{c}] \circ \boldsymbol{V}[\mathsf{d}]$*;*
- $\mathsf{INN}(\mathsf{a}, \mathsf{b}, \mathsf{c}, \mathsf{d} \in \{0,1\}^*)$*: aborts if* $\boldsymbol{V}[\mathsf{a}] \cdot \boldsymbol{V}[\mathsf{b}] \neq \boldsymbol{V}[\mathsf{c}] \cdot \boldsymbol{V}[\mathsf{d}]$*.*

If a query tries to store a vector in a key already in $\boldsymbol{V}$, or read a key not in $\boldsymbol{V}$, the oracle also aborts.

A Vector Oracle (VO) protocol is an interactive protocol where the parties have access to the vector oracle.

Definition 5 (Preprocessing VO Protocol). *A preprocessing VO protocol with vector size n for an indexed relation $\mathcal{R}$ is a tuple of PPT algorithms $(\mathsf{I}, \mathsf{P}, \mathsf{V})$ that have access to a vector oracle $\mathcal{O}$ with vector size n, such that:*

- $(\mathsf{I}, \mathsf{P}, \mathsf{V})$ *is a preprocessing interactive protocol for $\mathcal{R}$;*
- I *and* P*'s accesses to $\mathcal{O}$ are restricted to* VEC *queries;*
- V*'s accesses to $\mathcal{O}$ are restricted to queries except* VEC*;*
- P *does not send any messages to* V*; and*
- V *outputs* 1 *if and only if $\mathcal{O}$ does not abort.*

The notions public-coin, completeness and soundness follow from those of preprocessing interactive protocols.

Note that the verifier does not receive any information from the prover or $\mathcal{O}$ except if $\mathcal{O}$ aborts or not. Therefore, any VO protocols trivially satisfy HVSZK in Definition 2.

Theorem 1 shows that we can compile a VO protocol to an HVSZK PIOP with small overhead, no impact on completeness, and negligible degradation to soundness.

Theorem 1. *Assume that the preprocessing VO protocol* (VO.I, VO.P, VO.V) *with vector size n verifies the indexed relation $\mathcal{R}$ with completeness error ε_c and soundness error ε_s. Assume that* VO.I *submits m vectors,* VO.P *submits r vectors,* VO.V *issues t_H* HAD *queries, t_I* INN *queries. Then there exists a preprocessing PIOP protocol* $(\mathsf{I}, \mathsf{P}, \mathsf{V})$ *with degree bound $4n+1$ that verifies $\mathcal{R}$ with completeness error ε_c and soundness error $\varepsilon_s + \frac{t_H+t_I+6n+6}{|\mathbb{F}|-1}$, and HVSZK where the simulator* S *simulates the transcript of* $\langle \mathsf{I}, \mathsf{P}, \mathsf{V} \rangle(\mathbb{i}, \mathbb{x}, \mathbb{w})$ *with statistical distance* $\frac{1}{|\mathbb{F}|-1}$. *Moreover,* I *sends m polynomial oracles, and* P *sends $r+3$ polynomial oracles to* V, *and* V *makes at most $2(m+r)+6$ evaluation queries at 3 distinct points.*

If $t_I = 0$, i.e., the VO protocol does not use any INN *queries,* P *sends $r+2$ online polynomial oracles, and the verifier makes at most $2(m+r)+4$ evaluation queries at 3 distinct points.*

We provide an overview of the VO-to-PIOP compiler, followed by a sketch of the security proof for the compiled PIOP. We leave the complete proof to Appendix A. The compiler is partially inspired by the batched InnerProduct protocol in Claymore [22].

Compiler. Given any VO protocol $(\mathsf{VO.I}, \mathsf{VO.P}, \mathsf{VO.V})$, the compiled PIOP, namely $(\mathsf{I}, \mathsf{P}, \mathsf{V})$, works as follows:

1. I executes exactly as VO.I, except for every $\boldsymbol{v} \in \mathbb{F}^n$ submitted to $\mathcal{O}$, I instead sends a polynomial $[f_{\boldsymbol{v}}(X)]$ of degree at most $n-1$ to V;
2. P and V interact exactly as VO.P and VO.V, except that the queries to $\mathcal{O}$ are handled differently. Generally speaking, all the vectors involved in the protocol are replaced by polynomial oracles: the vectors submitted by VO.P are encapsulated in polynomial oracles and sent to V by P; the polynomial oracles for vectors submitted by VO.V can be simulated by V locally due to their special structures.
   - For every $\mathsf{VEC}(\mathsf{name}, \boldsymbol{v})$ query, P samples $\boldsymbol{\delta} \xleftarrow{\$} \mathbb{F}^q$, computes $\boldsymbol{v}_{\mathsf{name}} := \boldsymbol{v} \| \boldsymbol{\delta}$ and sends the polynomial $[f_{\boldsymbol{v}_{\mathsf{name}}}(X)]$ of degree at most $n+q-1$ to V, where q is a small integer decided later when we discuss ZK;
   - for every $\mathsf{POW}(\mathsf{name}, \alpha, k)$ query, P computes $\boldsymbol{v}_{\mathsf{name}} := \boldsymbol{\alpha}^k$, and V simulates the polynomial oracle $[f_{\boldsymbol{\alpha}^k}(X)]$ of degree $k-1$ by locally computing $f_{\boldsymbol{\alpha}^k}(z) = \frac{(\alpha z)^k - 1}{\alpha z - 1}$ for $\alpha z \neq 1$ or k for $\alpha z = 1$;
   - for every $\mathsf{SPA}(\mathsf{name}, \boldsymbol{v})$ query, P stores $\boldsymbol{v}_{\mathsf{name}} := \boldsymbol{v}$, and V simulates the polynomial oracle $[f_{\boldsymbol{v}}(X)]$ by evaluating this polynomial locally using the sparse representation of $\boldsymbol{v}$;
   - for every $\mathsf{ADD}(\mathsf{name}, \mathsf{left}, \mathsf{right})$ query, P computes $\boldsymbol{v}_{\mathsf{name}} := \boldsymbol{v}_{\mathsf{left}} + \boldsymbol{v}_{\mathsf{right}}$ and V simulates the polynomial oracle $[f_{\boldsymbol{v}_{\mathsf{name}}}(X)]$ of degree

at most $\max\{|\boldsymbol{v}_{\mathsf{left}}|, |\boldsymbol{v}_{\mathsf{right}}|\} - 1$ by querying the polynomial oracles $[f_{\boldsymbol{v}_{\mathsf{left}}}(X)]$ and $[f_{\boldsymbol{v}_{\mathsf{right}}}(X)]$ and adding the responses;
- for every $\mathsf{MUL}(\mathsf{name}, \mathsf{src}, \alpha)$ query, P computes $\boldsymbol{v}_{\mathsf{name}} := \alpha \cdot \boldsymbol{v}_{\mathsf{src}}$ and V simulates the polynomial oracle $[f_{\boldsymbol{v}_{\mathsf{name}}}(X)]$ of degree at most $|\boldsymbol{v}_{\mathsf{src}}| - 1$ by querying the polynomial oracle $[f_{\boldsymbol{v}_{\mathsf{src}}}(X)]$ and multiplying the response with α;
- for every $\mathsf{SHR}(\mathsf{name}, \mathsf{src}, k)$ query, P computes $\boldsymbol{v}_{\mathsf{name}} := \boldsymbol{0}^k \| \boldsymbol{v}_{\mathsf{src}}$ and V simulates the polynomial oracle $[f_{\boldsymbol{v}_{\mathsf{name}}}(X)]$ of degree at most $|\boldsymbol{v}_{\mathsf{src}}| + k - 1$ by querying the polynomial oracle $[f_{\boldsymbol{v}_{\mathsf{src}}}(X)]$ with z and multiplying the response with z^k;
- INN and HAD queries demand more complicated processing, whose details are in step 3 and step 4 to 7, respectively.

3. Let the vectors referred to by the INN queries be $\{\boldsymbol{a}^{(i)}, \boldsymbol{b}^{(i)}, \boldsymbol{c}^{(i)}, \boldsymbol{d}^{(i)}\}_{i=1}^{t_I}$. These vectors have sizes at least n. Note that P has computed all of them, and V has access to their polynomial oracles. The INN queries are equivalent to $\boldsymbol{a}^{(i)}_{[1..n]} \cdot \boldsymbol{b}^{(i)}_{[1..n]} = \boldsymbol{c}^{(i)}_{[1..n]} \cdot \boldsymbol{d}^{(i)}_{[1..n]}$ for every $i \in [t_I]$. To prove this, V samples $\beta \xleftarrow{\$} \mathbb{F}$ and sends β to P. P computes

$$\boldsymbol{r} := \sum_{i \in [t_I]} \beta^i \left(\boldsymbol{a}^{(i)}_{[1..n]} \circ \boldsymbol{b}^{(i)}_{[1..n]} - \boldsymbol{c}^{(i)}_{[1..n]} \circ \boldsymbol{d}^{(i)}_{[1..n]} \right),$$

samples $\boldsymbol{\delta} \xleftarrow{\$} \mathbb{F}^q$ and computes $\tilde{\boldsymbol{r}} := \left(\sum_{j=1}^{i} \boldsymbol{r}_{[j]} \right)_{i=1}^{n} \| \boldsymbol{\delta}$. P sends $[f_{\tilde{\boldsymbol{r}}}(X)]$ of degree $n + q - 1$ to V. It suffices for P to show that $\tilde{\boldsymbol{r}}_{[n]} = 0$ and that $(\tilde{\boldsymbol{r}} - (0 \| \tilde{\boldsymbol{r}}) - \boldsymbol{r})_{[1..n]} = 0$, which will be proved together with the HAD queries in the next step;

4. Let the vectors referred to by the HAD queries be $\{\bar{\boldsymbol{a}}^{(i)}, \bar{\boldsymbol{b}}^{(i)}, \bar{\boldsymbol{c}}^{(i)}, \bar{\boldsymbol{d}}^{(i)}\}_{i=1}^{t_H}$. We summarize what P needs to prove as follows:
   - $\bar{\boldsymbol{a}}^{(i)}_{[1..n]} \circ \bar{\boldsymbol{b}}^{(i)}_{[1..n]} - \bar{\boldsymbol{c}}^{(i)}_{[1..n]} \circ \bar{\boldsymbol{d}}^{(i)}_{[1..n]} = \boldsymbol{0}$ for every $i \in [t_H]$;
   - $\left(\tilde{\boldsymbol{r}} - (0 \| \tilde{\boldsymbol{r}}) - \sum_{i \in [t_I]} \beta^i \left(\boldsymbol{a}^{(i)} \circ \boldsymbol{b}^{(i)} - \boldsymbol{c}^{(i)} \circ \boldsymbol{d}^{(i)} \right) \right)_{[1..n]} = \boldsymbol{0}$;
   - $\tilde{\boldsymbol{r}} \circ \boldsymbol{e}_n = \boldsymbol{0}$.

   To prove the above, V samples $\alpha \xleftarrow{\$} \mathbb{F}$ and sends α to P. P samples $\boldsymbol{\delta} \xleftarrow{\$} \mathbb{F}^q$ and computes

$$\begin{aligned} \boldsymbol{t} := \boldsymbol{\delta} &+ \sum_{i \in [t_H]} \alpha^i \left(\bar{\boldsymbol{a}}^{(i)} \circ \bar{\boldsymbol{b}}^{(i)} - \bar{\boldsymbol{c}}^{(i)} \circ \bar{\boldsymbol{d}}^{(i)} \right) \\ &+ \alpha^{t_H+2} \left(\tilde{\boldsymbol{r}} - (0 \| \tilde{\boldsymbol{r}}) - \sum_{i \in [t_I]} \beta^j \left(\boldsymbol{a}^{(j)} \circ \boldsymbol{b}^{(j)} - \boldsymbol{c}^{(j)} \circ \boldsymbol{d}^{(j)} \right) \right). \end{aligned}$$

P sends the polynomial oracle $[f_{\boldsymbol{t}}(X)]$ of degree at most $2n+q-1$ to V. It suffices to show that

$$\mathbf{0} = -\boldsymbol{t} \circ (\mathbf{0}^n \| \mathbf{1}^{n+q}) + \sum_{i \in [t_H]} \alpha^i \left(\bar{\boldsymbol{a}}^{(i)} \circ \bar{\boldsymbol{b}}^{(i)} - \bar{\boldsymbol{c}}^{(i)} \circ \bar{\boldsymbol{d}}^{(i)} \right) + \alpha^{t_H+1} \tilde{\boldsymbol{r}} \circ \boldsymbol{e}_n$$
$$+ \alpha^{t_H+2} \left(\tilde{\boldsymbol{r}} - (0 \| \tilde{\boldsymbol{r}}) - \sum_{i \in [t_I]} \beta^j \left(\boldsymbol{a}^{(j)} \circ \boldsymbol{b}^{(j)} - \boldsymbol{c}^{(j)} \circ \boldsymbol{d}^{(j)} \right) \right). \tag{1}$$

5. V samples $\omega \xleftarrow{\$} \mathbb{F}$ and sends ω to P, and P computes

$$\begin{aligned} h(X) := & -(\omega X^{-1})^n f_{\mathbf{1}^{n+q}}(\omega X^{-1}) f_{\boldsymbol{t}}(X) \\ & + \sum_{i \in [t_H]} \alpha^i \left(f_{\bar{\boldsymbol{a}}^{(i)}}(\omega X^{-1}) f_{\bar{\boldsymbol{b}}^{(i)}}(X) - f_{\bar{\boldsymbol{c}}^{(i)}}(\omega X^{-1}) f_{\bar{\boldsymbol{d}}^{(i)}}(X) \right) \\ & + \alpha^{t_H+1} (\omega X^{-1})^{n-1} f_{\tilde{\boldsymbol{r}}}(X) + \alpha^{t_H+2} \left(f_{\mathbf{1}^{n+q}}(\omega X^{-1}) \cdot (1-X) f_{\tilde{\boldsymbol{r}}}(X) \right) \\ & - \alpha^{t_H+2} \sum_{i \in [t_I]} \beta^j \left(f_{\boldsymbol{a}^{(j)}}(\omega X^{-1}) f_{\boldsymbol{b}^{(j)}}(X) - f_{\boldsymbol{c}^{(j)}}(\omega X^{-1}) f_{\boldsymbol{d}^{(j)}}(X) \right), \end{aligned}$$

   i.e., replace every pair of Hadamard product $\boldsymbol{v} \circ \boldsymbol{v}'$ in Equation (1) by the polynomial $f_{\boldsymbol{v}}(\omega X^{-1}) f_{\boldsymbol{v}'}(X)$ whose constant term is $f_{\boldsymbol{v} \circ \boldsymbol{v}'}(\omega)$. It suffices to show that the constant term of $h(X)$ is 0.
6. Let γ be a generator of the multiplicative group $\mathbb{F}^*$. P samples $\delta \xleftarrow{\$} \mathbb{F}$ and computes $\bar{h}(X) := \delta X^{2n+q-1} + X^{2n+q-1} \sum_{i \neq 0} \frac{h_i}{\gamma^i - 1} X^i$. Note that $(\gamma X)^{-2n-q+1} \bar{h}(\gamma X) - X^{-2n-q+1} \bar{h}(X)$ is exactly $h(X)$ except that the constant term is set to zero.
7. P sends the polynomial oracle $[\bar{h}(X)]$ of degree at most $4n+q-2$ to V, and V checks the identity

$$h(X) = (\gamma X)^{-2n-q+1} \bar{h}(\gamma X) - X^{-2n-q+1} \bar{h}(X) \tag{2}$$

   at a random point $z \xleftarrow{\$} \mathbb{F}$, by querying $[\bar{h}(X)]$ and evaluating $h(z)$ according to its definition using all the polynomial oracles at hand.

Proof (Sketch). Completeness and soundness follow intuitively, and we refer to Appendix A for a more rigorous proof. HVSZK follows from the fact that every polynomial is queried at most twice (at z and ωz^{-1}, or at z and γz for $\bar{h}(X)$), and that each polynomial sent by the prover contains at least q uniformly random coefficient in $\mathbb{F}$, except $\bar{h}(X)$, which contains only one random coefficient. Therefore, by letting $q = 2$, we ensure that all the evaluation results are uniformly random and independent of each other, except the last query to $\bar{h}(X)$. However, the simulator can solve for the last query result using the identity in Equation (2). □

Note that the number of evaluation queries in Theorem 1 is only a worst-case upper bound, which is reached only if every prover vector is involved in both a HAD and an INN query. In practice, this number is much smaller due to a series of optimizations in Sect. 6.

Remark 1. The compiler presented in this paper uses the *monomial basis*, i.e., the vectors are embedded into the coefficients of the polynomials. Prior works often use the *Reed-Solomon code basis* where a vector is identified by its interpolation polynomials over a domain $H \subset \mathbb{F}$. The VO-to-PIOP compilation can also be implemented using the Reed-Solomon code basis. However, the VO model must be modified as follows to avoid several efficiency issues:

- In the monomial basis, the polynomial for power vector $\boldsymbol{\alpha}^k$ admits fast evaluation. This is not the case in the Reed-Solomon code basis. Instead, the verifier has access to the identity vector $(h)_{h\in H}$ corresponding to $f(X) := X$. Moreover, the vector $\left(\frac{1}{\alpha-h}\right)_{h\in H}$ is also available, as the polynomial $\frac{v_H(X)\cdot v_H(\alpha)^{-1}-1}{X-\alpha}$ has fast evaluation method when H has special structures, where $v_H(X)$ is the vanishing polynomial over H. We should replace the POW with queries for submitting these types of vectors instead.
- In the Reed-Solomon code basis, the Hadamard product is identified with the polynomial multiplication. Therefore, the functionality of the HAD query is changed to submitting $\boldsymbol{V}[\mathsf{name}] \leftarrow \boldsymbol{V}[\mathsf{left}] \circ \boldsymbol{V}[\mathsf{right}]$. Then we also need a ZERO(name) query to check that $\boldsymbol{V}[\mathsf{name}] \stackrel{?}{=} \mathbf{0}$.
- In the monomial basis, shifting a vector $\boldsymbol{v}$ effectively multiplies X^k to the polynomial $f_{\boldsymbol{v}}(X)$. In the Reed-Solomon code basis, the shifting is implemented by replacing $f(X)$ with $f(g^{-k}\cdot X)$ assuming H is a multiplicative subgroup generated by $g \in \mathbb{F}^*$. This shifting is cyclic rather than zero-padded, and the SHR query should be redefined accordingly.

In compiling this modified VO model, the INN queries can be batched together and checked by one invocation of the *univariate sumcheck* protocol [10,4,8]. We will not dive into the details which are off the topic of this paper. We choose the monomial basis in this work because our protocols rely heavily on the power vector and that the monomial basis permits a more flexible choice of vector sizes.

Theorem 2 shows that PIOP protocols (not necessarily HVSZK) can be compiled back to VO protocols. This implies that the VO protocols are sufficiently powerful to prove the same relations as PIOP protocols.

Theorem 2. *Assume the preprocessing PIOP protocol* (PIOP.I, PIOP.P, PIOP.V) *of degree bound n verifies the relation $\mathcal{R}$ with completeness error ε_c, and soundness error ε_s, where* I *sends m polynomial oracles,* P *sends r polynomial oracles, and* V *makes s evaluation queries. Assume that the* PIOP.V *is computed by an arithmetic circuit of size V. There exists a VO protocol* $(\mathsf{I}, \mathsf{P}, \mathsf{V})$ *with vector size n that verifies $\mathcal{R}$ with completeness error $\varepsilon_c + O(V/|\mathbb{F}|)$ and soundness error $\varepsilon_c + O(V/|\mathbb{F}|)$, where* I *submits m vectors,* P *submits $r+2$ vectors, and the verifier makes s* INN *queries,* 4 HAD *queries.*

We leave the proof of Theorem 2 to Appendix B.

4 Vector Oracle Protocols for R1CS and HPR

Now we apply the VO model to construct zkSNARKs for the linear-algebra-based relations R1CS and HPR, defined in Equation (3) and (4), respectively. In this section, we construct VO protocols for these relations. Afterward, these VO protocols can be compiled to zkSNARKs via the VO-to-PIOP compiler introduced in Sect. 3 followed by the cryptographic compilation explained in Sect. 2.4.

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

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

The matrices $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}$ are usually sparse. We start from the SparseMVP protocol to deal with the matrix-vector products in these relations.

4.1 Sparse Matrix Vector Product

Let $\boldsymbol{M} \in \mathbb{F}^{H \times K}$ be a sparse matrix with at most S nonzero elements. Given the input vector $\boldsymbol{a} \in \mathbb{F}^n$, the protocol SparseMVP checks that $\boldsymbol{a}_{[1..H]} = \boldsymbol{M}\boldsymbol{a}_{[H+1..H+K]}$, where "given a vector $\boldsymbol{a}$" means the vector is stored in the vector oracle, and the prover learns the content of $\boldsymbol{a}$.

Random Linear Combination. Lemma 1 allows the verifier to check that two vectors are the same when one or both of them are not stored in the vector oracle.

Lemma 1. *Let γ be a generator of the multiplicative group $\mathbb{F}^*$. For any nonzero vector $\boldsymbol{v} \in \mathbb{F}^H$, for uniformly random $\alpha \in \mathbb{F} \backslash \{\gamma^i\}_{i=1}$, the probability* $\Pr[\boldsymbol{r}_\alpha \cdot \boldsymbol{v} = 0] \leq \frac{H}{|\mathbb{F}|-|H|}$, *where* $\boldsymbol{r}_\alpha := \left(\frac{1}{\alpha-\gamma^i}\right)_{i=1}^{H}$.

Proof. Note that

$$\sum_{i=1}^{H} \frac{\boldsymbol{v}_{[i]}}{\alpha - \gamma^i} = 0 \Leftrightarrow \quad \sum_{i=1}^{H} \boldsymbol{v}_{[i]} \prod_{j \neq i} (\alpha - \gamma^j) = 0 \Leftrightarrow \quad \boldsymbol{v}^\mathsf{T} \boldsymbol{\Gamma} \boldsymbol{\alpha}^H = 0,$$

where $\boldsymbol{\Gamma}$ is a matrix of size $H \times H$ whose i-th row is the coefficient vector of polynomial $\prod_{j \neq i}(X - \gamma^j)$, which (after normalized) is the Lagrange basis polynomial over $\{\gamma^i\}_{i=1}^H$. Since the Lagrange basis polynomials are linearly independent, $\boldsymbol{\Gamma}$ is an invertible matrix, therefore $\boldsymbol{v}^\mathsf{T}\boldsymbol{\Gamma} \neq \boldsymbol{0}$. Since $\boldsymbol{v}^\mathsf{T}\boldsymbol{\Gamma}\boldsymbol{\alpha}^H = f_{\boldsymbol{v}^\mathsf{T}\boldsymbol{\Gamma}}(\alpha)$, the conclusion follows from Schwartz-Zippel Lemma. □

The verifier samples $\alpha \xleftarrow{\$} \mathbb{F}\backslash\{\gamma^i\}_{i=1}^H$ and sends to the prover. By Lemma 1, if the prover shows that $\boldsymbol{r}_\alpha \cdot \boldsymbol{a}_{[1..H]} = \boldsymbol{r}_\alpha^\mathsf{T}\boldsymbol{M}\boldsymbol{a}_{[H+1..H+K]}$, the verifier is confident that $\boldsymbol{a}_{[1..H]} = \boldsymbol{M}\boldsymbol{a}_{[H+1..H+K]}$ with high probability. To prove this, the prover submits the vector $\boldsymbol{c} = \boldsymbol{r}_\alpha^\mathsf{T}\boldsymbol{M} \in \mathbb{F}^K$ together with $\boldsymbol{r}_\alpha$ and shows the following statements:

1. $\boldsymbol{r}_\alpha$ is as defined;
2. $\boldsymbol{r}_\alpha \cdot \boldsymbol{a}_{[1..H]} = \boldsymbol{c} \cdot \boldsymbol{a}_{[H+1..H+K]}$; and
3. $\boldsymbol{c} = \boldsymbol{r}_\alpha^\mathsf{T}\boldsymbol{M}$.

The verifier checks the first condition by a HAD query for the relation $\boldsymbol{r}_\alpha \circ (\alpha \cdot \boldsymbol{1}^H - \boldsymbol{\gamma}^H) = \boldsymbol{1}^H$, and the second condition by an INN query for

$$\boldsymbol{r}_\alpha^{\to n-H} \cdot \boldsymbol{a}^{\to n-H} = \boldsymbol{c}^{\to n-K} \cdot \boldsymbol{a}^{\to n-H-K},$$

where n is the vector size of the vector oracle, and the right-shifts by $n-*$ align the inner-product ranges to the right boundary and effectively remove any redundant elements outside the range to prevent them from affecting the inner product.

Now we focus on the third condition. By applying Lemma 1 again, the verifier picks another random element $\beta \xleftarrow{\$} \mathbb{F}\backslash\{\gamma^i\}_{i=1}^K$ and asks the prover to show that $\boldsymbol{c} \cdot \boldsymbol{r}_\beta = \boldsymbol{r}_\alpha^\mathsf{T}\boldsymbol{M}\boldsymbol{r}_\beta$, where $\boldsymbol{r}_\beta := \left(\frac{1}{\beta-\gamma^i}\right)_{i=1}^{K}$. This proves that $\boldsymbol{c} = \boldsymbol{r}_\alpha^\mathsf{T}\boldsymbol{M}$. Therefore, the problem boils down to showing that $\boldsymbol{r}_\alpha^\mathsf{T}\boldsymbol{M}\boldsymbol{r}_\beta$ is equal to the inner product $\boldsymbol{c} \cdot \boldsymbol{r}_\beta$, which is solved next.

