Polynomial IOPs for Linear Algebra Relations

Alan Szepieniec
alan@nervos.org
Nervos Foundation

Yuncong Zhang
shjdzhangyuncong@sjtu.edu.cn
Shanghai Jiao Tong University

Abstract. This paper proposes new Polynomial IOPs for arithmetic circuits. They rely on the monomial coefficient basis to represent the matrices and vectors arising from the arithmetic constraint satisfaction system, and build on new protocols for establishing the correct computation of linear algebra relations such as matrix-vector products and Hadamard products. Our protocols give rise to concrete proof systems with succinct verification when compiled down with a cryptographic compiler whose role is abstracted away in this paper. Depending only on the compiler, the resulting SNARKs are either transparent or rely on a trusted setup.

Keywords: SNARK · Polynomial IOP · Zero-Knowledge · Succinct Verification

1 Introduction

Succinct Non-Interactive Arguments of Knowledge (SNARKs) enable a resource-constrained verifier to cryptographically verify the authentic computations of an untrusted prover. The technology is particularly well-suited to the cryptocurrency setting, where participants are typically anonymous, untrusted, and where the success of the network depends on the capability of lightweight nodes to verify the network's consensus (however that is defined). In this setting, there is a large monetary incentive for malicious behavior.

Despite the flurry of rapid related and unrelated developments by diverse parties, two trends are emerging as good practices in this domain.

1. *Functional separation in the compilation pipeline.* The compilation process for general purpose zero-knowledge proofs is separated into multiple steps with clear boundaries. At the input of this pipeline is a computation, represented either as program source code or as a circuit. A technique called arithmetization turns this computation into a constraint system involving native operations over a finite field. The next step transforms this constraint system into an abstract proof system between two parties, prover P and verifier V, that are interactive Turing machines with access to unrealistic or unrealizable resources such as PCP oracles. These abstract proof systems in this step are called *interactive oracle proofs (IOPs)* and typically achieve statistical or even perfect security. In the last step, the *cryptographic compilation*, the unrealistic resources are replaced by cryptographic approximations that achieve the same functionality at the expense of introducing computational hardness assumptions for security.

2. *Polynomial IOP formalism.* The abstract information-theoretical proof system in the step before cryptographic compilation could in principle rely on a variety of unrealistic resources, and build a sound proof system from their mathematical properties. However, for the purpose of establishing soundness, the Schwartz-Zippel lemma is an indispensable tool. The strategy is to reduce the satisfaction of arithmetic constraints arising from the constraint system to series of identities of *low-degree polynomials*. By evaluating these polynomials in random points, their equality is tested probabilistically. If the left and right hand sides of an equation represent identical polynomials, they are identical everywhere, but if they are unequal they are different *almost* everywhere. The Schwartz-Zippel lemma provides an exact concrete quantification of the security lost due to this probabilistic approximation. A *Polynomial IOP* is the abstract proof system tailored to this strategy. In this formalism, the prover sends low degree polynomials to the verifier, and rather than reading the entire list of coefficients, the verifier queries these polynomials in a given point through an oracle interface. The cryptographic compiler uses a *polynomial commitment scheme* to simulate this unrealistic resource.

These trends are visible in the rise of universal SNARKs with universal and updatable structured reference strings (SRS's) such as Sonic [11], PLONK [8], and Marlin [7]. The common idea here is to use the cryptographic pairing-based mathematics only to realize *polynomial commitment scheme*, typically the KZG scheme [10]. Since the SRS is used only for the KZG scheme, it is independent of the preceding abstract proof system and the circuit it encodes; this independence is precisely what enables updates to the SRS and its adaptation to any circuit. PLONK and Marlin independently formalize this abstraction and introduce the terms *Polynomial Protocol* and *Algebraic Holographic Proof (AHP)*, respectively. This paper adopts the terminology of Bünz *et al.* [6], who introduce a new polynomial commitment scheme (and hence a cryptographic compiler) based on groups of unknown order and in the process explore the landscape of protocols it can apply to.

These trends are *also* visible in the rise of IOPs based on Reed-Solomon codes [1,4,3]. The underlying abstract protocols here are not explicitly Polynomial IOPs. However, their common feature is the reliance on Reed-Solomon codewords as the proof oracles. Since Reed-Solomon codewords are obtained by evaluating polynomials in a domain of points whose cardinality is larger than the polynomials' degree, these proof oracles uniquely identify the originating low-degree polynomials. As a result, a Reed-Solomon IOP is a Polynomial IOP in disguise.

Despite the spontaneous convergence onto Polynomial IOPs as a useful formalism, there seems to be little agreement about the optimal interface between Polynomial IOPs and the arithmetic constraint systems that they realize. Arithmetic constraint systems typically express constraints using matrix algebra: in terms of vectors, and matrix multiplication, but also *Hadamard products*, which is a fancy word for the element-wise products of pairs of equal-length vectors. The

set of operations that Polynomial IOPs natively offer are somewhat different. As a result, how the Polynomial IOP represents the objects in the arithmetic constraint system and how it simulates the equations that constrain them, are the key questions in the design process of Polynomial IOPs. The various answers to these questions are what set the various Polynomial IOPs for arithmetic circuits apart.

- Marlin and Aurora represent the objects of the arithmetic constraint system as the Reed-Solomon codewords of polynomials. Standard techniques establish the correct computation of a Hadamard product of such codewords. The computation of a linear transform applied to such a codeword is reduced to checking the sum of a related codeword.
- PLONK represents the vector of wire values as the values of a polynomial in a domain of points. A permutation argument establishes the assignment of wires to gates and the standard techniques for Reed-Solomon codewords establish the consistency of inputs and outputs to addition and multiplication gates.
- Sonic represents the vectors of left, right, and output wires of a series of multiplication gates as the coefficient vectors of three polynomials. The consistency of these multiplication gates, and of a linear transform, is established by checking several properties of bivariate polynomials. The paper furthermore explains under which conditions these bivariate polynomials can be simulated with univariate ones.

Contributions. In this paper we propose a new Polynomial IOP for arithmetic circuits, called Claymore[1]. Succinct verification is achieved with a trusted preprocessing phase. When compiled with any polynomial commitment scheme, the result is a concrete zk-SNARK with universal updatable structured reference string, or transparent setup, depending only on the nature of the polynomial commitment scheme.

The arithmetic constraint system chosen to represent the arithmetic circuit is the Hadamard Product Relation (HPR), in which the witness consists of three vectors representing the left, right, and output wires of a list of multiplication gates. We note that Sonic realizes a similar constraint satisfaction relation by reducing both the multiplication and linear constraints into one large equation. In Claymore, the multiplication gate consistency and linear consistency are achieved in two separate steps, both of which rely on a collection of subprotocols for linear algebra relations that we develop along the way. The separate steps are later merged as an explicit optimization.

Like Sonic but unlike Marlin and PLONK, Claymore opts for the *monomial coefficient basis* to represent the vectors of the arithmetic constraint system. DenseClaymore represents the linear transform as a dense matrix in the monomial coefficient basis and this choice results in the smallest number of polynomials in the transcript across all Polynomial IOPs. The price to pay for this brevity is the $O(n^2)$ scaling of the polynomials' degree, where n is the size of the circuit.

[1] A type of Scottish sword.

A question that naturally arises when using this basis, is whether it is also equipped to deal with the sparse linear transformations that typically come from long-winded computations. We answer this question positively by providing methods for dealing with sparse linear algebra relations, culminating in a sparse variant of Claymore. This variant concretely outperforms Marlin and Sonic in terms of the number of polynomials in the transcript. While this number is smaller still for PLONK, one notes that PLONK does not support arbitrary fan-in for linear constraints, whereas Marlin and Claymore (both variants) do.

Additionally, we compare the new and existing Polynomial IOPs both abstractly and concretely. In the abtract comparison we determine how the key performance-driving parameters of the Polynomial IOP evolve as a function of the circuit size. In the concrete comparison, we transform the various Polynomial IOPs with three different cryptographic compilers into concrete zk-SNARKs in order to compare the size of the resulting proofs. In this comparison, all proofs establish the integrity of the same benchmark computation.

Motivation and applications. The motivation for this work is chiefly theoretical. We study the interface between arithmetic circuits and Polynomial IOPs in isolation of other constraints and demands. As a result of this focus, our protocol is arguably simpler than other protocols that achieve nominally the same thing. Complexity is the friend of mistakes, and our protocol may therefore be the preferred option for this reason even in circumstances where it is inferior in terms of performance.

The dense variant of Claymore performs extremely well for shallow arithmetic circuits, such as the verification circuits of lattice-based and MQ-based signature schemes, which typically involve operations on large matrices and vectors over a small finite field. As a result, a DenseClaymore-SNARK is an outstanding candidate for achieving post-quantum signature aggregation, or signature schemes with various fancy properties that zero-knowledge proofs enable.

Using SNARKs in combination with other cryptographic tools points to a useful property that SNARKs frequently lack — they typically require a finite field with a particular structure, such as a large multiplicative subgroup of smooth order. Marlin and PLONK have this property, while Sonic is defined for arbitrary fields. Using the SNARK in combination with a different cryptosystem that requires an incompatible field, requires the SNARK to simulate the cryptosystem's field operations using the arithmetic constraint system of the SNARK. In contrast, Claymore (like Sonic) induces no such costly simulation overhead as it works for any large enough finite field.

The protocols proposed here promote *simplicity through modularity*. However, we observe that once the basic protocol has been composed, there are available optimizations that improve its characteristics at the cost of violating the boundaries between modules. This observation highlights the utility of separating *design* from *optimization* considerations. Note that it is only the optimized SparseClaymore protocol that outperforms Marlin in the target metric, number of polynomials. The unoptimized version is inferior in all respects. Furthermore, the proof of zero-knowledge relies on a batching-related optimization that ap-

plies to both variants of Claymore; without this optimization the proof is tricky and complex. Lastly, the optimizations stand on their own, and can possibly improve other Polynomial IOPs beyond Claymore.

2 Preliminaries

2.1 Indexed Relations

Owing to their convenience, we use *indexed relations* [7]. An indexed relation is a set $\mathcal{R}$ of tuples $(\mathtt{i}, \mathtt{x}, \mathtt{w})$, whose three components are called the *index*, *instance*, and *witness*, respectively. The separation between index and instance captures the intuition that some properties of concrete proofs for $\mathcal{R}$ should be computable from $\mathtt{i}$ even before $\mathtt{x}$ is known. For instance, $\mathtt{i}$ can be the description of an arithmetic circuit, $\mathtt{x}$ the values of the output wires, and $\mathtt{w}$ an assignment of values to all wires that makes the all gates consistent. The projection $\{(\mathtt{i}, \mathtt{x}) \mid (\mathtt{i}, \mathtt{x}, \mathtt{w}) \in \mathcal{R}\}$ of triples in $\mathcal{R}$ onto the first two components is the *indexed language corresponding to* $\mathcal{R}$ and is denoted by $\mathcal{L}(\mathcal{R})$.

2.2 Constraint Systems

A constraint system is a representation of a computation in terms of equations with unknown variables. When there is an assignment to the unknown variables that satisfies all equations, we say the constraint system is *satisfiable*, and this assignment is the *witness*. The *index* determines all fixed constants in the equations, and the *instance* determines known variables that can vary independently of the index but are ultimately known by all parties involved.

The following constraint system is adapted from Bootle *et al.* [5].

Definition 1 (Hadamard Product Relation (HPR)). *Let $\mathbb{F}$ be a finite field. A triple $(\mathtt{i}, \mathtt{x}, \mathtt{w})$ where $\mathtt{i} = (m, n, M)$ with $m, n \in \mathbb{N}$, and $M \in \mathbb{F}^{m \times (1+3n)}$, where $\mathtt{x} = \mathbf{x} \in \mathbb{F}^m$, and where $\mathtt{w} = (\mathbf{w_l}, \mathbf{w_r}, \mathbf{w_o}) \in \mathbb{F}^n \times \mathbb{F}^n \times \mathbb{F}^n$; satisfies the* Hadamard Product Relation *iff both*

$$\mathbf{x} = M \begin{pmatrix} 1 \\ \hline \mathbf{w_l} \\ \hline \mathbf{w_r} \\ \hline \mathbf{w_o} \end{pmatrix} \tag{1}$$

and

$$\mathbf{w_l} \circ \mathbf{w_r} = \mathbf{w_o} \ , \tag{2}$$

where $\circ$ denotes the Hadamard (i.e., entry-wise) product; and in this case we write $(\mathtt{i}, \mathtt{x}, \mathtt{w}) \in \mathcal{R}_{\mathsf{HPR}}$.

2.3 Interactive Proof Systems

Definition 2 (Interactive Proof System). *Let $\mathcal{R}$ be an indexed relation with corresponding relation language $\mathcal{L}(\mathcal{R})$. An* interactive proof system *is a pair (P, V) of stateful interactive Turing machines such that: the input to P is $(\mathtt{i}, \mathtt{x}, \mathtt{w})$, the input to V is $(\mathtt{i}, \mathtt{x})$; P and V exchange $\mathsf{r} = \mathsf{r}(|\mathtt{i}|)$ messages in total; and in the last step of the protocol V outputs a single bit $b \in \{\top, \bot\}$. The system satisfies two more properties:*

- Completeness — V *accepts members of $\mathcal{L}(\mathcal{R})$:* $(\mathtt{i}, \mathtt{x}) \in \mathcal{L}(\mathcal{R}) \Rightarrow b = \top$.
- Soundness *(with* soundness error σ*)* — V *rejects non-members of $\mathcal{L}(\mathcal{R})$ except with probability at most σ taken over the all random coins involved:* $\Pr[(\mathtt{i}, \mathtt{x}) \notin \mathcal{L}(\mathcal{R}) \Rightarrow b = \bot] \geq 1 - \sigma$.

Soundness becomes a moot point when for the given index $\mathtt{i}$ every instance $\mathtt{x}$ has a matching witness $\mathtt{w}$ such that $(\mathtt{i}, \mathtt{x}, \mathtt{w}) \in \mathcal{R}$. In this case a stronger notion called *knowledge soundness* [2] is preferred, which informally requires that any adversary that successfully convinces the verifier can be made to leak a witness by an extractor machine that has the same interface as the verifier but can additionally reset the adversary to an earlier point in time without forgetting the observed transcripts. In our context, all witnesses are encoded into oracles, and the prover displays knowledge of them simply by providing the oracles to the verifier. As a result, at our level of abstraction, knowledge soundness follows automatically from soundness. When the oracles are simulated by a concrete cryptographic tool, knowledge soundness becomes an important consideration that is not automatically satisfied. However, this cryptographic instantiation is beyond the scope of this paper.

A proof system is zero-knowledge [9] if, informally, an authentic transcript could have been produced by an adversary who is ignorant of the witness. More formally, the distribution of authentic transcripts must be samplable with public information only.

Definition 3 (Honest-Verifier Zero-Knowledge). *Let $\mathcal{R}$ be an indexed relation and let (P, V) be a proof system for $\mathcal{R}$. Let $tr \leftarrow \langle \mathsf{P}(\mathtt{i}, \mathtt{x}, \mathtt{w}), \mathsf{V}(\mathtt{i}, \mathtt{x}) \rangle$ denote the assignment to the variable tr of the transcript arising from the interaction between P with input $(\mathtt{i}, \mathtt{x}, \mathtt{w})$ and V with input $(\mathtt{i}, \mathtt{x})$. The proof system (P, V) is* honest-verifier zero-knowledge *if there exists a polynomial-time Turing machine S such that the distribution $\mathcal{D}_0$ of authentic transcripts $tr \leftarrow \langle \mathsf{P}(\mathtt{i}, \mathtt{x}, \mathtt{w}), \mathsf{V}(\mathtt{i}, \mathtt{x}) \rangle$, is identical to the distribution $\mathcal{D}_1$ of simulated transcripts $tr \leftarrow \mathsf{S}(\mathtt{i}, \mathtt{x})$. When $\mathcal{D}_0$ and $\mathcal{D}_1$ are distinct, we consider the statistical distance and use the term* Statistical *Honest-Verifier Zero-Knowledge.*

2.4 Polynomial IOP

Informally, a Polynomial IOP is an abstract proof system, where the prover sends polynomials and the verifier, instead of reading the polynomials in their entirety, is allowed to query the polynomial as oracles in select points.

Definition 4 (Polynomial IOP). *Let $\mathcal{R}$ be an indexed relation with corresponding indexed language $\mathcal{L}(\mathcal{R})$, $\mathbb{F}$ some finite field, and $\mathsf{d} \in \mathbb{N}$ a degree bound. A* Polynomial IOP for $\mathcal{R}$ with degree bound d *is a pair of interactive machines (P, V), satisfying the following description.*

- (P, V) *is an interactive proof for $\mathcal{L}(\mathcal{R})$ with r rounds, and with soundness error σ.*
- P *sends polynomials $f_i(X) \in \mathbb{F}[X]$ of degree at most d to V.*
- V *is an oracle machine with access to a list of oracles, which contains one oracle for each polynomial it has received from the prover.*
- *When an oracle associated with a polynomial $f_i(X)$ is queried on a point $z_j \in \mathbb{F}$, the oracle responds with the value $f_i(z_j)$.*
- V *sends challenges $\alpha_k \in \mathbb{F}$ to P.*
- V *is public coin.*

Definition 4 stipulates one global degree bound d for all polynomials. In Appendix A we offer this alternative definition that stipulates individual degree bounds d_i for each polynomial $f_i(X)$. The same appendix presents a transformation between definitions to establish their equivalence. This transformation does lose some generality: queries in $z_j = 0$ are not allowed, the global-bound protocol has one polynomial more, and the soundness error increases by at most $\frac{2p+\mathsf{d}-1}{|\mathbb{F}|-1}$, where p is the original number of polynomials. However, these restrictions are not significant for typical applications of Polynomial IOPs, where the field $\mathbb{F}$ is large. Therefore, without too much loss of generality, we may assume for the sake of a simpler presentation that the polynomials come with individual degree bounds.

With a minor extension, Polynomial IOPs can appropriately capture preprocessing. This extension introduces third machine, the *indexer* I. As its name suggests, I reads only $\mathtt{i}$, and it outputs a list of polynomials to which V has oracle access.

Definition 5 (Polynomial IOP with Preprocessing). *Let $\mathcal{R}$ be an indexed relation with corresponding language $\mathcal{L}(\mathcal{R})$. A* Polynomial IOP with Preprocessing *is a tuple of interactive machines $(\mathsf{I}, \mathsf{P}, \mathsf{V})$ such that (P, V) is a Polynomial IOP for $\mathcal{L}(\mathcal{R})$ and such that*

- I *takes $\mathtt{i}$ for input and outputs a list of polynomials of degree at most d;*
- V *has oracle access to these polynomials in addition to the polynomials it receives from P.*

Some of the Polynomial IOPs in this paper are designed for modular composition. As a result, V does not begin with an empty list of polynomial oracles. In order to define the relations that these Polynomial IOPs realize, we denote by $[f_i(X)]$ a polynomial $f_i(X)$ that was sent to V by I or P at some earlier stage and to which V has oracle access.

3 Dense Linear Algebra Relations

3.1 Inner Product

Bünz *et al.* [6] are the first to sketch a Polynomial IOP that realizes an inner product relation between two vectors. It relies on the fact that the inner product of the coefficient vectors of $f_{\boldsymbol{a}}(X)$ and $f_{\boldsymbol{b}}(X)$ is the middle coefficient of $f_{\boldsymbol{a}}(X) \cdot X^{\mathsf{d}} \cdot f_{\boldsymbol{b}}(X^{-1})$, assuming that both $f_{\boldsymbol{a}}(X)$ and $f_{\boldsymbol{b}}(X)$ are of degree d. To verify the middle coefficient is indeed the claimed inner product c, V needs two polynomials: the left half $l(X)$ and the right half $r(X)$, both of degree $\mathsf{d}-1$. Then the identity $f_{\boldsymbol{a}}(X) \cdot X^{\mathsf{d}} \cdot f_{\boldsymbol{b}}(X^{-1}) = l(X) + X^{\mathsf{d}} \cdot c + X^{\mathsf{d}+1} \cdot r(X)$ cannot hold in more than $2\mathsf{d}$ points unless c is the correct inner product.

Our variant of this protocol achieves the same result with the same number of queries but with one polynomial oracle less. This trade-off induces a doubling of the polynomial's degree and an increase-by-one in the number of distinct evaluation points. To see how this is achieved, observe that the coefficient on X^{d} of the polynomial $f_{\boldsymbol{a}}(X) \cdot X^{\mathsf{d}} \cdot f_{\boldsymbol{b}}(X^{-1}) - c \cdot X^{\mathsf{d}}$ is zero. The same is true for $h(X) = \bar{h}(X) \cdot \gamma^{\mathsf{d}} - \bar{h}(\gamma X)$ for *any* $\bar{h}(X)$ and any γ with a large enough multiplicative order. If $f_{\boldsymbol{a}}(X) \cdot X^{\mathsf{d}} \cdot f_{\boldsymbol{b}}(X^{-1}) = \sum_{i=0}^{2\mathsf{d}} c_i X^i$ (with $c_{\mathsf{d}} = c$), then P can obtain $\bar{h}(X)$ by setting its ith coefficient to $c_i/(\gamma^{\mathsf{d}} - \gamma^i)$ when $i \neq \mathsf{d}$, or uniformly at random when $i = \mathsf{d}$. The verifier V tests that the coefficient of X^{d} is indeed zero by sampling the left and right hand sides of the polynomial identity

$$\bar{h}(X) \cdot \gamma^{\mathsf{d}} - \bar{h}(\gamma \cdot X) = f_{\boldsymbol{a}}(X) \cdot X^{\mathsf{d}} \cdot f_{\boldsymbol{b}}(X^{-1}) - c \cdot X^{\mathsf{d}} \tag{3}$$

in a uniformly random point $z \xleftarrow{\$} \mathbb{F}\backslash\{0\}$. The multiplicative order of γ must be larger than $2\mathsf{d}$ for this $\bar{h}(X)$ to exist; for simplicity set γ to the smallest element that generates $\mathbb{F}\backslash\{0\}, \times$.