Sparse Representation of Matrix. Since the matrix $\boldsymbol{M}$ has only S nonzero entries, we represent $\boldsymbol{M}$ by the row indices $(\mathsf{row}_1, \cdots, \mathsf{row}_S) \in [H]^S$, the column indices $(\mathsf{col}_1, \cdots, \mathsf{col}_S) \in [K]^S$ and the nonzero entries $\boldsymbol{v} \in \mathbb{F}^S$, such that for each $i \in [S]$, $\boldsymbol{M}_{[\mathsf{row}_i, \mathsf{col}_i]} = \boldsymbol{v}_{[i]}$. By the definition of $\boldsymbol{r}_\alpha, \boldsymbol{r}_\beta$ and matrix-vector product,

$$\boldsymbol{r}_\alpha^\mathsf{T} \boldsymbol{M} \boldsymbol{r}_\beta = \sum_{i,j=1}^{H,K} \boldsymbol{M}_{[i,j]} \cdot \boldsymbol{r}_{\alpha[i]} \cdot \boldsymbol{r}_{\beta[j]} = \sum_{i \in [S]} \frac{\boldsymbol{v}_{[i]}}{(\alpha - \gamma^{\mathsf{row}_i})(\beta - \gamma^{\mathsf{col}_i})}.$$

This is the inner product between $\boldsymbol{v}$ and $\boldsymbol{t} := \left(\frac{1}{(\alpha-\gamma^{\mathsf{row}_i})(\beta-\gamma^{\mathsf{col}_i})}\right)_{i=1}^S$. Since $\boldsymbol{v}$ depends only on the matrix, the indexer may preprocess and submit $\boldsymbol{v}$ to the vector oracle. The prover computes and submits the vector $\boldsymbol{t}$. The verifier checks $\boldsymbol{v} \cdot \boldsymbol{t} = \boldsymbol{c} \cdot \boldsymbol{r}_\beta$ by an INN query.

Checking $\boldsymbol{t}$. Finally, the vector $\boldsymbol{t}$ is validated by the identity

$$\boldsymbol{t} \circ (\alpha\beta \cdot \boldsymbol{1}^S - \alpha \cdot \boldsymbol{w} - \beta \cdot \boldsymbol{u} + \boldsymbol{u} \circ \boldsymbol{w}) = \boldsymbol{1}^S,$$

where $\boldsymbol{u} := (\gamma^{\mathsf{row}_i})_{i=1}^S$ and $\boldsymbol{w} := \left(\gamma^{\mathsf{col}_i}\right)_{i=1}^S$ can be preprocessed by the indexer. Moreover, the indexer also preprocesses the vector $\boldsymbol{u} \circ \boldsymbol{w}$ to save one online vector. Using these preprocessed vectors, the verifier can check the above identity by one HAD query, thus finishes the protocol.

Combining the Vectors. In the above protocol, the prover needs to submit four vectors: $\boldsymbol{r}_\alpha, \boldsymbol{c}, \boldsymbol{r}_\beta$ and $\boldsymbol{t}$. We suggest an alternative method in which the prover concatenates those four vectors into two: $\boldsymbol{s} := \boldsymbol{r}_\alpha \| (-\boldsymbol{c})$ and $\boldsymbol{h} := \boldsymbol{r}_\beta \| \boldsymbol{t}$. The HAD and INN queries are modified by shifting the vectors accordingly. See Algorithm 1 for more details.

4.2 R1CS

With the $\mathsf{SparseMVP}$ protocol ready, now we construct a Vector Oracle protocol for verifying $\mathsf{R1CS}$. Given an index of the $\mathsf{R1CS}$ relation, the indexer concatenates the matrices into $\boldsymbol{M} = (\boldsymbol{A}^\mathsf{T} \| \boldsymbol{B}^\mathsf{T} \| \boldsymbol{C}^\mathsf{T})^\mathsf{T} \in \mathbb{F}^{3H \times K}$ and invokes the indexer of $\mathsf{SparseMVP}$ with $\boldsymbol{M}$. The prover computes $\boldsymbol{z} = 1 \| \boldsymbol{x} \| \boldsymbol{w}$, $\boldsymbol{y} = \boldsymbol{M}\boldsymbol{z}$, and submits $\boldsymbol{u} = \boldsymbol{y} \| \boldsymbol{z}$ to $\mathcal{O}$. The $\mathsf{R1CS}$ relation is equivalent to the following conditions on $\boldsymbol{u}$:

1. $\boldsymbol{u}_{[1..H]} \circ \boldsymbol{u}_{[H+1..2H]} = \boldsymbol{u}_{[2H+1..3H]}$;
2. $\boldsymbol{u}_{[3H+1..3H+\ell+2]} = 1 \| \boldsymbol{x}$;

Algorithm 1 SparseMVP Protocol

Index: a sparse representation of a matrix $\boldsymbol{M} \in \mathbb{F}^{H \times K}$: $\{\mathsf{row}_i, \mathsf{col}_i, \mathsf{val}_i\}_{i=1}^{S}$
Input: $\boldsymbol{a} \in \mathbb{F}^n$ **Check:** $\boldsymbol{a}_{[1..H]} = \boldsymbol{M}\boldsymbol{a}_{[H+1..H+K]}$
Preprocessing:

1: I computes $\boldsymbol{u} = (\gamma^{\mathsf{row}_i})_{i=1}^{S} \| \boldsymbol{0}^{n-S}, \boldsymbol{w} = \left(\gamma^{\mathsf{col}_i}\right)_{i=1}^{S} \| \boldsymbol{0}^{n-S}, \boldsymbol{v} = (\mathsf{val}_i)_{i=1}^{S} \| \boldsymbol{0}^{n-S}$ and $\boldsymbol{y} = \boldsymbol{u} \circ \boldsymbol{w}$
2: I sends $H, K, S, \gamma, \boldsymbol{u}, \boldsymbol{w}, \boldsymbol{v}, \boldsymbol{y}$ to P, and H, K, S, γ to V, and submits $\boldsymbol{u}, \boldsymbol{w}, \boldsymbol{v}, \boldsymbol{y}$ to $\mathcal{O}$.

Online:

1: V samples $\alpha \xleftarrow{\$} \mathbb{F}^*$ and sends α to P;
2: P computes $\boldsymbol{r}_\alpha = \left(\frac{1}{\alpha - \gamma^i}\right)_{i=1}^{H}$, $\boldsymbol{c} = \boldsymbol{r}_\alpha^{\mathsf{T}} \boldsymbol{M}$;
3: P computes $\boldsymbol{s} = \boldsymbol{r}_\alpha \| (-\boldsymbol{c}) \| \boldsymbol{0}^{n-H-K}$ and submits $\boldsymbol{s}$ to $\mathcal{O}$;
4: V queries $\mathcal{O}$ to check that $\boldsymbol{s} \circ (\alpha \cdot \boldsymbol{1}^H - \boldsymbol{\gamma}^H) = \boldsymbol{1}^H$;
5: V queries $\mathcal{O}$ to check that $\boldsymbol{s}^{\to n-H-K} \cdot \boldsymbol{a}^{\to n-H-K} = 0$;
6: V samples $\beta \xleftarrow{\$} \mathbb{F}^*$ and sends β to P;
7: P computes $\boldsymbol{r}_\beta = \left(\frac{1}{\beta - \gamma^i}\right)_{i=1}^{K}$, $\boldsymbol{t} = \left(\frac{1}{(\alpha - \gamma^{\mathsf{row}_i})(\beta - \gamma^{\mathsf{col}_i})}\right)_{i=1}^{S}$
8: P computes $\boldsymbol{h} = \boldsymbol{r}_\beta \| \boldsymbol{t} \| \boldsymbol{0}^{n-S-K}$ and submits $\boldsymbol{h}$ to $\mathcal{O}$;
9: V queries $\mathcal{O}$ to check that $\boldsymbol{h} \circ (\beta \cdot \boldsymbol{1}^K - \boldsymbol{\gamma}^K) = \boldsymbol{1}^K$;
10: V queries $\mathcal{O}$ to check that $\boldsymbol{h} \circ (\alpha\beta \cdot \boldsymbol{1}^S - \alpha \cdot \boldsymbol{w} - \beta \cdot \boldsymbol{u} + \boldsymbol{y})^{\to K} = \boldsymbol{1}_{[K+1..K+S]}$;
11: V queries $\mathcal{O}$ to check that $-\boldsymbol{h}^{\to n-K} \cdot \boldsymbol{s}^{\to n-H-K} = \boldsymbol{h}^{\to n-S-K} \cdot \boldsymbol{v}^{\to n-S}$.

3. $\boldsymbol{u}_{[1..3H]} = \boldsymbol{M}\boldsymbol{u}_{[3H+1..3H+K]}$.

The verifier checks the first condition by a HAD query. For the second condition, the verifier subtracts $1 \| \boldsymbol{x}$ from the target range of $\boldsymbol{u}$ and tests if the result is zero by multiplying a *masking vector* consisting of zeros and ones. The masking vector can be obtained by first submitting a power vector $\boldsymbol{1}^{\ell+1}$ then right-shifting it appropriately. Finally, the verifier checks the third condition by SparseMVP. See Algorithm 2 for more details.

Algorithm 2 VOR1CS Protocol

Index: $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C} \in \mathbb{F}^{H \times K}$ where $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}$ have s nonzero entries in total
Instance: $\boldsymbol{x} \in \mathbb{F}^\ell$ **Witness:** $\boldsymbol{w} \in \mathbb{F}^{K-\ell-1}$
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 $\boldsymbol{M} = (\boldsymbol{A}^{\mathsf{T}} \| \boldsymbol{B}^{\mathsf{T}} \| \boldsymbol{C}^{\mathsf{T}})^{\mathsf{T}} \in \mathbb{F}^{3H \times K}$ of length s and invokes SparseMVP.I with this representation.

Online:

1: P submits $\boldsymbol{u} = (\boldsymbol{M}(1\|\boldsymbol{x}\|\boldsymbol{w})) \| 1 \| \boldsymbol{x} \| \boldsymbol{w} \| \boldsymbol{0}^{n-3H-K}$ to $\mathcal{O}$;
2: P and V run the protocol SparseMVP with inputs $\boldsymbol{u}$;
3: V queries $\mathcal{O}$ to check that $\boldsymbol{u}^{\to n-H} \circ \boldsymbol{u}^{\to n-2H} = \boldsymbol{1}_{[n-H+1..n]} \circ \boldsymbol{u}^{\to n-3H}$;
4: V queries $\mathcal{O}$ to check that $\boldsymbol{1}_{[3H+1..3H+\ell+1]} \circ (\boldsymbol{u} - \boldsymbol{e}_{3H+1} - \boldsymbol{x}^{\to 3H+1}) = \boldsymbol{0}$.

Theorem 3. *The* VOR1CS *protocol in Algorithm 2 is a VO protocol for the relation* $\mathcal{R}_{\mathsf{R1CS}}$ *with perfect completeness and soundness error* $\frac{3H+K}{|\mathbb{F}|-3H-K}$.

4.3 HPR

We construct the Vector Oracle protocol for HPR similarly. Let $\boldsymbol{M} = (\boldsymbol{d}\|\boldsymbol{A}\|\boldsymbol{B}\|\boldsymbol{C}) \in \mathbb{F}^{H\times(3K+1)}$, and the indexer preprocesses $\boldsymbol{M}$ by invoking the indexer of SparseMVP. The prover computes and submits $\boldsymbol{w} = \boldsymbol{w}_1\|\boldsymbol{w}_2\|\boldsymbol{w}_3$. Then the HPR relation is equivalent to $\boldsymbol{w}_{[1..K]} \circ \boldsymbol{w}_{[K+1..2K]} = \boldsymbol{w}_{[2K+1..3K]}$ and $\boldsymbol{M}(1\|\boldsymbol{w}) = \boldsymbol{x}\|\boldsymbol{0}^{H-\ell}$. The verifier checks these equations by a HAD query and SparseMVP. See Algorithm 3 for more details.

Algorithm 3 VOHPR Protocol

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

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

Online:

1: P submits $\boldsymbol{w} = \boldsymbol{w}_1\|\boldsymbol{w}_2\|\boldsymbol{w}_3\|\boldsymbol{0}^{n-3K}$ to $\mathcal{O}$;
2: P and V run the protocol SparseMVP with inputs $\boldsymbol{x} + \boldsymbol{e}_{H+1} + \boldsymbol{w}^{\to H+1}$;
3: V queries $\mathcal{O}$ to check that $\boldsymbol{w}^{\to n-K} \circ \boldsymbol{w}^{\to n-2K} = \boldsymbol{1}_{[n-K+1..n]} \circ \boldsymbol{w}^{\to n-3K}$.

Theorem 4. *The* VOHPR *protocol in Algorithm 3 is a VO protocol for the relation* $\mathcal{R}_{\mathsf{HPR}}$ *with perfect completeness and soundness error* $\frac{H+3K}{|\mathbb{F}|-H-3K}$.

5 Vector Oracle Protocol for PLONK

The constraint system of PLONK characterizes fan-in-two circuits. We formalize this system to an indexed relation referred to as the PLONK relation. To better explain our protocol, we start with a high-level overview of the transformation from fan-in-two circuits to the PLONK relation.

5.1 The PLONK Relation

Assume the circuit contains C gates, including C_a addition gates, C_m multiplication gates and C_c constant gates, where $C = C_c + C_a + C_m$.

For the i-th gate, denote its left input by a_i, its right input by b_i and its output by c_i. We view the circuit as the following constraints over the variables $\{a_i, b_i, c_i\}_{i=1}^{C}$:

- **addition constraints**: let $\mathcal{I}_a \subset [C]$ be the index set for the addition gates, these constraints require that $a_i + b_i = c_i$ for $i \in \mathcal{I}_a$;
- **multiplication constraints**: let $\mathcal{I}_m \subset [C]$ be the index set for the multiplication gates, then $a_i \cdot b_i = c_i$ for $i \in \mathcal{I}_m$;
- **constant constraints**: let $\mathcal{I}_c \subset [C]$ be the index set for the constant gates, and for $i \in \mathcal{I}_c$, denote the constant value of the i-th gate by d_i, which is publicly known as part of the circuit description, it is required that the output of the i-th gate is $c_i = d_i$ for $i \in \mathcal{I}_c$;
- **public input/output**: let $\boldsymbol{w} := (a_i)_{i=1}^{C} \| (b_i)_{i=1}^{C} \| (c_i)_{i=1}^{C} \in \mathbb{F}^{3C}$, and $\mathcal{I}_x \subset [3C]$ be the indices of public inputs/outputs in $\boldsymbol{w}$, then it is required that $\boldsymbol{w}_{[i]} = x_i$ for $i \in \mathcal{I}_x$ and public input/output values x_i;
- **copy constraints**: let $[3C]$ be partitioned into $[3C] = S_1 \cup \cdots \cup S_L$, then it is required that for any pair of $(i, j) \in [3C]^2$ in the same partition, $\boldsymbol{w}_{[i]} = \boldsymbol{w}_{[j]}$. This partition is decided by the wiring of the circuit such that variables connected by wires fall into the same partition.

PLONK collects these variables into three vectors $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c} \in \mathbb{F}^C$. To characterize the above constraints, PLONK introduces five vectors $\boldsymbol{q_L}$, $\boldsymbol{q_R}$, $\boldsymbol{q_O}$, $\boldsymbol{q_M}$ and $\boldsymbol{q_C}$. By properly setting the values of these five vectors, the first three types of constraints are equivalent to

$$\boldsymbol{a} \circ \boldsymbol{q_L} + \boldsymbol{b} \circ \boldsymbol{q_R} + \boldsymbol{c} \circ \boldsymbol{q_O} + \boldsymbol{a} \circ \boldsymbol{b} \circ \boldsymbol{q_M} + \boldsymbol{q_C} = \boldsymbol{0}. \tag{5}$$

PLONK handles the public input/output constraints by dummy gates and the copy constraints by a permutation argument. We omit these details in our definition, and refer interested readers to [17].

Before we present a formal definition of the PLONK relation, we apply an optimization by sorting the gates in the circuit by their types. Concretely, we index the constant gates by $[C_c]$, the multiplication gates by $[C_c + 1..C_c + C_m]$, and the addition gates by $[C_c + C_m + 1..C]$. As a result of sorting, we eliminate the vectors $\boldsymbol{q_L}, \boldsymbol{q_R}, \boldsymbol{q_O}$ and $\boldsymbol{q_M}$, and replace $\boldsymbol{q_C}$ by a shorter vector $\boldsymbol{d} \in \mathbb{F}^{C_c}$, i.e., the vector of all the constant-gate values.

Formally, we define the PLONK relation as

$$\mathcal{R}_{\mathsf{PLONK}} := \left\{ \left(\begin{pmatrix} C_c, C_a, \\ C_m, \boldsymbol{d}, \\ \Pi, \mathcal{I}_x \end{pmatrix}, \boldsymbol{x}, (\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}) \right) \middle| \begin{array}{l} C = C_c + C_a + C_m \\ \Pi \text{ is a partition over } [3C] \\ \boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c} \in \mathbb{F}^C, \boldsymbol{x} \in \mathbb{F}^{3C}, \boldsymbol{d} \in \mathbb{F}^{C_c} \\ \boldsymbol{c}_{[1..C_c]} = \boldsymbol{d} \\ \boldsymbol{a}_{[C_c+1..C_c+C_m]} \circ \boldsymbol{b}_{[C_c+1..C_c+C_m]} \\ \quad = \boldsymbol{c}_{[C_c+1..C_c+C_m]} \\ \boldsymbol{a}_{[C_c+C_m+1..C]} + \boldsymbol{b}_{[C_c+C_m+1..C]} \\ \quad = \boldsymbol{c}_{[C_c+C_m+1..C]} \\ (\boldsymbol{a} \| \boldsymbol{b} \| \boldsymbol{c})_{[i]} = \boldsymbol{x}_{[i]} \text{ for } i \in \mathcal{I}_x \\ \boldsymbol{a} \| \boldsymbol{b} \| \boldsymbol{c} \text{ satisfies the copy} \\ \quad \text{constraint of } \Pi \end{array} \right\}. \quad (6)$$

Note that the sorted PLONK relation is equivalent to the original one up to a quasi-linear-time reduction.

To prove the multiplication constraints, the prover splits $\boldsymbol{b}$ into $\boldsymbol{b}_\times := \boldsymbol{b}_{[1..C_c+C_m]}$ and $\boldsymbol{b}_+ := \boldsymbol{b} - \boldsymbol{b}_\times$. Therefore, the prover submits four vectors to $\mathcal{O}$: $\boldsymbol{a}, \boldsymbol{b}_\times, \boldsymbol{b}_+, \boldsymbol{c}$. The verifier checks the addition, multiplication constraints and constant constraints by the following HAD queries, where the first check ensures that $\boldsymbol{b}_\times$ and $\boldsymbol{b}_+$ contain zeros in appropriate positions:

$$\begin{aligned} \mathbf{1}_{[C_c+C_m+1..C]} \circ \boldsymbol{b}_\times &= \mathbf{1}_{[1..C_c+C_m]} \circ \boldsymbol{b}_+ \\ \mathbf{1}_{[C_c+1..C_c+C_m]} \circ \boldsymbol{c} &= \boldsymbol{a} \circ \boldsymbol{b}_\times \\ \mathbf{1}_{[C_c+C_m+1..C]} \circ (\boldsymbol{a} + \boldsymbol{b}_+ - \boldsymbol{c}) &= \mathbf{0} \\ \mathbf{1}_{[1..C_c]} \circ (\boldsymbol{c} - \boldsymbol{d}) &= \mathbf{0}. \end{aligned}$$

To check the public inputs and outputs, the verifier splits the sparse vector $\boldsymbol{x}$ into $\boldsymbol{x} = \boldsymbol{x}^{(1)} \| \boldsymbol{x}^{(2)} \| \boldsymbol{x}^{(3)}$ each of length C. The verifier then computes the sparse vectors $\boldsymbol{t}^{(1)} := \sum_{i \in [C] \cap \mathcal{I}_x} \boldsymbol{e}_i$, $\boldsymbol{t}^{(2)} := \sum_{i \in [C+1..2C] \cap \mathcal{I}_x} \boldsymbol{e}_i$, and $\boldsymbol{t}^{(3)} := \sum_{i \in [2C+1..3C] \cap \mathcal{I}_x} \boldsymbol{e}_i$. Using $\boldsymbol{t}^{(i)}$ as the masking vectors, the verifier checks the public inputs/outputs constraints by

$$\boldsymbol{t}^{(1)} \circ \left(\boldsymbol{a} - \boldsymbol{x}^{(1)} \right) = \mathbf{0}, \quad \boldsymbol{t}^{(2)} \circ \left(\boldsymbol{b} - \boldsymbol{x}^{(2)} \right) = \mathbf{0}, \quad \boldsymbol{t}^{(3)} \circ \left(\boldsymbol{c} - \boldsymbol{x}^{(3)} \right) = \mathbf{0}.$$

For the copy constraints, we will develop a CopyCheck protocol in the subsequent subsections. Given the vectors $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}$, the CopyCheck protocol verifies that $\boldsymbol{a} \| \boldsymbol{b} \| \boldsymbol{c}$ satisfies the copy constraint of a partition Π. The CopyCheck protocol is an adaption of the permutation-check in PLONK for the VO model.

Combining the Vectors. Similar to SparseMVP, we apply the vector concatenation technique to improve the verifier efficiency at the cost of increased vector size and prover computations.

In more detail, the prover computes the concatenated vector $\boldsymbol{w} := \boldsymbol{a}\|\boldsymbol{b}\|\boldsymbol{c}$ of size $3C$ and submits $\boldsymbol{w}$ to $\mathcal{O}$ instead of submitting $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}$ separately. The verifier then checks the addition and multiplication constraints by HAD queries involving $\boldsymbol{w}$ shifted to different positions. The verifier then checks the public input/output in the same way as before, except that the vectors $\boldsymbol{x}$ and $\boldsymbol{t}$ are not split. See Algorithm 4 for more details. We leave the version without vector concatenation to Appendix E.

Algorithm 4 VOPLONK Protocol

Index: $\boldsymbol{d} \in \mathbb{F}^{C_c}$, $C_a, C_m, \mathcal{I}_x \subset [3C]$, partition Π over $[3C]$ where $C = C_c + C_a + C_m$
Instance: $\boldsymbol{x} \in \mathbb{F}^{3C}$ which is sparse **Witness:** $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c} \in \mathbb{F}^{C}$
Check: $((C_c, C_a, C_m, \boldsymbol{d}, \Pi, \mathcal{I}_x), \boldsymbol{x}, (\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c})) \in \mathcal{R}_{\mathsf{PLONK}}$
Preprocessing:

1: I runs CopyCheck.I with index Π;
2: I submits $\boldsymbol{d}$ to $\mathcal{O}$, sends $\boldsymbol{d}, C_c, C_a, C_m, \mathcal{I}_x$ to P, and $C_c, C_a, C_m, \mathcal{I}_x$ to V.

Online:

1: P submits $\boldsymbol{w} = \boldsymbol{a}\|\boldsymbol{b}\|\boldsymbol{c}$ to $\mathcal{O}$;
2: V queries $\mathcal{O}$ to check that $\boldsymbol{w}^{\to n-C} \circ \boldsymbol{w}^{\to n-2C} = \mathbf{1}_{[n-C_m-C_a+1..n-C_m]} \circ \boldsymbol{w}^{\to n-3C}$;
3: V queries $\mathcal{O}$ to check that $\mathbf{1}_{[3C-C_a+1..3C]} \circ \left(\boldsymbol{w}^{\to 2C} + \boldsymbol{w}^{\to C} - \boldsymbol{w}\right) = \mathbf{0}$;
4: V submits $\boldsymbol{t} = \sum_{i \in \mathcal{I}_x} \boldsymbol{e}_i$ to $\mathcal{O}$ and V queries $\mathcal{O}$ to check that $\boldsymbol{t} \circ (\boldsymbol{w} - \boldsymbol{x}) = \mathbf{0}$;
5: V queries $\mathcal{O}$ to check that $\mathbf{1}_{[2C+1..2C+C_c]} \circ \left(\boldsymbol{w} - \boldsymbol{d}^{\to 2C}\right) = \mathbf{0}$;
6: P and V run the protocol CopyCheck with inputs $\boldsymbol{w}$.

Theorem 5. *The* VOPLONK *protocol in Algorithm 4 is a VO protocol that validates the relation* $\mathcal{R}_{\mathsf{PLONK}}$ *with completeness error* $\frac{3C}{|\mathbb{F}|-1}$*, soundness error* $\frac{15C}{|\mathbb{F}|-1}$.

The rest of this section is devoted to the CopyCheck protocol.