Formally, the relation realized by inner product protocols is

$$\mathcal{R}_{\text{ip}} = \left\{ (\mathbb{i}, \mathbb{x}, \mathbb{w}) \;\middle|\; \begin{array}{l} \mathbb{i} = \mathsf{d} \\ \mathbb{x} = ([f_{\boldsymbol{a}}(X)], [f_{\boldsymbol{b}}(X)], c) \\ \mathbb{w} = (f_{\boldsymbol{a}}(X), f_{\boldsymbol{b}}(X)) \\ f_{\boldsymbol{a}}(X) = \sum_{i=0}^{\mathsf{d}} a_i X^i \\ f_{\boldsymbol{b}}(X) = \sum_{i=0}^{\mathsf{d}} b_i X^i \\ c = \sum_{i=0}^{\mathsf{d}} a_i b_i \end{array} \right\} . \tag{4}$$

Theorem 1 (Security of InnerProduct). *Protocol* InnerProduct *of Protocol 1 is a Polynomial IOP for $\mathcal{L}(\mathcal{R}_{\text{ip}})$ with completeness and soundness with soundness error $\sigma = \frac{2\mathsf{d}}{|\mathbb{F}|-1}$.*

Proof. The protocol revolves around the polynomial identity of Equation (3). The verifier tests this identity by sampling left and right hand sides in a random point z. Since this is an identity whenever $c = \boldsymbol{a}^{\mathsf{T}}\boldsymbol{b}$, completeness follows. For soundness, observe that when $c \neq \boldsymbol{a}^{\mathsf{T}}\boldsymbol{b}$ then the coefficient of X^{d} on the right hand side of (3) is nonzero whereas the matching coefficient of the left hand side

description: decides $\mathcal{L}(\mathcal{R}_{\text{ip}})$
inputs: $\mathbb{i} : \mathsf{d}$
$\mathbb{x} : ([f_{\boldsymbol{a}}(X)], [f_{\boldsymbol{b}}(X)], c)$
$\mathbb{w} : (f_{\boldsymbol{a}}(X), f_{\boldsymbol{b}}(X))$
begin
P computes $f_{\boldsymbol{c}}(X) = \sum_{i=0}^{2\mathsf{d}} c_i X^i \leftarrow f_{\boldsymbol{a}}(X) \cdot X^{\mathsf{d}} \cdot f_{\boldsymbol{b}}(X^{-1})$
P computes $\bar{h}(X) = \sum_{i=0}^{2\mathsf{d}} \bar{h}_i X^i$ with $\bar{h}_i \leftarrow \frac{c_i}{\gamma^{\mathsf{d}} - \gamma^i}$ for all $i \neq \mathsf{d}$ and $\bar{h}_{\mathsf{d}} \xleftarrow{\$} \mathbb{F}$
P sends $\bar{h}(X)$ of degree at most $2\mathsf{d}$ to V
V samples $z \xleftarrow{\$} \mathbb{F} \backslash \{0\}$ and queries $([f_{\boldsymbol{a}}(X)], [f_{\boldsymbol{b}}(X)], [\bar{h}(X)], [\bar{h}(X)])$ in $(z, z^{-1}, z, \gamma \cdot z)$
V receives $y_a = f_{\boldsymbol{a}}(z)$, $y_b = f_{\boldsymbol{b}}(z^{-1})$, $y_h = \bar{h}(z)$, and $y_h^* = \bar{h}(\gamma \cdot z)$
V tests $y_h \cdot \gamma^{\mathsf{d}} - y_h^* \stackrel{?}{=} y_a \cdot y_b \cdot z^{\mathsf{d}} - c \cdot z^{\mathsf{d}}$

Protocol 1: InnerProduct

is zero. There are at most $2\mathsf{d}$ points z where left and right hand sides are equal, since both hands are bounded by this degree. By the Schwartz-Zippel lemma, the probability of a false accept is $\sigma = \frac{2\mathsf{d}}{|\mathbb{F}|-1}$. □

3.2 Batched Inner Product

We can batch multiple invocations of protocol InnerProduct into a single protocol that requires the prover to send only one polynomial oracle. Formally, the relation is given by

$$\mathcal{R}_{\text{bip}} = \left\{ (\mathbb{i}, \mathbb{x}, \mathbb{w}) \middle| \begin{array}{l} \mathbb{i} = (m, \mathsf{d}) \\ \mathbb{x} = \{([f_{\boldsymbol{a}_i}(X)], [f_{\boldsymbol{b}_i}(X)], c_i)\}_{i=1}^m \\ \mathbb{w} = \{(f_{\boldsymbol{a}_i}(X), f_{\boldsymbol{b}_i}(X))\}_{i=1}^m \\ \forall i \in \{0, \ldots, m-1\} \,.\, f_{\boldsymbol{a}_i}(X) = \sum_{j=0}^{\mathsf{d}} a_{ij} X^i \\ \forall i \in \{0, \ldots, m-1\} \,.\, f_{\boldsymbol{b}_i}(X) = \sum_{j=0}^{\mathsf{d}} b_{ij} X^i \\ \forall i \in \{0, \ldots, m-1\} \,.\, c_i = \sum_{j=0}^{\mathsf{d}} a_{ij} b_{ij} \end{array} \right\} \,. \quad (5)$$

Theorem 2 (Security of BatchedInnerProduct). *Protocol* BatchedInnerProduct *of Protocol 2 is a Polynomial IOP for* $\mathcal{L}(\mathcal{R}_{\text{bip}})$ *with completeness and soundness with soundness error* $\sigma = \frac{2\mathsf{d}+m-1}{|\mathbb{F}|-1}$.

Proof. Let $\sum_{j=0}^{2\mathsf{d}} c_{i,j} X^j = f_{\boldsymbol{a}_i}(X) \cdot f_{\boldsymbol{b}_i}(X^{-1}) \cdot X^{\mathsf{d}}$ for all $i \in \{1, \ldots, m\}$. Furthermore, let $\bar{H}(X, Y) = \sum_{i=1}^m \sum_{j=0}^{2\mathsf{d}} h_{i,j} X^j Y^{i-1}$ with $h_{i,j} = \frac{c_{i,j}}{\gamma^{\mathsf{d}} - \gamma^j}$ for $j \neq \mathsf{d}$, arbitrary $h_{i,\mathsf{d}}$ for $i > 0$, and $h_{0,\mathsf{d}}$ such that $\bar{h}(X) = \bar{H}(X, \alpha)$.

The protocol revolves around the bivariate polynomial identity

$$\bar{H}(X, Y) \cdot \gamma^{\mathsf{d}} - \bar{H}(\gamma \cdot X, Y) = X^{\mathsf{d}} \cdot \sum_{i=1}^m (f_{\boldsymbol{a}_i}(X) \cdot f_{\boldsymbol{b}_i}(X^{-1}) - c_i) \cdot Y^{i-1} \,. \quad (6)$$

description: decides $\mathcal{L}(\mathcal{R}_{\text{bip}})$
inputs: $\mathtt{i} : (m, \mathsf{d})$
$\mathtt{x} : \{([f_{\boldsymbol{a}_i}(X)], [f_{\boldsymbol{b}_i}(X)], c_i)\}_{i=1}^m$
$\mathtt{w} : \{(f_{\boldsymbol{a}_i}(X), f_{\boldsymbol{b}_i}(X))\}_{i=1}^m$
begin
P computes $f_{\boldsymbol{c}_i}(X) \leftarrow f_{\boldsymbol{a}_i}(X) \cdot f_{\boldsymbol{b}_i}(X^{-1}) \cdot X^{\mathsf{d}}$ for i from 1 to m
V samples $\alpha \xleftarrow{\$} \mathbb{F}\backslash\{0\}$ and sends α to P
P computes $f_{\boldsymbol{c}}(X) \leftarrow \sum_{i=1}^m f_{\boldsymbol{c}_i}(X) \cdot \alpha^{i-1}$
P computes $\bar{h}(X) = \sum_{j=0}^{2\mathsf{d}} \bar{h}_j X^j$ with $\bar{h}_j \leftarrow \frac{c_j}{\gamma^{\mathsf{d}} - \gamma^j}$ for all $j \neq \mathsf{d}$ and $\bar{h}_{\mathsf{d}} \xleftarrow{\$} \mathbb{F}$
P sends $\bar{h}(X)$ of degree at most $2\mathsf{d}$ to V
V samples $z \xleftarrow{\$} \mathbb{F}\backslash\{0\}$ and queries
$(\{([f_{\boldsymbol{a}_i}(X)], [f_{\boldsymbol{b}_i}(X)])\}_{i=1}^m, [\bar{h}(X)], [\bar{h}(X)])$ in $(\{z, z^{-1}\}_{i=1}^m, z, \gamma \cdot z)$
V receives $y_{a,i} = f_{\boldsymbol{a}_i}(z)$, $y_{b,i} = f_{\boldsymbol{b}_i}(z^{-1})$ for i from 1 to m, and $y_h = \bar{h}(z)$, $y_h^* = \bar{h}(\gamma \cdot z)$
V tests $y_h \cdot \gamma^{\mathsf{d}} - y_h^* \stackrel{?}{=} z^{\mathsf{d}} \cdot \sum_{i=1}^m (y_{a,i} \cdot y_{b,i} - c_i) \cdot \alpha^{i-1}$

Protocol 2: BatchedInnerProduct

The verifier tests this identity by sampling left and right hand sides in a random point (z, α). Since this is an identity whenever $c_i = \boldsymbol{a}_i^{\mathsf{T}} \boldsymbol{b}_i$ for all i from 1 to m, completeness follows. For soundness, consider when for some i, $c_i \neq \boldsymbol{a}_i^{\mathsf{T}} \boldsymbol{b}_i$. Then left and right hand sides of (6) are unequal. There are at most $2\mathsf{d} + m - 1$ points (z, α) in which left and right hand sides are equal, since both hands are bounded by this total degree. By the (two-dimensional) Schwartz-Zippel lemma, the probability of a false accept is $\sigma = \frac{2\mathsf{d}+m-1}{|\mathbb{F}|-1}$. □

This inner product protocol and its batched version are convenient for zero-knowledge. The verifier V makes two queries to the polynomial $\bar{h}(X)$: one in z and one in $\gamma \cdot z$. Since $\bar{h}(X)$ has one uniformly random coefficient, one of the responses is uniformly random. The other one is such that the tested equality is true. So the honest verifier learns no new information from $\bar{h}(X)$.

3.3 Modular Reduction

We start with a protocol that will be used as a subprotocol in the sequel. This protocol establishes that one polynomial, $r(X)$, is the remainder after division of a second polynomial $f(X)$, by a third, $d(X)$. This third polynomial is assumed to be known, but the protocol can be naturally amended to allow V only oracle access to $[d(X)]$. Formally, the relation is given by

$$\mathcal{R}_{\text{reduce}} = \left\{ (\mathtt{i}, \mathtt{x}, \mathtt{w}) \;\middle|\; \begin{array}{l} \mathtt{i} = (\mathsf{d}_f, \mathsf{d}_r) \\ \mathtt{x} = ([f(X)], [r(X)], d(X)) \\ \mathtt{w} = (f(X), r(X)) \\ \exists q(X) \in \mathbb{F}[X] \,.\, f(X) = q(X) \cdot d(X) + r(X) \\ \mathsf{deg}(f) \leq \mathsf{d}_f \\ \mathsf{deg}(r) \leq \mathsf{d}_r \end{array} \right\} . \tag{7}$$

description: decides $\mathcal{L}(\mathcal{R}_{\text{reduce}})$
inputs: $\mathtt{i} : (\mathsf{d}_f, \mathsf{d}_r)$
$\mathbb{x} : ([f(X)], [r(X)], d(X))$
$\mathbb{w} : (f(X), r(X))$
begin
P computes q such that $f(X) = q(X) \cdot d(X) + r(X)$
P sends $q(X)$ of degree at most $\mathsf{d}_f - \mathsf{deg}(d)$ to V
V samples $z \xleftarrow{\$} \mathbb{F}\backslash\{0\}$ and queries $[f(X)]$, $[q(X)]$, and $[r(X)]$ in z
V receives $y_f = f(z)$, $y_q = q(z)$, and $y_r = r(z)$
V tests $y_f \stackrel{?}{=} y_q \cdot d(z) + y_r$

Protocol 3: ModReduce

Theorem 3 (Security of ModReduce). *Protocol* ModReduce *of Protocol 3 is a Polynomial IOP for* $\mathcal{L}(\mathcal{R}_{\text{reduce}})$ *with completeness and soundness with soundness error* $\sigma = \mathsf{d}_f/|\mathbb{F}|$.

Proof. Completeness follows from construction: dividing $f(X)$ by $d(X)$ gives quotient $q(X)$ and remainder $r(X)$. Therefore, $f(X) = q(X) \cdot d(X) + r(X)$ is an identity of polynomials and guaranteed to hold everywhere including in the point z.

For soundness, observe that when $r(X) \not\equiv f(X) \bmod d(X)$ then $d(X)$ does not divide $f(X) - r(X)$. As a result, $f(X) \neq q(X) \cdot d(X) + r(X)$ is an inequality of polynomials with degree $\mathsf{deg}(d) + \mathsf{deg}(q) = \mathsf{d}_f$. Due to the Schwartz-Zippel lemma, the left and right hand sides can evaluate to the same value in at most d_f choices for z. The probability of V accepting when $r(X) \not\equiv f(X) \bmod d(X)$ is therefore $\sigma = \mathsf{d}_f/|\mathbb{F}|$.

What is left to argue is that P fails to convince V when the congruence $r(X) \equiv f(X) \bmod d(X)$ holds, but $r(X)$ is not equal to the remainder after division of $f(X)$ by $d(X)$. The representatives of the congruence class of $r(X)$ are apart by polynomials of degree at least $\mathsf{deg}(d)$, there is only one representative of degree at most $\mathsf{d}_r < \mathsf{deg}(d)$. The index value d_r therefore already constrains $r(X)$ to a unique polynomial. □

3.4 Matrix-Vector Product

The next protocol involves two polynomials that represent vectors in the monomial coefficient basis. It establishes that the one vector is the result of applying a linear transformation to the other. This linear transformation itself can be known and computed explicitly by the verifier. However, for succinct verifiers it is more appealing to encode this matrix into a polynomial oracle. Depending on the context, either the protocol's preprocessing phase produces this oracle, or another external protocol does.

Specifically, let $\boldsymbol{a} \in \mathbb{F}^n$ and $\boldsymbol{b} \in \mathbb{F}^m$ and $M \in \mathbb{F}^{m \times n}$ with the element in row i and column j (both indices starting at zero) indexed as $M_{[i,j]}$. These objects are represented as polynomials with $a_{[i]}$ being ith element of $\boldsymbol{a}$ and simultaneously

the coefficient of the monomial X^i in $f_{\boldsymbol{a}}(X)$, and similarly for $\boldsymbol{b}, b_{[i]}$, and $f_{\boldsymbol{b}}(X)$. When encoded into polynomial form, the matrix is encoded in row-first order, specifically $f_M(X) = \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} M_{[i,j]} X^{in+j}$. The protocol establishes that $\boldsymbol{b} = M\boldsymbol{a}$. Formally, the relation is given by

$$\mathcal{R}_{\text{mvp}} = \left\{ (\mathtt{i}, \mathtt{x}, \mathtt{w}) \;\middle|\; \begin{array}{l} \mathtt{i} = (m, n, M) \\ \mathtt{x} = ([f_{\boldsymbol{a}}(X)], [f_{\boldsymbol{b}}(X)]) \\ \mathtt{w} = (f_{\boldsymbol{a}}(X), f_{\boldsymbol{b}}(X)) \\ f_{\boldsymbol{a}}(X) = \sum_{i=0}^{n-1} \boldsymbol{a}_{[i]} X^i \text{ for some } \boldsymbol{a} \in \mathbb{F}^n \\ f_{\boldsymbol{b}}(X) = \sum_{i=0}^{m-1} \boldsymbol{b}_{[i]} X^i \text{ for some } \boldsymbol{b} \in \mathbb{F}^m \\ \boldsymbol{b} = M\boldsymbol{a} \end{array} \right\} . \tag{8}$$

description: decides $\mathcal{L}(\mathcal{R}_{\text{mvp}})$
inputs: $\mathtt{i} : (m, n, M)$
$\mathtt{x} : ([f_{\boldsymbol{a}}(X)], [f_{\boldsymbol{b}}(X)])$
$\mathtt{w} : (f_{\boldsymbol{a}}(X), f_{\boldsymbol{b}}(X))$
// pre-processing
begin
  I computes $f_M(X) \leftarrow \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} M_{[i,j]} X^{in+j}$
  I sends $f_M(X)$ of degree at most $mn-1$ to P and V
begin
  V samples $\alpha \xleftarrow{\$} \mathbb{F} \backslash \{0\}$ and sends α to P
  P computes $r(X) \leftarrow f_M(X) \bmod X^n - \alpha$
  P sends $r(X)$ of degree at most $n-1$ to V
  P and V run $\mathsf{ModReduce}$ with $\mathtt{i}^{(1)} = (mn-1, n-1)$,
   $\mathtt{x}^{(1)} = ([f_M(X)], [r(X)], X^n - \alpha)$, and $\mathtt{w}^{(1)} = (f_M(X), r(X))$
  V queries $[f_{\boldsymbol{b}}(X)]$ in α and receives $y_{\boldsymbol{\alpha}^\intercal \boldsymbol{b}} = f_{\boldsymbol{b}}(\alpha)$
  P and V run $\mathsf{InnerProduct}$ with $\mathtt{i}^{(2)} = n-1$, $\mathtt{x}^{(2)} = ([r(X)], [f_{\boldsymbol{a}}(X)], y_{\boldsymbol{\alpha}^\intercal \boldsymbol{b}})$,
   and $\mathtt{w}^{(2)} = (r(X), f_{\boldsymbol{a}}(X))$

Protocol 4: DenseMVP

Theorem 4 (Security of DenseMVP). *Protocol* DenseMVP *of Protocol 4 is a Polynomial IOP for* $\mathcal{L}(\mathcal{R}_{\text{mvp}})$ *with completeness and soundness with soundness error* $\sigma = \frac{mn+m+2n-4}{|\mathbb{F}|-1}$.

Proof. Let $\boldsymbol{\alpha}^\mathsf{T} = (\alpha^0, \alpha^1, \cdots)$ and $\boldsymbol{r}^\mathsf{T} = \boldsymbol{\alpha}^\mathsf{T} M$, and consider the equations

$$\boldsymbol{b} = M\boldsymbol{a} \tag{9}$$

$$\boldsymbol{\alpha}^\mathsf{T}\boldsymbol{b} = \boldsymbol{\alpha}^\mathsf{T} M\boldsymbol{a} \tag{10}$$

$$\sum_{i=0}^{m-1} \alpha^i b_{[i]} = \boldsymbol{r}^\mathsf{T}\boldsymbol{a} \tag{11}$$

$$f_{\boldsymbol{b}}(\alpha) = \sum_{i=0}^{n-1} r_{[i]} a_{[i]} \tag{12}$$

$$y_{\boldsymbol{\alpha}^\intercal \boldsymbol{b}} = \mathsf{coeffs}(r(X)) \cdot \mathsf{coeffs}(f_{\boldsymbol{a}}(X)) \tag{13}$$

$$(\mathbb{i}^{(2)}, \mathbb{x}^{(2)}) = (n-1, ([r(X)], [f_{\boldsymbol{a}}(X)], y_{\boldsymbol{\alpha}^\intercal \boldsymbol{b}})) \in \mathcal{L}(\mathcal{R}_{\text{ip}}) \ , \tag{14}$$

where $\mathsf{coeffs} : \mathbb{F}[X] \to \mathbb{F}^n$ is the function that returns the vector of coefficients of its argument. Observe that $\mathsf{coeffs}(r(X)) = \boldsymbol{r}$, by substituting X^n by α in the expression for $f_M(X)$:

$$\sum_{i=0}^{m-1}\sum_{j=0}^{n-1} M_{[i,j]} X^{in+j} \xrightarrow{X^n \mapsto \alpha} r(X) = \sum_{i=0}^{m-1}\sum_{j=0}^{n-1} M_{[i,j]} \alpha^i X^j \tag{15}$$

$$= \sum_{j=0}^{n-1} \left(\sum_{i=0}^{m-1} M_{[i,j]} \alpha^i \right) X^j \tag{16}$$

$$= \sum_{j=0}^{n-1} r_{[j]} X^j \ . \tag{17}$$

Completeness follows from the implications (9) $\Rightarrow$ (10) $\Leftrightarrow$ (11) $\Leftrightarrow$ (12) $\Rightarrow$ (13) $\Rightarrow$ (14).

For soundness, there are 3 events that can cause V to accept despite $\boldsymbol{b} \neq M\boldsymbol{a}$:

1. (9) $\not\Leftarrow$ (10). The probability of this event is at most $\frac{m-1}{|\mathbb{F}|-1}$ due to the Schwartz-Zippel lemma.
2. (12) $\not\Leftarrow$ (13) because $r(X)$ is not the remainder of $f_M(X)$ after division by $X^n - \alpha$. The probability of this event is at most $\frac{mn-1}{|\mathbb{F}|}$, the soundness error of $\mathsf{ModReduce}$.
3. (13) $\not\Leftarrow$ (14), because $y_{\boldsymbol{\alpha}^\intercal \boldsymbol{b}}$ is not the inner product of the coefficient vectors of $r(X)$ and $f_{\boldsymbol{a}}(X)$. The probability of this event is at most $\frac{2n-2}{|\mathbb{F}|}$, the soundness error of $\mathsf{InnerProduct}$.

By the union bound, the soundness error of $\mathsf{DenseMVP}$ is bounded by $\sigma = \frac{mn+m+2n-4}{|\mathbb{F}|-1}$. □

Note that after unrolling, the verifier the $\mathsf{DenseMVP}$ protocol tests two polynomial identities. One arises from expanding $\mathsf{ModReduce}$, and the other arises from $\mathsf{InnerProduct}$. Both polynomial identities involve the polynomial $r(X)$, and

as a result it can be eliminated and the identities merged. We present the unrolled and optimized version in Appendix C.1.