5.2 Product Equality

The most fundamental building block of CopyCheck is the ProductEq protocol, which allows the prover to prove that two vectors $\boldsymbol{u}, \boldsymbol{v} \in \mathbb{F}^n$ satisfy $\prod_{i=1}^{\ell} \boldsymbol{u}_{[i]} = \prod_{i=1}^{\ell} \boldsymbol{v}_{[i]}$ for some $\ell \leq n$. This protocol assumes that $\boldsymbol{u}$ and $\boldsymbol{v}$ contain no zero in their first ℓ elements.

First, the prover submits to $\mathcal{O}$ the *accumulated vector* $\boldsymbol{r}$ defined by:

$$\boldsymbol{r}_{[i]} := \prod_{j=1}^{i} \left(\boldsymbol{u}_{[j]}/\boldsymbol{v}_{[j]}\right) \text{ for } 1 \le i \le \ell.$$

To check that $\boldsymbol{r}$ is indeed computed as defined, note that the definition of $\boldsymbol{r}$ is equivalent to:

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

The verifier may check these equations in a single HAD query. Now $\prod_{i=1}^{\ell} \boldsymbol{u}_{[i]} = \prod_{i=1}^{\ell} \boldsymbol{v}_{[i]}$ is equivalent to $\boldsymbol{r}_{[\ell]} = 1$, which can be checked by another HAD query. See Algorithm 5 for more details.

Algorithm 5 ProductEq Protocol

Input: $\boldsymbol{u}, \boldsymbol{v}, 1 \le \ell \le n$ where $\boldsymbol{u}_{[i]}, \boldsymbol{v}_{[i]} \neq 0$ for any $i \in [\ell]$
Check: $\prod_{i=1}^{\ell} \boldsymbol{u}_{[i]} = \prod_{i=1}^{\ell} \boldsymbol{v}_{[i]}$

1: P computes $\boldsymbol{r} = \left(\prod_{j=1}^{i} (\boldsymbol{u}_{[j]}/\boldsymbol{v}_{[j]})\right)_{i=1}^{\ell} \| \boldsymbol{0}^{n-\ell}$;
2: P submits $\boldsymbol{r}$ to $\mathcal{O}$;
3: V queries $\mathcal{O}$ to check that $(\boldsymbol{r}^{\to n-\ell+1} + \boldsymbol{e}_{n-\ell+1}) \circ \boldsymbol{u}^{\to n-\ell} = \boldsymbol{r}^{\to n-\ell} \circ \boldsymbol{v}^{\to n-\ell}$;
4: V queries $\mathcal{O}$ to check that $(\boldsymbol{r} - \boldsymbol{e}_{\ell}) \circ \boldsymbol{e}_{\ell} = \boldsymbol{0}$.

For the VOPLONK protocol without vector concatenation, we will need the ProductEq protocol to check $\prod_{i\in[\ell],j\in[3]} \boldsymbol{u}_{[i]}^{(j)} = \prod_{i\in[\ell],j\in[3]} \boldsymbol{v}_{[i]}^{(j)}$ given three pairs of vectors $\boldsymbol{u}^{(j)}, \boldsymbol{v}^{(j)}$ for $j \in [3]$. This is accomplished by applying the ProductEq protocol to $\boldsymbol{u} := \boldsymbol{u}^{(1)} \circ \boldsymbol{u}^{(2)} \circ \boldsymbol{u}^{(3)}$ and $\boldsymbol{v} := \boldsymbol{v}^{(1)} \circ \boldsymbol{v}^{(2)} \circ \boldsymbol{v}^{(3)}$. However, a naive implementation of this requires the prover to submit five vectors to $\mathcal{O}$, i.e., $\boldsymbol{u}, \boldsymbol{u}^{(1)} \circ \boldsymbol{u}^{(2)}, \boldsymbol{v}, \boldsymbol{v}^{(1)} \circ \boldsymbol{v}^{(2)}$ and $\boldsymbol{r}$. The TripleProductEq protocol in Algorithm 8 applies a technique to reduce the number of submitted vectors to three. We leave this protocol to Appendix E.

5.3 Permutation

Given vectors $\boldsymbol{u}, \boldsymbol{v} \in \mathbb{F}^n$ and an integer $\ell \le n$, the protocol Permute checks that $\boldsymbol{u}_{[1..\ell]}$ is a reorder of $\boldsymbol{v}_{[1..\ell]}$, i.e., there exists a permutation $\sigma : [\ell] \to [\ell]$ such that $\boldsymbol{u}_{[\sigma(i)]} = \boldsymbol{v}_{[i]}$ for every $i \in [\ell]$, denoted by $\boldsymbol{u}_{[1..\ell]} \sim \boldsymbol{v}_{[1..\ell]}$.

Obviously, $\boldsymbol{u}_{[1..\ell]} \sim \boldsymbol{v}_{[1..\ell]}$ implies that $\mathsf{ProductEq}(\boldsymbol{u}, \boldsymbol{v}, \ell) = 1$. Although the reverse implication is not true, Lemma 3 (in Appendix C)

indicates that if $\boldsymbol{u}_{[1..\ell]} \not\sim \boldsymbol{v}_{[1..\ell]}$ then $\mathsf{ProductEq}(\boldsymbol{u} + \alpha \cdot \mathbf{1}, \boldsymbol{v} + \alpha \cdot \mathbf{1}, \ell) \neq 1$ with overwhelming probability for uniformly random $\alpha \in \mathbb{F}$. This idea is formalized in Algorithm 6, and the tripled version TriplePermute in Algorithm 9 is left to Appendix E.

Algorithm 6 Permute Protocol

Input: $\boldsymbol{u}, \boldsymbol{v} \in \mathbb{F}^n, \ell \leq n$
Check: $\boldsymbol{u}_{[1..\ell]} \sim \boldsymbol{v}_{[1..\ell]}$.

1: V samples $\alpha \xleftarrow{\$} \mathbb{F}^*$ and sends α to P;
2: P and V run the protocol ProductEq with inputs $\boldsymbol{u} + \alpha \cdot \mathbf{1}^\ell, \boldsymbol{v} + \alpha \cdot \mathbf{1}^\ell$ and ℓ.

5.4 Copy Check

Given a vector $\boldsymbol{v} \in \mathbb{F}^n$, the protocol CopyCheck verifies that $\boldsymbol{v}_{[1..\ell]}$ satisfies the copy constraint of some partition Π over $[\ell]$. Assume that Π partitions $[\ell]$ into the non-overlapping union of subsets $S_1 \cup S_2 \cup \cdots \cup S_t$. First, the indexer finds a permutation σ over $[\ell]$ such that for any $i \neq j \in [\ell]$, i and j are partitioned in the same subset S_k if and only if $\sigma^s(i) = j$ for some $s \in \mathbb{Z}$. In another word, the cycles of σ induce a partition, denoted by Π_σ, that is the same as Π. By group theory, such σ exists for every partition and can be found in $O(\ell)$ time.

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

Next, the verifier sends a uniformly random $\beta \xleftarrow{\$} \mathbb{F}$, and the prover and the verifier execute Permute to check that $\boldsymbol{v} + \beta\boldsymbol{\gamma}^\ell$ is a reorder of $\boldsymbol{v} + \beta\boldsymbol{\sigma}$. By Lemma 4, this implies that $\boldsymbol{v}$ remains unchanged after permuted by σ. Therefore, for any $i \neq j \in [\ell]$ in the same S_k, $\sigma^s(i) = j$ implies $\boldsymbol{v}_{[i]} = \boldsymbol{v}_{[\sigma(i)]} = \cdots = \boldsymbol{v}_{[\sigma^s(i)]} = \boldsymbol{v}_{[j]}$, and $\boldsymbol{v}$ satisfies the copy constraints of Π. This is formalized as the CopyCheck protocol in Algorithm 7, and the tripled version in Algorithm 10 in Appendix E.

6 Efficiency Analysis

We compile the VO protocols in the last two sections into PIOP protocols and compare them with other PIOP-based works with respect to the number of polynomial oracles and evaluation queries. Then we transform

Algorithm 7 CopyCheck Protocol

Index: A partition Π over $[\ell]$ **Input:** $\boldsymbol{v} \in \mathbb{F}^n$

Check: $\boldsymbol{v}_{[1..\ell]}$ satisfies the copy constraint of Π.

Preprocessing:

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

Online:

1: V samples $\beta \xleftarrow{\$} \mathbb{F}^*$ and sends β to P;
2: P and V run the protocol Permute with inputs $\boldsymbol{v} + \beta\boldsymbol{\gamma}^\ell, \boldsymbol{v} + \beta\boldsymbol{\sigma}$ and ℓ.

these protocols into zkSNARKs and compare them with concurrent works w.r.t. concrete efficiency. The PIOP protocols and zkSNARKs in our work improves over prior works in several respects. These improvements are attributed to the following three optimization techniques, among which the latter two are uncovered by the VO model.

The first optimization is inspired by a technique in PLONK (attributed to Mary Maller in Sect. 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 new 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 $\alpha \cdot z^{-1}$ 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 z. 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 corresponding polynomials of the right operands in all the HAD and INN queries will never be queried. Therefore, we have carefully ordered the operands in the HAD and INN queries to place most of the prover vectors in the right operands, eliminating more than half of the evaluation queries and one distinct evaluation point.

Finally, we apply the vector concatenation technique to reduce the number of prover vectors and consequently the number of polynomial commitments in the ultimate zkSNARK. However, this technique also increases the maximal polynomial degrees, thus sacrificing the proving

efficiency. To estimate the effects of the vector combination on the proving efficiency, we also include in our comparison the protocols without applying this technique.

6.1 Comparison

We first compare our work with other PIOP protocols in Table 1. For the number of (distinct) evaluations, the VOProof protocols have at most six evaluations and two distinct points, which are smaller than all prior works. For the number of polynomials, the VOProof protocols are also smaller than all prior works, except that the prover-efficient versions have one or two more online polynomials than PLONK. As for the maximal degree, the prover-efficient versions of VOProof are all roughly the same as prior works, except the verifier-efficient version of VOPLONK, whose maximal degree is doubled compared to PLONK.

Table 1. Comparison with other PIOPs. H, K, S are the numbers of rows, columns, and nonzero entries of the matrices in R1CS and HPR, C is the number of gates. The "*" refers to the prover-efficient versions.

relation	protocol	# polynomials offline/online	# evaluations	# distinct points	# max degree
R1CS	Marlin [4]	9/12	18	3	$6S$
	VOR1CS	4/7	4	2	$6S + K$
	VOR1CS*	4/10	6	2	$6S - K$
HPR	Sonic [5]	6/16	16	4	$7K$
	VOHPR	4/7	4	2	$6S + 5K$
	VOHPR*	4/9	5	2	$6S - K$
PLONK	PLONK [3]	8/7	7	2	$3C$
	VOPLONK	2/5	3	2	$5C + C_a$
	VOPLONK*	4/9	5	2	C

Next, we apply the KZG [14] polynomial commitment and the Fiat-Shamir heuristic to compile the VOProof protocols into zkSNARKs. Table 2 compares the proving costs, verification costs, and proof sizes of VOProof with other zkSNARKs with constant-sized verifiers.

For reference, we present in Appendix D the compiled zkSNARKs, which is generated automatically by a Python script powered by `Sympy`. PIOP can also be compiled to zkSNARKs using other polynomial commitments. In particular, DARK [6] also supports homomorphic addition

Table 2. Concrete comparison with other zkSNARKs with constant-sized verifiers. S is the number of nonzero matrix entries, K is the number of matrix columns, C is the number of 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. The "*" refers to the prover-efficient versions.

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]	$3S + 2K$	$O(S\log(S))$	3	ℓ	3	—
	Marlin [4]	$18S + 18K$	$O(S\log(S))$	2	$O(1)$	13	21
	VOR1CS	$27S + 13K$	$O(S\log(S))$	2	17	9	3
	VOR1CS*	$27S + 5K$	$O(S\log(S))$	2	20	12	5
HPR	Sonic [5]	$273K$	$O(K\log(K))$	13	$O(1)$	20	16
	VOHPR	$27S + 30K$	$O(S\log(S))$	2	17	9	3
	VOHPR*	$27S + 6K$	$O(S\log(S))$	2	19	11	4
PLONK	PLONK^1 [3]	$11C$	$O(C\log(C))$	2	16	7	7
	PLONK^2 [3]	$9C$	$O(C\log(C))$	2	18	9	7
	VOPLONK	$26C + 4C_a$	$O(C\log(C))$	2	12	7	2
	VOPLONK*	$10C$	$O(C\log(C))$	2	18	11	4

of commitments thus allows using the optimization techniques discussed before.

Proving Cost. The proving cost is dominated by the group exponentiations and the finite field operations in Fast-Fourier-Transforms (FFT). It is hard to compute the concrete numbers of field operations contributed by the FFT, so we only compare the concrete numbers of group exponentiations. In this metric, both versions of VOHPR outperform Sonic by five to ten times, while both versions of VOR1CS increase this number by 50% compared to Marlin. The prover-efficient version of VOPLONK is roughly the same as PLONK, but the verifier-efficient version increases this number by two to three times.

Verification Cost. Marlin and Sonic do not provide the concrete number of $\mathbb{G}_1$ exponentiations beside the two pairings. Since the pairings are typically far more expensive than group exponentiations, we consider VOProof has roughly the same verification cost as Marlin, while the efficiency of VOHPR improves over Sonic by six times. Meanwhile, the verifier-efficient version of VOPLONK outperforms PLONK by four to six group exponentiations.

Proof Size. The proof sizes of all the verifier-efficient versions of VOProof outperform the prior works (apart from Groth16, which relies on per-

circuit setups). In particular, VOPLONK sets a new record of proof size with two field elements and seven group elements. The prover-efficient versions of both VOR1CS and VOHPR are also smaller than both Marlin and Sonic, while the prover-efficient version of VOPLONK is roughly the same as PLONK.

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 the linear combination of polynomial commitments. Although it is possible to construct our protocols and 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.

For each constraint system, we provide two versions of zkSNARKs prioritizing the verifier or the prover, respectively. We prefer the verifier-efficient version because zkSNARKs have wide applications in blockchains, where the verifiers induce significantly higher costs than the provers. For example, on Ethereum, assuming an ETH is $2000, the transaction fee for verifying each SNARK proof is typically two magnitudes greater than the cost of generating this proof [26]. In scenarios where the prover is more critical, we also have the prover-efficient version available. This flexibility in choosing different trade-offs is another benefit brought by the simplicity and high modularity of VOProof.

Future work. As discussed in Section 3, the VO model, after being modified, can be compiled to PIOP using the Reed-Solomon code basis. This indicates the possibility to unify all prior zkSNARKs in one framework using the language of the VO model. Analyzing existing protocols in this framework may provide more thorough explanations for the different features in prior constructions, and potentially reveal new directions for improvements.

Acknowledgement. This work is partially supported by National Key Research and Development Project 2020YFA0712300.

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. Ariel Gabizon and Zachary J. Williamson. plookup: A simplified polynomial protocol for lookup tables. *IACR Cryptol. ePrint Arch.*, 2020:315, 2020.

18. 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.
19. 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.
20. Alessandro Chiesa, Fermi Ma, Nicholas Spooner, and Mark Zhandry. Post-quantum succinct arguments: Breaking the quantum rewinding barrier. Cryptology ePrint Archive, Report 2021/334, 2021. https://ia.cr/2021/334.
21. 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.
22. Alan Szepieniec and Yuncong Zhang. Polynomial iops for linear algebra relations. *IACR Cryptol. ePrint Arch.*, 2020:1022, 2020.
23. 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.
24. 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.
25. Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Advances in Cryptology - CRYPTO '86*, volume 263 of *LNCS*, pages 186–194. Springer, 1986. https://doi.org/10.1007/3-540-47721-7_12.
26. Zk-rollup development experience sharing, 2021. https://www.fluidex.io/en/blog/zkrollup-intro1/.

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 will define a sequence of models $\{\mathsf{VO}_i\}_{i=0}^{s}$ that starts from the VO model and ends with an alternative formalization of the PIOP model. Correspondingly, starting from the original VO protocol $(\mathsf{VO.I}, \mathsf{VO.P}, \mathsf{VO.V})$, we construct a sequence of

protocols $(\mathsf{VO.I}^{(i)}, \mathsf{VO.P}^{(i)}, \mathsf{VO.V}^{(i)})_{i=0}^{s}$, one for each of the models, and the last of the sequence would be the PIOP protocol with the desired properties.

VO_0. This is exactly the VO model.

VO_1. VO_1 is the same as VO_0, except the following changes:

- When receiving the query $\mathsf{SHR}(\mathsf{name}, \mathsf{src}, k)$, $\mathcal{O}$ sets $\boldsymbol{V}[\mathsf{name}] = \mathbf{0}^k \| \boldsymbol{V}[\mathsf{src}]$, instead of $\boldsymbol{V}[\mathsf{src}]^{\to k} := \mathbf{0}^k \| \boldsymbol{V}[\mathsf{src}]_{[1..n-k]}$, i.e., no longer remove the extra k elements exceeding the length limit. Therefore, in the VO_1 model, $\boldsymbol{V}$ may contain vectors of size larger than n.
- When receiving the query $\mathsf{HAD}(\mathsf{a}, \mathsf{b}, \mathsf{c}, \mathsf{d})$, let $\boldsymbol{a} := \boldsymbol{V}[\mathsf{a}]$, and $\boldsymbol{b}, \boldsymbol{c}, \boldsymbol{d}$ are defined similarly, $\mathcal{O}$ checks $\boldsymbol{a}_{[1..n]} \circ \boldsymbol{b}_{[1..n]} \stackrel{?}{=} \boldsymbol{c}_{[1..n]} \circ \boldsymbol{d}_{[1..n]}$, i.e., the Hadamard relation check is restricted to the "window" of indices from 1 to n.
- Similarly, when receiving the query $\mathsf{INN}(\mathsf{a}, \mathsf{b}, \mathsf{c}, \mathsf{d})$, $\mathcal{O}$ checks $\boldsymbol{a}_{[1..n]} \cdot \boldsymbol{b}_{[1..n]} \stackrel{?}{=} \boldsymbol{c}_{[1..n]} \cdot \boldsymbol{d}_{[1..n]}$.

We claim that VO_1 is equivalent to VO_0 in the sense that for any sequence of queries, the oracles in these two models will give the same sequence of replies. In another word, the aforementioned changes does not affect the behavior of the oracle from the point of view of the parties. To see the equivalence, note that the change to the SHR queries only affects the vectors outside the window, since the vectors are never shifted to the left. Therefore, we can see the oracle in VO_0 as a view of the oracle in VO_1 restricted to the window. As a result of this equivalence, the original VO protocol $(\mathsf{VO.I}, \mathsf{VO.P}, \mathsf{VO.V})$ directly works in the VO_1 model.

VO_2. VO_2 is the same as VO_1, except that the queries $\mathsf{VEC}(\mathsf{name}, \boldsymbol{v})$ allow $|\boldsymbol{v}| \geq n$. Clearly, this change only affects the part of the vectors outside the window, and VO_2 is equivalent to VO_1.

We also modify a bit of the protocol in this model. Specifically, let $\mathsf{VO.P}^{(2)}$ be the same as $\mathsf{VO.P}$ except that for any vector $\boldsymbol{v}$ submitted by $\mathsf{VO.P}$, $\mathsf{VO.P}^{(2)}$ uniformly randomly samples $\boldsymbol{\delta} \in \mathbb{F}^q$ and submits $\boldsymbol{v} \| \boldsymbol{\delta} \in \mathbb{F}^{n+q}$ instead, where q is a small integer to be determined later. This does not affect completeness since the appended $\boldsymbol{\delta}$ never affects the part of the vectors in the window, nor soundness which does not rely on the prover.

VO_3. VO_3 is the same as VO_2, except that the oracle allows a new type of extended Hadamard query. Specifically, $\mathcal{O}$ accepts a new query $\mathsf{EHAD}(\{\mathsf{a}_i, \mathsf{b}_i\}_{i=1}^s)$ and checks that $\sum_{i=1}^s \boldsymbol{a_i}_{[1..n]} \circ \boldsymbol{b_i}_{[1..n]} = \mathbf{0}$. Since this is a new query, this does not affect completeness. This new query is only useful to the verifier therefore soundness is not affected.

VO_4. VO_4 is the same as VO_3, except that the INN query is removed. Assume $\mathsf{VO.V}^{(3)}$ issues the following t_I queries $\{\mathsf{INN}(\mathsf{a}_i, \mathsf{b}_i, \mathsf{c}_i, \mathsf{d}_i)\}_{i=1}^{t_I}$. The protocol $(\mathsf{VO.I}^{(4)}, \mathsf{VO.P}^{(4)}, \mathsf{VO.V}^{(4)})$ is the same as the protocol in the last step, except that $\mathsf{VO.P}^{(4)}$ does not make any INN queries, and $\mathsf{VO.P}^{(4)}$ and $\mathsf{VO.V}^{(4)}$ execute the following steps at the end of the protocol:

1. $\mathsf{VO.V}^{(4)}$ uniformly randomly samples $\beta \in \mathbb{F}^*$ and sends β to $\mathsf{VO.P}^{(4)}$;
2. $\mathsf{VO.P}^{(4)}$ computes $\boldsymbol{r} := \sum_{i=1}^{t_I} \beta^{i-1} \left(\boldsymbol{a}_i \circ \boldsymbol{b}_i - \boldsymbol{c}_i \circ \boldsymbol{d}_i\right)_{[1..n]}$;
3. $\mathsf{VO.P}^{(4)}$ uniformly randomly samples $\boldsymbol{\delta} \in \mathbb{F}^q$, and computes $\tilde{\boldsymbol{r}} := \left(\sum_{j=1}^{i} \boldsymbol{r}_{[j]}\right)_{i=1}^{n} \| \boldsymbol{\delta}$;
4. $\mathsf{VO.P}^{(4)}$ submits $\tilde{\boldsymbol{r}}$;
5. $\mathsf{VO.V}^{(4)}$ submits $\boldsymbol{r'} = \tilde{\boldsymbol{r}} - (0 \| \tilde{\boldsymbol{r}})$;
6. $\mathsf{VO.V}^{(4)}$ queries for $\sum_{i=1}^{t_I} \beta^{i-1} \left(\boldsymbol{a_i}_{[1..n]} \circ \boldsymbol{b_i}_{[1..n]} - \boldsymbol{c_i}_{[1..n]} \circ \boldsymbol{d_i}_{[1..n]}\right) - \boldsymbol{r'}_{[1..n]} \circ \mathbf{1}^n = \mathbf{0}$;
7. $\mathsf{VO.V}^{(4)}$ queries for $\tilde{\boldsymbol{r}}_{[1..n]} \circ \boldsymbol{e}_n = \mathbf{0}$.

Note that step 6 uses an EHAD query and step 7 uses a HAD query.

To justify this change, consider the following statements (where $\boldsymbol{r}$ is defined as in step 2, and $\boldsymbol{r'}$ is defined as in step 5, while $\tilde{\boldsymbol{r}}$, submitted by the untrusted prover, can be arbitrary vector):

$$\boldsymbol{a_i}_{[1..n]} \cdot \boldsymbol{b_i}_{[1..n]} = \boldsymbol{c_i}_{[1..n]} \cdot \boldsymbol{d_i}_{[1..n]} \quad \forall i \in [t_I] \tag{7}$$

$$\sum_{i=1}^{t_I} \beta^{i-1} (\boldsymbol{a_i}_{[1..n]} \cdot \boldsymbol{b_i}_{[1..n]} - \boldsymbol{c_i}_{[1..n]} \cdot \boldsymbol{d_i}_{[1..n]}) = 0 \tag{8}$$

$$\sum_{i=1}^{n} \boldsymbol{r}_{[i]} = 0 \tag{9}$$

$$\tilde{\boldsymbol{r}}_{[i]} = \sum_{j=1}^{i} \boldsymbol{r}_{[j]} \quad \text{for } i \in [n] \tag{10}$$

$$\tilde{\boldsymbol{r}}_{[n]} = 0 \quad \text{(equivalent to step 7 accepted)} \tag{11}$$

$$\boldsymbol{r'}_{[1..n]} = \boldsymbol{r} \quad \text{(equivalent to step 6 accepted)} \tag{12}$$

Assume that the original INN queries are accepted, i.e., (7) is correct, then the best strategy of the prover is to follow the above steps, in which case (10) $\Rightarrow$ (12). Then we have (7) $\Rightarrow$ (8) $\Leftrightarrow$ (9), and (9)(10) $\Rightarrow$ (11).

On the other hand, when $\mathsf{VO.V}^{(4)}$ accepts, which implies (12) and (11) are correct, we have (12) $\Rightarrow$ (10), (10)(11) $\Rightarrow$ (9) $\Leftrightarrow$ (8). By Schwartz-Zippel Lemma, $\Pr[(8) \mid \neg(7)] \leq (t_I - 1)/(|\mathbb{F}| - 1)$. Therefore, if the original INN queries are not accepted, the probability that $\mathsf{VO.V}^{(4)}$ accepts is no more than $(t_I - 1)/(|\mathbb{F}| - 1)$. By the union bound, the soundness error of $(\mathsf{VO.I}^{(4)}, \mathsf{VO.P}^{(4)}, \mathsf{VO.V}^{(4)})$ is bounded by $\varepsilon_s + (t_I - 1)/(|\mathbb{F}| - 1)$.

In this step we eliminate the INN queries, at the cost of one more prover-submitted vector, one more HAD query, one EHAD query, and negligible increment to the soundness error.

VO_5. VO_5 modifies VO_4 by further removing the HAD query. We then modify the protocol and let the verifier use the extended Hadamard query instead. Recall that $\mathsf{VO.V}^{(4)}$ issues $t_H + 1$ HAD queries, denoted by $\{\mathsf{HAD}(\mathsf{a}_i, \mathsf{b}_i, \mathsf{c}_i, \mathsf{d}_i)\}_{i=1}^{t_H+1}$. $\mathsf{VO.V}^{(5)}$ is the same as $\mathsf{VO.V}^{(4)}$, except that it does not make these HAD queries, but replaces them with a single EHAD query in the end of the protocol. To accomplish this, $\mathsf{VO.V}^{(5)}$ uniformly randomly samples $\alpha \in \mathbb{F}^*$, and queries for

$$\sum_{i=1}^{t_H+1} \alpha^{i-1}(\boldsymbol{a_i}_{[1..n]} \circ \boldsymbol{b_i}_{[1..n]} - \boldsymbol{c_i}_{[1..n]} \circ \boldsymbol{d_i}_{[1..n]}) = \mathbf{0}.$$

Note that this corresponds to an EHAD query with $s = 2t_H + 2$ pairs.

Obviously, if the original HAD queries are accepted, this new query will also be accepted. On the other hand, if there exists at least one pair of $i \in [t_H + 1], j \in [n]$ such that $\boldsymbol{a_i}_{[j]} \cdot \boldsymbol{b_i}_{[j]} - \boldsymbol{c_i}_{[j]} \cdot \boldsymbol{d_i}_{[j]} \neq 0$, by the Schwartz-Zippel Lemma, the probability that $\sum_{i=1}^{t_H+1} \alpha^{i-1}(\boldsymbol{a_i}_{[j]} \cdot \boldsymbol{b_i}_{[j]} - \boldsymbol{c_i}_{[j]} \cdot \boldsymbol{d_i}_{[j]}) = 0$ is bounded by $t_H/(|\mathbb{F}| - 1)$, so is the probability that the EHAD query is accepted. By the union bound, the soundness error of $\mathsf{VO.V}^{(5)}$ is $\varepsilon_s + (t_I + t_H - 1)/(|\mathbb{F}| - 1)$.

VO_6. VO_6 modifies VO_5 by requiring that the verifier can only make one EHAD query. $\mathsf{VO.V}^{(5)}$ uses 2 EHAD queries of size $t_I + 1$ and $t_H + 1$ respectively. Denote them by $\mathsf{EHAD}(\{\mathsf{a}_i, \mathsf{b}_i\}_{i=1}^{t_I+1})$ and $\mathsf{EHAD}(\{\mathsf{a}'_i, \mathsf{b}'_i\}_{i=1}^{t_H+1})$. $\mathsf{VO.V}^{(6)}$ is the same as $\mathsf{VO.V}^{(5)}$ except that it replaces the two EHAD queries by a single one. Specifically, $\mathsf{VO.V}^{(6)}$ uniformly randomly samples $\zeta \in \mathbb{F}^*$ and queries for $\mathsf{EHAD}(\{\mathsf{a}_i, \mathsf{b}_i\}_{i=1}^{t_I+1} \cup \{\zeta \cdot \mathsf{a}'_i, \zeta \cdot \mathsf{b}'_i\}_{i=1}^{t_H+1})$. Obviously, if the original two EHAD queries are accepted, this new query is also

accepted; if the original queries are invalid, this new query is accepted with probability at most $1/(|\mathbb{F}|-1)$. Therefore, the soundness error of $\mathsf{VO.V}^{(6)}$ is bounded by $\varepsilon_s + (t_I + t_H)/(|\mathbb{F}|-1)$.

VO_7. VO_7 modifies VO_6 by eliminating the window restriction on the extended Hadamard check. Instead of checking $\sum_{i=1}^{s} \boldsymbol{a_i}_{[1..n]} \circ \boldsymbol{b_i}_{[1..n]} = \mathbf{0}$, the EHAD query checks $\sum_{i=1}^{s} \boldsymbol{a}_i \circ \boldsymbol{b}_i = \mathbf{0}$, where the vectors are zero-padded to the size of the longest vector.

Assume that the EHAD query made by $\mathsf{VO.I}^{(6)}$ is $\mathsf{EHAD}(\{\mathsf{a}_i, \mathsf{b}_i\}_{i=1}^{s})$. Let the protocol $(\mathsf{VO.I}^{(7)}, \mathsf{VO.P}^{(7)}, \mathsf{VO.V}^{(7)})$ be the same as in the last model, except that the EHAD query is replaced by the following steps:

1. $\mathsf{VO.P}^{(7)}$ uniformly randomly samples $\boldsymbol{\delta} \in \mathbb{F}^q$, computes and submits $\boldsymbol{t} = \boldsymbol{\delta} + \sum_{i=1}^{s} \boldsymbol{a}_i \circ \boldsymbol{b}_i$;
2. $\mathsf{VO.V}^{(7)}$ queries for $\sum_{i=1}^{s} \boldsymbol{a}_i \circ \boldsymbol{b}_i - \boldsymbol{t} \circ (\mathbf{0}^n \| \mathbf{1}^{|\boldsymbol{t}|-n}) = \mathbf{0}$.

Obviously, step 2 is accepted if and only if the original EHAD query is accepted, i.e., $\sum_{i=1}^{s} \boldsymbol{a_i}_{[1..n]} \circ \boldsymbol{b_i}_{[1..n]} = \mathbf{0}$, assuming $q \le n$ and the prover is trying the best effort to make the verifier accept. Therefore, completeness error and soundness error are the same as before, while the prover submits one more vector to the oracle.

VO_8. VO_8 modifies VO_7 by adding a new query $\mathsf{EVAL}(\mathsf{name}, z \in \mathbb{F})$ which replies with $f_{\boldsymbol{V}[\mathsf{name}]}(z)$. This query is only available to the verifier, thus does not affect soundness of the protocol.

VO_9. VO_9 modifies VO_8 by removing the EHAD query. We then modify the protocol to replace the only EHAD query by the following procedure.

Let the EHAD query issued by $\mathsf{VO.V}^{(8)}$ be $\mathsf{EHAD}(\{\mathsf{a}_i, \mathsf{b}_i\}_{i=1}^{s})$. Instead of this query, $\mathsf{VO.P}^{(9)}$ and $\mathsf{VO.V}^{(9)}$ execute the following steps:

1. $\mathsf{VO.V}^{(9)}$ uniformly samples $\omega \in \mathbb{F}^*$ and sends ω to $\mathsf{VO.P}^{(9)}$;
2. $\mathsf{VO.P}^{(9)}$ computes $h(X) := X^{2n+q} \cdot \sum_{i=1}^{s} f_{\boldsymbol{a}_i}(\omega X^{-1}) \cdot f_{\boldsymbol{b}_i}(X)$;
3. $\mathsf{VO.P}^{(9)}$ uniformly samples $\delta \in \mathbb{F}$ and computes $\bar{h}(X)$ such that $h(X) = \bar{h}(\gamma X) - \gamma^{2n+q}\bar{h}(X)$ and the X^{2n+q} term of $\bar{h}(X)$ is δ (we will later show that this is only possible if the EHAD query is accepted), where γ is a fixed and publicly known generator of $\mathbb{F}^*$;
4. $\mathsf{VO.P}^{(9)}$ submits the coefficient vector $\boldsymbol{h}$ of $\bar{h}(X)$ to $\mathcal{O}$;
5. $\mathsf{VO.V}^{(9)}$ uniformly samples $z \in \mathbb{F}$. Then for $i \in [s]$, let $u_i := \mathsf{EVAL}(\mathsf{a}_i, \omega \cdot z^{-1})$, $v_i := \mathsf{EVAL}(\mathsf{b}_i, z)$. Let $y_1 := \mathsf{EVAL}(\mathsf{h}, \gamma \cdot z)$, and $y_2 := \mathsf{EVAL}(\mathsf{h}, z)$.
6. $\mathsf{VO.V}^{(9)}$ checks $\sum_{i=1}^{s} u_i \cdot v_i = y_1 - \gamma^{2n+q} \cdot y_2$.

We show that the probability of verifier acceptance is negligibly different from that of the original EHAD. Consider the following statements:

$$\sum_{i=1}^{s} \boldsymbol{a}_i \circ \boldsymbol{b}_i = \mathbf{0} \tag{13}$$

$$\text{The } X^{2n+q} \text{ term of } h(X) \text{ is } 0 \tag{14}$$

$$h(X) = \bar{h}(\gamma X) - \gamma^{2n+q}\bar{h}(X) \tag{15}$$

$$h(z) = \bar{h}(\gamma \cdot z) - \gamma^{2n+q}\bar{h}(z) \tag{16}$$

$$\sum_{i=1}^{s} u_i \cdot v_i = y_1 - \gamma^{2n+q} \cdot y_2 \tag{17}$$

By the definition of $h(X)$, the X^{2n+q} term of $h(X)$ is

$$X^{2n+q} \cdot \sum_{i=1}^{s} \sum_{j=1}^{2n+q} \omega^{j-1} \boldsymbol{a}_{\boldsymbol{i}[j]} \cdot \boldsymbol{b}_{\boldsymbol{i}[j]} = X^{2n+q} f_{\sum_{i=1}^{s} \boldsymbol{a}_i \circ \boldsymbol{b}_i}(\omega).$$

Therefore, (13) $\Rightarrow$ (14), and by Schwartz-Zippel Lemma, (14) $\Rightarrow$ (13) except with probability $(2n+q)/(|\mathbb{F}|-1)$.

If the X^{2n+q} term of $h(X)$ is 0, let $\bar{h}_i = h_i/(\gamma^i - \gamma^{2n+q})$ for $i \in [4n]\backslash\{2n+q\}$ where h_i is the coefficient for X^i in $h(X)$, and for $i = 2n+q$, $\bar{h}_i$ can be arbitrary. It is easy to check that the $\bar{h}(X)$ defined as above satisfies (15). On the other hand, for arbitrary $\bar{h}(X)$, $\bar{h}(\gamma X) - \gamma^{2n+q}\bar{h}(X)$ is guaranteed to have a zero X^{2n+q} term. Therefore, assuming the prover is trying its best to convince the verifier, (14) $\Leftrightarrow$ (15).

(15) $\Rightarrow$ (16) is straightforward, and by Schwartz-Zippel Lemma, (16) $\Rightarrow$ (15) except with probability $(4n+2q)/|\mathbb{F}|$.

Finally, (16) $\Leftrightarrow$ (17) by definition of u_i, v_i and y_1, y_2. In conclusion, if the original EHAD check passes, then $\mathsf{VO.V}^{(9)}$ also accepts, and completeness is preserved. On the other hand, the soundness error is increased by less than $(6n+3q)/(|\mathbb{F}|-1)$.

VO_{10}. VO_{10} modifies VO_9 by requiring that in any EVAL(name, z) query, name must refer to a vector submitted by a VEC query.

We modify the protocol subjecting to the above restriction. Let $\mathsf{VO.V}^{(10)}$ be the same as $\mathsf{VO.V}^{(9)}$ except that for every EVAL(name, z) query, $\mathsf{VO.V}^{(10)}$ executes one of the following according to the type of vector referred to by name:

1. if name is submitted by a POW query, say $\mathsf{POW}(\mathsf{name}, \alpha, k)$, $\mathsf{VO.V}^{(10)}$ computes the reply by $y = \frac{(\alpha z)^k - 1}{\alpha z - 1}$ or $y = k \cdot 1$ if $\alpha z = 1$;
2. if name is submitted by a SPA query, say $\mathsf{SPA}(\mathsf{name}, \boldsymbol{v})$, $\mathsf{VO.V}^{(10)}$ directly computes $y = f_{\boldsymbol{v}}(z)$ using the sparse representation of $\boldsymbol{v}$;
3. if name is submitted by an ADD query, say $\mathsf{ADD}(\mathsf{name}, \mathsf{left}, \mathsf{right})$, let $y_l = \mathsf{EVAL}(\mathsf{left}, z)$ and $y_r = \mathsf{EVAL}(\mathsf{right}, z)$, and $\mathsf{VO.V}^{(10)}$ computes the reply of this query by $y = y_l + y_r$;
4. if name is submitted by an MUL query, say $\mathsf{MUL}(\mathsf{name}, \mathsf{src}, \alpha)$, let $y_s = \mathsf{EVAL}(\mathsf{src}, z)$, then $\mathsf{VO.V}^{(10)}$ computes the reply of this query by $y = \alpha \cdot y_s$;
5. if name is submitted by an SHR query, say $\mathsf{SHR}(\mathsf{name}, \mathsf{src}, k)$, let $y_s = \mathsf{EVAL}(\mathsf{src}, z)$, then $\mathsf{VO.V}^{(10)}$ computes the reply of this query by $y = z^k \cdot y_s$.

The equivalence between $\mathsf{VO.V}^{(10)}$ and $\mathsf{VO.V}^{(9)}$ follows directly from the definition of EVAL query and the POW, SPA, ADD, MUL and SHR queries.

VO_{11}. VO_{11} modifies VO_{10} by removing the POW, SPA, ADD, MUL and SHR queries. Since the vectors submitted by these types of queries are never referenced, we can safely remove all these queries from the protocol.

VO_{11} has only two types of queries left: VEC and EVAL. This model is equivalent to the PIOP model, and the transformation from the protocol $(\mathsf{VO.I}^{(11)}, \mathsf{VO.P}^{(11)}, \mathsf{VO.V}^{(11)})$ to a PIOP protocol $(\mathsf{I}, \mathsf{P}, \mathsf{V})$ is straightforward: every VEC query corresponds to a polynomial oracle sent from the indexer or the prover to the verifier, and every EVAL query corresponds to an evaluation query to a polynomial oracle. Note that the degree bound for the protocol is *at least* $4n + q - 1$, which is one more than the maximal degree of the polynomial $\bar{h}(X)$. However, in PIOP model with higher degree bound, the protocol also works, since VO_{11} does not limit the size of the vectors submitted by the prover.

The number of polynomials sent by the indexer is still m, while the number of polynomials sent by the prover is $r + 3$, where the three additional polynomials correspond to the vectors $\tilde{\boldsymbol{r}}$, $\boldsymbol{t}$ and $\boldsymbol{h}$ respectively. Note that in $\mathsf{VO.V}^{(11)}$ the vector $\tilde{\boldsymbol{r}}$ is queried at most once, $\boldsymbol{t}$ is queried at most once, and $\boldsymbol{h}$ is queried at most twice, i.e., at z and $\gamma \cdot z$ respectively. For every vector submitted by the VEC query in the original protocol, this vector is queried at most twice in the end, once at z and another time at $\omega \cdot z^{-1}$. Therefore, there are at most $2(m + r) + 4$ evaluation queries at 3 distinct points. If the original protocol does not make any INN query, i.e., $t_I = 0$, the vector $\tilde{\boldsymbol{r}}$ is no longer necessary, the prover polynomials becomes $r+2$ and the number of evaluation queries becomes $2(m+r)+3$.

The completeness error is ε_c, and the soundness error is bounded by $\varepsilon_s + (t_I + t_H + 6n + 3q)/(|\mathbb{F}| - 1)$.

Honest-Verifier Statistical Zero-Knowledge. We show that the PIOP protocol $(\mathsf{I}, \mathsf{P}, \mathsf{V})$ is honest-verifier zero-knowledge. Note that every polynomial sent by P, except $f_{\boldsymbol{h}}(X)$, contains q fresh uniformly random coefficients $\boldsymbol{\delta} \in \mathbb{F}^q$, while $f_{\boldsymbol{h}}(X)$ contains a single uniformly random coefficient $\delta \in \mathbb{F}$.

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., the verifier messages in the original protocol together with α, β, ω, z, and the responses from the evaluation queries, i.e., $f_{\boldsymbol{v}_i}(\omega \cdot z^{-1}), f_{\boldsymbol{v}_i}(z)$ for $i \in [m + r + 2]$ where $\boldsymbol{v}_i$ is the i-th vector submitted by the VEC query, and $y_1 = \bar{h}(\gamma \cdot z), y_2 = \bar{h}(z)$.

The simulator S samples the verifier view by simulating a run of the protocol that differs from an honest run in the following respects:

- the prover sends dummy polynomial oracles to the verifier;
- for each evaluation query:
  - if it is a query for $f_{\boldsymbol{v}_i}(\omega z^{-1})$ or $f_{\boldsymbol{v}_i}(z)$ where $i \in [m]$, i.e., this vector is submitted by the indexer, since S has access to the index, S may compute $\boldsymbol{v}_i$ and therefore the polynomial evaluations accordingly;
  - if it is a query for $f_{\boldsymbol{v}_i}(\omega z^{-1})$ or $f_{\boldsymbol{v}_i}(z)$ where $i \in [m+1..m+r+2]$ or for $y_2 = \bar{h}(z)$, i.e., this vector is submitted by the prover, S uniformly randomly sample the query result from $\mathbb{F}$;
  - finally, for the query $y_1 = \bar{h}(\gamma \cdot z)$, compute y_1 according to the identity (17).

We show that the output of the above-defined S only has a negligible statistical difference from the verifier view. Since S has access to all the information that the verifier has, the verifier messages simulated by S follow exactly the same distribution of that of an honest run of the protocol. We only need to show that the query results u_i, v_i for $i \in [m+1..m+r+2]$ and y_2 in the real execution are uniformly random over $\mathbb{F}$ independent of the rest of the verifier view. Consider the following matrices:

$$\boldsymbol{V} = \begin{pmatrix} \boldsymbol{v}_{m+1}^{\mathsf{T}} \\ \boldsymbol{v}_{m+2}^{\mathsf{T}} \\ \vdots \\ \boldsymbol{v}_{m+r+2}^{\mathsf{T}} \end{pmatrix} \quad \boldsymbol{X} = \begin{pmatrix} 1 & 1 \\ \omega \cdot z^{-1} & z \\ (\omega \cdot z^{-1})^2 & z^2 \\ \vdots & \vdots \\ (\omega \cdot z^{-1})^{n+q-1} & z^{n+q-1} \end{pmatrix}.$$

Note that every 2 rows of matrix $\boldsymbol{X}$ form an invertible sub-matrix except when $\omega \cdot z^{-1} = z$ which happens with probability bounded by $\frac{1}{|\mathbb{F}|-1}$. Also note that every row of matrix $\boldsymbol{V}$ contains q uniformly random elements in $\mathbb{F}$. Let $q = 2$, then we have

$$\begin{pmatrix} f_{\boldsymbol{v}_{m+1}}(\omega z^{-1}) & f_{\boldsymbol{v}_{m+1}}(z) \\ f_{\boldsymbol{v}_{m+2}}(\omega z^{-1}) & f_{\boldsymbol{v}_{m+2}}(z) \\ \vdots & \vdots \\ f_{\boldsymbol{v}_{m+r+2}}(\omega z^{-1}) & f_{\boldsymbol{v}_{m+r+2}}(z) \end{pmatrix} = \boldsymbol{V}\boldsymbol{X}$$

is uniformly random over $\mathbb{F}^{(r+2)\times 2}$. Finally, since $\boldsymbol{h}$ contains one element $\delta \in \mathbb{F}$ that is uniformly random over $\mathbb{F}$, y_2 is also uniformly random. In conclusion, the statistical difference between the output of S and the verifier view is at most $\frac{1}{|\mathbb{F}|-1}$.

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. Note that in the end, the verifier verifies a single identity $\sum_{i=1}^{s} u_i v_i = y_1 - \gamma^{2n+q} \cdot y_2$, where u_i and v_i are linear combinations of $\{f_{\boldsymbol{v}_i}(\omega z^{-1})\}_{i\in[m+r+2]} \cup \{1\}$, and linear combinations of $\{f_{\boldsymbol{v}_i}(z)\}_{i\in[m+r+2]} \cup \{1\}$, respectively. Let $\boldsymbol{u} = (f_{\boldsymbol{v}_1}(\omega z^{-1}), \cdots, f_{\boldsymbol{v}_{m+r+2}}(\omega z^{-1}), y_1, 1)$ and $\boldsymbol{v} = (f_{\boldsymbol{v}_1}(z), \cdots, f_{\boldsymbol{v}_{m+r+2}}(z), y_2, 1)$, the above identity can be rewritten in the form

$$\boldsymbol{u}^{\mathsf{T}}\boldsymbol{M}\boldsymbol{v} = 0$$

where the matrix $\boldsymbol{M}$ is known to the verifier.

Since all the variables in $\boldsymbol{v}$ are queried at point z, V checks the above identity using the following strategy. V makes at most $m+r+3$ evaluation queries to obtain the vector $\boldsymbol{u}$. Then V computes $\boldsymbol{c} := \boldsymbol{u}^{\mathsf{T}}\boldsymbol{M}$. Finally, V linearly combines the polynomial oracles of $f_{\boldsymbol{v}_1}(X), \cdots, f_{\boldsymbol{v}_{m+r+2}}(X), \bar{h}(X)$ and 1 using $\boldsymbol{c}$ as the coefficients, and obtains the oracle for a polynomial $g(X)$. V queries $g(X)$ at z and checks if $g(z) = 0$. This optimization also

reduces the number of query points for each polynomial oracle to 1, i.e., at ωz^{-1} only, therefore $q = 1$ suffices to ensure zero-knowledge.

The ability to linearly combine the polynomial oracles also allows another strategy that eliminates the evaluation point $\gamma \cdot z$ at the cost of one more polynomial oracle. This trade-off is worthwhile because it results in smaller zkSNARK proofs and also reduces the degree bound from $4n + q - 1$ to $2n + q - 1$.

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^{2n+q} 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 $2n+q-1$, such that $h(X) = \bar{h}_1(X)+X^{2n+q+1}\cdot\bar{h}_2(X)$. Now soundness relies on the ability to limit the degree of the polynomials sent by the prover, $\bar{h}_1(X)$ and $\bar{h}_2(X)$ should be shifted to align with the degree bound D. In detail, P sends the polynomials $\bar{h}'_1(X) := X^{D-(2n+q-1)} \cdot \bar{h}_1(X)$ and $\bar{h}'_2(X) := X^{D-(2n+q-1)} \cdot \bar{h}_1(X)$ to V, who checks $\sum u_i v_i = (\bar{h}_1(z) + z^{2n+q+1}\bar{h}_2(z)) \cdot z^{2n+q-1-D}$ by putting both $\bar{h}_1(z)$ and $\bar{h}_2(z)$ in the vector $\boldsymbol{v}$. This optimization eliminates the query response $\bar{h}(\gamma \cdot z)$ from the zkSNARK proof together with one distinct evaluation point (i.e., $\gamma \cdot x$), and reduces the degree bound to $2n + q - 1$, 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 simulate polynomial oracle using VEC and INN queries, and prove the verifier computation result using the $\mathsf{VOPLONK}$ protocol.

The algorithms I and P work in the same way as $\mathsf{PIOP.I}$ and, $\mathsf{PIOP.P}$, except that whenever $\mathsf{PIOP.I}$ or $\mathsf{PIOP.P}$ sends to $\mathsf{PIOP.V}$ a polynomial oracle $[f(X)]$ of degree at most $n - 1$ to $\mathsf{PIOP.V}$, P instead submits its coefficient vector $\boldsymbol{v}$ of size n to $\mathcal{O}$.

Finally, V sends to P all the random coins that are needed to compute the evaluation queries $z_1, \cdots, z_s$. Denote the polynomial oracle to which z_i is queried to by $[f^{(k_i)}(X)]$. P computes the query results $y_1, \cdots, y_s$ where $y_i = f^{(k_i)}(z_i)$.

Since the PIOP protocol is public coin, at the end of the interaction, all the verifier randomnesses are public coins which are already learned by the prover. Denote by ω all the public coins the verifier used, then the verifier response is a deterministic function of $(\omega, \mathbb{i}_V, \mathbb{x}, y_1, \cdots, y_s)$. Denote this function by F. Then P convinces V that $\mathsf{F}(\omega, \mathbb{i}_V, \mathbb{x}, y_1, \cdots, y_s) = 1$ using the $\mathsf{VOPLONK}$ protocol presented in Algorithm 11, with the following

differences: the preprocessed vectors are hard-coded into P and V, and submitted to $\mathcal{O}$ 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 $\mathsf{PIOP.V}$.

Recall that in Algorithm 11, P sends to V a vector $\boldsymbol{w}$ that contains all the input/output of the gates in the circuit, including all of the query results $y_1, \cdots, y_s$. Assume the positions of these values in $\boldsymbol{v}$ are $\ell_1, \cdots, \ell_s$ respectively. 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 $i \in [s]$, V checks $f^{(k_i)}(z_i) = y_i$ by an INN query for

$$\boldsymbol{w}_{k_i} \cdot (1, z_i, \cdots, z_i^{n-1}) = \boldsymbol{w} \cdot \boldsymbol{e}_{\ell_i}$$

where $\boldsymbol{v}_{k_i}$ is the coefficient vector of $f^{(k_i)}(X)$.