To see that this merger has no effect on soundness, observe that the inequality $lhs_1 \neq lhs_2$ implies $r(X) \neq lhs_1$ or $r(X) \neq lhs_2$. The verifier therefore accepts this false instance with a probability bounded by the same soundness error as the unoptimized protocol. This optimization strategy translates more generally to (some) other Polynomial IOPs: to eliminate a polynomial that is common to two identities, move it to the right hand side and then equate both left hand sides.

3.5 Hadamard Product

The next protocol establishes that the Hadamard (or component-wise) product of two vectors is equal to a third. These vectors are represented as the coefficient vectors of polynomials $f_{\boldsymbol{a}}(X)$, $f_{\boldsymbol{b}}(X)$, and $f_{\boldsymbol{c}}(X)$ such that $\boldsymbol{c} = \boldsymbol{a} \circ \boldsymbol{b}$ and $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c} \in \mathbb{F}^{\mathsf{d}+1}$. The protocol relies on the fact that $\boldsymbol{c} = \boldsymbol{a} \circ \boldsymbol{b}$ implies $\boldsymbol{\alpha}^{\mathsf{T}}(\boldsymbol{a} \circ \boldsymbol{b}) = \boldsymbol{\alpha}^{\mathsf{T}}\boldsymbol{c}$ for all vectors $\boldsymbol{\alpha}$. In other words, one can simply sample a random scalar $\alpha \xleftarrow{\$} \mathbb{F}\backslash\{0\}$, and check the inner product of $\boldsymbol{\alpha} \circ \boldsymbol{a}$ with $\boldsymbol{b}$ against the inner product $\boldsymbol{\alpha}^{\mathsf{T}}\boldsymbol{c}$. Note that the right hand side of this check amounts to $f_{\boldsymbol{c}}(\alpha)$ and the operands in the left hand side amount to the coefficient vectors of $f_{\boldsymbol{a}}(\alpha X)$ and $f_{\boldsymbol{b}}(X)$, respectively. Formally, the relation is given by

$$\mathcal{R}_{\text{hadamard}} = \left\{ (\mathtt{i}, \mathtt{x}, \mathtt{w}) \;\middle|\; \begin{array}{l} \mathtt{i} = \mathsf{d} \\ \mathtt{x} = ([f_{\boldsymbol{a}}(X)], [f_{\boldsymbol{b}}(X)], [f_{\boldsymbol{c}}(X)]) \\ \mathtt{w} = (f_{\boldsymbol{a}}(X), f_{\boldsymbol{b}}(X), f_{\boldsymbol{c}}(X)) \\ \forall i \in \{0, \ldots, \mathsf{d}\} \,.\, a_i b_i = c_i \end{array} \right\} . \tag{18}$$

description: decides $\mathcal{L}(\mathcal{R}_{\text{hadamard}})$
inputs: $\mathtt{i}$: d
$\mathtt{x}$: $[f_{\boldsymbol{a}}(X)], [f_{\boldsymbol{b}}(X)], [f_{\boldsymbol{c}}(X)]$
$\mathtt{w}$: $f_{\boldsymbol{a}}(X), f_{\boldsymbol{b}}(X), f_{\boldsymbol{c}}(X)$
begin
V samples $\alpha \xleftarrow{\$} \mathbb{F}\backslash\{0\}$ and sends α to P
P evaluates $y \leftarrow f_{\boldsymbol{c}}(\alpha)$
V queries $[f_{\boldsymbol{c}}(X)]$ in α and receives $y = f_{\boldsymbol{c}}(\alpha)$
P and V run InnerProduct with $\mathtt{i}^{(1)} = \mathsf{d}$, $\mathtt{x}^{(1)} = ([f_{\boldsymbol{a}}(\alpha X)], [f_{\boldsymbol{b}}(X)], y)$, $\mathtt{w}^{(1)} = (f_{\boldsymbol{a}}(\alpha X), f_{\boldsymbol{b}}(X))$, where V simulates $[f_{\boldsymbol{a}}(\alpha X)]$ using $[f_{\boldsymbol{a}}(X)]$ and the scalar α

Protocol 5: Hadamard

Theorem 5 (Security of Hadamard). *Protocol* Hadamard *of Protocol 9 is a Polynomial IOP for* $\mathcal{L}(\mathcal{R}_{\text{hadamard}})$ *with completeness and soundness with soundness error* $\sigma = 3\mathsf{d}/(|\mathbb{F}| - 1)$.

Proof. Consider the following sequence of equations.

$$\boldsymbol{a} \circ \boldsymbol{b} = \boldsymbol{c} \tag{19}$$

$$\boldsymbol{\alpha}^{\mathsf{T}} \cdot (\boldsymbol{a} \circ \boldsymbol{b}) = \boldsymbol{\alpha}^{\mathsf{T}} \cdot \boldsymbol{c} \tag{20}$$

$$\sum_{i=0}^{\mathsf{d}} (\alpha^i \boldsymbol{a}_{[i]}) \boldsymbol{b}_{[i]} = \sum_{i=0}^{\mathsf{d}} \alpha^i \boldsymbol{c}_{[i]} \tag{21}$$

$$\mathsf{coeffs}(f_{\boldsymbol{a}}(\alpha X)) \cdot \mathsf{coeffs}(f_{\boldsymbol{b}}(X)) = f_{\boldsymbol{c}}(\alpha) \tag{22}$$

$$\mathsf{coeffs}(f_{\boldsymbol{a}}(\alpha X)) \cdot \mathsf{coeffs}(f_{\boldsymbol{b}}(X)) = y \tag{23}$$

$$(\mathbb{i}, \mathbb{x}) = (\mathsf{d}, ([f_{\boldsymbol{a}}(\alpha X)], [f_{\boldsymbol{b}}(X)], y)) \in \mathcal{L}(\mathcal{R}_{\mathsf{InnerProduct}}) \tag{24}$$

Completeness follows from the sequence of implications (19) $\Rightarrow$ (20) $\Leftrightarrow$ (21) $\Leftrightarrow$ (22) $\Leftrightarrow$ (23) $\Rightarrow$ (24).

For soundness, consider when the reverse implications fail.

- (19) $\not\Leftarrow$ (20). This event happens with probability at most $\mathsf{d}/(|\mathbb{F}| - 1)$ due to the Schwartz-Zippel lemma.
- (23) $\not\Leftarrow$ (24). This event happens with probability at most $2\mathsf{d}/(|\mathbb{F}| - 1)$, the soundness error of $\mathsf{InnerProduct}$.

Therefore, the probability that V accepts even though $\boldsymbol{a} \circ \boldsymbol{b} \neq \boldsymbol{c}$ is bounded by $\sigma = 3\mathsf{d}/(|\mathbb{F}| - 1)$. □

4 Sparse Linear Algebra Relations

The purpose of this section is to present an analogue of the $\mathsf{DenseMVP}$ Polynomial IOP but that works when the matrix M is represented sparsely, *i.e.*, as a list of nonzero coefficients and their coordinates. The full, formal presentation of this protocol is rather lengthy, and so we defer it to Appendix B. Here we present an intuitive, high-level overview with just enough detail so that the reader could reconstruct the deferred formal presentation.

4.1 High-Level Overview

Let $M \in \mathbb{F}^{m \times n}$ be a matrix with only K nonzero elements. It can be represented by a triple of functions $(\mathsf{col}, \mathsf{row}, \mathsf{val})$ via $M = \sum_{k=0}^{K-1} \mathbf{e}_{\mathsf{row}(k)} \mathbf{e}_{\mathsf{col}(k)}^{\mathsf{T}} \cdot \mathsf{val}(k)$, where $\mathbf{e}_i$ is the ith unit vector, where $\mathsf{row}, \mathsf{col} : \mathbb{N} \to \mathbb{N}$ indicate the column and row of the kth element, and where $\mathsf{val} : \mathbb{N} \to \mathbb{F}$ indicates its value. We detail a protocol to establish that $\mathbf{y} = M\mathbf{x}$. We first explain the steps from a high level point of view.

From MVP to *bivariate polynomial evaluation.* A key component of the dense matrix-vector multiplication protocol is the evaluation of $(f_M(X) \mod X^n - \alpha)$ at the point z, where $f_M(X)$ is the polynomial associated with the matrix M, *i.e.*, $f_M(X) = \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} M_{[i,j]} X^{in+j}$. This step can equivalently be interpreted as the evaluation of the bivariate polynomial $f_M(X,Y) = \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} M_{[i,j]} X^i Y^j$ in the point (α, z). In other words, if we can achieve sparse bivariate polynomial evaluation, then we can achieve sparse matrix-vector products.

From *bivariate polynomials* to *univariate monomial vectors.* The reduction goes one step further: it is possible to achieve sparse bivariate polynomial evaluation given a procedure that establishes that the vector of coefficients of a dense polynomial is the same as the vector of monomials of a sparse univariate polynomial when evaluated in a given point. To see this, observe that a sparse bivariate polynomial $f(X,Y) = \sum_{k=0}^{K-1} c_k X^{a_k} Y^{b_k}$ can be evaluated in a point (x,y) using the polynomials $f_c(X) = \sum_{k=0}^{K-1} c_k X^k$, $f_x(X) = \sum_{k=0}^{K-1} x^{a_k} X^k$, and $f_y(X) = \sum_{k=0}^{K-1} y^{b_k} X^k$, simply by performing one Hadamard and one InnerProduct subprotocol. This reduction does introduce a problem, namely $f_x(X)$ and $f_y(X)$ cannot be known before V supplies x and y. So how does P commit to them, and how does V verify that the received oracles match with the commitment?

From *univariate monomial vector* to *bit matrix.* Let's focus on $f_x(X)$, as $f_y(X)$ proceeds analogously. This polynomial can be represented by a bit matrix B, which takes the value 1 in cells (a_k, k) and 0 elsewhere. Let H denote the largest such a_k, *i.e.*, $H = \max_k a_k$. Let furthermore $\mathbf{x} = (x^0, x, x^2, \ldots, x^{H-1})^\mathsf{T}$, and $\mathbf{z} = (z^0, z, z^2, \ldots, z^{K-1})^\mathsf{T}$. Then B represents the polynomial $f_x(X)$ since $f_x(z) = \mathbf{x}B\mathbf{z}$.

From *bit matrix* to *Lagrange and Vandermonde* matrices. The idea is to decompose the matrix $B \in \mathbb{F}^{H \times K}$ as the product of two matrices, $L \in \mathbb{F}^{H \times H}$ and $R \in \mathbb{F}^{H \times K}$. Let $\mathcal{H} \subset \mathbb{F}$ be a set of H distinct elements of $\mathbb{F}$ and $\varphi : \mathbb{N} \to \mathcal{H}$ any mapping from $\{0, \ldots, H-1\}$ to $\mathcal{H}$. L is the Lagrange matrix, whose hth row is the coefficient vector of $\mathcal{L}_h(X)$, which is the Lagrange polynomial taking the value 1 in $\varphi(h)$ and 0 in all other points of $\mathcal{H}$. Symbolically:

$$\mathcal{L}_h(X) = \sum_{i=0}^{H-1} L_{[h,i]} X^i = \prod_{\substack{i=0 \\ i \neq h}}^{H-1} \frac{X - \varphi(i)}{\varphi(h) - \varphi(i)} \quad . \tag{25}$$

R is the Vandermonde matrix, whose rows are the (Hadamard) powers of $(\varphi(a_k))_{k=0}^{K-1}$. Specifically:

$$R = \begin{pmatrix} 1 & 1 & \cdots & 1 \\ \varphi(a_0) & \varphi(a_1) & \cdots & \varphi(a_{K-1}) \\ \varphi(a_0)^2 & \varphi(a_1)^2 & \cdots & \varphi(a_{K-1})^2 \\ \vdots & \vdots & \cdots & \vdots \\ \varphi(a_0)^{H-1} & \varphi(a_1)^{H-1} & \cdots & \varphi(a_{K-1})^{H-1} \end{pmatrix} . \tag{26}$$

To verify that $LR = B$, observe that the inner product between $L_{[h,:]}$ and $R_{[:,k]}$ is equal to $\mathcal{L}_h(\varphi(a_k))$. When V provides (x, z) hoping to obtain $\mathbf{x}^\mathsf{T} B\mathbf{z}$, P will respond with $[\mathbf{x}^\mathsf{T} L]$ and $[R\mathbf{z}]$ (in polynomial form), and both proceed to an $\mathsf{InnerProduct}$ protocol. We will refer to these vectors as the Lagrange and Vandermonde vectors, respectively. The next question is, how and against what does V verify them?

Verifying the Lagrange vector. After sending x and receiving the vector (encoded as a polynomial oracle) $[\mathbf{x}^\mathsf{T} L]$, V sends γ to P, who responds with the vector $[L\boldsymbol{\gamma}]$, where $\boldsymbol{\gamma} = (1, \gamma, \gamma^2, \ldots, \gamma^{H-1})^\mathsf{T}$. Let $Z_\mathcal{H}(X)$ be the unique monic polynomial of degree $H-1$ that vanishes on $\mathcal{H}$. By repeating the equation

$$\mathcal{L}_h(\gamma) \cdot (\gamma - \varphi(h)) = \frac{Z_\mathcal{H}(\gamma)}{\prod\limits_{\substack{i=0 \\ i \neq h}}^{H-1} (\varphi(h) - \varphi(i))} \tag{27}$$

for every h, V can check $L\boldsymbol{\gamma}$ using a $\mathsf{Hadamard}$ subprotocol, assuming that P or I previously committed to oracles for $f_\varphi(X) = \sum_{h=0}^{H-1} \varphi(h) X^h$ and $f_\mathcal{H}(X) = \sum_{h=0}^{H-1} \left(\prod_{i \in \{0 \ldots H-1\} \setminus \{h\}} (\varphi(h) - \varphi(i)) \right)^{-1} \cdot X^h$. The next step is to query the oracles $[\mathbf{x}L]$ and $[L\boldsymbol{\gamma}]$ in γ and x, respectively and verify that the responses match.

Verifying the Vandermonde Vector. A similar technique allows V to verify the Vandermonde vector. After sending z and receiving $[R\mathbf{z}]$ back, V sends δ, and P responds with $[\boldsymbol{\delta}^\mathsf{T} R]$. Next, V checks that for every $k \in \{0, \ldots, K-1\}$,

$$\left(\sum_{i=0}^{H-1} (\delta \cdot \varphi(a_k))^i \right) \cdot (\delta\varphi(a_k) - 1) = (\delta\varphi(a_k))^H - 1 \tag{28}$$

using another $\mathsf{Hadamard}$ protocol and the precommitted oracle $f_a(X) = \sum_{k=0}^{K-1} \varphi(a_k) X^k$. Lastly, V queries $[R\mathbf{z}]$ in δ to see if the response matches with $[\boldsymbol{\delta} R]$ when queried in z.

Batching Lagrange and Vandermonde Vectors. In order to establish the correct evaluation of the bivariate polynomial, the prover must establish the correct production of two univariate monomial vectors. A naïve implementation invokes the Lagrange vector and the Vandermonde vector procedure twice. However, it turns out to be possible to merge these two invocations, and save a total of 4 polynomials. We treat this optimization explicitly in Appendix C.

5 A Polynomial IOP for Arithmetic Circuits

5.1 The Protocol

The next protocol, Protocol 6 puts many of the previously developed tools together into a Polynomial IOP (with preprocessing) for arithmetic circuits as captured by the HPR. To differentiate our protocol from other similar ones, we name it Claymore.

description: realizes $\mathcal{R}_{\text{hpr}}$
inputs: $\mathbb{i}$: (m, n, M) with $M \in \mathbb{F}^{m \times (3n+1)}$
$\mathbb{x}$: $\mathbf{x} \in \mathbb{F}^n$
$\mathbb{w}$: $(\mathbf{w_l}, \mathbf{w_r}, \mathbf{w_o}) \in \mathbb{F}^n \times \mathbb{F}^n \times \mathbb{F}^n$
// preprocessing
begin
  I runs MVP.I on $\mathbb{i}^{(1)} = (m, 3n+1, M)$
// online
begin
  P computes $f_{wl} \leftarrow \sum_{j=0}^{n-1} \mathbf{w_l}_{[j]} X^j$, $f_{wr} \leftarrow \sum_{j=0}^{n-1} \mathbf{w_r}_{[j]} X^j$, and $f_{wo} \leftarrow \sum_{j=0}^{n-1} \mathbf{w_o}_{[j]} X^j$
  P sends $f_{wl}(X)$, $f_{wr}(X)$, and $f_{wo}(X)$, all of degrees at most $n-1$, to V
  P computes $f_{1w}(X) \leftarrow 1 + X f_{wl}(X) + X^{n+1} f_{wr}(X) + X^{2n+1} f_{wo}(X)$
  P computes $f_x(X)$, whose coefficient vectors correspond to $\mathbf{x} = M\,(1|\mathbf{w_l}^\mathsf{T}|\mathbf{w_r}^\mathsf{T}|\mathbf{w_o}^\mathsf{T})^\mathsf{T}$
  P and V run MVP with $\mathbb{i}^{(1)} = (m, 3n+1, M)$, $\mathbb{x}^{(1)} = ([f_{1w}(X)], [f_x(X)])$, $\mathbb{w}^{(1)} = (f_{1w}(X), f_x(X))$ where V simulates $[f_{1w}(X)]$ using $f_{1w}(X) = 1 + X f_{wl}(X) + X^{n+1} f_{wr}(X) + X^{2n+1} f_{wo}(X)$, $[f_{wl}(X)]$, $[f_{wr}(X)]$, and $[f_{wo}(X)]$; and where V computes $[f_x(X)]$ locally using $\mathbf{x} = \mathbb{x}$
  P and V run Hadamard with $\mathbb{i}^{(2)} = n-1$, $\mathbb{x}^{(2)} = ([f_{wl}(X)], [f_{wr}(X)], [f_{wo}(X)])$, $\mathbb{w}^{(2)} = (f_{wl}(X), f_{wr}(X), f_{wo}(X))$

Protocol 6: Claymore

Theorem 6 (Security of Claymore). *Protocol* Claymore *of Protocol 6 is a Polynomial IOP for* $\mathcal{R}_{\text{HPR}}$ *with completeness and soundness error* $\sigma \leq \sigma_{\mathsf{Hadamard}} + \sigma_{\mathsf{MVP}}$.

Proof. Completeness follows from construction. Since the arguments are computed honestly, the subprotocols succeed and guarantee equalities (1) and (2), respectively.

Soundness. If the HPR instance is a false instance, then $\mathbf{x} \neq M(1|\mathbf{w_l}^\mathsf{T}|\mathbf{w_r}^\mathsf{T}|\mathbf{w_o}^\mathsf{T})^\mathsf{T}$ or $\mathbf{w_l} \circ \mathbf{w_r} \neq \mathbf{w_o}$. As a result either the Hadamard protocol succeeds despite being run on a false instance, or the MVP protocol succeeds despite being run on a false instance. The probabilities of these events are respectively at most $\sigma_{\mathsf{Hadamard}}$ and at most σ_{MVP}. □

5.2 The Role of Preprocessing

The preprocessing phase can be omitted. In this case, V must compute $f_M(X)$ locally. This task requires $O(mn)$ work, or only $O(K)$ if the matrix M has only K nonzero elements and is represented as such. When this phase is omitted, Claymore should be compared to the Polynomial IOP underlying Aurora [4].

When used with preprocessing, Claymore achieves fast verification. Specifically, the matrix M which determines the circuit being proved, is processed by the indexer. For long and drawn-out computations, this matrix is typically sparse and the SparseMVP is suitable. However, for short or shallow computations, DenseMVP is the better option. Depending on the choice of MVP protocol, the matching soundness error should be considered.

5.3 Optimizations

Reuse $\boldsymbol{\alpha}$ Across Hadamard Protocols. DenseClaymore has only one invocation of the Hadamard subprotocol, but the (partially unrolled) SparseClaymore has many more. It is worth reusing the same α for all these invocations as this reduces the number of unique evaluation points.

First, observe that all invocations to Hadamard can be shuffled around until they can all be run simultaneously – none of the inputs to any of the Hadamard protocols depend on the outputs of any other. Second, we can concatenate all the Hadamard relations and prove one batched relation

$$\boldsymbol{a}_0\|\boldsymbol{a}_1\|\cdots\|\boldsymbol{a}_{k-1} \circ \boldsymbol{b}_0\|\boldsymbol{b}_1\|\cdots\|\boldsymbol{b}_{k-1} = \boldsymbol{c}_0\|\boldsymbol{c}_1\|\cdots\|\boldsymbol{c}_{k-1} \tag{29}$$

instead of k individual relations separately. This batching comes with no soundness degradation.

The batched equation can be verified with k separate InnerProduct protocols that prove the same inner product relations as would be proved without batching – except that α is now the same everywhere. So neither P nor any other observer can determine whether V is verifying Equation 29 or k separate equations.

Batch the Inner Product Protocols. The unrolled SparseClaymore protocol consists of 10 invocations of InnerProduct protocol, and the unrolled DenseClaymore protocol consists of 2. We can replace these InnerProduct protocols with the

Batched InnerProduct protocol presented in Protocol 2. To see that this replacement does not affect the soundness, note that the InnerProduct subprotocols do not involve any verifier randomness and we can safely postpone them to the end of the Claymore protocol. Next, we replace them with a BatchedInnerProduct, unifying the degrees by the maximal degree of these polynomials. The negligible soundness degradation of this modification is captured concretely by Lemma 2 of Appendix A.

Batch the Sparse Vector Protocols. We also present an alternative version of SparseBiEval by batching the two instances of VandermondeVector and the two LagrangeVector protocols. This optimization eliminates four polynomial oracles at the cost of doubling the polynomial degrees. We present the protocol details and security proofs in Appendix C.2.

Concatenate Left and Right Wire Vectors. Instead of sending three witness polynomials $(f_{wl}(X), f_{wr}(X), f_{wo}(X))$, the prover can get away with sending only two: $(f_{wi}(X), f_{wo}(X))$ where $f_{wi}(X) = f_{wl}(X) + X^n \cdot f_{wr}(X)$. This concatenation is already implicit in the matrix-vector product subprotocol. The input to the Hadamard subprotocol should be $\mathbb{x} = ([f_{wi}(X)], [X^n \cdot f_{wi}(X)], [X^n \cdot f_{wo}(X)])$. The subprotocol then establishes that

$$\begin{pmatrix} \mathbf{w_l} \\ \hline \mathbf{w_r} \\ \hline \mathbf{0}_n \end{pmatrix} \circ \begin{pmatrix} \mathbf{0}_n \\ \hline \mathbf{w_l} \\ \hline \mathbf{w_r} \end{pmatrix} = \begin{pmatrix} \mathbf{0}_n \\ \hline \mathbf{w_o} \\ \hline \mathbf{0}_n \end{pmatrix} \quad , \tag{30}$$

which is clearly equivalent to the original Hadamard relation. With this technique, the polynomials are of degree $3n - 1$, and so the soundness error is $(9n - 3)/(|\mathbb{F}| - 1)$ instead of $(3n - 3)/(|\mathbb{F}| - 1)$.

This optimization also preserves zero knowledge. To see this, observe that any distinguisher D that uses $f_{wi}(X)$ can be simulated with a distinguisher D' that uses $f_{wl}(X)$ and $f_{wr}(X)$. As a result, the optimized protocol lacks zero knowledge only if the protocol before applying the optimization also lacks it.

6 Zero-Knowledge

The strategy for achieving zero-knowledge consists of appending $3q$ coefficients to the initial wire vectors $\mathbf{w_l}$, $\mathbf{w_r}$, and $\mathbf{w_o}$ such that each new vector has q uniformly random coefficients and such that their Hadamard relation remains. The randomizers will make the witness polynomials q-wise independent, meaning that no distinguisher restricted to at most q queries will obtain any information about the witness.

It is tricky to define zero-knowledge the context of Polynomial IOPs. The distinguisher D can always query the received oracles in enough points to interpolate and then extract the witness. The notion is only meaningful when the

description: realizes $\mathcal{R}_{\text{HPR}}$
inputs: $\mathbb{i}$: (m, n, M) with $M \in \mathbb{F}^{m \times n}$
$\mathbb{x}$: $\mathbf{x} \in \mathbb{F}^n$
$\mathbb{w}$: $(\mathbf{w_l}, \mathbf{w_r}, \mathbf{w_o}) \in \mathbb{F}^n \times \mathbb{F}^n \times \mathbb{F}^n$
additional parameters: q
offline preprocessing:
P runs Claymore.I on $\mathbb{i}^{(1)} = (m, n+3q, M' = \left(M_{[:,0:(n+1)]} \middle| 0_{m\times 3q} \middle| M_{[:,(n+1):(2n+1)]} \middle| 0_{m\times 3q} \middle| M_{[:,(2n+1):(3n+1)]} \middle| 0_{m\times 3q}\right))$
online phase:
// compute witness polynomial with randomizers
P samples $\mathbf{r}^{(l)}_{[0:q]}, \mathbf{r}^{(r)}_{[q:2q]}, \mathbf{r}^{(o)}_{[2q:3q]} \xleftarrow{\$} \mathbb{F}^q$ and sets $\mathbf{r}^{(l)}_{[q:2q]} = \mathbf{r}^{(r)}_{[0:q]} = 0_{q\times 1}$, $\mathbf{r}^{(o)}_{[0:2q]} = 0_{2q\times 1}$, $\mathbf{r}^{(l)}_{[2q:3q]} = 1_{q\times 1}$, and $\mathbf{r}^{(r)}_{[2q:3q]} = \mathbf{r}^{(o)}_{[4:6]}$ *// $\mathbf{r}^{(l)} \circ \mathbf{r}^{(r)} = \mathbf{r}^{(o)}$*
P and V run Claymore with $\mathbb{i}^{(1)} = (m, n+3q, M')$, $\mathbb{x}^{(1)} = \mathbf{x}$, $\mathbb{w}^{(1)} = ((\mathbf{w_l}^\mathsf{T} | \mathbf{r}^{(l)\,\mathsf{T}}), (\mathbf{w_r}^\mathsf{T} | \mathbf{r}^{(r)\,\mathsf{T}}), (\mathbf{w_o}^\mathsf{T} | \mathbf{r}^{(o)\,\mathsf{T}}))$

Protocol 7: ZKClaymore

number of queries bounded by some parameter. We furthermore restrict the distinguisher's queries to be distributed identically to that of an honest verifier; this restriction therefore corresponds to *honest-verifier* zero knowledge. As a result, we are not concerned with finding a complete description of the polynomials that make up the transcript. Instead, we are only concerned with the verifier's view of the transcript. This view corresponds to the list of queries and responses to the various oracles.

Theorem 7. *When $q \geq 2$, the Polynomial IOP* ZKClaymore *of protocol 7 has statistical honest-verifier zero-knowledge if all the* InnerProduct *subprotocols are replaced by a single invocation of* BatchedInnerProduct*. Concretely, the statistical distance between the verifier's view of authentic transcript versus the verifier's view of simulated transcript is bounded by $\frac{3}{|\mathbb{F}|-1}$, which is negligible in the field size.*

Proof. We show how S produces the verifier view for $(\mathbb{i}, \mathbb{x})$ without knowledge of $\mathbb{w}$. In the process, we establish that this view is indistinguishable from that of an authentic protocol execution.

The protocol ZKClaymore consists of an invocation to Hadamard protocol and an invocation to either the dense or sparse variant of MVP. Note that both protocols DenseMVP and SparseMVP consists of:

1. a query to $f_x(X)$ at uniformly random $\alpha \xleftarrow{\$} \mathbb{F}\backslash\{0\}$;
2. a protocol invocation (ModReduce in DenseMVP, or SparseBiEval in SparseMVP) with inputs that are independent of $\mathbf{w_l}, \mathbf{w_r}, \mathbf{w_o}$;
3. an invocation of InnerProduct on input $f_{1w}(X)$ and another polynomial ($r(X)$ in DenseMVP or $f_{\boldsymbol{\alpha}^\intercal M}(X)$ in SparseMVP), denoted by $f_t(X)$ hereafter, that is also independent of $\mathbf{w_l}, \mathbf{w_r}, \mathbf{w_o}$.

Since S knows M and $\mathbf{x}$, S can compute all polynomials that do not depend on witnesses honestly, *i.e.*, as the honest P would. We therefore restrict attention to polynomials that depend on the witness.

What remains is to demonstrate how to sample the verifier view for InnerProduct on input $f_{1w}(X)$ and $f_t(X)$, and for Hadamard on input $f_{wl}(X)$, $f_{wr}(X)$ and $f_{wo}(X)$. These two subprotocols contribute two polynomial pairs each to the BatchedInnerProduct protocol. It suffices to sample the verifier view for the BatchedInnerProduct protocol just for these two polynomial pairs, because the remaining pairs are independent of the witness.

This verifier view consists of several elements, namely:

1. Uniformly random z, α^* (the symbol α^* is used for batching the various inner product relations into one)
2. $y_h = \bar{h}(z), y_h^* = \bar{h}(\gamma \cdot z)$
3. The verifier view contributed by the InnerProduct protocol in MVP:
   (a) $y_{l_1} = f_{wl}(z)$
   (b) $y_{r_1} = f_{wr}(z)$
   (c) $y_{o_1} = f_{wo}(z)$
   (d) $y_t = f_t(z^{-1})$ (S samples this one honestly)
4. The verifier view contributed by the top-level Hadamard protocol:
   (a) Uniformly random β
   (b) $y_{o_2} = f_{wo}(\beta)$
   (c) $y_{l_2} = f_{wl}(\beta z)$
   (d) $y_{r_2} = f_{wr}(z^{-1})$

In the verifier view of an honest run, the above values satisfy:

$$\begin{aligned} y_h \cdot \gamma^{3n+3q} - y_h^* = \quad & (y_{1w} \cdot y_t - f_x(\alpha)) \cdot z^{3n+3q} \\ & + \alpha^* \cdot \left((y_{l_2} \cdot y_{r_2} - y_{o_2}) \cdot z^{n+q-1} \right) \\ & + \alpha^{*2} \cdot (\cdots) \ , \end{aligned} \tag{31}$$

where the ellipses omit terms that are independent of the witness and thus already known to S, and where $y_{1w} = 1 + z^{-1} y_{l_1} + z^{-n-q-1} y_{r_1} + z^{-2n-2q-1} y_{o_1}$.

S samples uniformly random $\alpha^*, z, \beta \xleftarrow{\$} \mathbb{F} \backslash \{0\}$ and computes y_t honestly.

Consider the matrices

$$Z_l = \begin{pmatrix} 1 & z & z^2 & \cdots & z^{n+3q-1} \\ 1 & \beta z & (\beta z)^2 & \cdots & (\beta z)^{n+3q-1} \end{pmatrix} \tag{32}$$

$$Z_r = \begin{pmatrix} 1 & z & z^2 & \cdots & z^{n+3q-1} \\ 1 & z^{-1} & z^{-2} & \cdots & z^{-n-3q+1} \end{pmatrix} \tag{33}$$

$$Z_o = \begin{pmatrix} 1 & z & z^2 & \cdots & z^{n+3q-1} \\ 1 & \beta & \beta^2 & \cdots & \beta^{n+3q-1} \end{pmatrix} \tag{34}$$

which satisfy $(y_{l_1}, y_{l_2})^\mathsf{T} = Z_l \, (\mathbf{w_l}^\mathsf{T} | \mathbf{r}^{(l)\,\mathsf{T}})^\mathsf{T}$, $(y_{r_1}, y_{r_2})^\mathsf{T} = Z_r \, (\mathbf{w_r}^\mathsf{T} | \mathbf{r}^{(r)\,\mathsf{T}})^\mathsf{T}$, and $(y_{o_1}, y_{o_2})^\mathsf{T} = Z_o \, (\mathbf{w_o}^\mathsf{T} | \mathbf{r}^{(o)\,\mathsf{T}})^\mathsf{T}$. Capture the relation between polynomials' values

and randomizers into a single equation:

$$\begin{pmatrix} y_{l_1} \\ y_{l_2} \\ y_{r_1} \\ y_{r_2} \\ y_{o_1} \\ y_{o_2} \end{pmatrix} = \begin{pmatrix} z^n & z^{n+1} & 0 & 0 & 0 & 0 \\ (\beta z)^n & (\beta z)^{n+1} & 0 & 0 & 0 & 0 \\ 0 & 0 & z^{n+q} & z^{n+q+1} & z^{n+2q} & z^{n+2q+1} \\ 0 & 0 & z^{-n-q} & z^{-n-q-1} & z^{-n-2q} & z^{-n-2q-1} \\ 0 & 0 & 0 & 0 & z^{n+2q} & z^{n+2q+1} \\ 0 & 0 & 0 & 0 & \beta^{n+2q} & \beta^{n+2q+1} \end{pmatrix} \begin{pmatrix} \mathbf{r}^{(l)}_{[0]} \\ \mathbf{r}^{(l)}_{[1]} \\ \mathbf{r}^{(r)}_{[q]} \\ \mathbf{r}^{(r)}_{[q+1]} \\ \mathbf{r}^{(o)}_{[2q]} \\ \mathbf{r}^{(o)}_{[2q+1]} \end{pmatrix} + \mathbf{c} \ , \tag{35}$$

where $\mathbf{c} \in \mathbb{F}^6$ is a constant vector independent of the randomizers.

Let $Z_l^\star, Z_r^\star, Z_o^\star$ be the 2×2 submatrices of Z_l, Z_r, Z_o whose columns are multiplied by $\mathbf{r}^{(l)}_{[0:2]}, \mathbf{r}^{(r)}_{[q:q+2]}$ and $\mathbf{r}^{(o)}_{[2q:2q+2]}$, respectively; or equivalently, the 2×2 submatrices on the diagonal of Equation 35. If these 2×2 submatrices are all invertible, then $(y_{l_1}, y_{l_2}, y_{r_1}, y_{r_2}, y_{o_1}, y_{o_2})$ are uniform because the randomizers $\mathbf{r}^{(l)}_{[0:q]}, \mathbf{r}^{(r)}_{[q:2q]}, \mathbf{r}^{(o)}_{[2q:3q]}$ are. So S samples $(y_{l_1}, y_{l_2}, y_{r_1}, y_{r_2}, y_{o_1}, y_{o_2})$ uniformly at random from $\mathbb{F}^6$.

The 2×2 submatrices of $Z_l^\star, Z_r^\star, Z_o^\star$ are not all invertible if $z = 1$, if $\beta = 1$, or if $\beta = z$. The probability of this event is $3/(|\mathbb{F}| - 1)$.

Since we can solve for one of y_h, y_h^* given the other, we only need to show that y_h is uniformly random over $\mathbb{F}$. This is where the convenient arbitrary coefficient $\bar{h}_{\mathsf{d}}$ of $\bar{h}(X)$ comes into play. This coefficient is chosen uniformly at random, and so S samples $y_h \xleftarrow{\$} \mathbb{F}$. Lastly, S solves Eqn. (31) for y_h^*.

To complete the argument, except with a negligible failure probability corresponding to the 2×2 submatrices $Z_l^\star$, $Z_r^\star$, $Z_o^\star$ being singular, S samples a verifier view from a distribution that is identical to the distribution of verifier views of an authentic protocol execution. The distinguishing advantage of any distinguisher D is bounded by the S's failure probability, which is $3/(|\mathbb{F}| - 1)$. This number also bounds the statistical distance in distributions of the view of the verifier of authentic transcripts versus simulated transcripts. □

7 Comparison

7.1 Abstract Comparison

We compare both variants of Claymore to some other Polynomial IOPs from the literature, namely Sonic, PLONK, Marlin, and Aurora. Of these Polynomial IOPs, the first three give rise to SNARKs after cryptographic compilation. In contrast, Aurora gives rise to a proof system generating short proofs but whose verifier complexity is linear in the size of the witness. Importantly, Claymore is comparable to both types of proof system: with preprocessing, it gives rise to a SNARK; when preprocessing is omitted, the proofs remain short at the expense of linear verifier complexity.

Table 1 contains an overview of the comparison. It considers the following key performance indicators for Polynomial IOPs.

– The number of polynomials sent by I during the offline preprocessing phase. This number determines the size of the universal or structured reference strings. While this number contributes to the complexity of I, this complexity is generally speaking not a make or break factor.
– The number of polynomials sent by P during the online proving phase. This number contributes to the size of the proof and to the complexity of both P and V.
– The number of evaluations. This number contributes to the size of the proof, as well as indirectly to the complexity of P and V.
– The number of distinct points for evaluation. Some cryptographic compilers (*e.g.*, [6]) enable the merger of two polynomial evaluations provided that they are being evaluated in the same point. This number limits the number of times this optimization can be applied.
– The maximum degree of all polynomials. This number contributes to indexer and prover complexity in two ways. First, before cryptographic compilation, I and P operate on polynomials of this degree and their complexity is affected accordingly. The exception is if the polynomials are sparse, or otherwise exhibit a structure that enables fast computation. Second, some cryptographic compilers induce overheads that are superlinear in this degree.

Table 1. Comparison between Claymore and other Polynomial IOPs from the literature, with respect to key performance indicators.

	# polynomials offline / online	# evaluations	# distinct points	max. degree
Sonic [11]	$12M/3M+7$	$11M+3$	$9M+2$	$O(n)$
PLONK [8]	8 / 6	7	2	$12(n+a)$
Marlin [7]	9 / 12	18	3	$6k+6$
Aurora [4]	- / 7	8	2	$\mathsf{max}(m,n)$
DenseClaymore	1 / 4	10	6	$m(3n+1)-1$
SparseClaymore	8 / 10	30	10	$3K-1$

For Sonic, n refers to the number of multiplication gates and the degree of the largest polynomial is $7n$. However, due to their technique for simulating bivariate polynomials, the addition gates have fan-in bounded by a parameter M, corresponding to having at most M nonzero elements in every row of the linear transform matrix. The same bound applies to the number of nonzero elements in every column. As a result of converting the original circuit into one with this property, a number of multiplication gates may have to be added, thus explaining the Landau notation.

For PLONK, n refers to the number of multiplication gates and a refers to the number of addition gates, all of which have fan-in 2. We note that there is a variant of PLONK with larger proofs and smaller prover time, which is not shown in the table.

Aurora does not have a preprocessing phase and as a result the verifier's complexity is linear in the number of nonzero elements in the matrices A, B, C from the R1CS tuple. Marlin uses the same mechanics but uses preprocessing to shrink the verifier's workload for the matrix multiplication; this technique requires 9 polynomials in the uniform or structured reference string (3 per matrix) and a few more in the online protocol. The parameter k denotes the largest number of nonzero elements of $\{A, B, C\}$.

For Claymore the linear transform is either represented densely as an $m \times n$ matrix, or sparsely as a list of K nonzero coefficients in this matrix.

Marlin, PLONK, and Aurora work in the Reed-Solomon codeword basis and crucially rely on the structure of the field or of its multiplicative group. In contrast, Sonic and Claymore work for any field.

7.2 Concrete Comparison

To make the comparison more concrete, we compile the various Polynomial IOPs into concrete SNARKs with various compilers. In the following we use P to denote the number of polynomials, Q for the number of queries, U for the number of unique points, $|\mathbb{F}|$ for the size of a field element, and $|\mathbb{G}|$ for the size of a group element.

For the benchmark computation we choose the following: to prove the membership of an element in a set by verifying the Merkle tree authentication path in zero knowledge. The set holds 1024 elements and its Merkle root is known, as is the member element. The witness consists of the element's position in the tree, and the authentication path. The Merkle tree is constructed using the zero-knowledge-proof-friendly Rescue-Prime hash function [12] with $m = 3$ state elements, rate equal to $r = 2$, over a prime field with $p > 2^{256}$ elements, with $N = 18$ rounds, targeting a security level of $\lambda = 128$ bits against collisions.

After arithmetization, this computation can be cast into one of three constraint systems for arithmetic circuits.

- The computation can be represented as a PLONK-relation, in which case there are $a = 13021$ additions, $m = 17101$ multiplications, and 31234 distinct wires in total.
- The computation can be represented as a Hadamard Product Relation (HPR) with $n = 4631$ multiplications and $m = 7571$ linear relations with arbitrary fan-in. After coercing the constraint system to one whose matrix has at most $M = 3$ nonzero elements on every row and on every column, there are $n = 23850$ multiplications and $m = 26790$ linear relations.
- The computation can be represented as a Rank-1 Constraint Satisfaction System (R1CS) with 12202 rank-1 constraints and no more than $k = 38694$ nonzero elements in $\{A, B, C\}$.

The various cryptographic compilers differ in how they simulate evaluation queries. A polynomial oracle $[f(X)]$ is represented by a cryptographic commitment. When V queries it in z and obtains the response y, one option is to simply

run the polynomial commitment's evaluation protocol to prove that $f(z) = y$. Another option is for V to use the commitment to $f(X)$ to derive the commitment to $\frac{f(X)-y}{X-z}$, and for P to prove that this polynomial has an appropriately bounded degree. The point is that degree bound checks might be simpler to combine than evaluation queries, depending on the nature of the polynomial commitment.

- KZG polynomial commitments. A commitment to a polynomial $f(X)$ is a single group element. When V queries its value in z, P responds with its value $y = f(z)$ along with another group element, a commitment to $\frac{f(X)-y}{X-z}$. Note that the zerofier $X-z$ must divide $f(X)-f(z)$. One pairing evaluation allows V to verify the correct relation between the received commitments. Evaluation queries in the same point z but to different polynomials can be batched using random weights supplied by V. However, the scheme does not support batching evaluations in distinct points, at least in terms of communication cost. To establish the proper degree bounds, P must supply a commitment to a *degree bound check polynomial* $F(X) = \sum_i \omega^{2i} \cdot f_i(X) + \omega^{2i+1} \cdot X^{\mathsf{d}-\delta(i)} \cdot f_i(X)$ where the sum ranges over all prior polynomials $f_i(X)$, where $\delta(i)$ is its degree bound, and where ω is a random challenge supplied by V. One more batch-evaluation is necessary to establish that the commitment to $F(X)$ is well formed. So the total size of a Polynomial IOP compiled with this KZG-based compiler is

$$(P+1) \times |\mathbb{G}| + (U+1) \times (|\mathbb{F}| + |\mathbb{G}|) \ . \tag{36}$$

  Using the BLS-384 pairing group, we can represent group elements with $|\mathbb{G}| = 385$ bits and scalar field elements with $|\mathbb{F}| = 256$ bits.
- DARK polynomial commitments. Evaluation queries to the same polynomial in distinct points can be batched, and evaluation queries to distinct polynomials in the same point can be batched. Evaluation queries to distinct polynomials in distinct points cannot be batched.[2] We choose to batch all queries into one evaluation proof for each polynomial, as opposed to batching all polynomials for each distinct point. Every evaluation protocol consists of at most $\lfloor \log_2 \mathsf{d} \rfloor$ rounds where d is the global degree bound. In every round except the last, P sends two group elements and at most U field elements, where U is the number of unique evaluation points. In the last round, P sends an integer of $\lfloor \log_2 \mathsf{d} \rfloor \times \lambda + |\mathbb{F}|$ bits. So the total size of a Polynomial

[2] There is a natural method for this type of batching that relies on dividing out the zerofier. In response to queries z_1 to $[f(X)]$ and z_2 to $[g(X)]$, P supplies $y_1 = f(z_1), y_2 = g(z_2)$ and commitments to the polynomials $\frac{f(X)-y_1}{X-z_1}$ and $\frac{g(X)-y_2}{X-z_2}$. At this point, V can verify that the new commitments are correctly related to the old, and he can use all commitments to derive a new commitment to "zero" polynomial. Precisely speaking, this commitment is to an integer polynomial whose coefficients are multiples of p. Then *one* evaluation protocol suffices to establish that $y_1 = f(z_1)$ and $y_2 = g(z_2)$. This batching method has not been formally analyzed and such an analysis is out of the scope of this paper.

IOP proof compiled with this DARK strategy is

$$P \times \lfloor \log_2 \mathsf{d} \rfloor \times (2|\mathbb{G}| + U|\mathbb{F}|) + \lfloor \log_2 \mathsf{d} \rfloor \times \lambda + |\mathbb{F}| \ . \tag{37}$$

Concretely we use 2000 bit class group elements, so $|\mathbb{G}| = 2000$. The field elements are integers modulo p, a $2\lambda = 256$ bit prime, so $|\mathbb{F}| = \lfloor \log_2 p \rfloor = 256$.