Completeness follows from those of the PIOP protocol and the $\mathsf{VOPLONK}$ protocol by design. Therefore, the completeness error is $\varepsilon_c + O(V/|\mathbb{F}|)$. For soundness, note that if V accepts then $f^{(k_i)}(z_i) = y_i$ for every $i \in [s]$. Therefore, by soundness of the PIOP protocol, the probability that all the INN queries are accepted when $(\mathtt{i}, \mathtt{x})$ is invalid is bounded by ε_s. Finally, by soundness of the $\mathsf{VOPLONK}$ protocol, the probability that V accepts while $\mathsf{F}(\omega, \mathtt{i}_V, \mathtt{x}, y_1, \cdots, y_s) \neq 1$ 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 $\mathsf{VOPLONK}$ protocol. The number of HAD queries is 4 brought by the $\mathsf{VOPLONK}$ protocol. The number of INN queries is s, i.e., one for each evaluation query.

C Security Proofs for Vector Oracle Protocols

C.1 Lemmas

Lemma 3 (restate of Lemma A.3 of [17]). *Let $\boldsymbol{u}, \boldsymbol{v} \in \mathbb{F}^\ell$. If $\boldsymbol{u}$ and $\boldsymbol{v}$ are not permutations of each other, then for uniformly random $\alpha \in \mathbb{F}$, the probability that $\prod_{i\in[\ell]}(\boldsymbol{u}_{[i]} + \alpha) = \prod_{i\in[\ell]}(\boldsymbol{v}_{[i]} + \alpha)$ is bounded by $\frac{\ell}{|\mathbb{F}|}$.*

Definition 6 (Simultaneous Permutation). *Let $\{\boldsymbol{u}^{(j)}\}_{j=1}^m, \{\boldsymbol{v}^{(j)}\}_{j=1}^m$ be two groups of vectors in $\mathbb{F}^\ell$. We say they are simultaneous permutations of each other, denoted by $\{\boldsymbol{u}^{(j)}\}_{j=1}^m \sim \{\boldsymbol{v}^{(j)}\}_{j=1}^m$, if there exists a permutation σ over $[\ell]$ such that $\boldsymbol{u}^{(j)}_{[\sigma(i)]} = \boldsymbol{v}^{(j)}_{[i]}$ for any $i \in [\ell]$ and $j \in [m]$.*

Lemma 4. *Let* $\{\boldsymbol{u}^{(j)}\}_{j=1}^m, \{\boldsymbol{v}^{(j)}\}_{j=1}^m$ *be two groups of vectors in* $\mathbb{F}^\ell$*. If* $\{\boldsymbol{u}^{(j)}\}_{j=1}^m \not\sim \{\boldsymbol{v}^{(j)}\}_{j=1}^m$*, then for uniformly random* $\alpha \in \mathbb{F}$*, the probability that* $\sum_{j=1}^m \alpha^{j-1}\boldsymbol{u}^{(j)} \sim \sum_{j=1}^m \alpha^{j-1}\boldsymbol{v}^{(j)}$ *is bounded by* $\frac{2(m-1)\ell}{|\mathbb{F}|}$*.*

Proof. Consider two sequences of tuples $U := \{(\boldsymbol{u}_{[i]}^{(1)}, \cdots, \boldsymbol{u}_{[i]}^{(m)})\}_{i\in[\ell]}$ and $V := \{(\boldsymbol{v}_{[i]}^{(1)}, \cdots, \boldsymbol{v}_{[i]}^{(m)})\}_{i\in[\ell]}$. Since $\{\boldsymbol{u}^{(j)}\}_{j=1}^m \not\sim \{\boldsymbol{v}^{(j)}\}_{j=1}^m$, obviously U and V are not permutations of each other. Therefore, there exists some tuple $(a_1, a_2, \cdots, a_m)$ that appears different number of times in U and V. Let $\mathcal{I}_u$ be the index set where $(\boldsymbol{u}_{[i]}^{(1)}, \cdots, \boldsymbol{u}_{[i]}^{(m)}) = (a_1, a_2, \cdots, a_m)$ and $\mathcal{I}_v$ be the similarly defined for V. Then $|\mathcal{I}_u| \neq |\mathcal{I}_v|$. For any tuple $(b_1, b_2, \cdots, b_m) \neq (a_1, a_2, \cdots, a_m)$, for uniformly random α over $\mathbb{F}$, the probability that $\sum_{j=1}^m \alpha^{j-1}a_j = \sum_{j=1}^m \alpha^{j-1}b_j$ is bounded by $\frac{m-1}{|\mathbb{F}|}$, due to Schwartz-Zippel Lemma. By the union bound, the probability that $\sum_{j=1}^m \alpha^{j-1}a_j = \sum_{j=1}^m \alpha^{j-1}\boldsymbol{u}_{[i]}^{(j)}$ for any $i \notin \mathcal{I}_u$ or $\sum_{j=1}^m \alpha^{j-1}a_j = \sum_{j=1}^m \alpha^{j-1}\boldsymbol{v}_{[i]}^{(j)}$ for any $i \notin \mathcal{I}_v$ is less than $\frac{2(m-1)\ell}{|\mathbb{F}|}$.

Therefore, except with probability $\frac{2(m-1)\ell}{|\mathbb{F}|}$, the value $\sum_{j=1}^m \alpha^{j-1}a_j$ appears exactly $|\mathcal{I}_u|$ times in vector $\sum_{j=1}^m \alpha^{j-1}\boldsymbol{u}^{(j)}$ and $|\mathcal{I}_v|$ (i.e. $\neq |\mathcal{I}_u|$) times in vector $\sum_{j=1}^m \alpha^{j-1}\boldsymbol{v}^{(j)}$, which ensures that the two vectors are not permutations of each other.

C.2 Security Proofs for the Protocols

Theorem 6. *The* SparseMVP *protocol in Algorithm 1 has perfect completeness and soundness error* $\frac{H+K}{|\mathbb{F}|-H-K}$*.*

Proof. Consider the following statements:

$$\boldsymbol{a}_{[1..H]} = \boldsymbol{M}\boldsymbol{a}_{[H+1..H+K]} \tag{18}$$

$$\boldsymbol{s}_{[1..H+K]} = \boldsymbol{r}_\alpha \| (-\boldsymbol{r}_\alpha^\mathsf{T}\boldsymbol{M}) \tag{19}$$

$$\boldsymbol{s} \circ \left(\alpha \cdot \mathbf{1}^H - \boldsymbol{\gamma}^H\right) = \mathbf{1}^H \tag{20}$$

$$\boldsymbol{s}^{\to n-H-K} \cdot \boldsymbol{a}^{\to n-H-K} = 0 \tag{21}$$

$$\boldsymbol{h}_{[1..S+K]} = \boldsymbol{r}_\beta \| \left(\frac{1}{(\alpha - \boldsymbol{u}_{[i]})(\beta - \boldsymbol{w}_{[i]})}\right)_{i=1}^S \tag{22}$$

$$\boldsymbol{h} \circ \left(\beta \cdot \mathbf{1}^K - \boldsymbol{\gamma}^K\right) = \mathbf{1}^K \tag{23}$$

$$\boldsymbol{h} \circ (\alpha\beta \cdot \mathbf{1}^S - \alpha \cdot \boldsymbol{w} - \beta \cdot \boldsymbol{u} + \boldsymbol{y})^{\to K} = (\mathbf{1}^S)^{\to K} \tag{24}$$

$$- \boldsymbol{h}^{\to n-K} \cdot \boldsymbol{s}^{\to n-H-K} = \boldsymbol{h}^{\to n-S-K} \cdot \boldsymbol{v}^{\to n-S} \tag{25}$$

Completeness follows from the fact that if the prover is honest then all of (18)(19) and (22) are true, and the following implications:

- (18)(19) ⇒ (21)
- (19) ⇒ (20)
- (22) ⇒ (23)(24)
- (19)(22) ⇒ (25)

Therefore, completeness error is 0.

Soundness follows from the fact that (20)(21)(23)(24) and (25) hold if the verifier accepts, and the following implications:

- (23)(24) ⇒ (22)
- (20)(22)(25) ⇒ (19) which fails with probability $\frac{K}{|\mathbb{F}|-K}$
- (19)(21) ⇒ (18) which fails with probability $\frac{H}{|\mathbb{F}|-H}$

Therefore, soundness error of the SparseMVP protocol is less than $\frac{H+K}{|\mathbb{F}|-H-K}$. □

Theorem 3. *The* VOR1CS *protocol in Algorithm 2 is a VO protocol for the relation* $\mathcal{R}_{\mathsf{R1CS}}$ *with perfect completeness and soundness error* $\frac{3H+K}{|\mathbb{F}|-3H-K}$.

Proof. Consider the following statements:

$$(\boldsymbol{A}(1\|\boldsymbol{x}\|\boldsymbol{w})) \circ (\boldsymbol{B}(1\|\boldsymbol{x}\|\boldsymbol{w})) = \boldsymbol{C}(1\|\boldsymbol{x}\|\boldsymbol{w}) \tag{26}$$

$$\boldsymbol{u}_{[1..3H+K]} = (\boldsymbol{M}\|1\|\boldsymbol{x}\|\boldsymbol{w})\|1\|\boldsymbol{x}\|\boldsymbol{w} \tag{27}$$

$$\boldsymbol{u}_{[1..3H]} = \boldsymbol{M}\boldsymbol{u}_{[3H+1..3H+K]} \tag{28}$$

$$\mathsf{V} \text{ accepts in the subprotocol } \mathsf{SparseMVP} \tag{29}$$

$$\boldsymbol{u}_{[1..H]} \circ \boldsymbol{u}_{[H+1..2H]} = \boldsymbol{u}_{[2H+1..3H]} \tag{30}$$

$$\boldsymbol{u}^{\to n-H} \circ \boldsymbol{u}^{\to n-2H} = \mathbf{1}_{[n-H+1..n]} \circ \boldsymbol{u}^{\to n-3H} \tag{31}$$

$$\mathbf{1}_{[3H+1..3H+\ell+1]} \circ (\boldsymbol{u} - \boldsymbol{e}_{3H+1} - \boldsymbol{x}^{\to 3H+1}) = \mathbf{0} \tag{32}$$

Completeness follows from the fact that (26) and (27) hold if the prover is honest, and the sequence of implications (27) ⇒ (28) ⇒ (29) that follows from completeness of SparseMVP, (26)(27) ⇒ (30) ⇒ (31), and (27) ⇒ (32). Therefore, the protocol VOR1CS is perfectly complete.

Soundness follows from the fact that (29)(31) and (32) hold if the verifier accepts, and the sequence of implications (29) ⇒ (28) due to

soundness of protocol SpaseMVP and fails with probability $\frac{3H+K}{|\mathbb{F}|-3H-K}$, (28)(32) $\Rightarrow$ (27), and (27)(30) $\Rightarrow$ (26). Therefore, soundness error of the VOR1CS protocol is bounded by $\frac{3H+K}{|\mathbb{F}|-3H-K}$. □

Theorem 4. *The* VOHPR *protocol in Algorithm 3 is a VO protocol for the relation* $\mathcal{R}_{\mathsf{HPR}}$ *with perfect completeness and soundness error* $\frac{H+3K}{|\mathbb{F}|-H-3K}$.

Proof. Consider the following statements:

$$\boldsymbol{A}\boldsymbol{w}_{[1..K]} + \boldsymbol{B}\boldsymbol{w}_{[K+1..2K]} + \boldsymbol{C}\boldsymbol{w}_{[2K+1..3K]} + \boldsymbol{d} = \boldsymbol{x}\|\boldsymbol{0}^{H-\ell} \tag{33}$$

$$\boldsymbol{w}_{[1..K]} \circ \boldsymbol{w}_{[K+1..2K]} = \boldsymbol{w}_{[2K+1..3K]} \tag{34}$$

$$\boldsymbol{M}(\boldsymbol{e}_1 + \boldsymbol{w}^{\to 1}) = \boldsymbol{x}\|\boldsymbol{0}^{H-\ell} \tag{35}$$

$$\mathsf{V} \text{ accepts in the subprotocol } \mathsf{SparseMVP} \tag{36}$$

$$\boldsymbol{w}^{\to n-K} \circ \boldsymbol{w}^{\to n-2K} = \boldsymbol{1}_{[n-K+1..n]} \circ \boldsymbol{w}^{\to n-3K} \tag{37}$$

Completeness follows from the fact that (33) and (34) hold if the prover is honest, and the sequence of implications (33) $\Rightarrow$ (35) $\Rightarrow$ (36) where the second implication follows from completeness of SparseMVP, and (34) $\Rightarrow$ (37). Therefore, the protocol VOHPR is perfectly complete.

Soundness follows from the fact that (36) and (37) hold if the verifier accepts, and the sequence of implications (36) $\Rightarrow$ (35) $\Rightarrow$ (33) where the first implication follows from soundness of protocol SpaseMVP and fails with probability $\frac{H+3K}{|\mathbb{F}|-H-3K}$, and (37) $\Rightarrow$ (34). Therefore, soundness error of the VOR1CS protocol is bounded by $\frac{H+3K}{|\mathbb{F}|-H-3K}$. □

Theorem 5. *The* VOPLONK *protocol in Algorithm 4 is a VO protocol that validates the relation* $\mathcal{R}_{\mathsf{PLONK}}$ *with completeness error* $\frac{3C}{|\mathbb{F}|-1}$, *soundness error* $\frac{15C}{|\mathbb{F}|-1}$.

Proof. Consider the following statements:

$$\boldsymbol{a}_{[C_c+1..C_c+C_m]} \circ \boldsymbol{b}_{[C_c+1..C_c+C_m]} = \boldsymbol{c}_{[C_c+1..C_c+C_m]} \tag{38}$$

$$\boldsymbol{a}_{[C_c+C_m+1..C]} + \boldsymbol{b}_{[C_c+C_m+1..C]} = \boldsymbol{c}_{[C_c+C_m+1..C]} \tag{39}$$

$$\boldsymbol{c}_{[1..C_c]} = \boldsymbol{d} \tag{40}$$

$$\boldsymbol{t}_{[i]} \neq 0 \Rightarrow (\boldsymbol{a}\|\boldsymbol{b}\|\boldsymbol{c})_{[i]} = \boldsymbol{x}_{[i]} \qquad \forall i \in [3C] \tag{41}$$

$$\boldsymbol{a}\|\boldsymbol{b}\|\boldsymbol{c} \text{ satisfies the copy constraints of } \Pi \tag{42}$$

$$\boldsymbol{1}_{[3C-C_a+1..3C]} \circ (\boldsymbol{w}^{\to 2C} + \boldsymbol{w}^{\to C} - \boldsymbol{w}) = \boldsymbol{0} \tag{43}$$

$$\boldsymbol{w}^{\to n-C} \circ \boldsymbol{w}^{\to n-2C} = \mathbf{1}_{[n-C_m-C_a+1..n-C_a]} \circ \boldsymbol{w}^{\to n-3C} \tag{44}$$

$$\boldsymbol{t} \circ (\boldsymbol{w} - \boldsymbol{x}) = \mathbf{0} \tag{45}$$

$$\mathbf{1}_{[2C+1..2C+C_c]} \circ \left(\boldsymbol{w} - \boldsymbol{d}^{\to 2C}\right) = \mathbf{0} \tag{46}$$

$$\mathsf{V} \text{ accepts in the subprotocol } \mathsf{CopyCheck} \tag{47}$$

Note that $\boldsymbol{w} = \boldsymbol{a} \| \boldsymbol{b} \| \boldsymbol{c}$ holds unconditionally, because the existence of valid $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}$ is equivalent to the existence of valid $\boldsymbol{w}$.

Completeness follows from the fact that if the prover is honest then all of (38)(39)(40)(41) and (42) are true, and the following implications: (38) $\Leftrightarrow$ (44), (39) $\Leftrightarrow$ (43), (41) $\Leftrightarrow$ (45), (40) $\Leftrightarrow$ (46) and (42) $\Rightarrow$ (47) that follows from completeness of the subprotocol CopyCheck which fails with probability $\frac{3C}{|\mathbb{F}|-1}$.

Soundness follows from the fact that (43)(44)(45)(46) and (47) hold if the verifier accepts. The reverse implication fails only if (47) does not imply (42), which happens with probability at most $\frac{15C}{|\mathbb{F}|-1}$, due to soundness error of CopyCheck. □

Theorem 7. *The* ProductEq *protocol in Algorithm 5 is perfectly complete and perfectly sound.*

Proof. Consider the following statements:

$$\prod_{i=1}^{\ell} \boldsymbol{u}_{[i]} = \prod_{i=1}^{\ell} \boldsymbol{v}_{[i]} \tag{48}$$

$$\boldsymbol{r}_{[1..\ell]} = \left(\prod_{j=1}^{i} \boldsymbol{u}_{[i]} / \boldsymbol{v}_{[i]} \right)_{i=1}^{\ell} \tag{49}$$

$$\boldsymbol{r}_{[\ell]} = 1 \tag{50}$$

$$(\boldsymbol{r} - \boldsymbol{e}_\ell) \circ \boldsymbol{e}_\ell = \mathbf{0} \tag{51}$$

$$\boldsymbol{u}_{[1]} \cdot 1 = \boldsymbol{r}_{[1]} \cdot \boldsymbol{v}_{[1]} \text{ and } \forall i \in [2..\ell], \boldsymbol{u}_{[i]} \cdot \boldsymbol{r}_{[i-1]} = \boldsymbol{r}_{[i]} \cdot \boldsymbol{v}_{[i]}, \tag{52}$$

$$(\boldsymbol{r}^{\to n-\ell+1} + \boldsymbol{e}_{n-\ell+1}) \circ \boldsymbol{u}^{\to n-\ell} = \boldsymbol{r}^{\to n-\ell} \circ \boldsymbol{v}^{\to n-\ell} \tag{53}$$

Completeness follows from the fact that (48) and (49) hold when the prover is honest, and the sequences of implications (48)(49) $\Rightarrow$ (50) $\Rightarrow$ (51), and (49) $\Rightarrow$ (52) $\Rightarrow$ (53).

Soundness follows from the fact that (51) and (53) hold when the verifier accepts, and the sequences of implications (51) $\Rightarrow$ (50), (53) $\Rightarrow$ (52), and (52)(50) $\Rightarrow$ (48). □

Theorem 8. *The* Permute *protocol in Algorithm 6 has completeness error* $\frac{\ell}{|\mathbb{F}|-1}$*, soundness error* $\frac{3\ell}{|\mathbb{F}|-1}$*.*

Proof. For completeness, note that if $\boldsymbol{u}_{[1..\ell]} \sim \boldsymbol{v}_{[1..\ell]}$ then $\prod_{i=1}^{\ell}(\boldsymbol{u}_{[i]} + \alpha) = \prod_{i=1}^{\ell}(\boldsymbol{v}_{[i]} + \alpha)$. Except with probability $\frac{\ell}{|\mathbb{F}|-1}$, $-\alpha \notin \boldsymbol{u}_{[1..\ell]}$, which is equivalent to $-\alpha \notin \boldsymbol{v}_{[1..\ell]}$. By the perfect completeness of ProductEq protocol, completeness error of Permute protocol is $\frac{\ell}{|\mathbb{F}|-1}$.

For soundness, note that if $\boldsymbol{u}_{[1..\ell]} \not\sim \boldsymbol{v}_{[1..\ell]}$, then by Lemma 3, $\prod_{i=1}^{\ell}(\boldsymbol{u}_{[i]}+\alpha) \neq \prod_{i=1}^{\ell}(\boldsymbol{v}_{[i]} + \alpha)$ except with probability $\frac{\ell}{|\mathbb{F}|}$. Moreover, except with probability $\frac{2\ell}{|\mathbb{F}|-1}$, $-\alpha \notin \boldsymbol{u}_{[1..\ell]} \cup \boldsymbol{v}_{[1..\ell]}$. By the perfect soundness of ProductEq protocol, soundness error of Permute is bounded by $\frac{3\ell}{|\mathbb{F}|-1}$.

□

Theorem 9. *The* CopyCheck *protocol in Algorithm 7 has completeness error* $\frac{\ell}{|\mathbb{F}|-1}$*, soundness error* $\frac{5\ell}{|\mathbb{F}|-1}$*.*

Proof. For completeness, note that if $\boldsymbol{v}_{[1..\ell]}$ satisfies the copy constraint of Π, then $\boldsymbol{v}_{[i]} = \boldsymbol{v}_{[\sigma(i)]}$ for every $i \in [\ell]$, and $\boldsymbol{v}_{[1..\ell]} + \beta \cdot \boldsymbol{\gamma}$ is a reorder of $\boldsymbol{v}_{[1..\ell]} + \beta \cdot \boldsymbol{\sigma}$. By completeness of Permute protocol, completeness error of CopyCheck is at most $\frac{\ell}{|\mathbb{F}|-1}$.

For soundness, note that if $\boldsymbol{v}_{[1..\ell]}$ does not satisfy the copy constraint of Π, then $\boldsymbol{v}_{[i]} \neq \boldsymbol{v}_{[\sigma(i)]}$ for some i, therefore $\{\boldsymbol{v}_{[1..\ell]}, \boldsymbol{\gamma}\}$ and $\{\boldsymbol{v}_{[1..\ell]}, \boldsymbol{\sigma}\}$ are not simultaneous permutations of each other. By Lemma 4, except with probability $\frac{2\ell}{|\mathbb{F}|}$, $\boldsymbol{v}_{[1..\ell]} + \beta \cdot \boldsymbol{\gamma}$ is not a reorder of $\boldsymbol{v}_{[1..\ell]} + \beta \cdot \boldsymbol{\sigma}$. By soundness of Permute protocol, soundness error of CopyCheck is bounded by $\frac{5\ell}{|\mathbb{F}|-1}$.

□

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. The latex code for the provers and the verifiers are produced by a Python script which implements the VO compiler and the PIOP compiler in a symbolic way. This script uses the open-source python library `Sympy`[4] for symbolic computations.

[4] https://github.com/sympy/sympy

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

$$e(F+z\cdot W+r'z'\cdot W'+r''z''\cdot W'', [1]_2)\cdot e(-W-r'\cdot W'-r''\cdot W'', [x]_2) = 1.$$

Remark 2. 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 $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C} \in \mathbb{F}^{H\times K}$ be matrices each has at most S nonzero entries. For an instance $\boldsymbol{x} \in \mathbb{F}^\ell$, the zkSNARK $\mathsf{VOR1CS}$ 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 $D \geq 2K + 6S + 2$.

Indexer. The inputs are the sparse representations for the matrices, 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 $\mathsf{row}_i \in [H], \mathsf{col}_i \in [K]$ and $\mathsf{val}_i \in \mathbb{F}$.

1. $\boldsymbol{u} := (\gamma^{\mathsf{row}_i})_{i=1}^{3S}$
2. $\boldsymbol{w} := \left(\gamma^{\mathsf{col}_i}\right)_{i=1}^{3S}$
3. $\boldsymbol{v} := (\mathsf{val}_i)_{i=1}^{3S}$
4. $\boldsymbol{y} := \boldsymbol{u} \circ \boldsymbol{w}$
5. $\mathsf{cm}_{\boldsymbol{u}} := \mathsf{com}\left(f_{\boldsymbol{u}}(X), \mathsf{srs}\right)$

6. $\mathsf{cm}_{\boldsymbol{w}} := \mathsf{com}\left(f_{\boldsymbol{w}}(X), \mathsf{srs}\right)$
7. $\mathsf{cm}_{\boldsymbol{v}} := \mathsf{com}\left(f_{\boldsymbol{v}}(X), \mathsf{srs}\right)$
8. $\mathsf{cm}_{\boldsymbol{y}} := \mathsf{com}\left(f_{\boldsymbol{y}}(X), \mathsf{srs}\right)$
9. Output
   - $\mathsf{pk} := (\mathsf{cm}_{\boldsymbol{u}}, \mathsf{cm}_{\boldsymbol{w}}, \mathsf{cm}_{\boldsymbol{v}}, \mathsf{cm}_{\boldsymbol{y}})$
   - $\mathsf{vk} := (\boldsymbol{u}, \boldsymbol{w}, \boldsymbol{v}, \boldsymbol{y}, \mathsf{cm}_{\boldsymbol{u}}, \mathsf{cm}_{\boldsymbol{w}}, \mathsf{cm}_{\boldsymbol{v}}, \mathsf{cm}_{\boldsymbol{y}})$

Prover.