– FRI polynomial commitments. A FRI commitment is a Merkle root of a Reed-Solomon codeword, obtained by evaluating the polynomial in a domain that is ρ times larger than its degree, where ρ is known as the expansion factor. The FRI protocol establishes the bounded degree by opening Merkle leafs. As a result, the technique for dividing out the zerofier applies even without supplying a new commitment. The Reed-Solomon codeword of the polynomial $\frac{f(X)-y}{X-z}$ can be computed given the Reed-Solomon codeword of the polynomial $f(X)$. So every query generates one field element. The FRI protocol is run on the degree bound check polynomial $F(X) = \sum_i \omega^{2i} \cdot f_i(X) + \omega^{2i+1} \cdot X^{\mathsf{d}-\delta(i)} \cdot f_i(X)$ where the sum ranges over all polynomials after dividing out the zerofiers. In this expression, $\delta(i)$ is its updated degree bound, *i.e.*, after dividing out the zerofiers, and ω is a random challenge supplied by V. The evaluations of this degree bound check polynomial on the codeword domain can be computed from evaluations of the constituent polynomials. The FRI protocol consists of $\lfloor \log_2 \mathsf{d} \rfloor$ rounds where d is the global degree bound. In every round, P supplies a new Merkle root along with s field elements and as many authentication paths of length at most $\lfloor \log_2 \rho \cdot \mathsf{d} \rfloor$. The protocol bounds the polynomial's degree with soundness error conjecturally approximately ρ^{-s} when $\mathbb{F}$ is large enough. So the total size of a Polynomial IOP compiled with FRI is less than

$$P \times 2 \cdot \lambda + Q \times |\mathbb{F}| + \lfloor \log_2 \mathsf{d} \rfloor \times s \times (\lfloor \log_2 \rho \cdot \mathsf{d} \rfloor \times 2 \cdot \lambda + |\mathbb{F}|) \ . \tag{38}$$

Concretely, we set $\rho = 16$, $s = 32$, and use $|\mathbb{F}| = 256$. Note that the hash function with which the FRI Merkle trees are built, must have at least $2 \cdot \lambda = 256$ bit outputs.

Table 2 summarizes the results. It shows the size of the SNARK in bytes obtained from the given Polynomial IOP and compiled with the given compiler. The computation whose integrity is proved, is the Merkle tree membership relation described above. The source code for reproducing these numbers is available at https://github.com/aszepieniec/claymore-benchmark.

8 Conclusion

The protocols proposed in this paper challenge the notion that the Reed-Solomon codeword basis is the appropriate basis for representing objects from the arithmetic constraint system in a Polynomial IOP. Instead, the monomial coefficient basis provides a natural and intuitive representation for these objects. In this basis, the native operations on polynomials are identifiable with the matrix operations in the arithmetic constraint system. Moreover, this basis does not impose

Table 2. Comparison of concrete SNARK size.

	KZG	DARK	FRI
Sonic $P = 16$, $Q = 36$, $U = 20$, $M = 3$, $\mathsf{d} = 166950$	3 222	388 720	384 640
PLONK $P = 6$, $Q = 7$, $U = 2$, $\mathsf{d} = 361464$	578	61 232	424 352
Marlin $P = 12$, $Q = 18$, $U = 3$, $\mathsf{d} = 232170$	947	121 888	383 936
DenseClaymore $P = 4$, $Q = 10$, $U = 6$, $\mathsf{d} = 105327751$	802	72 416	825 792
SparseClaymore $P = 10$, $Q = 30$, $U = 10$, $\mathsf{d} = 158951$	1 411	139 704	384 256

any restrictions on the structure of the field. The resulting Polynomial IOP for arithmetic constraint systems outperforms similar constructions based on the Reed-Solomon codeword basis, at least as far as the number of polynomials in the transcript is concerned.

The modular approach followed in this paper admits a piece by piece presentation and analysis that benefits simplicity and accessibility. Nevertheless, some optimizations violate the boundaries implicit in this modular structure. These optimizations are of independent interest as they may also apply elsewhere; perhaps they have straightforward analogues in the Reed-Solomon codeword domain. Some of the more important optimizations are summarized as follows.

- In some cases it is possible to eliminate polynomials. In particular, when a polynomial is queried exactly twice and is involved in exactly two polynomial identities. By moving this polynomial to the left hand side, and equating the right hand sides, the polynomial identities are merged and the polynomial in question is eliminated, all without impacting soundness.
- After unrolling a Polynomial IOP, InnerProduct subprotocols appear in great numbers and many places. They can all be batched. In addition to saving polynomials, this batching facilitates a simpler and more direct proof of zero-knowledge that would not be possible otherwise.
- Batching may apply in more places still. For instance, the sparse MVP procedure benefits from two invocations of a subprotocol that establishes that a given vector is a Lagrange vector, and two more that another vector is a Vandermonde vector. The Lagrange and Vandermonde vector protocols can be merged in order to save polynomials. In fact, this merger extends to multivariate polynomials in more than two variables.
- Concatenating the vectors of left and right wires saves one polynomial, but only if the Hadamard subprotocol can be made to work with the concatenated vector. In particular, this adaptation requires shifting the protocol's second and third arguments. Performing this shift in either basis is possible, but this shift highlights an interesting difference. In the Reed-Solomon codeword basis, the query to the polynomial oracle is multiplied by a constant factor; in

the monomial coefficient basis, however, it is the response that is multiplied by a factor that depends on the query.

One of the questions that led to the present line of research was to find the Polynomial IOP with the smallest possible number of polynomials in the transcript. In this respect, we report a mitigated success: DenseClaymore has only four polynomials, down 33% from runner-up PLONK; and when restricting to polynomials whose degrees grow linearly with the size of the circuit, then SparseClaymore achieves two polynomials less than Marlin, the only competitor that also admits arbitrary fan-in for linear gates.

However, the concrete comparison shows that we were optimizing the wrong metric. Contrary to our expectation, the number of polynomials in the transcript is *not* the dominant factor of proof size. Indeed, this comparison highlights the importance of balancing a Polynomial IOP's number of polynomials against its other parameters.

Acknowledgements. Both authors are supported by the Nervos Foundation.

References

1. Ames, S., Hazay, C., Ishai, Y., Venkitasubramaniam, M.: Ligero: Lightweight Sublinear Arguments Without a Trusted Setup. In: Thuraisingham, B.M., Evans, D., Malkin, T., Xu, D. (eds.) ACM CCS 2017. pp. 2087–2104. ACM (2017), https://doi.org/10.1145/3133956.3134104
2. Bellare, M., Goldreich, O.: On defining proofs of knowledge. In: Brickell, E.F. (ed.) CRYPTO 1992s. Lecture Notes in Computer Science, vol. 740, pp. 390–420. Springer (1992), https://doi.org/10.1007/3-540-48071-4_28
3. Ben-Sasson, E., Bentov, I., Horesh, Y., Riabzev, M.: Scalable zero knowledge with no trusted setup. In: Boldyreva, A., Micciancio, D. (eds.) CRYPTO 2019, Part III. Lecture Notes in Computer Science, vol. 11694, pp. 701–732. Springer (2019). https://doi.org/10.1007/978-3-030-26954-8_23
4. Ben-Sasson, E., Chiesa, A., Riabzev, M., Spooner, N., Virza, M., Ward, N.P.: Aurora: Transparent Succinct Arguments for R1CS. In: Ishai, Y., Rijmen, V. (eds.) EUROCRYPT 2019 , Part I. Lecture Notes in Computer Science, vol. 11476, pp. 103–128. Springer (2019), https://doi.org/10.1007/978-3-030-17653-2_4
5. Bootle, J., Cerulli, A., Chaidos, P., Groth, J., Petit, C.: Efficient Zero-Knowledge Arguments for Arithmetic Circuits in the Discrete Log Setting. In: Fischlin, M., Coron, J. (eds.) EUROCRYPT 2016, Part II. Lecture Notes in Computer Science, vol. 9666, pp. 327–357. Springer (2016), https://doi.org/10.1007/978-3-662-49896-5_12
6. Bünz, B., Fisch, B., Szepieniec, A.: Transparent SNARKs from DARK Compilers. In: Canteaut, A., Ishai, Y. (eds.) EUROCRYPT 2020 Part I. Lecture Notes in Computer Science, vol. 12105, pp. 677–706. Springer (2020), https://eprint.iacr.org/2019/1229.pdf
7. Chiesa, A., Hu, Y., Maller, M., Mishra, P., Vesely, N., Ward, N.P.: Marlin: Preprocessing zksnarks with universal and updatable SRS. In: Canteaut, A., Ishai, Y. (eds.) EUROCRYPT 2020 Part I. Lecture Notes in Computer Science, vol. 12105, pp. 738–768. Springer (2020), https://eprint.iacr.org/2019/1047.pdf

8. Gabizon, A., Williamson, Z.J., Ciobotaru, O.: PLONK: Permutations over Lagrange-bases for Oecumenical Noninteractive arguments of Knowledge. IACR Cryptology ePrint Archive **2019**, 953 (2019), https://eprint.iacr.org/2019/953
9. Goldwasser, S., Micali, S., Rackoff, C.: The knowledge complexity of interactive proof-systems (extended abstract). In: Sedgewick, R. (ed.) ACM STOC. pp. 291–304. ACM (1985). https://doi.org/10.1145/22145.22178
10. Kate, A., Zaverucha, G.M., Goldberg, I.: Constant-size commitments to polynomials and their applications. In: Abe, M. (ed.) ASIACRYPT 2010. Lecture Notes in Computer Science, vol. 6477, pp. 177–194. Springer (2010), https://doi.org/10.1007/978-3-642-17373-8_11
11. Maller, M., Bowe, S., Kohlweiss, M., Meiklejohn, S.: Sonic: Zero-knowledge snarks from linear-size universal and updatable structured reference strings. In: Cavallaro, L., Kinder, J., Wang, X., Katz, J. (eds.) ACM CCS 2019. pp. 2111–2128. ACM (2019), https://eprint.iacr.org/2019/099.pdf
12. Szepieniec, A., Ashur, T., Dhooghe, S.: Rescue-Prime: a standard specification (sok). IACR Cryptology ePrint Archive **2020**, 1143 (2020), https://eprint.iacr.org/2020/1143

A Definitional Equivalence: Polynomial IOPs with and without Individual Degree Bounds

Definition 4 presents one possible definition for the notion that is Polynomial IOP, namely where there is one degree bound d that applies to all polynomials. However, all the protocols presented in this paper provide individual degree bounds for each polynomial, corresponding to another possible definition. The discussion following Definition 4 argues that no generality is lost by switching between the two options. We now formalize this intuition.

Definition 6 (Polynomial IOP with Individual Degree Bounds). *Let $\mathcal{R}$ be an indexed relation with corresponding indexed language $\mathcal{L}(\mathcal{R})$, $\mathbb{F}$ some finite field, and $\delta : \mathbb{N} \to \mathbb{N}$ a function that maps polynomial indices to their respective degree bounds. A* Polynomial IOP for $\mathcal{R}$ with individual degree bounds δ *is a pair of interactive machines* (P, V)*, satisfying the following description.*

- (P, V) *is an interactive proof for $\mathcal{L}(\mathcal{R})$ with r rounds, and with soundness error σ.*
- P *sends polynomials $f_i(X) \in \mathbb{F}[X]$ of degree at most $\delta(i)$ to V.*
- V *is an oracle machine with access to a list of oracles, which contains one oracle for each polynomial it has received from the prover.*
- *When an oracle associated with a polynomial $f_i(X)$ is queried on a point $z_j \in \mathbb{F}$, the oracle responds with the value $f_i(z_j)$.*
- V *sends challenges $\alpha_k \in \mathbb{F}$ to P.*
- V *is public coin.*

Showing the equivalence in one direction turns out to be trivial. The next lemma establishes that a Polynomial IOP with individual degree bounds (6) can simulate a Polynomial IOP with a single degree bound (4).

Lemma 1 ((4) ⇒ (6)). *Let* (P, V) *be a Polynomial IOP with degree bound* d *for a relation* $\mathcal{R}$ *with soundness error* σ *according to Definition 4. Then* (P, V) *is a Polynomial IOP with individual degree bounds* $\delta(i) = \mathsf{d}$ *for* $\mathcal{R}$ *with soundness error* σ *according to Definition 6.*

Proof. The difference between the two definitions is that the ith polynomial $f_i(X)$ is guaranteed to be of degree at most $\delta(i)$ rather than d. In this particular case, $\delta(i) = \mathsf{d}$, so there is no difference at all. □

The other direction of the equivalence is more involved as it requires a compiler and comes with a (negligible) soundness degradation as well as one extra polynomial oracle. Let $\mathsf{C}(\mathsf{P})$ and $\mathsf{C}(\mathsf{V})$ denote the compiled prover and verifier, respectively. The compiler starts by computing $\mathsf{d} = \mathsf{max}_i \delta(i)$. The compiled prover and verifier then mimick the original prover and verifier, and finalize with the following steps:

- $\mathsf{C}(\mathsf{V})$ sends a random scalar $\omega \xleftarrow{\$} \mathbb{F}\backslash\{0\}$.
- $\mathsf{C}(\mathsf{P})$ sends a single polynomial called the *degree bound check polynomial* $F(X) = \sum_i \left(\omega^{2i} \cdot f_i(X) + \omega^{2i+1} \cdot X^{\mathsf{d}-\delta(i)} \cdot f_i(X)\right)$ of degree at most d.
- $\mathsf{C}(\mathsf{V})$ samples all polynomial oracles in a random point $z \xleftarrow{\$} \mathbb{F}\backslash\{0\}$ and tests $F(z) \stackrel{?}{=} \sum_i \left(\omega^{2i} \cdot f_i(z) + \omega^{2i+1} \cdot z^{\mathsf{d}-\delta(i)} \cdot f_i(z)\right)$.

Lemma 2 ((6) ⇒ (4)). *Let* (P, V) *be a Polynomial IOP with individual degree bounds* $\delta(i)$ *for a relation* $\mathcal{R}$ *with soundness error* σ *and* q *queries to* p *polynomials in* t *distinct points from* $\mathbb{F}\backslash\{0\}$ *according to Definition 6. Then* $(\mathsf{C}(\mathsf{P}), \mathsf{C}(\mathsf{V}))$ *is a Polynomial IOP with global degree bound* $\mathsf{d} = \mathsf{max}_i \delta(i)$ *for* $\mathcal{R}$ *with soundness error at most* $\sigma + \frac{2p+\mathsf{d}-1}{|\mathbb{F}|-1}$ *and* $q + p + 1$ *queries to* $p + 1$ *polynomials in* $t + 1$ *distinct points from* $\mathbb{F}\backslash\{0\}$ *according to Definition 4.*

Proof. The numbers of polynomials, queries, and unique points are trivial. Completeness follows from construction. The degree of $F(X)$ is at most $\mathsf{d} = \max_i \delta(i)$ and this polynomial is tested against its definition.

In terms of soundness, we distinguish two cases. Case one: all the polynomials $f_i(X)$ satisfy their individual degree bounds. The $\mathsf{C}(\mathsf{V})$ accepts a false instance with probability σ. Case two: some or all of the polynomials $f_i(X)$ do not satisfy their individual degree bounds bounds. In this case, there is a Y for which the equation

$$F(X, Y) = \sum_i \left(Y^{2i} \cdot f_i(X) + Y^{2i+1} \cdot X^{\mathsf{d}-\delta(i)} \cdot f_i(X)\right) \tag{39}$$

is not a univariate polynomial identity. Therefore, Equation 39 is not a bivariate polynomial identity either. There are at most $2p - 1 + \mathsf{d}$ points (X, Y) where left and right hand sides are equal. By the (two-dimensional) Schwartz-Zippel lemma, the probability of sampling one of these points is $\frac{2p+\mathsf{d}-1}{|\mathbb{F}|-1}$. By the union bound, the probability of false accept in either case is bounded by $\sigma + \frac{2p+\mathsf{d}-1}{|\mathbb{F}|-1}$. □

B Sparse Matrix-Vector Product

B.1 Formal Presentation

The formal presentation of the sparse matrix-vector multiplication protocol, which follows, builds the protocol hierarchically. We therefore follow the high-level overview but in reverse order.

Vandermonde Vector. The VandermondeVector protocol realizes the following relation

$$\mathcal{R}_{\text{vv}} = \left\{ (\mathbb{i}, \mathbb{x}, \mathbb{w}) \;\middle|\; \begin{array}{l} \mathbb{i} = (H, K, \varphi(h), \{a_k\}_{k=0}^{K-1}) \\ \mathbb{x} = (z, [f_{\hat{\mathbf{z}}}(X)]) \\ \mathbb{w} = f_{\hat{\mathbf{z}}}(X) \\ R = \begin{pmatrix} \varphi(a_0)^0 & \varphi(a_1)^0 & \cdots \\ \varphi(a_0)^1 & \varphi(a_1)^1 & \cdots \\ \vdots & \vdots & \ddots \end{pmatrix} \\ f_{\hat{\mathbf{z}}}(X) = \sum_{i=0}^{H-1} \hat{z}_i X^i \\ \mathbf{z} = (1\,,\, z\,,\, z^2\,,\, \cdots\,,\, z^{K-1})^{\mathsf{T}} \\ \hat{\mathbf{z}} = R\mathbf{z} \end{array} \right\} . \tag{40}$$

Theorem 8 (Security of VandermondeVector). *Protocol* VandermondeVector *is a Polynomial IOP for* $\mathcal{R}_{\text{vv}}$ *with completeness and soundness with soundness error* $\sigma \leq \frac{H+5K-5}{|\mathbb{F}|-1}$.

Proof. Consider the following sequences of equations.

$$\hat{\mathbf{z}} = R\mathbf{z} \tag{41}$$

$$\boldsymbol{\delta}^{\mathsf{T}}\hat{\mathbf{z}} = \boldsymbol{\delta}^{\mathsf{T}} R\mathbf{z} \tag{42}$$

$$f_{\hat{\mathbf{z}}}(\delta) = f_{\hat{\boldsymbol{\delta}}}(z) \tag{43}$$

$$y_1 = y_0 \;\;, \tag{44}$$

and

$$\forall k \in \{0, \ldots, K-1\} \,.\, \left(\sum_{h=0}^{H-1} (\delta \cdot \varphi(k))^h \right) \cdot (1 - \delta \cdot \varphi(a_k)) = 1 - (\delta \cdot \varphi(a_k))^H \tag{45}$$

$$\left(\boldsymbol{\delta}^{\mathsf{T}} R\right) \circ (1 - \delta \cdot \varphi(a_k))_{k=0}^{K-1} = \left(1 - (\delta \cdot \varphi(a_k))^H\right)_{k=0}^{K-1} \tag{46}$$

$$f_{\hat{\boldsymbol{\delta}}}(X) \circ \left(\delta \cdot f_{\varphi(a_k)}(X) - f_I(X)\right) = \delta^H \cdot f_{\varphi(a_k)^H}(X) - f_I(X) \tag{47}$$

$$(\mathbb{i}^{(1)}, \mathbb{x}^{(1)}) = \tag{48}$$

$$(K-1, [f_{\hat{\boldsymbol{\delta}}}(X)], [\delta \cdot f_{\varphi(a_k)}(X) - f_I(X)], [\delta^H \cdot f_{\varphi(a_k)^H}(X) - f_I(X)]) \in \mathcal{L}(\mathcal{R}_{\text{hadamard}}) \;\;.$$

Completeness follows from the sequences of implications a) (41) $\Rightarrow$ (42) $\Leftrightarrow$ (43) $\Leftrightarrow$ (44), and b) (45) $\Leftrightarrow$ (46) $\Leftrightarrow$ (47) $\Rightarrow$ (48). Sequence (a) holds for any

description: decides $\mathcal{L}(\mathcal{R}_{\text{vv}})$
inputs: $\mathbb{i}$: $H, K, \varphi(h), \{a_k\}_{k=0}^{K-1}$
$\mathbb{x}$: $z, [f_{\hat{\boldsymbol{z}}}(X)]$
$\mathbb{w}$: $f_{\hat{\boldsymbol{z}}}(X)$
// pre-processing
begin
  I computes $f_{\varphi(a_k)}(X) \leftarrow \sum_{k=0}^{K-1} \varphi(a_k) X^k$ and
  $f_{\varphi(a_k)^H}(X) \leftarrow \sum_{k=0}^{K-1} \varphi(a_k)^H X^k$
  I sends $f_{\varphi(a_k)}(X)$ and $f_{\varphi(a_k)^H}(X)$, both of degree at most $K-1$, to P and V
// online
begin
  V samples $\delta \xleftarrow{\$} \mathbb{F}\backslash\{0\}$ and sends δ to P
  P computes $\boldsymbol{\delta} \leftarrow (\delta^0, \delta^1, \ldots, \delta^{H-1})^\mathsf{T}$, $\hat{\boldsymbol{\delta}} \leftarrow R^\mathsf{T}\boldsymbol{\delta}$, and $f_{\hat{\boldsymbol{\delta}}}(X) \leftarrow \sum_{i=0}^{K-1} \hat{\delta}_i X^i$
  P sends $f_{\hat{\boldsymbol{\delta}}}(X)$ of degree at most $K-1$ to V
  V queries $[f_{\hat{\boldsymbol{\delta}}}(X)]$ in z and receives $y_0 = f_{\hat{\boldsymbol{\delta}}}(z)$
  V queries $[f_{\hat{\boldsymbol{z}}}(X)]$ in δ and receives $y_1 = f_{\hat{\boldsymbol{z}}}(\delta)$
  V checks $y_1 \stackrel{?}{=} y_0$
  P and V run Hadamard with $\mathbb{i}^{(1)} = H - 1$,
  $\mathbb{x}^{(1)} = ([f_{\hat{\boldsymbol{\delta}}}(X)], [\delta \cdot f_{\varphi(a_k)}(X) - f_I(X)], [\delta \cdot f_{\varphi(a_k)^H}(X) - f_I(X)])$,
  $\mathbb{w}^{(1)} = (f_{\hat{\boldsymbol{\delta}}}(X), \delta^H \cdot f_{\varphi(a_k)}(X) - f_I(X), \delta^H \cdot f_{\varphi(a_k)^H}(X) - f_I(X))$, where
  V simulates $[\delta \cdot f_{\varphi(a_k)}(X) - f_I(X)], [\delta^H \cdot f_{\varphi(a_k)^H}(X) - f_I(X)]$ using the oracles $[f_{\varphi(a_k)}(X)], [f_{\varphi(a_k)^H}(X)]$ and the known scalar δ, in combination with computing $f_I(X) = \sum_{k=0}^{K-1} X^k$ locally

Protocol 8: VandermondeVector

matrix R. Sequence (b) starts with the equation for a geometric sum derived from the matrix R as defined as in Eqn. 26.