1. $\boldsymbol{u}_1 := (\boldsymbol{M}\,(1 \| \boldsymbol{x} \| \boldsymbol{w}))\, \| 1 \| \boldsymbol{x} \| \boldsymbol{w}$
2. $\boldsymbol{\delta} \xleftarrow{\$} \mathbb{F}, u(X) := f_{\boldsymbol{u}_1 \| \boldsymbol{\delta}}(X)$
3. $\mathsf{cm}_u := \mathsf{com}\left(u(X), \mathsf{srs}\right)$
4. $\mu := \mathsf{H}_1(\mathsf{cm}_{\boldsymbol{u}}, \mathsf{cm}_{\boldsymbol{w}}, \mathsf{cm}_{\boldsymbol{v}}, \mathsf{cm}_{\boldsymbol{y}}, \mathsf{cm}_u)$
5. $\boldsymbol{r} := \left(\frac{1}{\alpha - \gamma^i}\right)_{i=1}^{3H}$
6. $\boldsymbol{c} := \boldsymbol{r}^{\mathsf{T}} \boldsymbol{M}$
7. $\boldsymbol{s} := \boldsymbol{r} \| (-\boldsymbol{c})$
8. $\boldsymbol{\delta}_1 \xleftarrow{\$} \mathbb{F}, s(X) := f_{\boldsymbol{s} \| \boldsymbol{\delta}_1}(X)$
9. $\mathsf{cm}_s := \mathsf{com}\left(s(X), \mathsf{srs}\right)$
10. $\nu := \mathsf{H}_1(\mathsf{cm}_{\boldsymbol{u}}, \mathsf{cm}_{\boldsymbol{w}}, \mathsf{cm}_{\boldsymbol{v}}, \mathsf{cm}_{\boldsymbol{y}}, \mathsf{cm}_u, \mathsf{cm}_s)$
11. $\boldsymbol{h} := \left(\frac{1}{\beta - \gamma^i}\right)_{i=1}^{K} \| \left(\frac{1}{(\alpha - \boldsymbol{u}_{[i]})(\beta - \boldsymbol{w}_{[i]})}\right)_{i=1}^{3S}$
12. $\boldsymbol{\delta}_2 \xleftarrow{\$} \mathbb{F}, h(X) := f_{\boldsymbol{h} \| \boldsymbol{\delta}_2}(X)$
13. $\mathsf{cm}_h := \mathsf{com}\left(h(X), \mathsf{srs}\right)$
14. $\beta := \mathsf{H}_1(\mathsf{cm}_{\boldsymbol{u}}, \mathsf{cm}_{\boldsymbol{w}}, \mathsf{cm}_{\boldsymbol{v}}, \mathsf{cm}_{\boldsymbol{y}}, \mathsf{cm}_u, \mathsf{cm}_s, \mathsf{cm}_h)$
15. $\boldsymbol{r}_1 :=$ the sum of:
   - $\boldsymbol{u}_1{}^{\to -3H+3S} \circ \boldsymbol{s}^{\to -3H+3S}$
   - $\beta \cdot \left(-\boldsymbol{h}^{\to 3S} \circ \boldsymbol{s}^{\to -3H+3S} - \boldsymbol{h} \circ \boldsymbol{v}^{\to K}\right)$
16. $\boldsymbol{\delta}_3 \xleftarrow{\$} \mathbb{F}, \tilde{\boldsymbol{r}} := \left(\sum_{j=1}^{i} \boldsymbol{r}_{1[j]}\right)_{i=1}^{K+3S} \| \boldsymbol{\delta}_3$
17. $\mathsf{cm}_{\tilde{\boldsymbol{r}}} := \mathsf{com}\left(f_{\tilde{\boldsymbol{r}}}(X), \mathsf{srs}\right)$
18. $\alpha := \mathsf{H}_1(\mathsf{cm}_{\boldsymbol{u}}, \mathsf{cm}_{\boldsymbol{w}}, \mathsf{cm}_{\boldsymbol{v}}, \mathsf{cm}_{\boldsymbol{y}}, \mathsf{cm}_u, \mathsf{cm}_s, \mathsf{cm}_h, \mathsf{cm}_{\tilde{\boldsymbol{r}}})$
19. $\boldsymbol{t} :=$ the sum of:
   - $\left(\mu \cdot \boldsymbol{1}^{3H} - \boldsymbol{\gamma}^{3H}\right) \circ \boldsymbol{s}$
   - $-\boldsymbol{1}^{3H} \circ \boldsymbol{1}^{3H}$
   - $\left(\alpha\nu \cdot \boldsymbol{1}^{K} - \alpha \cdot \boldsymbol{\gamma}^{K}\right) \circ \boldsymbol{h}$
   - $-\alpha \cdot \boldsymbol{1}^{K} \circ \boldsymbol{1}^{K}$
   - $\alpha^2 \cdot \boldsymbol{h} \circ \left(-\mu \cdot \boldsymbol{w}^{\to K} - \nu \cdot \boldsymbol{u}^{\to K} + \boldsymbol{y}^{\to K} + \mu\nu \cdot \boldsymbol{1}^{K \to K}\right)$
   - $-\alpha^2 \cdot \boldsymbol{1}^{K} \circ \boldsymbol{1}^{K}$
   - $\alpha^3 \cdot \boldsymbol{u}_1{}^{\to -H+K+3S} \circ \boldsymbol{u}_1{}^{\to -2H+K+3S}$

- $-\alpha^3 \cdot \mathbf{1}^{H\to -H+K+3S} \circ \boldsymbol{u}_1{}^{\to -3H+K+3S}$
- $\alpha^5 \cdot \boldsymbol{u}_1{}^{\to -3H+3S} \circ \boldsymbol{s}^{\to -3H+3S}$
- $-\alpha^5\beta \cdot \boldsymbol{h}^{\to 3S} \circ \boldsymbol{s}^{\to -3H+3S}$
- $-\alpha^5\beta \cdot \boldsymbol{h} \circ \boldsymbol{v}^{\to K}$
- $\left(-\alpha^5 \cdot \tilde{\boldsymbol{r}} + \alpha^5 \cdot \tilde{\boldsymbol{r}}^{\to 1}\right) \circ \mathbf{1}^{K+3S}$

20. $\boldsymbol{\delta}_4 \xleftarrow{\$} \mathbb{F}, t(X) := f_{\boldsymbol{\delta}_4 \| \boldsymbol{t}_{[K+3S+1..]}}(X)$
21. $\mathsf{cm}_t := \mathsf{com}\left(t(X), \mathsf{srs}\right)$
22. $\omega := \mathsf{H}_1(\mathsf{cm}_{\boldsymbol{u}}, \mathsf{cm}_{\boldsymbol{w}}, \mathsf{cm}_{\boldsymbol{v}}, \mathsf{cm}_{\boldsymbol{y}}, \mathsf{cm}_u, \mathsf{cm}_s, \mathsf{cm}_h, \mathsf{cm}_{\tilde{\boldsymbol{r}}}, \mathsf{cm}_t)$
23. $h_1(X) :=$ the sum of:
    - $\frac{X\left(-\mu(X-\gamma\omega)\left(\left(\frac{\omega}{X}\right)^{3H}-1\right)+(X-\omega)\left(\left(\frac{\gamma\omega}{X}\right)^{3H}-1\right)\right)}{(X-\omega)(X-\gamma\omega)} \cdot s(X)$
    - $\frac{X\left(X^{3H}-1\right)\left(\left(\frac{\omega}{X}\right)^{3H}-1\right)}{(X-1)(X-\omega)}$
    - $\frac{X\alpha\left(-\nu(X-\gamma\omega)\left(\left(\frac{\omega}{X}\right)^{K}-1\right)+(X-\omega)\left(\left(\frac{\gamma\omega}{X}\right)^{K}-1\right)\right)}{(X-\omega)(X-\gamma\omega)} \cdot h(X)$
    - $\frac{X\alpha\left(X^{K}-1\right)\left(\left(\frac{\omega}{X}\right)^{K}-1\right)}{(X-1)(X-\omega)}$
    - $-X^K\alpha^2\mu \cdot h(\frac{\omega}{X}) \cdot f_{\boldsymbol{w}}(X)$
    - $\frac{X^K\alpha^2\mu\nu\left(X^K-1\right)}{X-1} \cdot h(\frac{\omega}{X})$
    - $-X^K\alpha^2\nu \cdot h(\frac{\omega}{X}) \cdot f_{\boldsymbol{u}}(X)$
    - $X^K\alpha^2 \cdot h(\frac{\omega}{X}) \cdot f_{\boldsymbol{y}}(X)$
    - $\frac{X\alpha^2\left(X^K-1\right)\left(\left(\frac{\omega}{X}\right)^{K}-1\right)}{(X-1)(X-\omega)}$
    - $X^{-2H+K+3S}\alpha^3\left(\frac{\omega}{X}\right)^{-H+K+3S} \cdot u(\frac{\omega}{X}) \cdot u(X)$
    - $\frac{X^{-3H+K+3S+1}\alpha^3\left(\frac{\omega}{X}\right)^{-H+K+3S}\left(\left(\frac{\omega}{X}\right)^{H}-1\right)}{X-\omega} \cdot u(X)$
    - $\frac{\alpha^4\left(\frac{\omega}{X}\right)^{3H}\left(X-\omega\left(\frac{\omega}{X}\right)^{\ell}\right)}{X-\omega} \cdot u(X)$
    - $\frac{X^{3H+2}\alpha^4\left(\frac{\omega}{X}\right)^{3H}\left(\left(\frac{\omega}{X}\right)^{\ell+1}-1\right)}{X-\omega} \cdot f_{\boldsymbol{x}}(X)$
    - $\frac{X^{3H}\alpha^4\left(\frac{\omega}{X}\right)^{3H}\left(-X+\omega\left(\frac{\omega}{X}\right)^{\ell}\right)}{X-\omega}$
    - $X^{-3H+3S}\alpha^5\left(\frac{\omega}{X}\right)^{-3H+3S} \cdot u(\frac{\omega}{X}) \cdot s(X)$
    - $-X^{-3H+3S}\alpha^5\beta\left(\frac{\omega}{X}\right)^{3S} \cdot h(\frac{\omega}{X}) \cdot s(X)$
    - $-X^K\alpha^5\beta \cdot h(\frac{\omega}{X}) \cdot f_{\boldsymbol{v}}(X)$
    - $-\frac{\alpha^5(X-\omega)\left(X^{K+3S}-1\right)}{X(X-1)} \cdot f_{\tilde{\boldsymbol{r}}}(\frac{\omega}{X})$
    - $X^{K+3S-1}\alpha^6 \cdot f_{\tilde{\boldsymbol{r}}}(\frac{\omega}{X})$
    - $\frac{X^{K+3S}\left(\frac{\omega}{X}\right)^{K+3S}\left(\left(\frac{\omega}{X}\right)^{3S+1}-1\right)}{X-\omega} \cdot t(X)$
24. $h_2(X) := h_1(X) \cdot X^D \bmod X^D$

25. $h_3(X) := \frac{h_1(X)}{X} - X^{-D-1} \cdot h_2(X)$
26. $\mathsf{cm}_{h_2} := \mathsf{com}\left(h_2(X), \mathsf{srs}\right)$
27. $\mathsf{cm}_{h_3} := \mathsf{com}\left(h_3(X), \mathsf{srs}\right)$
28. $z := \mathsf{H}_1(\mathsf{cm}_{\boldsymbol{u}}, \mathsf{cm}_{\boldsymbol{w}}, \mathsf{cm}_{\boldsymbol{v}}, \mathsf{cm}_{\boldsymbol{y}}, \mathsf{cm}_u, \mathsf{cm}_s, \mathsf{cm}_h, \mathsf{cm}_{\tilde{\boldsymbol{r}}}, \mathsf{cm}_t, \mathsf{cm}_{h_2}, \mathsf{cm}_{h_3})$
29. $y := h(\frac{\omega}{z}), y_1 := u(\frac{\omega}{z}), y_2 := f_{\tilde{\boldsymbol{r}}}(\frac{\omega}{z})$
30. $c :=$ the sum of:
    - $\frac{z\left(\mu(\gamma\omega - z)\left(\left(\frac{\omega}{z}\right)^{3H} - 1\right) - (\omega - z)\left(\left(\frac{\gamma\omega}{z}\right)^{3H} - 1\right)\right)}{(\omega - z)(\gamma\omega - z)}$
    - $\alpha^5 y_1 z^{-3H+3S} \left(\frac{\omega}{z}\right)^{-3H+3S}$
    - $-\alpha^5 \beta y z^{-3H+3S} \left(\frac{\omega}{z}\right)^{3S}$
31. $c_1 :=$ the sum of:
    - $-\frac{z\left(z^{3H} - 1\right)\left(\left(\frac{\omega}{z}\right)^{3H} - 1\right)}{(\omega - z)(z - 1)}$
    - $-\frac{\alpha z\left(z^{K} - 1\right)\left(\left(\frac{\omega}{z}\right)^{K} - 1\right)}{(\omega - z)(z - 1)}$
    - $\frac{\alpha^2 \mu\nu y z^K \left(z^K - 1\right)}{z - 1}$
    - $-\frac{\alpha^2 z\left(z^K - 1\right)\left(\left(\frac{\omega}{z}\right)^K - 1\right)}{(\omega - z)(z - 1)}$
    - $\frac{\alpha^4 z^{3H+1}\left(\frac{\omega}{z}\right)^{3H}\left(-\omega\left(\frac{\omega}{z}\right)^{\ell} + z\right)}{\omega - z} \cdot f_{\boldsymbol{x}}(z)$
    - $\frac{\alpha^4 z^{3H}\left(\frac{\omega}{z}\right)^{3H}\left(-\omega\left(\frac{\omega}{z}\right)^{\ell} + z\right)}{\omega - z}$
    - $\frac{\alpha^5 y_2 (\omega - z)\left(z^{K+3S} - 1\right)}{z(z - 1)}$
    - $\alpha^6 y_2 z^{K+3S-1}$
32. $c_2 := \frac{\alpha z\left(\nu(\gamma\omega - z)\left(\left(\frac{\omega}{z}\right)^K - 1\right) - (\omega - z)\left(\left(\frac{\gamma\omega}{z}\right)^K - 1\right)\right)}{(\omega - z)(\gamma\omega - z)}$
33. $c_3 := -\alpha^2 \mu y z^K$
34. $c_4 := -\alpha^2 \nu y z^K$
35. $c_5 := \alpha^2 y z^K$
36. $c_6 :=$ the sum of:
    - $\alpha^3 y_1 z^{-2H+K+3S} \left(\frac{\omega}{z}\right)^{-H+K+3S}$
    - $\frac{\alpha^3 z^{-3H+K+3S+1}\left(\frac{\omega}{z}\right)^{-H+K+3S}\left(1 - \left(\frac{\omega}{z}\right)^H\right)}{\omega - z}$
    - $\frac{\alpha^4\left(\frac{\omega}{z}\right)^{3H}\left(\omega\left(\frac{\omega}{z}\right)^{\ell} - z\right)}{\omega - z}$
37. $c_7 := -\alpha^5 \beta y z^K$
38. $c_8 := \frac{z^{K+3S}\left(\frac{\omega}{z}\right)^{K+3S}\left(1 - \left(\frac{\omega}{z}\right)^{3S+1}\right)}{\omega - z}$
39. $c_9 := -z^D$
40. $c_{10} := -z$
41. $g(X) := c \cdot s(X) + c_2 \cdot h(X) + c_3 \cdot f_{\boldsymbol{w}}(X) + c_4 \cdot f_{\boldsymbol{u}}(X) + c_5 \cdot f_{\boldsymbol{y}}(X) + c_6 \cdot u(X) + c_7 \cdot f_{\boldsymbol{v}}(X) + c_8 \cdot t(X) + c_9 \cdot h_2(X) + c_{10} \cdot h_3(X) + c_1$

42. $\mathsf{cm}_g := c \cdot \mathsf{cm}_s + c_2 \cdot \mathsf{cm}_h + c_3 \cdot \mathsf{cm}_{\boldsymbol{w}} + c_4 \cdot \mathsf{cm}_{\boldsymbol{u}} + c_5 \cdot \mathsf{cm}_{\boldsymbol{y}} + c_6 \cdot \mathsf{cm}_u + c_7 \cdot \mathsf{cm}_{\boldsymbol{v}} + c_8 \cdot \mathsf{cm}_t + c_9 \cdot \mathsf{cm}_{h_2} + c_{10} \cdot \mathsf{cm}_{h_3} + c_1 \cdot \mathsf{com}(1)$
43. $(W, W_1) := \mathsf{open}\left(\begin{array}{l}\{(\mathsf{cm}_h, y, h(X)), (\mathsf{cm}_u, y_1, u(X)), (\mathsf{cm}_{\tilde{\boldsymbol{r}}}, y_2, f_{\tilde{\boldsymbol{r}}}(X))\},\\ \{(\mathsf{cm}_g, 0, g(X))\},\\ \left\{\frac{\omega}{z}, z\right\}\end{array}\right)$
44. Output $\pi := (\mathsf{cm}_u, \mathsf{cm}_s, \mathsf{cm}_h, \mathsf{cm}_{\tilde{\boldsymbol{r}}}, \mathsf{cm}_t, \mathsf{cm}_{h_2}, \mathsf{cm}_{h_3}, y, y_1, y_2, W, W_1)$

Verifier.

1. $\mu := \mathsf{H}_1(\mathsf{cm}_{\boldsymbol{u}}, \mathsf{cm}_{\boldsymbol{w}}, \mathsf{cm}_{\boldsymbol{v}}, \mathsf{cm}_{\boldsymbol{y}}, \mathsf{cm}_u)$
2. $\nu := \mathsf{H}_1(\mathsf{cm}_{\boldsymbol{u}}, \mathsf{cm}_{\boldsymbol{w}}, \mathsf{cm}_{\boldsymbol{v}}, \mathsf{cm}_{\boldsymbol{y}}, \mathsf{cm}_u, \mathsf{cm}_s)$
3. $\beta := \mathsf{H}_1(\mathsf{cm}_{\boldsymbol{u}}, \mathsf{cm}_{\boldsymbol{w}}, \mathsf{cm}_{\boldsymbol{v}}, \mathsf{cm}_{\boldsymbol{y}}, \mathsf{cm}_u, \mathsf{cm}_s, \mathsf{cm}_h)$
4. $\alpha := \mathsf{H}_1(\mathsf{cm}_{\boldsymbol{u}}, \mathsf{cm}_{\boldsymbol{w}}, \mathsf{cm}_{\boldsymbol{v}}, \mathsf{cm}_{\boldsymbol{y}}, \mathsf{cm}_u, \mathsf{cm}_s, \mathsf{cm}_h, \mathsf{cm}_{\tilde{\boldsymbol{r}}})$
5. $\omega := \mathsf{H}_1(\mathsf{cm}_{\boldsymbol{u}}, \mathsf{cm}_{\boldsymbol{w}}, \mathsf{cm}_{\boldsymbol{v}}, \mathsf{cm}_{\boldsymbol{y}}, \mathsf{cm}_u, \mathsf{cm}_s, \mathsf{cm}_h, \mathsf{cm}_{\tilde{\boldsymbol{r}}}, \mathsf{cm}_t)$
6. $z := \mathsf{H}_1(\mathsf{cm}_{\boldsymbol{u}}, \mathsf{cm}_{\boldsymbol{w}}, \mathsf{cm}_{\boldsymbol{v}}, \mathsf{cm}_{\boldsymbol{y}}, \mathsf{cm}_u, \mathsf{cm}_s, \mathsf{cm}_h, \mathsf{cm}_{\tilde{\boldsymbol{r}}}, \mathsf{cm}_t, \mathsf{cm}_{h_2}, \mathsf{cm}_{h_3})$
7. $c :=$ the sum of:
   - $\frac{z\left(\mu(\gamma\omega - z)\left(\left(\frac{\omega}{z}\right)^{3H} - 1\right) - (\omega - z)\left(\left(\frac{\gamma\omega}{z}\right)^{3H} - 1\right)\right)}{(\omega - z)(\gamma\omega - z)}$
   - $\alpha^5 y_1 z^{-3H+3S} \left(\frac{\omega}{z}\right)^{-3H+3S}$
   - $-\alpha^5 \beta y z^{-3H+3S} \left(\frac{\omega}{z}\right)^{3S}$
8. $c_1 :=$ the sum of:
   - $-\frac{z\left(z^{3H} - 1\right)\left(\left(\frac{\omega}{z}\right)^{3H} - 1\right)}{(\omega - z)(z - 1)}$
   - $-\frac{\alpha z\left(z^{K} - 1\right)\left(\left(\frac{\omega}{z}\right)^{K} - 1\right)}{(\omega - z)(z - 1)}$
   - $\frac{\alpha^2 \mu\nu y z^K \left(z^K - 1\right)}{z - 1}$
   - $-\frac{\alpha^2 z\left(z^K - 1\right)\left(\left(\frac{\omega}{z}\right)^K - 1\right)}{(\omega - z)(z - 1)}$
   - $\frac{\alpha^4 z^{3H+1}\left(\frac{\omega}{z}\right)^{3H}\left(-\omega\left(\frac{\omega}{z}\right)^{\ell} + z\right)}{\omega - z} \cdot f_{\boldsymbol{x}}(z)$
   - $\frac{\alpha^4 z^{3H}\left(\frac{\omega}{z}\right)^{3H}\left(-\omega\left(\frac{\omega}{z}\right)^{\ell} + z\right)}{\omega - z}$
   - $\frac{\alpha^5 y_2 (\omega - z)\left(z^{K+3S} - 1\right)}{z(z - 1)}$
   - $\alpha^6 y_2 z^{K+3S-1}$
9. $c_2 := \frac{\alpha z\left(\nu(\gamma\omega - z)\left(\left(\frac{\omega}{z}\right)^K - 1\right) - (\omega - z)\left(\left(\frac{\gamma\omega}{z}\right)^K - 1\right)\right)}{(\omega - z)(\gamma\omega - z)}$
10. $c_3 := -\alpha^2 \mu y z^K$
11. $c_4 := -\alpha^2 \nu y z^K$
12. $c_5 := \alpha^2 y z^K$
13. $c_6 :=$ the sum of:
   - $\alpha^3 y_1 z^{-2H+K+3S} \left(\frac{\omega}{z}\right)^{-H+K+3S}$

$$- \frac{\alpha^3 z^{-3H+K+3S+1}\left(\frac{\omega}{z}\right)^{-H+K+3S}\left(1-\left(\frac{\omega}{z}\right)^{H}\right)}{\omega - z}$$

$$- \frac{\alpha^4\left(\frac{\omega}{z}\right)^{3H}\left(\omega\left(\frac{\omega}{z}\right)^{\ell} - z\right)}{\omega - z}$$

14. $c_7 := -\alpha^5 \beta y z^K$
15. $c_8 := \frac{z^{K+3S}\left(\frac{\omega}{z}\right)^{K+3S}\left(1-\left(\frac{\omega}{z}\right)^{3S+1}\right)}{\omega - z}$
16. $c_9 := -z^D$
17. $c_{10} := -z$
18. $\mathsf{cm}_g := c \cdot \mathsf{cm}_s + c_2 \cdot \mathsf{cm}_h + c_3 \cdot \mathsf{cm}_{\boldsymbol{w}} + c_4 \cdot \mathsf{cm}_{\boldsymbol{u}} + c_5 \cdot \mathsf{cm}_{\boldsymbol{y}} + c_6 \cdot \mathsf{cm}_u + c_7 \cdot \mathsf{cm}_{\boldsymbol{v}} + c_8 \cdot \mathsf{cm}_t + c_9 \cdot \mathsf{cm}_{h_2} + c_{10} \cdot \mathsf{cm}_{h_3} + c_1 \cdot \mathsf{com}(1)$
19. $\mathsf{vrfy}\left(\begin{array}{l}\{(\mathsf{cm}_h, y), (\mathsf{cm}_u, y_1), (\mathsf{cm}_{\tilde{\boldsymbol{r}}}, y_2)\}, \\ \{(\mathsf{cm}_g, 0)\}, \\ \left\{\frac{\omega}{z}, z\right\}\{W, W_1\}, [x]_2\end{array}\right) \stackrel{?}{=} 1$

D.3 The zkSNARK for HPR

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

Setup. Output $\mathsf{srs} := \mathsf{gen}(D)$ for $D \geq 6K + 6S + 2S' + 4$.

Indexer. The inputs are the sparse representations for the matrices, 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 $\mathsf{row}_i \in [H], \mathsf{col}_i \in [K]$ and $\mathsf{val}_i \in \mathbb{F}$.

1. $\boldsymbol{u} := (\gamma^{\mathsf{row}_i})_{i=1}^{3S+S'}$
2. $\boldsymbol{w} := \left(\gamma^{\mathsf{col}_i}\right)_{i=1}^{3S+S'}$
3. $\boldsymbol{v} := (\mathsf{val}_i)_{i=1}^{3S+S'}$
4. $\boldsymbol{y} := \boldsymbol{u} \circ \boldsymbol{w}$
5. $\mathsf{cm}_{\boldsymbol{u}} := \mathsf{com}\left(f_{\boldsymbol{u}}(X), \mathsf{srs}\right)$
6. $\mathsf{cm}_{\boldsymbol{w}} := \mathsf{com}\left(f_{\boldsymbol{w}}(X), \mathsf{srs}\right)$
7. $\mathsf{cm}_{\boldsymbol{v}} := \mathsf{com}\left(f_{\boldsymbol{v}}(X), \mathsf{srs}\right)$
8. $\mathsf{cm}_{\boldsymbol{y}} := \mathsf{com}\left(f_{\boldsymbol{y}}(X), \mathsf{srs}\right)$
9. Output
   - $\mathsf{pk} := (\mathsf{cm}_{\boldsymbol{u}}, \mathsf{cm}_{\boldsymbol{w}}, \mathsf{cm}_{\boldsymbol{v}}, \mathsf{cm}_{\boldsymbol{y}})$
   - $\mathsf{vk} := (\boldsymbol{u}, \boldsymbol{w}, \boldsymbol{v}, \boldsymbol{y}, \mathsf{cm}_{\boldsymbol{u}}, \mathsf{cm}_{\boldsymbol{w}}, \mathsf{cm}_{\boldsymbol{v}}, \mathsf{cm}_{\boldsymbol{y}})$

Prover.