For soundness, consider when the reverse implications fail. There are two such cases:

- (41) $\not\Leftarrow$ (42). By the Schwartz-Zippel lemma, the probability of this event is bounded by $\frac{H-1}{|\mathbb{F}|-1}$.
- (47) $\not\Leftarrow$ (48). The probability of this event is captured by the soundness error of $\mathsf{Hadamard}$, $\sigma_{\mathsf{Hadamard}} \leq \frac{5K-4}{|\mathbb{F}|-1}$.

By the Union Bound, the soundness error of $\mathsf{VandermondeVector}$ is bounded by $\sigma \leq \frac{H+5K-5}{|\mathbb{F}|-1}$. □

Lagrange Vector. The $\mathsf{LagrangeVector}$ protocol realizes the following relation

$$\mathcal{R}_{\text{lv}} = \left\{ (\mathbb{i}, \mathbb{x}, \mathbb{w}) \;\middle|\; \begin{array}{l} \mathbb{i} = (H, \varphi(h)) \\ \mathbb{x} = (x, [f_{\hat{\mathbf{x}}}(X)]) \\ \mathbb{w} = f_{\hat{\mathbf{x}}}(X) \\ L = \begin{pmatrix} \mathcal{L}_{\varphi(0),0} & \mathcal{L}_{\varphi(0),1} & \cdots \\ \mathcal{L}_{\varphi(1),0} & \mathcal{L}_{\varphi(1),1} & \cdots \\ \vdots & \vdots & \ddots \end{pmatrix} \\ f_{\hat{\mathbf{x}}}(X) = \sum_{i=0}^{H-1} \hat{x}_i X^i \\ \mathbf{x} = (1\,,\, x\,,\, x^2\,,\, \cdots\,,\, x^{H-1})^{\mathsf{T}} \\ \hat{\mathbf{x}} = \mathbf{x}^{\mathsf{T}} L \end{array} \right\} . \tag{49}$$

Theorem 9 (Security of $\mathsf{LagrangeVector}$). *Protocol $\mathsf{LagrangeVector}$ is a Polynomial IOP for $\mathcal{R}_{\text{lv}}$ with completeness and soundness with soundness error $\sigma \leq \frac{6H-5}{|\mathbb{F}|-1}$.*

Proof. Consider the following sequences of equations.

$$\hat{\mathbf{x}}^{\mathsf{T}} = \mathbf{x}^{\mathsf{T}} L \tag{50}$$

$$\hat{\mathbf{x}}^{\mathsf{T}} \boldsymbol{\gamma} = \mathbf{x}^{\mathsf{T}} L \boldsymbol{\gamma} \tag{51}$$

$$f_{\hat{\mathbf{x}}}(\gamma) = f_{\hat{\boldsymbol{\gamma}}}(x) \tag{52}$$

$$y_1 = y_0 \;\;, \tag{53}$$

description: decides $\mathcal{L}(\mathcal{R}_{\text{lv}})$
inputs: $\mathbb{i}$: $H, \varphi(h)$
$\mathbb{x}$: $x, [f_{\hat{\mathbf{x}}}(X)]$
$\mathbb{w}$: $f_{\hat{\mathbf{x}}}(X)$
// pre-processing
begin
I computes $f_{\varphi(h)}(X) \leftarrow \sum_{h=0}^{H-1} \varphi(h) X^h$, $Z_{\mathcal{H}}(X) \leftarrow \prod_{h=0}^{H-1}(X - \varphi(h))$ and $f_{\mathcal{H}}(X) \leftarrow \sum_{h=0}^{H-1} \left(\prod_{i \in \{0 \ldots H-1\} \setminus \{h\}} (\varphi(h) - \varphi(i))\right)^{-1} \cdot X^h$
I sends $f_{\varphi(h)}(X)$, $f_{\mathcal{H}}(X)$, both of degree at most $H-1$, and $Z_{\mathcal{H}}(X)$ of degree at most H, to P and V
// online
begin
V samples $\gamma \xleftarrow{\$} \mathbb{F} \setminus \{0\}$ and sends γ to P
P computes $\boldsymbol{\gamma} \leftarrow (\gamma^0, \gamma^1, \ldots, \gamma^{H-1})^{\mathsf{T}}$, $\hat{\boldsymbol{\gamma}} \leftarrow L\boldsymbol{\gamma}$, and $f_{\hat{\boldsymbol{\gamma}}}(X) \leftarrow \sum_{i=0}^{H-1} \hat{\gamma}_i X^i$
P sends $f_{\hat{\boldsymbol{\gamma}}}(X)$ of degree at most $H-1$ to V
V queries $[f_{\hat{\boldsymbol{\gamma}}}(X)]$ in x and receives $y_0 = f_{\hat{\boldsymbol{\gamma}}}(x)$
V queries $[f_{\hat{\mathbf{x}}}(X)]$ in γ and receives $y_1 = f_{\hat{\mathbf{x}}}(\gamma)$
V checks $y_1 \stackrel{?}{=} y_0$
V queries $[Z_{\mathcal{H}}(X)]$ in γ and receives $u = Z_{\mathcal{H}}(\gamma)$
P and V run Hadamard with $\mathbb{i}^{(1)} = H - 1$,
$\mathbb{x}^{(1)} = ([f_{\hat{\boldsymbol{\gamma}}}(X)], [\gamma f_I(X) - f_{\varphi(h)}(X)], [u f_{\mathcal{H}}(X)])$,
$\mathbb{w}^{(1)} = (f_{\hat{\boldsymbol{\gamma}}}(X), \gamma f_I(X) - f_{\varphi(h)}(X), u f_{\mathcal{H}}(X))$, where V simulates $[\gamma f_I(X) - f_{\varphi(h)}(X)], [u f_{\mathcal{H}}(X)]$ using the oracles $[f_{\varphi(h)}(X)], [f_{\mathcal{H}}(X)]$ and the known scalar γ, in combination with computing $f_I(X) = \sum_{h=0}^{H-1} X^h$ locally

Protocol 9: LagrangeVector

and

$$\forall h \in \{0, \ldots, H-1\} \,.\, \mathcal{L}_h(\gamma) = \prod_{\substack{i=0 \\ i \neq h}}^{H-1} \frac{\gamma - \varphi(i)}{\varphi(h) - \varphi(i)} \tag{54}$$

$$\forall h \in \{0, \ldots, H-1\} \,.\, \mathcal{L}_h(\gamma) \cdot (\gamma - \varphi(h)) = \frac{Z_{\mathcal{H}}(\gamma)}{\prod_{\substack{i=0 \\ i \neq h}}^{H-1} (\varphi(h) - \varphi(i))} \tag{55}$$

$$(L\boldsymbol{\gamma}) \circ (\gamma - \varphi(h))_{h=0}^{H-1} = Z_{\mathcal{H}}(\gamma) \left(\left(\prod_{i \in \{0 \ldots H-1\} \setminus \{h\}} (\varphi(h) - \varphi(i)) \right)^{-1} \right)_{h=0}^{H-1} \tag{56}$$

$$f_{\hat{\boldsymbol{\gamma}}}(X) \circ (\gamma f_I(X) - f_{\varphi}(X)) = u f_{\mathcal{H}}(X) \tag{57}$$

$$(\mathbb{i}^{(1)}, \mathbb{X}^{(1)}) = \tag{58}$$
$$(H-1, [f_{\hat{\boldsymbol{\gamma}}}(X)], [\gamma f_I(X) - f_{\varphi(h)}(X)], [u f_{\mathcal{H}}(X)]) \in \mathcal{L}(\mathcal{R}_{\text{hadamard}}) \ .$$

Completeness follows from the sequences of implications a) (50) $\Rightarrow$ (51) $\Leftrightarrow$ (52) $\Leftrightarrow$ (53), and b) (54) $\Leftrightarrow$ (55) $\Leftrightarrow$ (56) $\Leftrightarrow$ (57) $\Rightarrow$ (58). Sequence (a) holds for any matrix L. Sequence (b) starts with the definition of Lagrange basis polynomials, and holds when the rows of L are exactly the coefficient vectors of the Lagrange basis polynomials. Recall that Lagrange basis polynomial indexed by h takes the value 1 in $\varphi(h)$ and 0 in all other points of $\mathcal{H}$.

For soundness, consider when the reverse implications fail. There are two such cases:

- (50) $\not\Leftarrow$ (51). By the Schwartz-Zippel lemma, the probability of this event is bounded by $\frac{H-1}{|\mathbb{F}|-1}$.
- (57) $\not\Leftarrow$ (58). The probability of this event is captured by the soundness error of Hadamard, $\sigma_{\mathsf{Hadamard}} \leq \frac{5H-4}{|\mathbb{F}|-1}$.

By the Union Bound, the soundness error of LagrangeVector is bounded by $\sigma \leq \frac{6H-5}{|\mathbb{F}|-1}$. □

Sparse Univariate Polynomial Evaluation. In terms of sparse univariate polynomial evaluation, we actually achieve something more generic. Let $f(X) = \sum_{k=0}^{K-1} X^{a_k}$. The next protocol establishes that $f_z(X) = \sum_{k=0}^{K-1} z^{a_k} X^k$ is indeed the polynomial whose vector of coefficients corresponds to the monomials of $f(X)$ when evaluated in z. By evaluating $f_z(X)$ in $X = 1$, one obtains the evaluation $f(z)$.

The protocol SparseMonomialVector realizes the following relation.

$$\mathcal{R}_{\text{smv}} = \left\{ (\mathbb{i}, \mathbb{x}, \mathbb{w}) \;\middle|\; \begin{array}{l} \mathbb{i} = (H, K, \{a_k\}_{k=0}^{K-1}) \\ \mathbb{x} = (z, [f_z(X)]) \\ \mathbb{w} = (f_z(X)) \\ f_z(X) = \sum_{k=0}^{K-1} z^{a_k} X^k \\ H = \max_k a_k \end{array} \right\} \tag{59}$$

description: decides $\mathcal{L}(\mathcal{R}_{\text{smv}})$
inputs: $\mathbb{i}$: $H, K, \{a_k\}_{k=0}^{K-1}$
$\mathbb{x}$: $z, [f_z(X)]$
$\mathbb{w}$: $f_z(X)$
// pre-processing
begin
- I selects any mapping $\varphi : \mathbb{N} \to \mathbb{F}$ such that $\{0, \ldots, H-1\}$ are mapped to distinct elements
- I runs VandermondeVector.I with $\mathbb{i}^{(1)} = (H, K, \varphi(h), \{a_k\}_{k=0}^{K-1})$
- I runs LagrangeVector.I with $\mathbb{i}^{(2)} = (H, \varphi(h))$

// online
begin
- P computes $f_{\hat{\mathbf{z}}}(X) = \sum_{h=0}^{H-1} z^h \mathcal{L}_h(X)$ where $\mathcal{L}_h(X)$ is the unique monic polynomial that evaluates to 1 in $\varphi(h)$ and to 0 in all other points of $\mathcal{H}$
- P sends $f_{\hat{\mathbf{z}}}(X)$ of degree at most $H-1$ to V
- P and V run LagrangeVector with $\mathbb{i}^{(2)} = (H, \varphi(h))$, $\mathbb{x}^{(2)} = (z, [f_{\hat{\mathbf{z}}}(X)])$, and $\mathbb{w}^{(2)} = f_{\hat{\mathbf{z}}}(X)$
- V samples $y \xleftarrow{\$} \mathbb{F} \backslash \{0\}$ and sends y to P
- V queries $[f_z(X)]$ in y and receives $x = f_z(y)$
- P computes $f_{\hat{\mathbf{y}}}(X) \leftarrow \sum_{h=0}^{H-1} \left(\sum_{k=0}^{K-1} \varphi(a_k)^h y^h \right) X^h$, whose coefficient vector corresponds to $R\mathbf{y}$ with $\mathbf{y} = (1, y, \ldots, y^{K-1})^\mathsf{T} \in \mathbb{F}^K$
- P sends $f_{\hat{\mathbf{y}}}(X)$ of degree at most $H-1$ to V
- P and V run VandermondeVector with $\mathbb{i}^{(1)} = (H, K, \varphi(h), \{a_k\}_{k=0}^{K-1})$, $\mathbb{x}^{(1)} = (y, [f_{\hat{\mathbf{y}}}(X)])$, and $\mathbb{w}^{(1)} = f_{\hat{\mathbf{y}}}(X)$
- P and V run InnerProduct with $\mathbb{i}^{(3)} = H-1$, $\mathbb{x}^{(3)} = ([f_{\hat{\mathbf{z}}}(X)], [f_{\hat{\mathbf{y}}}(X)], x)$, and $\mathbb{w}^{(3)} = (f_{\hat{\mathbf{z}}}(X), f_{\hat{\mathbf{y}}}(X))$

Protocol 10: SparseMonomialVector

Theorem 10. *(Security of* SparseMonomialVector*) Protocol* SparseMonomialVector *is a Polynomial IOP for $\mathcal{R}_{\text{smv}}$ with completeness and soundness with soundness error $\sigma \leq \frac{11H+6K-14}{|\mathbb{F}|-1}$.*

Proof. Consider the following sequence of equations.

$$f_z(X) = \sum_{k=0}^{K-1} z^{a_k} X^k \tag{60}$$

$$f_z(y) = \sum_{k=0}^{K-1} z^{a_k} y^k \tag{61}$$

$$f_z(y) = \mathbf{z}^{\mathsf{T}} B \mathbf{y} \tag{62}$$

$$f_z(y) = \mathbf{z}^{\mathsf{T}} L R \mathbf{y} \tag{63}$$

$$f_z(y) = \hat{\mathbf{z}}^{\mathsf{T}} R \mathbf{y} \tag{64}$$

$$f_z(y) = \hat{\mathbf{z}}^{\mathsf{T}} \hat{\mathbf{y}} \tag{65}$$

$$(\mathbb{i}, \mathbb{x}) = (H-1, ([f_{\hat{\mathbf{z}}}(X)], [f_{\hat{\mathbf{y}}}(X)], x)) \in \mathcal{R}_{\text{ip}} \tag{66}$$

In these formulae, we define $\hat{\mathbf{z}}$ and $\hat{\mathbf{y}}$ as the vectors of coefficients of $f_{\hat{\mathbf{z}}}(X)$ and $f_{\hat{\mathbf{y}}}(X)$, respectively.

Completeness follows from the sequence of implications (60) $\Rightarrow$ (61) $\Leftrightarrow$ (62) $\Leftrightarrow$ (63) $\Rightarrow$ (64) $\Rightarrow$ (65) $\Rightarrow$ (66). For soundness, consider when the reverse implications fail.

- (60) $\not\Leftarrow$ (61). This event happens with probability at most $\frac{K-1}{|\mathbb{F}|-1}$ due to the Schwartz-Zippel lemma.
- (63) $\not\Leftarrow$ (64). This event implies that LagrangeVector succeeded despite being run on a false instance. This happens with probability at most equal to the soundness error of that protocol, $\sigma_{\mathsf{LagrangeVector}} \leq \frac{6H-5}{|\mathbb{F}|-1}$.
- (64) $\not\Leftarrow$ (65). This event implies that VandermondeVector succeeded despite being run on a false instance. This happens with probability at most equal to the soundness error of that protocol, $\sigma_{\mathsf{VandermondeVector}} \leq \frac{H+5K-5}{|\mathbb{F}|-1}$.
- (65) $\not\Leftarrow$ (66). This event implies that InnerProduct succeeded despite being run on a false instance. This happens with probability at most equal to the soundness error of that protocol, $\sigma_{\mathsf{InnerProduct}} \leq \frac{4H-3}{|\mathbb{F}|-1}$.

By the Union Bound, the probability that any one of these events occurs is bounded by $\sigma \leq \frac{11H+6K-14}{|\mathbb{F}|-1}$. □

An important class of sparse univariate polynomials are those that represent permutation matrices. As a consequence, protocol SparseMonomialVector can be used to establish that two polynomials have the same coefficients, except permuted according to a committed permutation. We omit the details of this digression.

Sparse Bivariate Polynomial Evaluation. The relation realized by the protocol SparseBiEval is given as follows.

$$\mathcal{R}_{\text{sbe}} = \left\{ (\mathbb{i}, \mathbb{x}, \mathbb{w}) \;\middle|\; \begin{array}{l} \mathbb{i} = (H, K, \{a_k\}_{k=0}^{K-1}, \{b_k\}_{k=0}^{K-1}, \{c_k\}_{k=0}^{K-1}) \\ \mathbb{x} = (u, v, w) \\ \mathbb{w} = \varnothing \\ H = \max(\max_k\{a_k\}, \max_k\{b_k\}) \\ f(X,Y) = \sum_{k=0}^{K-1} c_k X^{a_k} Y^{b_k} \\ f(u,v) = w \end{array} \right\} \tag{67}$$

description: decides $\mathcal{L}(\mathcal{R}_{\text{sbe}})$
inputs: $\mathbb{i}$: $H, K, \{a_k\}_{k=0}^{K-1}, \{b_k\}_{k=0}^{K-1}, \{c_k\}_{k=0}^{K-1}$
$\mathbb{x}$: u, v, w
$\mathbb{w}$: $\varnothing$
// pre-processing
begin
- I selects any mapping $\varphi : \mathbb{N} \to \mathbb{F}$ such that $\{0, \ldots, H-1\}$ are mapped to distinct elements
- I runs SparseMonomialVector.I with $\mathbb{i}^{(1)} = (H, K, \{a_k\}_{k=0}^{K-1})$
- I runs SparseMonomialVector.I with $\mathbb{i}^{(2)} = (H, K, \{b_k\}_{k=0}^{K-1})$
- I computes $f_c(X) \leftarrow \sum_{k=0}^{K-1} c_k X^k$
- I sends $f_c(X)$ of degree at most $K-1$ to P and V

// online
begin
- P computes $f_u(X) \leftarrow \sum_{k=0}^{K-1} u^{a_k} X^k$, $f_v(X) \leftarrow \sum_{k=0}^{K-1} v^{b_k} X^k$, and $f_{uv}(X) \leftarrow \sum_{k=0}^{K-1} u^{a_k} v^{b_k} X^k$
- P sends $f_u(X)$, $f_v(X)$, and $f_{uv}(X)$, all of degree at most $K-1$, to V
- P and V run SparseMonomialVector with $\mathbb{i}^{(1)} = (H, K, \{a_k\}_{k=0}^{K-1})$, $\mathbb{x}^{(1)} = (u, [f_u(X)])$, and $\mathbb{w}^{(1)} = f_u(X)$
- P and V run SparseMonomialVector with $\mathbb{i}^{(2)} = (H, K, \{b_k\}_{k=0}^{K-1})$, $\mathbb{x}^{(2)} = (v, [f_v(X)])$, and $\mathbb{w}^{(2)} = f_v(X)$
- P and V run Hadamard with $\mathbb{i}^{(3)} = K-1$, $\mathbb{x}^{(3)} = ([f_u(X)], [f_v(X)], [f_{uv}(X)])$, and $\mathbb{w}^{(3)} = (f_u(X), f_v(X), f_{uv}(X))$
- P and V run InnerProduct with $\mathbb{i}^{(4)} = K-1$, $\mathbb{x}^{(4)} = ([f_{uv}(X)], [f_c(X)], w)$, and $\mathbb{w}^{(4)} = (f_{uv}(X), f_c(X))$

Protocol 11: SparseBiEval

Theorem 11 (Security of SparseBiEval). *Protocol* SparseBiEval *is a Polynomial IOP for* $\mathcal{R}_{\text{sbi}}$ *with completeness and soundness with soundness error* $\sigma = \frac{22H+21K-35}{|\mathbb{F}|-1}$.

Proof. Consider the following sequence of equations.

$$f(u,v) = w \tag{68}$$

$$\sum_{k=0}^{K-1} c_k u^{a_k} v^{b_k} = w \tag{69}$$

$$\mathsf{coeffs}(f_{uv}(X)) \cdot \mathsf{coeffs}(f_c(X)) = w \tag{70}$$

$$(\mathsf{coeffs}(f_u(X)) \circ \mathsf{coeffs}(f_v(X))) \cdot \mathsf{coeffs}(f_c(X)) = w \tag{71}$$

$$(\mathsf{coeffs}(f_u(X)) \circ \mathsf{coeffs}(f_v(X))) \cdot (c_k)_{k=0}^{K-1} = w \tag{72}$$

$$\left(\mathsf{coeffs}(f_u(X)) \circ \left(v^{b_k}\right)_{k=0}^{K-1}\right) \cdot (c_k)_{k=0}^{K-1} = w \tag{73}$$

$$\left((u^{a_k})_{k=0}^{K-1} \circ \left(v^{b_k}\right)_{k=0}^{K-1}\right) \cdot (c_k)_{k=0}^{K-1} = w \tag{74}$$

Completeness follows from the sequence of implications (68) $\Leftrightarrow$ (69) $\Rightarrow$ (70) $\Rightarrow$ (71) $\Leftrightarrow$ (72) $\Rightarrow$ (73) $\Rightarrow$ (74). For soundness, consider when the reverse implications fail.

- (69) $\not\Leftarrow$ (70). This event happens with probability at most equal to the soundness error of $\mathsf{InnerProduct}$, $\sigma_{\mathsf{InnerProduct}} \leq \frac{4K-3}{|\mathbb{F}|-1}$.
- (70) $\not\Leftarrow$ (71). This event happens with probability at most equal to the soundness error of $\mathsf{Hadamard}$, $\sigma_{\mathsf{Hadamard}} \leq \frac{5K-4}{|\mathbb{F}|-1}$.
- (72) $\not\Leftarrow$ (73). This event happens with probability at most equal to the soundness error of $\mathsf{SparseMonomialVector}$, $\sigma_{\mathsf{SparseMonomialVector}} \leq \frac{11H+6K-14}{|\mathbb{F}|-1}$.
- (73) $\not\Leftarrow$ (74). This event happens with probability at most equal to the soundness error of $\mathsf{SparseMonomialVector}$, $\sigma_{\mathsf{SparseMonomialVector}} \leq \frac{11H+6K-14}{|\mathbb{F}|-1}$.

By the Union Bound, the soundness error of $\mathsf{SparseBiEval}$ is bounded by $\sigma \leq \frac{22H+21K-35}{|\mathbb{F}|-1}$. □

Sparse Matrix-Vector Product. The relation that is realized by the sparse matrix-vector product protocol is essentially the same as the one realized by the regular (dense) matrix-vector product protocol. However, we restate this relation in such a way so as to stress that the matrix is represented sparsely, *i.e.*, as a list of position-coefficient tuples rather than an array of coefficients.

$$\mathcal{R}_{\text{smvp}} = \left\{ (\mathbb{i}, \mathbb{x}, \mathbb{w}) \;\middle|\; \begin{array}{l} \mathbb{i} = (m, n, \{a_k\}_{k=0}^{K-1}, \{b_k\}_{k=0}^{K-1}, \{c_k\}_{k=0}^{K-1}) \\ \mathbb{x} = ([f_{\boldsymbol{a}}(X)], [f_{\boldsymbol{b}}(X)]) \\ \mathbb{w} = (f_{\boldsymbol{a}}(X), f_{\boldsymbol{b}}(X)) \\ f_{\boldsymbol{a}}(X) = \sum_{i=0}^{n-1} \boldsymbol{a}_{[i]} X^i \text{ for some } \boldsymbol{a} \in \mathbb{F}^n \\ f_{\boldsymbol{b}}(X) = \sum_{i=0}^{m-1} \boldsymbol{b}_{[i]} X^i \text{ for some } \boldsymbol{b} \in \mathbb{F}^m \\ M = \sum_{k=0}^{K-1} c_k \mathbf{e}_{a_k}^{\mathsf{T}} \mathbf{e}_{b_k} \\ \boldsymbol{b} = M\boldsymbol{a} \end{array} \right\} . \tag{75}$$

description: decides $\mathcal{L}(\mathcal{R}_{\text{smvp}})$
inputs: $\mathbb{i}$: $m, n, \{a_k\}_{k=0}^{K-1}, \{b_k\}_{k=0}^{K-1}, \{c_k\}_{k=0}^{K-1}$
$\mathbb{x}$: $[f_{\boldsymbol{a}}(X)], [f_{\boldsymbol{b}}(X)]$
$\mathbb{w}$: $f_{\boldsymbol{a}}(X), f_{\boldsymbol{b}}(X)$
// pre-processing
begin
  I computes $H \leftarrow \max(m, n)$
  I runs $\mathsf{SparseBiEval.I}$ with $\mathbb{i}^{(1)} = (H, K, \{a_k\}_{k=0}^{K-1}, \{b_k\}_{k=0}^{K-1}, \{c_k\}_{k-1}^{K-1})$
// online
begin
  V samples $\alpha \xleftarrow{\$} \mathbb{F}\backslash\{0\}$ and sends α to P
  P computes $f_{\boldsymbol{\alpha}^\intercal M}(X) \leftarrow \sum_{k=0}^{K-1} c_k \alpha^{a_k} X^{b_k}$
  P sends $f_{\boldsymbol{\alpha}^\intercal M}(X)$ of degree at most $n-1$ to V
  P computes $s \leftarrow f_{\boldsymbol{b}}(\alpha)$
  V queries $[f_{\boldsymbol{b}}(X)]$ in α and receives $s = f_{\boldsymbol{b}}(\alpha)$
  P and V run $\mathsf{InnerProduct}$ with $\mathbb{i}^{(2)} = n-1$, $\mathbb{x}^{(2)} = ([f_{\boldsymbol{a}}(X)], [f_{\boldsymbol{\alpha}^\intercal M}(X)], s)$, and $\mathbb{w}^{(2)} = (f_{\boldsymbol{a}}(X), f_{\boldsymbol{\alpha}^\intercal M}(X))$
  V samples $\beta \xleftarrow{\$} \mathbb{F}\backslash\{0\}$
  P computes $w \leftarrow f_{\boldsymbol{\alpha}^\intercal M}(\beta)$
  V queries $[f_{\boldsymbol{\alpha}^\intercal M}(X)]$ in β and receives $w = f_{\boldsymbol{\alpha}^\intercal M}(\beta)$
  P and V run $\mathsf{SparseBiEval}$ with $\mathbb{i}^{(1)} = (H, K, \{a_k\}_{k=0}^{K-1}, \{b_k\}_{k=0}^{K-1}, \{c_k\}_{k=0}^{K-1})$, $\mathbb{x}^{(1)} = (\alpha, \beta, w)$, and $\mathbb{w}^{(1)} = \varnothing$

Protocol 12: SparseMVP

Theorem 12 (Security of SparseMVP). *Protocol* SparseMVP *is a Polynomial IOP for* $\mathcal{R}_{\text{smvp}}$ *with completeness and soundness with soundness error* $\sigma \leq \frac{5m+22H+21K-39}{|\mathbb{F}|-1}$.

Proof. Consider the following sequence of equations.

$$\boldsymbol{b} = M\boldsymbol{a} \tag{76}$$
$$\boldsymbol{\alpha}^\intercal \boldsymbol{b} = \boldsymbol{\alpha}^\intercal M\boldsymbol{a} \tag{77}$$
$$f_{\boldsymbol{b}}(\alpha) = \boldsymbol{\alpha}^\intercal M\boldsymbol{a} \tag{78}$$
$$s = \mathsf{coeffs}(f_{\boldsymbol{\alpha}^\intercal M}(X)) \cdot \mathsf{coeffs}(f_{\boldsymbol{a}}(X)) \tag{79}$$
$$s = \mathsf{coeffs}(f_M(\alpha, X)) \cdot \mathsf{coeffs}(f_{\boldsymbol{a}}(X)) \tag{80}$$

Completeness follows from the sequence of implications (76) $\Rightarrow$ (77) $\Leftrightarrow$ (78) $\Rightarrow$ (79) $\Rightarrow$ (80). For soundness, consider when the reverse implications fail.

- (76) $\not\Leftarrow$ (77). Due to the Schwartz-Zippel lemma, this event occurs with probability $\frac{m-1}{|\mathbb{F}|-1}$.
- (78) $\not\Leftarrow$ (79). This event occurs with probability bounded by the soundness error of $\mathsf{InnerProduct}$, $\sigma_{\mathsf{InnerProduct}} \leq \frac{4m-3}{|\mathbb{F}|-1}$.

– (79) $\not\Leftarrow$ (80). This event implies that $f_{\boldsymbol{a}^\intercal M}(X)$ and $f_M(\alpha, X)$ are distinct polynomials. The probability that this distinction is not caught by the SparseBiEval protocol is bounded by that protocol's soundness error, $\sigma_{\mathsf{SparseBiEval}} \leq \frac{22H+21K-35}{|\mathbb{F}|-1}$.

By the Union Bound, the soundness error of SparseMVP is bounded by $\sigma \leq \frac{5m+22H+21K-39}{|\mathbb{F}|-1}$. □

C Alternative and Optimized Protocols

C.1 DenseMVP

The protocol DenseMVP admits one merger of polynomial identities after unrolling. We present the optimized protocol in several steps to simplify their verification by the reader.

description: decides $\mathcal{L}(\mathcal{R}_{\text{mvp}})$
inputs: $\mathtt{i} : (m, n, M)$
$\mathtt{x} : ([f_{\boldsymbol{a}}(X)], [f_{\boldsymbol{b}}(X)])$
$\mathtt{w} : (f_{\boldsymbol{a}}(X), f_{\boldsymbol{b}}(X))$
// pre-processing
begin
  I computes $f_M(X) \leftarrow \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} M_{[i,j]} X^{in+j}$
  I sends $f_M(X)$ of degree at most $mn-1$ to P and V
begin
  V samples $\alpha \xleftarrow{\$} \mathbb{F}$ and sends α to P
  P computes $r(X) \leftarrow f_M(X) \bmod X^n - \alpha$
  P sends $r(X)$ of degree at most $n-1$ to V
  P and V run ModReduce with $\mathtt{i}^{(1)} = (mn-1, n-1)$,
   $\mathtt{x}^{(1)} = ([f_M(X)], [r(X)], X^n - \alpha)$, and $\mathtt{w}^{(1)} = (f_M(X), r(X))$
  V queries $[f_{\boldsymbol{b}}(X)]$ in α and receives $y_{\boldsymbol{a}^\intercal \boldsymbol{b}} = f_{\boldsymbol{b}}(\alpha)$
  P and V run InnerProduct with $\mathtt{i}^{(2)} = n-1$, $\mathtt{x}^{(2)} = ([r(X)], [f_{\boldsymbol{a}}(X)], y_{\boldsymbol{a}^\intercal \boldsymbol{b}})$,
   and $\mathtt{w}^{(2)} = (r(X), f_{\boldsymbol{a}}(X))$

Protocol 13: DenseMVP, original

description: decides $\mathcal{L}(\mathcal{R}_{\text{mvp}})$
inputs: $\mathtt{i} : (m, n, M)$
$\mathtt{x} : ([f_{\boldsymbol{a}}(X)], [f_{\boldsymbol{b}}(X)])$
$\mathtt{w} : (f_{\boldsymbol{a}}(X), f_{\boldsymbol{b}}(X))$
// pre-processing
begin
  I computes $f_M(X) \leftarrow \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} M_{[i,j]} X^{in+j}$
  I sends $f_M(X)$ of degree at most $mn-1$ to P and V
begin
  V samples $\alpha \xleftarrow{\$} \mathbb{F}$ and sends α to P
  P computes $r(X) \leftarrow f_M(X) \bmod X^n - \alpha$
  P sends $r(X)$ of degree at most $n-1$ to V
  P and V run ModReduce with $\mathtt{i}^{(1)} = (mn-1, n-1)$,
  $\mathtt{x}^{(1)} = ([f_M(X)], [r(X)], X^n - \alpha)$, and $\mathtt{w}^{(1)} = (f_M(X), r(X))$
  V queries $[f_{\boldsymbol{b}}(X)]$ in α and receives $y_{\boldsymbol{a}^\intercal \boldsymbol{b}} = f_{\boldsymbol{b}}(\alpha)$
  // InnerProduct
  begin
    P computes $h(X) \leftarrow r(X) \cdot f_{\boldsymbol{a}}(X^{-1}) \cdot X^{\mathtt{n}-1}$
    P computes $\bar{h}(X) = \sum_{i=0}^{2n-1} \bar{h}_i X^i$ with $\bar{h}_i \leftarrow \frac{h_i}{\gamma^{n-1} - \gamma^i}$ for $i \neq n-1$ and
    $\bar{h}_{n-1} \xleftarrow{\$} \mathbb{F}$
    P sends $\bar{h}(X)$ of degree at most $2n-1$ to V
    V samples $z \xleftarrow{\$} \mathbb{F} \backslash \{0\}$ and queries $([r(X)], [f_{\boldsymbol{a}}(X)], [\bar{h}(X)], [\bar{h}(X)])$ in
    $(z, z^{-1}, z, \gamma \cdot z)$
    V receives $y_r = f(z)$, $y_a = f_{\boldsymbol{a}}(z^{-1})$, $y_h = \bar{h}(z)$, and $y_h^* = \bar{h}(\gamma \cdot z)$
    V tests $y_h \cdot \gamma^{n-1} - y_h^* \stackrel{?}{=} y_r \cdot y_a \cdot z^{n-1} - y_{\boldsymbol{a}^\intercal \boldsymbol{b}}$

Protocol 14: DenseMVP, unrolled InnerProduct

description: decides $\mathcal{L}(\mathcal{R}_{\text{mvp}})$
inputs: $\mathtt{i} : (m, n, M)$
$\mathtt{x} : ([f_{\boldsymbol{a}}(X)], [f_{\boldsymbol{b}}(X)])$
$\mathtt{w} : (f_{\boldsymbol{a}}(X), f_{\boldsymbol{b}}(X))$
// pre-processing
begin
  I computes $f_M(X) \leftarrow \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} M_{[i,j]} X^{in+j}$
  I sends $f_M(X)$ of degree at most $mn - 1$ to P and V
begin
  V samples $\alpha \xleftarrow{\$} \mathbb{F}$ and sends α to P
  P computes $r(X) \leftarrow f_M(X) \bmod X^n - \alpha$
  P sends $r(X)$ of degree at most $n - 1$ to V
  // ModReduce
  begin
    P computes $q(X)$ such that $f_M(X) = q(X) \cdot (X^n - \alpha) + r(X)$
    P sends $q(X)$ of degree at most $(m-1)n$ to V
    V samples $z \xleftarrow{\$} \mathbb{F}\backslash\{0\}$ and queries $[f_M(X)]$, $[q(X)]$, and $[r(X)]$ in z
    V receives $y_f = f_M(z)$, $y_q = q(z)$, and $y_r = r(z)$
    V tests $y_f \stackrel{?}{=} y_q \cdot (z^n - \alpha) + y_r$
  V queries $[f_{\boldsymbol{b}}(X)]$ in α and receives $y_{\boldsymbol{\alpha}^\intercal \boldsymbol{b}} = f_{\boldsymbol{b}}(\alpha)$
  // InnerProduct
  begin
    P computes $h(X) \leftarrow r(X) \cdot f_{\boldsymbol{a}}(X^{-1}) \cdot X^{\mathtt{n}-1}$
    P computes $\bar{h}(X) = \sum_{i=0}^{2n-1} \bar{h}_i X^i$ with $\bar{h}_i \leftarrow \frac{h_i}{\gamma^{n-1} - \gamma^i}$ for $i \neq n-1$ and $\bar{h}_{n-1} \xleftarrow{\$} \mathbb{F}$
    P sends $\bar{h}(X)$ of degree at most $2n - 1$ to V
    V samples $z \xleftarrow{\$} \mathbb{F}\backslash\{0\}$ and queries $([r(X)], [f_{\boldsymbol{a}}(X)], [\bar{h}(X)], [\bar{h}(X)])$ in $(z, z^{-1}, z, \gamma \cdot z)$
    V receives $y_r = f(z)$, $y_a = f_{\boldsymbol{a}}(z^{-1})$, $y_h = \bar{h}(z)$, and $y_h^* = \bar{h}(\gamma \cdot z)$
    V tests $y_h \cdot \gamma^{n-1} - y_h^* \stackrel{?}{=} y_r \cdot y_a \cdot z^{n-1} - y_{\boldsymbol{\alpha}^\intercal \boldsymbol{b}}$

Protocol 15: DenseMVP, unrolled ModReduce

description: decides $\mathcal{L}(\mathcal{R}_{\text{mvp}})$
inputs: $\mathtt{i} : (m, n, M)$
$\mathtt{x} : ([f_{\boldsymbol{a}}(X)], [f_{\boldsymbol{b}}(X)])$
$\mathtt{w} : (f_{\boldsymbol{a}}(X), f_{\boldsymbol{b}}(X))$
// pre-processing
begin
  I computes $f_M(X) \leftarrow \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} M_{[i,j]} X^{in+j}$
  I sends $f_M(X)$ of degree at most $mn-1$ to P and V
begin
  V samples $\alpha \xleftarrow{\$} \mathbb{F}$ and sends α to P
  P computes $r(X) \leftarrow f_M(X) \bmod X^n - \alpha$
  // ModReduce
  begin
    P computes $q(X)$ such that $f_M(X) = q(X) \cdot (X^n - \alpha) + r(X)$
    P sends $q(X)$ of degree at most $(m-1)n$ to V
    V samples $z \xleftarrow{\$} \mathbb{F} \backslash \{0\}$ and queries $[f_M(X)]$ and $[q(X)]$ in z
    V receives $y_f = f_M(z)$, and $y = q(z)$
    V sets $y_r \leftarrow y_f - y_q \cdot (z^n - \alpha)$
  V queries $[f_{\boldsymbol{b}}(X)]$ in α and receives $y_{\boldsymbol{\alpha}^\intercal \boldsymbol{b}} = f_{\boldsymbol{b}}(\alpha)$
  // InnerProduct
  begin
    P computes $h(X) \leftarrow r(X) \cdot f_{\boldsymbol{a}}(X^{-1}) \cdot X^{\mathtt{n}-1}$
    P computes $\bar{h}(X) = \sum_{i=0}^{2n-1} \bar{h}_i X^i$ with $\bar{h}_i \leftarrow \frac{h_i}{\gamma^{n-1} - \gamma^i}$ for $i \neq n-1$ and $\bar{h}_{n-1} \xleftarrow{\$} \mathbb{F}$
    P sends $\bar{h}(X)$ of degree at most $2n-1$ to V
    V queries $([f_{\boldsymbol{a}}(X)], [\bar{h}(X)], [\bar{h}(X)])$ in $(z^{-1}, z, \gamma \cdot z)$
    V receives $y_a = f_{\boldsymbol{a}}(z^{-1})$, $y_h = \bar{h}(z)$, and $y_h^* = \bar{h}(\gamma \cdot z)$
    // Merged test
    V tests $y_h \cdot \gamma^{n-1} - y_h^* \stackrel{?}{=} y_r \cdot y_a \cdot z^{n-1} - y_{\boldsymbol{\alpha}^\intercal \boldsymbol{b}}$

Protocol 16: DenseMVP, eliminated $r(X)$; optimized

C.2 Batched SparseBiEval

BiVandermonde Vector. We batch two VandermondeVector protocols into a single protocol for efficiency. The BiVandermondeVector protocol realizes the following relation

$$\mathcal{R}_{\text{bvv}} = \left\{ (\mathbb{i}, \mathbb{x}, \mathbb{w}) \;\middle|\; \begin{array}{l} \mathbb{i} = (H, K, \varphi(h), \{a_k\}_{k=0}^{K-1}, \{b_k\}_{k=0}^{K-1}) \\ \mathbb{x} = (z, [f_{\hat{\mathbf{z}}}(X)]) \\ \mathbb{w} = f_{\hat{\mathbf{z}}}(X) \\ R_a = \begin{pmatrix} \varphi(a_0)^0 & \varphi(a_1)^0 & \cdots \\ \varphi(a_0)^1 & \varphi(a_1)^1 & \cdots \\ \vdots & \vdots & \ddots \end{pmatrix} \\ R_b = \begin{pmatrix} \varphi(b_0)^0 & \varphi(b_1)^0 & \cdots \\ \varphi(b_0)^1 & \varphi(b_1)^1 & \cdots \\ \vdots & \vdots & \ddots \end{pmatrix} \\ R = \begin{pmatrix} R_a & 0 \\ 0 & R_b \end{pmatrix} \\ f_{\hat{\mathbf{z}}}(X) = \sum_{i=0}^{2H-1} \hat{z}_i X^i \\ \mathbf{z} = (1\,,\, z\,,\, z^2\,,\, \cdots\,,\, z^{2K-1})^{\mathsf{T}} \\ \hat{\mathbf{z}} = R\mathbf{z} \end{array} \right\} . \tag{81}$$

Theorem 13 (Security of BiVandermondeVector). *Protocol* BiVandermondeVector *is a Polynomial IOP for* $\mathcal{R}_{\text{bvv}}$ *with completeness and soundness with soundness error* $\sigma \leq \frac{2H+10K-5}{|\mathbb{F}|-1}$.