1. $\boldsymbol{w}_1 := \boldsymbol{w} \| \boldsymbol{w}_1 \| \boldsymbol{w}_2$
2. $\boldsymbol{\delta} \xleftarrow{\$} \mathbb{F}, w(X) := f_{\boldsymbol{w}_1 \| \boldsymbol{\delta}}(X)$
3. $\mathsf{cm}_w := \mathsf{com}\left(w(X), \mathsf{srs}\right)$
4. $\mu := \mathsf{H}_1(\mathsf{cm}_{\boldsymbol{u}}, \mathsf{cm}_{\boldsymbol{w}}, \mathsf{cm}_{\boldsymbol{v}}, \mathsf{cm}_{\boldsymbol{y}}, \mathsf{cm}_w)$
5. $\boldsymbol{r} := \left(\frac{1}{\alpha - \gamma^i}\right)_{i=1}^{H}$
6. $\boldsymbol{c} := \boldsymbol{r}^{\mathsf{T}} \boldsymbol{M}$
7. $\boldsymbol{s} := \boldsymbol{r} \| \left(-\boldsymbol{c}\right)$
8. $\boldsymbol{\delta}_1 \xleftarrow{\$} \mathbb{F}, s(X) := f_{\boldsymbol{s} \| \boldsymbol{\delta}_1}(X)$
9. $\mathsf{cm}_s := \mathsf{com}\left(s(X), \mathsf{srs}\right)$
10. $\nu := \mathsf{H}_1(\mathsf{cm}_{\boldsymbol{u}}, \mathsf{cm}_{\boldsymbol{w}}, \mathsf{cm}_{\boldsymbol{v}}, \mathsf{cm}_{\boldsymbol{y}}, \mathsf{cm}_w, \mathsf{cm}_s)$
11. $\boldsymbol{h} := \left(\frac{1}{\beta - \gamma^i}\right)_{i=1}^{3K+1} \| \left(\frac{1}{(\alpha - \boldsymbol{u}_{[i]})(\beta - \boldsymbol{w}_{[i]})}\right)_{i=1}^{3S+S'}$
12. $\boldsymbol{\delta}_2 \xleftarrow{\$} \mathbb{F}, h(X) := f_{\boldsymbol{h} \| \boldsymbol{\delta}_2}(X)$
13. $\mathsf{cm}_h := \mathsf{com}\left(h(X), \mathsf{srs}\right)$
14. $\beta := \mathsf{H}_1(\mathsf{cm}_{\boldsymbol{u}}, \mathsf{cm}_{\boldsymbol{w}}, \mathsf{cm}_{\boldsymbol{v}}, \mathsf{cm}_{\boldsymbol{y}}, \mathsf{cm}_w, \mathsf{cm}_s, \mathsf{cm}_h)$
15. $\boldsymbol{r}_1 :=$ the sum of:
    - $\left(\boldsymbol{x}^{\to -H+3S+S'} + \boldsymbol{w}_1{}^{\to -H+3S+S'+\ell+1} + \boldsymbol{e}_{-H+3S+S'+\ell+1}\right) \circ \boldsymbol{s}^{\to -H+3S+S'}$
    - $\beta \cdot \left(-\boldsymbol{h}^{\to 3S+S'} \circ \boldsymbol{s}^{\to -H+3S+S'} - \boldsymbol{h} \circ \boldsymbol{v}^{\to 3K+1}\right)$
16. $\boldsymbol{\delta}_3 \xleftarrow{\$} \mathbb{F}, \tilde{\boldsymbol{r}} := \left(\sum_{j=1}^{i} \boldsymbol{r}_{1[j]}\right)_{i=1}^{3K+3S+S'+1} \| \boldsymbol{\delta}_3$
17. $\mathsf{cm}_{\tilde{\boldsymbol{r}}} := \mathsf{com}\left(f_{\tilde{\boldsymbol{r}}}(X), \mathsf{srs}\right)$
18. $\alpha := \mathsf{H}_1(\mathsf{cm}_{\boldsymbol{u}}, \mathsf{cm}_{\boldsymbol{w}}, \mathsf{cm}_{\boldsymbol{v}}, \mathsf{cm}_{\boldsymbol{y}}, \mathsf{cm}_w, \mathsf{cm}_s, \mathsf{cm}_h, \mathsf{cm}_{\tilde{\boldsymbol{r}}})$
19. $\boldsymbol{t} :=$ the sum of:
    - $\left(\mu \cdot \mathbf{1}^H - \boldsymbol{\gamma}^H\right) \circ \boldsymbol{s}$
    - $-\mathbf{1}^H \circ \mathbf{1}^H$
    - $\left(\alpha\nu \cdot \mathbf{1}^{3K+1} - \alpha \cdot \boldsymbol{\gamma}^{3K+1}\right) \circ \boldsymbol{h}$
    - $-\alpha \cdot \mathbf{1}^{3K+1} \circ \mathbf{1}^{3K+1}$
    - $\alpha^2 \cdot \boldsymbol{h} \circ \left(-\mu \cdot \boldsymbol{w}^{\to 3K+1} - \nu \cdot \boldsymbol{u}^{\to 3K+1} + \boldsymbol{y}^{\to 3K+1} + \mu\nu \cdot \mathbf{1}^{3K+1 \to 3K+1}\right)$
    - $-\alpha^2 \cdot \mathbf{1}^{3K+1} \circ \mathbf{1}^{3K+1}$
    - $\alpha^3 \cdot \boldsymbol{w}_1{}^{\to 2K+3S+S'+1} \circ \boldsymbol{w}_1{}^{\to K+3S+S'+1}$
    - $-\alpha^3 \cdot \mathbf{1}^{K \to 2K+3S+S'+1} \circ \boldsymbol{w}_1{}^{\to 3S+S'+1}$
    - $\left(\alpha^4 \cdot \boldsymbol{x}^{\to -H+3S+S'} + \alpha^4 \cdot \boldsymbol{w}_1{}^{\to -H+3S+S'+\ell+1} + \alpha^4 \cdot \boldsymbol{e}_{-H+3S+S'+\ell+1}\right) \circ \boldsymbol{s}^{\to -H+3S+S'}$
    - $-\alpha^4\beta \cdot \boldsymbol{h}^{\to 3S+S'} \circ \boldsymbol{s}^{\to -H+3S+S'}$
    - $-\alpha^4\beta \cdot \boldsymbol{h} \circ \boldsymbol{v}^{\to 3K+1}$
    - $\left(-\alpha^4 \cdot \tilde{\boldsymbol{r}} + \alpha^4 \cdot \tilde{\boldsymbol{r}}^{\to 1}\right) \circ \mathbf{1}^{3K+3S+S'+1}$

20. $\boldsymbol{\delta}_4 \xleftarrow{\$} \mathbb{F}, t(X) := f_{\boldsymbol{\delta}_4 \| \boldsymbol{t}_{[3K+3S+S'+2..]}}(X)$
21. $\mathsf{cm}_t := \mathsf{com}\,(t(X), \mathsf{srs})$
22. $\omega := \mathsf{H}_1(\mathsf{cm}_{\boldsymbol{u}}, \mathsf{cm}_{\boldsymbol{w}}, \mathsf{cm}_{\boldsymbol{v}}, \mathsf{cm}_{\boldsymbol{y}}, \mathsf{cm}_w, \mathsf{cm}_s, \mathsf{cm}_h, \mathsf{cm}_{\tilde{\boldsymbol{r}}}, \mathsf{cm}_t)$
23. $h_1(X) :=$ the sum of:
    - $\frac{X\left(-\mu(X-\gamma\omega)\left(\left(\frac{\omega}{X}\right)^H-1\right)+(X-\omega)\left(\left(\frac{\gamma\omega}{X}\right)^H-1\right)\right)}{(X-\omega)(X-\gamma\omega)} \cdot s(X)$
    - $\frac{X(X^H-1)\left(\left(\frac{\omega}{X}\right)^H-1\right)}{(X-1)(X-\omega)}$
    - $\frac{X\alpha\left(-\nu(X-\gamma\omega)\left(\left(\frac{\omega}{X}\right)^{3K+1}-1\right)+(X-\omega)\left(\left(\frac{\gamma\omega}{X}\right)^{3K+1}-1\right)\right)}{(X-\omega)(X-\gamma\omega)} \cdot h(X)$
    - $\frac{X\alpha\left(X^{3K+1}-1\right)\left(\left(\frac{\omega}{X}\right)^{3K+1}-1\right)}{(X-1)(X-\omega)}$
    - $-X^{3K+1}\alpha^2\mu \cdot h(\frac{\omega}{X}) \cdot f_{\boldsymbol{w}}(X)$
    - $\frac{X^{3K+1}\alpha^2\mu\nu\left(X^{3K+1}-1\right)}{X-1} \cdot h(\frac{\omega}{X})$
    - $-X^{3K+1}\alpha^2\nu \cdot h(\frac{\omega}{X}) \cdot f_{\boldsymbol{u}}(X)$
    - $X^{3K+1}\alpha^2 \cdot h(\frac{\omega}{X}) \cdot f_{\boldsymbol{y}}(X)$
    - $\frac{X\alpha^2\left(X^{3K+1}-1\right)\left(\left(\frac{\omega}{X}\right)^{3K+1}-1\right)}{(X-1)(X-\omega)}$
    - $X^{K+3S+S'+1}\alpha^3\left(\frac{\omega}{X}\right)^{2K+3S+S'+1} \cdot w(\frac{\omega}{X}) \cdot w(X)$
    - $\frac{X^{3S+S'+2}\alpha^3\left(\frac{\omega}{X}\right)^{2K+3S+S'+1}\left(\left(\frac{\omega}{X}\right)^K-1\right)}{X-\omega} \cdot w(X)$
    - $X^{-H+3S+S'}\alpha^4\left(\frac{\omega}{X}\right)^{-H+3S+S'} \cdot f_{\boldsymbol{x}}(\frac{\omega}{X}) \cdot s(X)$
    - $X^{-H+3S+S'}\alpha^4\left(\frac{\omega}{X}\right)^{-H+3S+S'+\ell+1} \cdot w(\frac{\omega}{X}) \cdot s(X)$
    - $X^{-H+3S+S'}\alpha^4\left(\frac{\omega}{X}\right)^{-H+3S+S'+\ell} \cdot s(X)$
    - $-X^{-H+3S+S'}\alpha^4\beta\left(\frac{\omega}{X}\right)^{3S+S'} \cdot h(\frac{\omega}{X}) \cdot s(X)$
    - $-X^{3K+1}\alpha^4\beta \cdot h(\frac{\omega}{X}) \cdot f_{\boldsymbol{v}}(X)$
    - $-\frac{\alpha^4(X-\omega)\left(X^{3K+3S+S'+1}-1\right)}{X(X-1)} \cdot f_{\tilde{\boldsymbol{r}}}(\frac{\omega}{X})$
    - $X^{3K+3S+S'}\alpha^5 \cdot f_{\tilde{\boldsymbol{r}}}(\frac{\omega}{X})$
    - $\frac{X^{3K+3S+S'+1}\left(\frac{\omega}{X}\right)^{3K+3S+S'+1}\left(\left(\frac{\omega}{X}\right)^{2K+3S+S'+2}-1\right)}{X-\omega} \cdot t(X)$
24. $h_2(X) := h_1(X) \cdot X^D \bmod X^D$
25. $h_3(X) := \frac{h_1(X)}{X} - X^{-D-1} \cdot h_2(X)$
26. $\mathsf{cm}_{h_2} := \mathsf{com}\,(h_2(X), \mathsf{srs})$
27. $\mathsf{cm}_{h_3} := \mathsf{com}\,(h_3(X), \mathsf{srs})$
28. $z := \mathsf{H}_1(\mathsf{cm}_{\boldsymbol{u}}, \mathsf{cm}_{\boldsymbol{w}}, \mathsf{cm}_{\boldsymbol{v}}, \mathsf{cm}_{\boldsymbol{y}}, \mathsf{cm}_w, \mathsf{cm}_s, \mathsf{cm}_h, \mathsf{cm}_{\tilde{\boldsymbol{r}}}, \mathsf{cm}_t, \mathsf{cm}_{h_2}, \mathsf{cm}_{h_3})$
29. $y := h(\frac{\omega}{z}), y_1 := w(\frac{\omega}{z}), y_3 := f_{\tilde{\boldsymbol{r}}}(\frac{\omega}{z})$
30. $y_2 := f_{\boldsymbol{x}}(\frac{\omega}{z})$
31. $c :=$ the sum of:

- $\frac{z\left(\mu(\gamma\omega-z)\left(\left(\frac{\omega}{z}\right)^H-1\right)-(\omega-z)\left(\left(\frac{\gamma\omega}{z}\right)^H-1\right)\right)}{(\omega-z)(\gamma\omega-z)}$
- $\alpha^4 z^{-H+3S+S'-1}\left(\frac{\omega}{z}\right)^{-H+3S+S'}\left(\omega y_1\left(\frac{\omega}{z}\right)^{\ell}+y_2 z+z\left(\frac{\omega}{z}\right)^{\ell}\right)$
- $-\alpha^4\beta y z^{-H+3S+S'}\left(\frac{\omega}{z}\right)^{3S+S'}$

32. $c_1 :=$ the sum of:
    - $-\frac{z\left(z^H-1\right)\left(\left(\frac{\omega}{z}\right)^H-1\right)}{(\omega-z)(z-1)}$
    - $-\frac{\alpha\left(z^{3K+1}-1\right)\left(\omega\left(\frac{\omega}{z}\right)^{3K}-z\right)}{(\omega-z)(z-1)}$
    - $\frac{\alpha^2\mu\nu y z^{3K+1}\left(z^{3K+1}-1\right)}{z-1}$
    - $-\frac{\alpha^2\left(z^{3K+1}-1\right)\left(\omega\left(\frac{\omega}{z}\right)^{3K}-z\right)}{(\omega-z)(z-1)}$
    - $\frac{\alpha^4 y_3(\omega-z)\left(z^{3K+3S+S'+1}-1\right)}{z(z-1)}$
    - $\alpha^5 y_3 z^{3K+3S+S'}$
33. $c_2 := \frac{\alpha z\left(\nu(\gamma\omega-z)\left(\left(\frac{\omega}{z}\right)^{3K+1}-1\right)-(\omega-z)\left(\left(\frac{\gamma\omega}{z}\right)^{3K+1}-1\right)\right)}{(\omega-z)(\gamma\omega-z)}$
34. $c_3 := -\alpha^2\mu y z^{3K+1}$
35. $c_4 := -\alpha^2\nu y z^{3K+1}$
36. $c_5 := \alpha^2 y z^{3K+1}$
37. $c_6 :=$ the sum of:
    - $\alpha^3 y_1 z^{K+3S+S'+1}\left(\frac{\omega}{z}\right)^{2K+3S+S'+1}$
    - $\frac{\alpha^3 z^{3S+S'+2}\left(\frac{\omega}{z}\right)^{2K+3S+S'+1}\left(1-\left(\frac{\omega}{z}\right)^K\right)}{\omega-z}$
38. $c_7 := -\alpha^4\beta y z^{3K+1}$
39. $c_8 := \frac{z^{3K+3S+S'+1}\left(\frac{\omega}{z}\right)^{3K+3S+S'+1}\left(1-\left(\frac{\omega}{z}\right)^{2K+3S+S'+2}\right)}{\omega-z}$
40. $c_9 := -z^D$
41. $c_{10} := -z$
42. $g(X) := c\cdot s(X)+c_2\cdot h(X)+c_3\cdot f_{\boldsymbol{w}}(X)+c_4\cdot f_{\boldsymbol{u}}(X)+c_5\cdot f_{\boldsymbol{y}}(X)+c_6\cdot w(X)+c_7\cdot f_{\boldsymbol{v}}(X)+c_8\cdot t(X)+c_9\cdot h_2(X)+c_{10}\cdot h_3(X)+c_1$
43. $\mathsf{cm}_g := c\cdot\mathsf{cm}_s+c_2\cdot\mathsf{cm}_h+c_3\cdot\mathsf{cm}_{\boldsymbol{w}}+c_4\cdot\mathsf{cm}_{\boldsymbol{u}}+c_5\cdot\mathsf{cm}_{\boldsymbol{y}}+c_6\cdot\mathsf{cm}_w+c_7\cdot\mathsf{cm}_{\boldsymbol{v}}+c_8\cdot\mathsf{cm}_t+c_9\cdot\mathsf{cm}_{h_2}+c_{10}\cdot\mathsf{cm}_{h_3}+c_1\cdot\mathsf{com}(1)$
44. $(W, W_1) := \mathsf{open}\left(\begin{array}{l}\{(\mathsf{cm}_h, y, h(X)), (\mathsf{cm}_w, y_1, w(X)), (\mathsf{cm}_{\tilde{\boldsymbol{r}}}, y_3, f_{\tilde{\boldsymbol{r}}}(X))\},\\ \{(\mathsf{cm}_g, 0, g(X))\},\\ \left\{\frac{\omega}{z}, z\right\}\end{array}\right)$
45. Output $\pi := (\mathsf{cm}_w, \mathsf{cm}_s, \mathsf{cm}_h, \mathsf{cm}_{\tilde{\boldsymbol{r}}}, \mathsf{cm}_t, \mathsf{cm}_{h_2}, \mathsf{cm}_{h_3}, y, y_1, y_3, W, W_1)$

Verifier.

1. $\mu := \mathsf{H}_1(\mathsf{cm}_{\boldsymbol{u}}, \mathsf{cm}_{\boldsymbol{w}}, \mathsf{cm}_{\boldsymbol{v}}, \mathsf{cm}_{\boldsymbol{y}}, \mathsf{cm}_w)$

2. $\nu := \mathsf{H}_1(\mathsf{cm}_{\boldsymbol{u}}, \mathsf{cm}_{\boldsymbol{w}}, \mathsf{cm}_{\boldsymbol{v}}, \mathsf{cm}_{\boldsymbol{y}}, \mathsf{cm}_w, \mathsf{cm}_s)$
3. $\beta := \mathsf{H}_1(\mathsf{cm}_{\boldsymbol{u}}, \mathsf{cm}_{\boldsymbol{w}}, \mathsf{cm}_{\boldsymbol{v}}, \mathsf{cm}_{\boldsymbol{y}}, \mathsf{cm}_w, \mathsf{cm}_s, \mathsf{cm}_h)$
4. $\alpha := \mathsf{H}_1(\mathsf{cm}_{\boldsymbol{u}}, \mathsf{cm}_{\boldsymbol{w}}, \mathsf{cm}_{\boldsymbol{v}}, \mathsf{cm}_{\boldsymbol{y}}, \mathsf{cm}_w, \mathsf{cm}_s, \mathsf{cm}_h, \mathsf{cm}_{\tilde{\boldsymbol{r}}})$
5. $\omega := \mathsf{H}_1(\mathsf{cm}_{\boldsymbol{u}}, \mathsf{cm}_{\boldsymbol{w}}, \mathsf{cm}_{\boldsymbol{v}}, \mathsf{cm}_{\boldsymbol{y}}, \mathsf{cm}_w, \mathsf{cm}_s, \mathsf{cm}_h, \mathsf{cm}_{\tilde{\boldsymbol{r}}}, \mathsf{cm}_t)$
6. $z := \mathsf{H}_1(\mathsf{cm}_{\boldsymbol{u}}, \mathsf{cm}_{\boldsymbol{w}}, \mathsf{cm}_{\boldsymbol{v}}, \mathsf{cm}_{\boldsymbol{y}}, \mathsf{cm}_w, \mathsf{cm}_s, \mathsf{cm}_h, \mathsf{cm}_{\tilde{\boldsymbol{r}}}, \mathsf{cm}_t, \mathsf{cm}_{h_2}, \mathsf{cm}_{h_3})$
7. $y_2 := f_{\boldsymbol{x}}(\frac{\omega}{z})$
8. $c :=$ the sum of:
   - $\frac{z\left(\mu(\gamma\omega - z)\left(\left(\frac{\omega}{z}\right)^H - 1\right) - (\omega - z)\left(\left(\frac{\gamma\omega}{z}\right)^H - 1\right)\right)}{(\omega - z)(\gamma\omega - z)}$
   - $\alpha^4 z^{-H+3S+S'-1}\left(\frac{\omega}{z}\right)^{-H+3S+S'}\left(\omega y_1\left(\frac{\omega}{z}\right)^{\ell} + y_2 z + z\left(\frac{\omega}{z}\right)^{\ell}\right)$
   - $-\alpha^4\beta y z^{-H+3S+S'}\left(\frac{\omega}{z}\right)^{3S+S'}$
9. $c_1 :=$ the sum of:
   - $-\frac{z\left(z^H - 1\right)\left(\left(\frac{\omega}{z}\right)^H - 1\right)}{(\omega - z)(z - 1)}$
   - $-\frac{\alpha\left(z^{3K+1} - 1\right)\left(\omega\left(\frac{\omega}{z}\right)^{3K} - z\right)}{(\omega - z)(z - 1)}$
   - $\frac{\alpha^2\mu\nu y z^{3K+1}\left(z^{3K+1} - 1\right)}{z - 1}$
   - $-\frac{\alpha^2\left(z^{3K+1} - 1\right)\left(\omega\left(\frac{\omega}{z}\right)^{3K} - z\right)}{(\omega - z)(z - 1)}$
   - $\frac{\alpha^4 y_3(\omega - z)\left(z^{3K+3S+S'+1} - 1\right)}{z(z - 1)}$
   - $\alpha^5 y_3 z^{3K+3S+S'}$
10. $c_2 := \frac{\alpha z\left(\nu(\gamma\omega - z)\left(\left(\frac{\omega}{z}\right)^{3K+1} - 1\right) - (\omega - z)\left(\left(\frac{\gamma\omega}{z}\right)^{3K+1} - 1\right)\right)}{(\omega - z)(\gamma\omega - z)}$
11. $c_3 := -\alpha^2\mu y z^{3K+1}$
12. $c_4 := -\alpha^2\nu y z^{3K+1}$
13. $c_5 := \alpha^2 y z^{3K+1}$
14. $c_6 :=$ the sum of:
   - $\alpha^3 y_1 z^{K+3S+S'+1}\left(\frac{\omega}{z}\right)^{2K+3S+S'+1}$
   - $\frac{\alpha^3 z^{3S+S'+2}\left(\frac{\omega}{z}\right)^{2K+3S+S'+1}\left(1 - \left(\frac{\omega}{z}\right)^K\right)}{\omega - z}$
15. $c_7 := -\alpha^4\beta y z^{3K+1}$
16. $c_8 := \frac{z^{3K+3S+S'+1}\left(\frac{\omega}{z}\right)^{3K+3S+S'+1}\left(1 - \left(\frac{\omega}{z}\right)^{2K+3S+S'+2}\right)}{\omega - z}$
17. $c_9 := -z^D$
18. $c_{10} := -z$
19. $\mathsf{cm}_g := c \cdot \mathsf{cm}_s + c_2 \cdot \mathsf{cm}_h + c_3 \cdot \mathsf{cm}_{\boldsymbol{w}} + c_4 \cdot \mathsf{cm}_{\boldsymbol{u}} + c_5 \cdot \mathsf{cm}_{\boldsymbol{y}} + c_6 \cdot \mathsf{cm}_w + c_7 \cdot \mathsf{cm}_{\boldsymbol{v}} + c_8 \cdot \mathsf{cm}_t + c_9 \cdot \mathsf{cm}_{h_2} + c_{10} \cdot \mathsf{cm}_{h_3} + c_1 \cdot \mathsf{com}(1)$
20. $\mathsf{vrfy}\left(\begin{array}{l}\{(\mathsf{cm}_h, y), (\mathsf{cm}_w, y_1), (\mathsf{cm}_{\tilde{\boldsymbol{r}}}, y_3)\}, \\ \{(\mathsf{cm}_g, 0)\}, \\ \left\{\frac{\omega}{z}, z\right\}\{W, W_1\}, [x]_2\end{array}\right) \stackrel{?}{=} 1$

D.4 The zkSNARK for PLONK

The zkSNARK $\mathsf{VOPLONK}$ generates a proof for the satisfiability of a fan-in-2 circuit with C_c constant gates, C_a addition gates, C_m multiplication gates. Let $C = C_c + C_a + C_m$.

Setup. Output $\mathsf{srs} := \mathsf{gen}(D)$ for $D \geq 6C - C_m$.

Indexer. On input the values of the constant gates $\boldsymbol{d} \in \mathbb{F}^{C_c}$ and the partition Π over $[3C]$, the indexer computes the permutation σ over $[3C]$ such that the induced partition of σ is Π.

1. $\boldsymbol{\sigma} := \left(\gamma^{\sigma(i)-1}\right)_{i=1}^{3C}$
2. $\mathsf{cm}_{\boldsymbol{\sigma}} := \mathsf{com}\left(f_{\boldsymbol{\sigma}}(X), \mathsf{srs}\right)$
3. $\mathsf{cm}_{\boldsymbol{d}} := \mathsf{com}\left(f_{\boldsymbol{d}}(X), \mathsf{srs}\right)$
4. Output
   - $\mathsf{pk} := (\mathsf{cm}_{\boldsymbol{\sigma}}, \mathsf{cm}_{\boldsymbol{d}})$
   - $\mathsf{vk} := (\boldsymbol{\sigma}, \boldsymbol{d}, \mathsf{cm}_{\boldsymbol{\sigma}}, \mathsf{cm}_{\boldsymbol{d}})$

Prover.

1. $\boldsymbol{w} := \boldsymbol{a}_{[C-C_a-C_m+1..C]} \| \boldsymbol{b} \| \boldsymbol{c}$
2. $\boldsymbol{\delta} \xleftarrow{\$} \mathbb{F}, w(X) := f_{\boldsymbol{w}\|\boldsymbol{\delta}}(X)$
3. $\mathsf{cm}_w := \mathsf{com}\left(w(X), \mathsf{srs}\right)$
4. $\zeta := \mathsf{H}_1(\mathsf{cm}_{\boldsymbol{\sigma}}, \mathsf{cm}_{\boldsymbol{d}}, \mathsf{cm}_w)$
5. $\zeta_1 := \mathsf{H}_2(\mathsf{cm}_{\boldsymbol{\sigma}}, \mathsf{cm}_{\boldsymbol{d}}, \mathsf{cm}_w)$
6. $\boldsymbol{u} := \boldsymbol{w}^{\to C-C_a-C_m} + \zeta \cdot \boldsymbol{\gamma}^{3C} + \zeta_1 \cdot \mathbf{1}^{3C}$
7. $\boldsymbol{v} := \boldsymbol{w}^{\to C-C_a-C_m} + \zeta \cdot \boldsymbol{\sigma} + \zeta_1 \cdot \mathbf{1}^{3C}$
8. $\boldsymbol{r} := \left(\prod_{j=1}^{i} \left(\boldsymbol{u}_{[j]} / \boldsymbol{v}_{[j]}\right)\right)_{i=1}^{3C}$
9. $\boldsymbol{\delta}_1 \xleftarrow{\$} \mathbb{F}, r(X) := f_{\boldsymbol{r}\|\boldsymbol{\delta}_1}(X)$
10. $\mathsf{cm}_r := \mathsf{com}\left(r(X), \mathsf{srs}\right)$
11. $\alpha := \mathsf{H}_1(\mathsf{cm}_{\boldsymbol{\sigma}}, \mathsf{cm}_{\boldsymbol{d}}, \mathsf{cm}_w, \mathsf{cm}_r)$
12. $\boldsymbol{t}_1 :=$ the sum of:
    - $\boldsymbol{w}^{\to 3C-C_m} \circ \boldsymbol{w}^{\to 2C-C_m}$
    - $-\mathbf{1}^{C_m \to 3C-C_m} \circ \boldsymbol{w}^{\to C-C_m}$
    - $\alpha^2 \cdot \boldsymbol{t} \circ \left(\boldsymbol{w}^{\to C-C_a-C_m} - \boldsymbol{x}\right)$
    - $\left(\alpha^4 \cdot \boldsymbol{r}^{\to 1} + \alpha^4 \cdot \boldsymbol{e}_1\right) \circ \left(\boldsymbol{w}^{\to C-C_a-C_m} + \zeta \cdot \boldsymbol{\gamma}^{3C} + \zeta_1 \cdot \mathbf{1}^{3C}\right)$
    - $-\alpha^4 \cdot \boldsymbol{r} \circ \left(\boldsymbol{w}^{\to C-C_a-C_m} + \zeta \cdot \boldsymbol{\sigma} + \zeta_1 \cdot \mathbf{1}^{3C}\right)$
13. $\boldsymbol{\delta}_2 \xleftarrow{\$} \mathbb{F}, t(X) := f_{\boldsymbol{\delta}_2 \| \boldsymbol{t}_{1\,[3C+1..]}}(X)$

14. $\mathsf{cm}_t := \mathsf{com}\,(t(X), \mathsf{srs})$
15. $\omega := \mathsf{H}_1(\mathsf{cm}_{\boldsymbol{\sigma}}, \mathsf{cm}_{\boldsymbol{d}}, \mathsf{cm}_w, \mathsf{cm}_r, \mathsf{cm}_t)$
16. $h(X) :=$ the sum of:
    - $X^{2C-C_m}\left(\frac{\omega}{X}\right)^{3C-C_m} \cdot w(\frac{\omega}{X}) \cdot w(X)$
    - $\frac{X^{C-C_m+1}\left(\frac{\omega}{X}\right)^{3C-C_m}\left(\left(\frac{\omega}{X}\right)^{C_m}-1\right)}{X-\omega} \cdot w(X)$
    - $-\frac{X\alpha\left(\frac{\omega}{X}\right)^{2C+C_m}\left(\left(\frac{\omega}{X}\right)^{C_a}-1\right)\left(X^{2C}+X^C-1\right)}{X-\omega} \cdot w(X)$
    - $X^{C-C_a-C_m}\alpha^2 \cdot f_{\boldsymbol{t}}(\frac{\omega}{X}) \cdot w(X)$
    - $-\alpha^2 \cdot f_{\boldsymbol{t}}(\frac{\omega}{X}) \cdot f_{\boldsymbol{x}}(X)$
    - $\frac{X\alpha^3\left(\frac{\omega}{X}\right)^{C+C_a+C_m}\left(1-\left(\frac{\omega}{X}\right)^{C-C_a-C_m}\right)}{X-\omega} \cdot w(X)$
    - $\frac{X^{C+C_a+C_m+1}\alpha^3\left(\frac{\omega}{X}\right)^{C+C_a+C_m}\left(\left(\frac{\omega}{X}\right)^{C-C_a-C_m}-1\right)}{X-\omega} \cdot f_{\boldsymbol{d}}(X)$
    - $X^{C-C_a-C_m-1}\alpha^4\omega \cdot r(\frac{\omega}{X}) \cdot w(X)$
    - $\frac{\alpha^4\omega\left(\zeta(X-1)\left((X\gamma)^{3C}-1\right)+\zeta_1\left(X^{3C}-1\right)(X\gamma-1)\right)}{X(X-1)(X\gamma-1)} \cdot r(\frac{\omega}{X})$
    - $X^{C-C_a-C_m}\alpha^4 \cdot w(X)$
    - $\frac{\alpha^4\left(\zeta(X-1)\left((X\gamma)^{3C}-1\right)+\zeta_1\left(X^{3C}-1\right)(X\gamma-1)\right)}{(X-1)(X\gamma-1)}$
    - $-X^{C-C_a-C_m}\alpha^4 \cdot r(\frac{\omega}{X}) \cdot w(X)$
    - $-\alpha^4\zeta \cdot r(\frac{\omega}{X}) \cdot f_{\boldsymbol{\sigma}}(X)$
    - $\frac{\alpha^4\zeta_1\left(1-X^{3C}\right)}{X-1} \cdot r(\frac{\omega}{X})$
    - $X^{3C-1}\alpha^5 \cdot r(\frac{\omega}{X})$
    - $-\frac{X^{3C}\alpha^5\left(\frac{\omega}{X}\right)^{3C}}{\omega}$
    - $\frac{X^{3C}\left(\frac{\omega}{X}\right)^{3C}\left(\left(\frac{\omega}{X}\right)^{3C-C_m+1}-1\right)}{X-\omega} \cdot t(X)$
17. $h_1(X) := h(X) \cdot X^D \bmod X^D$
18. $h_2(X) := \frac{h(X)}{X} - X^{-D-1} \cdot h_1(X)$
19. $\mathsf{cm}_{h_1} := \mathsf{com}\,(h_1(X), \mathsf{srs})$
20. $\mathsf{cm}_{h_2} := \mathsf{com}\,(h_2(X), \mathsf{srs})$
21. $z := \mathsf{H}_1(\mathsf{cm}_{\boldsymbol{\sigma}}, \mathsf{cm}_{\boldsymbol{d}}, \mathsf{cm}_w, \mathsf{cm}_r, \mathsf{cm}_t, \mathsf{cm}_{h_1}, \mathsf{cm}_{h_2})$
22. $y := w(\frac{\omega}{z}), y_2 := r(\frac{\omega}{z})$
23. $y_1 := f_{\boldsymbol{t}}(\frac{\omega}{z})$
24. $c :=$ the sum of:
    - $yz^{2C-C_m}\left(\frac{\omega}{z}\right)^{3C-C_m}$
    - $\frac{z^{C-C_m+1}\left(\frac{\omega}{z}\right)^{3C-C_m}\left(1-\left(\frac{\omega}{z}\right)^{C_m}\right)}{\omega-z}$
    - $\frac{\alpha\left(\frac{\omega}{z}\right)^{2C+C_m}\left(\left(\frac{\omega}{z}\right)^{C_a}-1\right)\left(-z+z^{C+1}+z^{2C+1}\right)}{\omega-z}$
    - $\alpha^2 y_1 z^{C-C_a-C_m}$

- $\frac{\alpha^3 z\left(\frac{\omega}{z}\right)^{C+C_a+C_m}\left(\left(\frac{\omega}{z}\right)^{C-C_a-C_m}-1\right)}{\omega-z}$
- $\alpha^4 z^{C-C_a-C_m-1}(\omega y_2+z)$
- $-\alpha^4 y_2 z^{C-C_a-C_m}$

25. $c_1 :=$ the sum of:
    - $-\alpha^2 y_1 \cdot f_{\boldsymbol{x}}(z)$
    - $\frac{\alpha^4(\omega y_2+z)\left(\zeta(z-1)\left((\gamma z)^{3C}-1\right)+\zeta_1\left(z^{3C}-1\right)(\gamma z-1)\right)}{z(z-1)(\gamma z-1)}$
    - $\frac{\alpha^4 y_2\zeta_1\left(1-z^{3C}\right)}{z-1}$
    - $\alpha^5 z^{3C-1}\left(y_2-\left(\frac{\omega}{z}\right)^{3C-1}\right)$
26. $c_2 := \frac{\alpha^3 z^{C+C_a+C_m+1}\left(\frac{\omega}{z}\right)^{C+C_a+C_m}\left(1-\left(\frac{\omega}{z}\right)^{C-C_a-C_m}\right)}{\omega-z}$
27. $c_3 := -\alpha^4 y_2\zeta$
28. $c_4 := \frac{z^{3C}\left(\frac{\omega}{z}\right)^{3C}\left(1-\left(\frac{\omega}{z}\right)^{3C-C_m+1}\right)}{\omega-z}$
29. $c_5 := -z^D$
30. $c_6 := -z$
31. $g(X) := c\cdot w(X)+c_2\cdot f_{\boldsymbol{d}}(X)+c_3\cdot f_{\boldsymbol{\sigma}}(X)+c_4\cdot t(X)+c_5\cdot h_1(X)+c_6\cdot h_2(X)+c_1$
32. $\mathsf{cm}_g := c\cdot\mathsf{cm}_w+c_2\cdot\mathsf{cm}_{\boldsymbol{d}}+c_3\cdot\mathsf{cm}_{\boldsymbol{\sigma}}+c_4\cdot\mathsf{cm}_t+c_5\cdot\mathsf{cm}_{h_1}+c_6\cdot\mathsf{cm}_{h_2}+c_1\cdot\mathsf{com}(1)$
33. $(W, W_1) := \mathsf{open}\left(\begin{array}{l}\{(\mathsf{cm}_w, y, w(X)), (\mathsf{cm}_r, y_2, r(X))\},\\ \{(\mathsf{cm}_g, 0, g(X))\},\\ \left\{\frac{\omega}{z}, z\right\}\end{array}\right)$
34. Output $\pi := (\mathsf{cm}_w, \mathsf{cm}_r, \mathsf{cm}_t, \mathsf{cm}_{h_1}, \mathsf{cm}_{h_2}, y, y_2, W, W_1)$

Verifier.

1. $\zeta := \mathsf{H}_1(\mathsf{cm}_{\boldsymbol{\sigma}}, \mathsf{cm}_{\boldsymbol{d}}, \mathsf{cm}_w)$
2. $\zeta_1 := \mathsf{H}_2(\mathsf{cm}_{\boldsymbol{\sigma}}, \mathsf{cm}_{\boldsymbol{d}}, \mathsf{cm}_w)$
3. $\alpha := \mathsf{H}_1(\mathsf{cm}_{\boldsymbol{\sigma}}, \mathsf{cm}_{\boldsymbol{d}}, \mathsf{cm}_w, \mathsf{cm}_r)$
4. $\omega := \mathsf{H}_1(\mathsf{cm}_{\boldsymbol{\sigma}}, \mathsf{cm}_{\boldsymbol{d}}, \mathsf{cm}_w, \mathsf{cm}_r, \mathsf{cm}_t)$
5. $z := \mathsf{H}_1(\mathsf{cm}_{\boldsymbol{\sigma}}, \mathsf{cm}_{\boldsymbol{d}}, \mathsf{cm}_w, \mathsf{cm}_r, \mathsf{cm}_t, \mathsf{cm}_{h_1}, \mathsf{cm}_{h_2})$
6. $y_1 := f_{\boldsymbol{t}}(\frac{\omega}{z})$
7. $c :=$ the sum of:
    - $yz^{2C-C_m}\left(\frac{\omega}{z}\right)^{3C-C_m}$
    - $\frac{z^{C-C_m+1}\left(\frac{\omega}{z}\right)^{3C-C_m}\left(1-\left(\frac{\omega}{z}\right)^{C_m}\right)}{\omega-z}$
    - $\frac{\alpha\left(\frac{\omega}{z}\right)^{2C+C_m}\left(\left(\frac{\omega}{z}\right)^{C_a}-1\right)\left(-z+z^{C+1}+z^{2C+1}\right)}{\omega-z}$
    - $\alpha^2 y_1 z^{C-C_a-C_m}$
    - $\frac{\alpha^3 z\left(\frac{\omega}{z}\right)^{C+C_a+C_m}\left(\left(\frac{\omega}{z}\right)^{C-C_a-C_m}-1\right)}{\omega-z}$

- $\alpha^4 z^{C-C_a-C_m-1}(\omega y_2 + z)$
- $-\alpha^4 y_2 z^{C-C_a-C_m}$

8. $c_1 :=$ the sum of:
   - $-\alpha^2 y_1 \cdot f_{\boldsymbol{x}}(z)$
   - $\frac{\alpha^4(\omega y_2+z)\left(\zeta(z-1)\left((\gamma z)^{3C}-1\right)+\zeta_1\left(z^{3C}-1\right)(\gamma z-1)\right)}{z(z-1)(\gamma z-1)}$
   - $\frac{\alpha^4 y_2 \zeta_1\left(1-z^{3C}\right)}{z-1}$
   - $\alpha^5 z^{3C-1}\left(y_2 - \left(\frac{\omega}{z}\right)^{3C-1}\right)$
9. $c_2 := \frac{\alpha^3 z^{C+C_a+C_m+1}\left(\frac{\omega}{z}\right)^{C+C_a+C_m}\left(1-\left(\frac{\omega}{z}\right)^{C-C_a-C_m}\right)}{\omega-z}$
10. $c_3 := -\alpha^4 y_2 \zeta$
11. $c_4 := \frac{z^{3C}\left(\frac{\omega}{z}\right)^{3C}\left(1-\left(\frac{\omega}{z}\right)^{3C-C_m+1}\right)}{\omega-z}$
12. $c_5 := -z^D$
13. $c_6 := -z$
14. $\mathsf{cm}_g := c\cdot\mathsf{cm}_w + c_2\cdot\mathsf{cm}_{\boldsymbol{d}} + c_3\cdot\mathsf{cm}_{\boldsymbol{\sigma}} + c_4\cdot\mathsf{cm}_t + c_5\cdot\mathsf{cm}_{h_1} + c_6\cdot\mathsf{cm}_{h_2} + c_1\cdot\mathsf{com}(1)$
15. $\mathsf{vrfy}\left(\begin{array}{l}\{(\mathsf{cm}_w, y), (\mathsf{cm}_r, y_2)\},\\ \{(\mathsf{cm}_g, 0)\},\\ \left\{\frac{\omega}{z}, z\right\}\{W, W_1\}, [x]_2\end{array}\right) \stackrel{?}{=} 1$

E Alternative Protocols

Algorithm 8 TripleProductEq Protocol

Input: $\boldsymbol{u}^{(1)}, \boldsymbol{u}^{(2)}, \boldsymbol{u}^{(3)}, \boldsymbol{v}^{(3)}, \boldsymbol{v}^{(3)}, \boldsymbol{v}^{(3)}, 1 \le \ell \le n$ where $\boldsymbol{u}_{[i]}^{(j)}, \boldsymbol{v}_{[i]}^{(j)} \neq 0$

Check: $\prod_{i\in[\ell], j\in[3]} \boldsymbol{u}_{[i]}^{(j)} = \prod_{i\in[\ell], j\in[3]} \boldsymbol{v}_{[i]}^{(j)}$

1: P computes $\boldsymbol{u} = \boldsymbol{u}^{(1)} \circ \boldsymbol{u}^{(2)} \circ \boldsymbol{u}^{(3)}$;
2: P computes $\boldsymbol{v} = \boldsymbol{v}^{(1)} \circ \boldsymbol{v}^{(2)} \circ \boldsymbol{v}^{(3)}$;
3: P computes $\boldsymbol{r} = \left(\prod_{j=1}^{i}(\boldsymbol{u}_{[j]}/\boldsymbol{v}_{[j]})\right)_{i=1}^{\ell} \| \mathbf{0}^{n-\ell}$;
4: P computes $\boldsymbol{s} = \left(\boldsymbol{r}_{[i]}/\boldsymbol{v}_{[i]}^{(1)} \cdot \boldsymbol{u}_{[i]}^{(1)}\right)_{i=1}^{\ell} \| \mathbf{0}^{n-\ell}$;
5: P computes $\boldsymbol{t} = \left(\boldsymbol{r}_{[i-1]}/\boldsymbol{u}_{[i]}^{(2)} \cdot \boldsymbol{v}_{[i]}^{(2)}\right)_{i=1}^{\ell} \| \mathbf{0}^{n-\ell}$ where $\boldsymbol{r}_{[0]}$ is defined as 1;
6: P submits $\boldsymbol{r}, \boldsymbol{s}, \boldsymbol{t}$ to $\mathcal{O}$;
7: V queries $\mathcal{O}$ to check that $\boldsymbol{s} \circ \boldsymbol{v}^{(1)} = \boldsymbol{r} \circ \boldsymbol{u}^{(1)}$;
8: V queries $\mathcal{O}$ to check that $\boldsymbol{t} \circ \boldsymbol{u}^{(2)} = (\boldsymbol{e}_1 + \boldsymbol{r}^{\to 1}) \circ \boldsymbol{v}^{(2)}$;
9: V queries $\mathcal{O}$ to check that $\boldsymbol{s} \circ \boldsymbol{u}^{(3)} = \boldsymbol{t} \circ \boldsymbol{v}^{(3)}$;
10: V queries $\mathcal{O}$ to check that $(\boldsymbol{r} - \boldsymbol{e}_\ell) \circ \boldsymbol{e}_\ell = \mathbf{0}$.

Algorithm 9 TriplePermute Protocol

Input: $\boldsymbol{u}^{(1)}, \boldsymbol{u}^{(2)}, \boldsymbol{u}^{(3)}, \boldsymbol{v}^{(1)}, \boldsymbol{v}^{(2)}, \boldsymbol{v}^{(3)} \in \mathbb{F}^n, \ell \leq n$
Check: $(\boldsymbol{u}^{(1)}_{[1..\ell]} \| \boldsymbol{u}^{(2)}_{[1..\ell]} \| \boldsymbol{u}^{(3)}_{[1..\ell]}) \sim (\boldsymbol{v}^{(1)}_{[1..\ell]} \| \boldsymbol{v}^{(2)}_{[1..\ell]} \| \boldsymbol{v}^{(3)}_{[1..\ell]})$.

1: V samples $\alpha \xleftarrow{\$} \mathbb{F}^*$ and sends α to P;
2: P and V run the protocol TripleProductEq with inputs $\boldsymbol{u}^{(1)} + \alpha \cdot \mathbf{1}^\ell, \boldsymbol{u}^{(2)} + \alpha \cdot \mathbf{1}^\ell, \boldsymbol{u}^{(3)} + \alpha \cdot \mathbf{1}^\ell, \boldsymbol{v}^{(1)} + \alpha \cdot \mathbf{1}^\ell, \boldsymbol{v}^{(2)} + \alpha \cdot \mathbf{1}^\ell, \boldsymbol{v}^{(3)} + \alpha \cdot \mathbf{1}^\ell$ and ℓ.

Algorithm 10 TripleCopyCheck Protocol

Index: A partition Π over $[3\ell]$ **Input:** $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c} \in \mathbb{F}^n$
Check: $\boldsymbol{a}_{[1..\ell]} \| \boldsymbol{b}_{[1..\ell]} \| \boldsymbol{c}_{[1..\ell]}$ satisfies the copy constraint of Π.
Preprocessing:

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

Online:

1: V samples $\beta \xleftarrow{\$} \mathbb{F}^*$ and sends β to P;
2: P and V run the protocol TriplePermute with inputs $\boldsymbol{a} + \beta\boldsymbol{\gamma}^\ell$, $\boldsymbol{b} + \beta\gamma^\ell \cdot \boldsymbol{\gamma}^\ell$, $\boldsymbol{c} + \beta\gamma^{2\ell} \cdot \boldsymbol{\gamma}^\ell$, $\boldsymbol{a} + \beta\boldsymbol{\sigma}^{(1)}$, $\boldsymbol{b} + \beta\boldsymbol{\sigma}^{(2)}$, $\boldsymbol{c} + \beta\boldsymbol{\sigma}^{(3)}$ and ℓ.

Algorithm 11 VOProof/POV Protocol

Index: $\boldsymbol{d} \in \mathbb{F}^{C_c}$, $C_a, C_m, \mathcal{I}_x \subset [3C]$, partition Π over $[3C]$ where $C = C_c + C_a + C_m$
Instance: $\boldsymbol{x} \in \mathbb{F}^{3C}$ which is sparse **Witness:** $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c} \in \mathbb{F}^C$
Check: $((C_c, C_a, C_m, \boldsymbol{d}, \Pi, \mathcal{I}_x), \boldsymbol{x}, (\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c})) \in \mathcal{R}_{\mathsf{POV}}$
Preprocessing:

1: I runs TripleCopyCheck.I with index Π;
2: I submits $\boldsymbol{d}$ to $\mathcal{O}$, sends $\boldsymbol{d}, C_c, C_a, C_m, \mathcal{I}_x$ to P, and $C_c, C_a, C_m, \mathcal{I}_x$ to V.

Online:

1: P submits $\boldsymbol{a}, \boldsymbol{c}$ to $\mathcal{O}$;
2: P submits $\boldsymbol{b}_\times = \boldsymbol{a}_{[1..C_c+C_m]}$, $\boldsymbol{b}_+ = \boldsymbol{b} - \boldsymbol{b}_\times$ to $\mathcal{O}$
3: V queries $\mathcal{O}$ to check that $\mathbf{1}_{[C_c+C_m+1..C]} \circ \boldsymbol{b}_\times = \mathbf{1}_{[1..C_c+C_m]} \circ \boldsymbol{b}_+$
4: V queries $\mathcal{O}$ to check that $\boldsymbol{a} \circ \boldsymbol{b}_\times = \mathbf{1}_{[C_c+1..C_c+C_m]} \circ \boldsymbol{c}$
5: V queries $\mathcal{O}$ to check that $\mathbf{1}_{[C_c+C_m+1..C]} \circ (\boldsymbol{a} + \boldsymbol{b}_+ - \boldsymbol{c}) = \mathbf{0}$
6: V queries $\mathcal{O}$ to check that $\mathbf{1}_{[1..C_c]} \circ (\boldsymbol{c} - \boldsymbol{d}) = \mathbf{0}$
7: V submits $\boldsymbol{x}^{(1)} = \boldsymbol{x}_{[1..C]}$, $\boldsymbol{x}^{(2)} = \boldsymbol{x}_{[C+1..2C]}$, $\boldsymbol{x}^{(3)} = \boldsymbol{x}_{[2C+1..3C]}$ to $\mathcal{O}$
8: V submits $\boldsymbol{t}^{(1)} = \sum_{i \in [C] \cap \mathcal{I}_x} \boldsymbol{e}_i$, $\boldsymbol{t}^{(2)} = \sum_{i \in [C+1..2C] \cap \mathcal{I}_x} \boldsymbol{e}_i$, $\boldsymbol{t}^{(3)} = \sum_{i \in [2C+1..3C] \cap \mathcal{I}_x} \boldsymbol{e}_i$ to $\mathcal{O}$
9: V queries $\mathcal{O}$ to check that $\left(\boldsymbol{a} - \boldsymbol{x}^{(1)}\right) \circ \boldsymbol{t}^{(1)} = \mathbf{0}$
10: V queries $\mathcal{O}$ to check that $\left(\boldsymbol{b}_\times + \boldsymbol{b}_+ - \boldsymbol{x}^{(2)}\right) \circ \boldsymbol{t}^{(2)} = \mathbf{0}$
11: V queries $\mathcal{O}$ to check that $\left(\boldsymbol{c} - \boldsymbol{x}^{(3)}\right) \circ \boldsymbol{t}^{(3)} = \mathbf{0}$
12: P and V run the protocol TripleCopyCheck with inputs $\boldsymbol{a}$, $\boldsymbol{b}_\times + \boldsymbol{b}_+$, $\boldsymbol{c}$.