Proof. Consider the following sequences of equations.

$$\hat{\mathbf{z}} = R\mathbf{z} \tag{82}$$

$$\boldsymbol{\delta}^{\mathsf{T}}\hat{\mathbf{z}} = \boldsymbol{\delta}^{\mathsf{T}}R\mathbf{z} \tag{83}$$

$$f_{\hat{\mathbf{z}}}(\delta) = f_{\hat{\boldsymbol{\delta}}}(z) \tag{84}$$

$$y_1 = y_0 \;\;, \tag{85}$$

and

$$\forall k \in \{0, \ldots, K-1\} \;.\; \left(\sum_{h=0}^{H-1} (\delta \cdot \varphi(a_k))^h\right) \cdot (1 - \delta \cdot \varphi(a_k)) = 1 - (\delta \cdot \varphi(a_k))^H \tag{86}$$

$$\left(\sum_{h=0}^{H-1} \delta^{H+h} \cdot \varphi(a_k)^h\right) \cdot (1 - \delta \cdot \varphi(b_k)) = \delta^H \cdot \left(1 - (\delta \cdot \varphi(b_k))^H\right)$$

$$\left(\boldsymbol{\delta}^{\mathsf{T}}R\right) \circ \left((1 - \delta \cdot \varphi(a_k))_{k=0}^{K-1} \,\|\, (1 - \delta \cdot \varphi(b_k))_{k=0}^{K-1}\right) =$$

$$\left(1 - (\delta \cdot \varphi(a_k))^H\right)_{k=0}^{K-1} \Big\| \left(1 - (\delta \cdot \varphi(b_k))^H\right)_{k=0}^{K-1} \cdot \delta^H \tag{87}$$

description: decides $\mathcal{L}(\mathcal{R}_{\text{bvv}})$
inputs: $\mathbb{i}$: $H, K, \varphi(h), \{a_k\}_{k=0}^{K-1}, \{b_k\}_{k=0}^{K-1}$
$\mathbb{x}$: $z, [f_{\hat{\boldsymbol{z}}}(X)]$
$\mathbb{w}$: $f_{\hat{\boldsymbol{z}}}(X)$
// pre-processing
begin
I computes $f_{\varphi(a_k)}(X) \leftarrow \sum_{k=0}^{K-1} \varphi(a_k) X^k$, $f_{\varphi(a_k)^H}(X) \leftarrow \sum_{k=0}^{K-1} \varphi(a_k)^H X^k$, $f_{\varphi(b_k)}(X) \leftarrow \sum_{k=0}^{K-1} \varphi(b_k) X^k$ and $f_{\varphi(b_k)^H}(X) \leftarrow \sum_{k=0}^{K-1} \varphi(b_k)^H X^k$
I sends $f_{\varphi(a_k)}(X)$, $f_{\varphi(a_k)^H}(X)$, $f_{\varphi(b_k)}(X)$ and $f_{\varphi(b_k)^H}(X)$, all of degree at most $K-1$, to P and V
// online
begin
V samples $\delta \xleftarrow{\$} \mathbb{F}\backslash\{0\}$ and sends δ to P
P computes $\boldsymbol{\delta} \leftarrow (\delta^0, \delta^1, \ldots, \delta^{2H-1})^\mathsf{T}$, $\hat{\boldsymbol{\delta}} \leftarrow R^\mathsf{T}\boldsymbol{\delta}$, and $f_{\hat{\boldsymbol{\delta}}}(X) \leftarrow \sum_{i=0}^{2K-1} \hat{\delta}_i X^i$
P sends $f_{\hat{\boldsymbol{\delta}}}(X)$ of degree at most $2K-1$ to V
V queries $[f_{\hat{\boldsymbol{\delta}}}(X)]$ in z and receives $y_0 = f_{\hat{\boldsymbol{\delta}}}(z)$
V queries $[f_{\hat{\boldsymbol{z}}}(X)]$ in δ and receives $y_1 = f_{\hat{\boldsymbol{z}}}(\delta)$
V checks $y_1 \stackrel{?}{=} y_0$
P and V run Hadamard with $\mathbb{i}^{(1)} = 2K-1$,
$\mathbb{x}^{(1)} = ([f_{\hat{\boldsymbol{\delta}}}(X)], [\delta \cdot f_{\varphi(a_k)}(X) - f_I(X) + (\delta \cdot f_{\varphi(b_k)}(X) - f_I(X)) \cdot \delta^H \cdot X^K]$, $[\delta^H \cdot f_{\varphi(a_k)^H}(X) - f_I(X) + (\delta^H \cdot f_{\varphi(b_k)^H}(X) - f_I(X)) \cdot \delta^H \cdot X^K])$,
$\mathbb{w}^{(1)} = (f_{\hat{\boldsymbol{\delta}}}(X), \delta^H \cdot f_{\varphi(a_k)}(X) - f_I(X) + (\delta^H \cdot f_{\varphi(b_k)}(X) - f_I(X)) \cdot \delta^H \cdot X^K$, $\delta^H \cdot f_{\varphi(a_k)^H}(X) - f_I(X) + (\delta^H \cdot f_{\varphi(b_k)^H}(X) - f_I(X)) \cdot \delta^H \cdot X^K)$, where V simulates $[f_{\varphi(a_k)}(\delta X) - f_I(X) + (f_{\varphi(b_k)}(\delta X) - f_I(X)) \cdot \delta^H \cdot X^K]$ and $[f_{\varphi(a_k)^H}(\delta X) - f_I(X) + (f_{\varphi(b_k)^H}(\delta X) - f_I(X)) \cdot \delta^H \cdot X^K]$ using the oracles $[f_{\varphi(a_k)}(X)], [f_{\varphi(a_k)^H}(X)], [f_{\varphi(b_k)}(X)], [f_{\varphi(b_k)^H}(X)]$ and the known scalar δ, in combination with computing $f_I(X) = \sum_{k=0}^{2K-1} X^k$ locally

Protocol 17: BiVandermondeVector

$$f_{\hat{\boldsymbol{\delta}}}(X) \circ \left(\delta \cdot f_{\varphi(a_k)}(X) - f_I(X) + (\delta \cdot f_{\varphi(b_k)}(X) - f_I(X)) \cdot \delta^H \cdot X^K\right) = \delta^H \cdot f_{\varphi(a_k)^H}(X) - f_I(X) + \left(\delta^H \cdot f_{\varphi(b_k)^H}(X) - f_I(X)\right) \cdot \delta^H \cdot X^K \quad (88)$$

$$\begin{aligned}(\mathbb{i}^{(1)}, \mathbb{x}^{(1)}) = (2K-1, [f_{\hat{\boldsymbol{\delta}}}(X)], \\ [\delta \cdot f_{\varphi(a_k)}(X) - f_I(X) + (\delta \cdot f_{\varphi(b_k)}(X) - f_I(X)) \cdot \delta^H \cdot X^K], \\ [\delta^H \cdot f_{\varphi(a_k)^H}(X) - f_I(X) + (\delta^H \cdot f_{\varphi(b_k)^H}(X) - f_I(X)) \cdot \delta^H \cdot X^K)] \\ \in \mathcal{L}(\mathcal{R}_{\text{hadamard}}) \ . \end{aligned} \quad (89)$$

Completeness follows from the sequences of implications a) (82) $\Rightarrow$ (83) $\Leftrightarrow$ (84) $\Leftrightarrow$ (85), and b) (86) $\Leftrightarrow$ (87) $\Leftrightarrow$ (88) $\Rightarrow$ (89). Sequence (a) holds for any matrix R. Sequence (b) starts with the equation for two geometric sums derived from the matrix R_a and R_b as defined as in Eqn. 26.

For soundness, consider when the reverse implications fail. There are two such cases:

- (82) $\not\Leftarrow$ (83). By the Schwartz-Zippel lemma, the probability of this event is bounded by $\frac{2H-1}{|\mathbb{F}|-1}$.
- (88) $\not\Leftarrow$ (89). The probability of this event is captured by the soundness error of Hadamard, $\sigma_{\mathsf{Hadamard}} \leq \frac{10K-4}{|\mathbb{F}|-1}$.

By the Union Bound, the soundness error of BiVandermondeVector is bounded by $\sigma \leq \frac{2H+10K-5}{|\mathbb{F}|-1}$. □

BiLagrange Vector. We batch two instances of LagrangeVector protocol into a single protocol for efficiency. The BiLagrangeVector protocol realizes the following relation

$$\mathcal{R}_{\text{lv}} = \left\{ (\mathbb{i}, \mathbb{x}, \mathbb{w}) \;\middle|\; \begin{array}{l} \mathbb{i} = (H, \varphi(h)) \\ \mathbb{x} = (u, v, [f_{\hat{\mathbf{u}}\hat{\mathbf{v}}}(X)]) \\ \mathbb{w} = f_{\hat{\mathbf{u}}\hat{\mathbf{v}}}(X) \\ L = \begin{pmatrix} \mathcal{L}_{\varphi(0),0} & \mathcal{L}_{\varphi(0),1} & \cdots \\ \mathcal{L}_{\varphi(1),0} & \mathcal{L}_{\varphi(1),1} & \cdots \\ \vdots & \vdots & \ddots \end{pmatrix} \\ f_{\hat{\mathbf{u}}\hat{\mathbf{v}}}(X) = \sum_{i=0}^{H-1} \hat{u}_i X^i + \sum_{i=0}^{H-1} \hat{v}_i X^{H+i} \\ \mathbf{u} = (1, u, u^2, \cdots, u^{H-1})^\mathsf{T} \\ \mathbf{v} = (1, v, v^2, \cdots, v^{H-1})^\mathsf{T} \\ \hat{\mathbf{u}} = \mathbf{u}^\mathsf{T} L \quad \hat{\mathbf{v}} = \mathbf{v}^\mathsf{T} L \end{array} \right\} . \quad (90)$$

Theorem 14 (Security of BiLagrangeVector). *Protocol* BiLagrangeVector *is a Polynomial IOP for* $\mathcal{R}_{\text{blv}}$ *with completeness and soundness with soundness error* $\sigma \leq \frac{7H-5}{|\mathbb{F}|-1}$.

description: decides $\mathcal{L}(\mathcal{R}_{\text{blv}})$
inputs: $\mathbb{i}$: $H, \varphi(h)$
$\mathbb{x}$: $u, v, [f_{\mathbf{u\hat{v}}}(X)]$
$\mathbb{w}$: $f_{\mathbf{u\hat{v}}}(X)$
// pre-processing
begin
I computes $f_{\varphi(h)}(X) \leftarrow \sum_{h=0}^{H-1} \varphi(h) X^h$, $Z_{\mathcal{H}}(X) \leftarrow \prod_{h=0}^{H-1}(X - \varphi(h))$ and $f_{\mathcal{H}}(X) \leftarrow \sum_{h=0}^{H-1} \left(\prod_{i \in \{0 \dots H-1\} \setminus \{h\}} (\varphi(h) - \varphi(i))\right)^{-1} \cdot X^h$
I sends $f_{\varphi(h)}(X)$, $f_{\mathcal{H}}(X)$, both of degree at most $H-1$, and $Z_{\mathcal{H}}(X)$ of degree at most H, to P and V
// online
begin
V samples $\gamma \xleftarrow{\$} \mathbb{F} \setminus \{0\}$ and sends γ to P
P computes $\boldsymbol{\gamma} \leftarrow (\gamma^0, \gamma^1, \dots, \gamma^{H-1})^{\mathsf{T}}$, $\hat{\boldsymbol{\gamma}} \leftarrow L\boldsymbol{\gamma}$, and $f_{\hat{\boldsymbol{\gamma}}}(X) \leftarrow \sum_{i=0}^{H-1} \hat{\gamma}_i X^i$
P sends $f_{\hat{\boldsymbol{\gamma}}}(X)$ of degree at most $H-1$ to V
V queries $[f_{\hat{\boldsymbol{\gamma}}}(X)]$ in u, v and receives $y_0 = f_{\hat{\boldsymbol{\gamma}}}(u), y_1 = f_{\hat{\boldsymbol{\gamma}}}(v)$
V queries $[f_{\mathbf{u\hat{v}}}(X)]$ in γ and receives $y_2 = f_{\mathbf{u\hat{v}}}(\gamma)$
V checks $y_2 \stackrel{?}{=} y_0 + \gamma^H \cdot y_1$
V queries $[Z_{\mathcal{H}}(X)]$ in γ and receives $w = Z_{\mathcal{H}}(\gamma)$
P and V run Hadamard with $\mathbb{i}^{(1)} = H - 1$,
$\mathbb{x}^{(1)} = ([f_{\hat{\boldsymbol{\gamma}}}(X)], [\gamma f_I(X) - f_{\varphi(h)}(X)], [w f_{\mathcal{H}}(X)])$,
$\mathbb{w}^{(1)} = (f_{\hat{\boldsymbol{\gamma}}}(X), \gamma f_I(X) - f_{\varphi(h)}(X), w f_{\mathcal{H}}(X))$, where V simulates $[\gamma f_I(X) - f_{\varphi(h)}(X)], [w f_{\mathcal{H}}(X)]$ using the oracles $[f_{\varphi(h)}(X)], [f_{\mathcal{H}}(X)]$ and the known scalar γ, in combination with computing $f_I(X) = \sum_{h=0}^{H-1} X^h$ locally

Protocol 18: BiLagrangeVector

Proof. Consider the following sequences of equations.

$$\hat{\mathbf{u}}^{\mathsf{T}} = \mathbf{u}^{\mathsf{T}} L \qquad \hat{\mathbf{v}}^{\mathsf{T}} = \mathbf{v}^{\mathsf{T}} L \tag{91}$$

$$\hat{\mathbf{u}}^{\mathsf{T}}\boldsymbol{\gamma} + \hat{\mathbf{v}}^{\mathsf{T}}\boldsymbol{\gamma} \cdot \gamma^H = \mathbf{u}^{\mathsf{T}} L\boldsymbol{\gamma} + \mathbf{v}^{\mathsf{T}} L\boldsymbol{\gamma} \cdot \gamma^H \tag{92}$$

$$f_{\hat{\mathbf{u}}\hat{\mathbf{v}}}(\gamma) = f_{\hat{\boldsymbol{\gamma}}}(u) + f_{\hat{\boldsymbol{\gamma}}}(v) \cdot \gamma^H \tag{93}$$

$$y_2 = y_0 + \gamma^H \cdot y_1 \ , \tag{94}$$

and

$$\forall h \in \{0, \ldots, H-1\} \,.\, \mathcal{L}_h(\gamma) = \prod_{\substack{i=0 \\ i \neq h}}^{H-1} \frac{\gamma - \varphi(i)}{\varphi(h) - \varphi(i)} \tag{95}$$

$$\forall h \in \{0, \ldots, H-1\} \,.\, \mathcal{L}_h(\gamma) \cdot (\gamma - \varphi(h)) = \frac{Z_{\mathcal{H}}(\gamma)}{\prod_{\substack{i=0 \\ i \neq h}}^{H-1} (\varphi(h) - \varphi(i))} \tag{96}$$

$$(L\boldsymbol{\gamma}) \circ (\gamma - \varphi(h))_{h=0}^{H-1} = Z_{\mathcal{H}}(\gamma) \left(\left(\prod_{i \in \{0 \ldots H-1\} \setminus \{h\}} (\varphi(h) - \varphi(i)) \right)^{-1} \right)_{h=0}^{H-1} \tag{97}$$

$$f_{\hat{\boldsymbol{\gamma}}}(X) \circ (\gamma f_I(X) - f_{\varphi}(X)) = w f_{\mathcal{H}}(X) \tag{98}$$

$$(\mathbb{i}^{(1)}, \mathbb{x}^{(1)}) = \tag{99}$$

$$(H-1, [f_{\hat{\boldsymbol{\gamma}}}(X)], [\gamma f_I(X) - f_{\varphi(h)}(X)], [w f_{\mathcal{H}}(X)]) \in \mathcal{L}(\mathcal{R}_{\text{hadamard}}) \ .$$

Completeness follows from the sequences of implications a) (91) $\Rightarrow$ (92) $\Leftrightarrow$ (93) $\Leftrightarrow$ (94), and b) (95) $\Leftrightarrow$ (96) $\Leftrightarrow$ (97) $\Leftrightarrow$ (98) $\Rightarrow$ (99). Sequence (a) holds for any matrix L. Sequence (b) starts with the definition of Lagrange basis polynomials, and holds when the rows of L are exactly the coefficient vectors of the Lagrange basis polynomials. Recall that Lagrange basis polynomial indexed by h takes the value 1 in $\varphi(h)$ and 0 in all other points of $\mathcal{H}$.

For soundness, consider when the reverse implications fail. There are two such cases:

- (91) $\not\Leftarrow$ (92). By the Schwartz-Zippel lemma, the probability of this event is bounded by $\frac{2H-1}{|\mathbb{F}|-1}$.
- (98) $\not\Leftarrow$ (99). The probability of this event is captured by the soundness error of $\mathsf{Hadamard}$, $\sigma_{\mathsf{Hadamard}} \leq \frac{5H-4}{|\mathbb{F}|-1}$.

By the Union Bound, the soundness error of $\mathsf{BiLagrangeVector}$ is bounded by $\sigma \leq \frac{7H-5}{|\mathbb{F}|-1}$. □

Batched SparseBiEval Using the above two protocols, we obtain an improved version of the $\mathsf{SparseBiEval}$ protocol that saves four polynomial oracles at the relatively minor expense of doubling their degree. Protocol 19 verifies the relation $\mathcal{R}_{\text{sbi}}$ defined by Eqn. 67.

description: decides $\mathcal{L}(\mathcal{R}_{\text{sbe}})$
inputs: $\mathbb{i}$: $H, K, \{a_k\}_{k=0}^{K-1}, \{b_k\}_{k=0}^{K-1}, \{c_k\}_{k=0}^{K-1}$
$\mathbb{x}$: u, v, w
$\mathbb{w}$: $\varnothing$

// pre-processing
begin
- I selects any mapping $\varphi : \mathbb{N} \to \mathbb{F}$ such that $\{0, \ldots, H-1\}$ are mapped to distinct elements
- I runs BiVandermondeVector.I with $\mathbb{i}^{(1)} = (H, K, \{a_k\}_{k=0}^{K-1}, \{b_k\}_{k=0}^{K-1})$
- I runs BiLagrangeVector.I with $\mathbb{i}^{(2)} = (H, \varphi(h))$
- I computes $f_c(X) \leftarrow \sum_{k=0}^{K-1} c_k X^k$
- I sends $f_c(X)$ of degree at most $K-1$ to P and V

// online
begin
- P computes $f_{u\|v}(X) \leftarrow \sum_{k=0}^{K-1} u^{a_k} X^k + \sum_{k=0}^{K-1} v^{b_k} X^{k+K}$ and $f_{uv}(X) \leftarrow \sum_{k=0}^{K-1} u^{a_k} v^{b_k} X^k$
- P sends $f_{u\|v}(X)$ of degree at most $2K-1$ and $f_{uv}(X)$ of degree at most $K-1$, to V
- **Batched Lagrange vector:**
  - P computes $\mathbf{u} \leftarrow (1, u, u^2, \cdots, u^{H-1})^\mathsf{T}$, $\hat{\mathbf{u}} \leftarrow \mathbf{u}^\mathsf{T} L$, $\mathbf{v} \leftarrow (1, v, v^2, \cdots, v^{H-1})^\mathsf{T}$, $\hat{\mathbf{v}} \leftarrow \mathbf{v}^\mathsf{T} L$
  - P computes $f_{\hat{\mathbf{u}}\hat{\mathbf{v}}}(X) \leftarrow \sum_{i=0}^{H-1} \hat{u}_i X^i + \sum_{i=0}^{H-1} \hat{v}_i X^{i+H}$
  - P sends $f_{\hat{\mathbf{u}}\hat{\mathbf{v}}}(X)$ of degree at most $2H-1$ to V
  - P and V run BiLagrangeVector with $\mathbb{i}^{(2)} = (H, \varphi(h))$, $\mathbb{x}^{(2)} = (u, v, [f_{\hat{\mathbf{u}}\hat{\mathbf{v}}}(X)])$, and $\mathbb{w}^{(2)} = f_{\hat{\mathbf{u}}\hat{\mathbf{v}}}(X)$
- **Batched Vandermonde vector:**
  - V samples $y \xleftarrow{\$} \mathbb{F}\backslash\{0\}$ and sends y to P
  - P computes $\mathbf{y} \leftarrow (1, y, y^2, \cdots, y^{2K-1})$, $\hat{\mathbf{y}} \leftarrow R\mathbf{y}$ where R is as defined in Eqn. 81
  - P computes $f_{\hat{\mathbf{y}}}(X) \leftarrow \sum_{i=0}^{2K-1} \hat{y}_i X^i$
  - P sends $f_{\hat{\mathbf{y}}}(X)$ of degree at most $2K-1$ to V
  - P and V run BiVandermondeVector with $\mathbb{i}^{(1)} = (H, K, \{a_k\}_{k=0}^{K-1}, \{b_k\}_{k=0}^{K-1})$, $\mathbb{x}^{(1)} = (y, [f_{\hat{\mathbf{y}}}(X)])$, and $\mathbb{w}^{(1)} = f_{\hat{\mathbf{y}}}(X)$
- V queries $[f_{u\|v}(X)]$ in y and receives $s = f_{u\|v}(y)$
- P and V run InnerProduct with $\mathbb{i}^{(3)} = 2H-1$, $\mathbb{x}^{(3)} = ([f_{\hat{\mathbf{u}}\hat{\mathbf{v}}}(X)], [f_{\hat{\mathbf{y}}}(X)], s)$, and $\mathbb{w}^{(3)} = (f_{\hat{\mathbf{u}}\hat{\mathbf{v}}}(X), f_{\hat{\mathbf{y}}}(X))$
- P and V run Hadamard with $\mathbb{i}^{(4)} = 3K-1$, $\mathbb{x}^{(4)} = ([f_{u\|v}(X)], [f_{u\|v}(X) \cdot X^K], [f_{uv}(X) \cdot X^K])$, and $\mathbb{w}^{(4)} = (f_{u\|v}(X), f_{u\|v}(X) \cdot X^K, f_{uv}(X) \cdot X^K)$ where V simulates $[f_{u\|v}(X) \cdot X^K]$ and $[f_{uv}(X) \cdot X^K]$ using $[f_{u\|v}(X)]$ and $[f_{uv}(X)]$ respectively
- P and V run InnerProduct with $\mathbb{i}^{(5)} = K-1$, $\mathbb{x}^{(5)} = ([f_{uv}(X)], [f_c(X)], w)$, and $\mathbb{w}^{(5)} = (f_{uv}(X), f_c(X))$

Protocol 19: BatchedSparseBiEval

Theorem 15 (Security of BatchedSparseBiEval). *Protocol* BatchedSparseBiEval *is a Polynomial IOP for* $\mathcal{R}_{\text{sbi}}$ *with completeness and soundness with soundness error* $\sigma = \frac{17H+31K-21}{|\mathbb{F}|-1}$.

Proof. Consider the following sequences of equations.

$$f_{u\|v}(X) = \sum_{k=0}^{K-1} u^{a_k} X^k + \sum_{k=0}^{K-1} v^{b_k} X^{k+K} \tag{100}$$

$$f_{u\|v}(y) = \sum_{k=0}^{K-1} u^{a_k} y^k + \sum_{k=0}^{K-1} v^{b_k} y^{k+K} \tag{101}$$

$$f_{u\|v}(y) = \mathbf{u}^{\mathsf{T}}(\mathbf{e}_{a_0}, \cdots, \mathbf{e}_{a_{K-1}})\mathbf{y}_{[0:K]} + \mathbf{v}^{\mathsf{T}}(\mathbf{e}_{b_0}, \cdots, \mathbf{e}_{b_{K-1}})\mathbf{y}_{[K:2K]} \tag{102}$$

$$f_{u\|v}(y) = \mathbf{u}^{\mathsf{T}} L R_a \mathbf{y}_{[0:K]} + \mathbf{v}^{\mathsf{T}} L R_b \mathbf{y}_{[K:2K]} \tag{103}$$

$$f_{u\|v}(y) = \hat{\mathbf{u}}^{\mathsf{T}} R_a \mathbf{y}_{[0:K]} + \hat{\mathbf{v}}^{\mathsf{T}} R_b \mathbf{y}_{[K:2K]} \tag{104}$$

$$f_{u\|v}(y) = (\hat{\mathbf{u}}^{\mathsf{T}} \| \hat{\mathbf{v}}^{\mathsf{T}}) R \mathbf{y} \tag{105}$$

$$f_{u\|v}(y) = (\hat{\mathbf{u}}^{\mathsf{T}} \| \hat{\mathbf{v}}^{\mathsf{T}}) \cdot \hat{\mathbf{y}} \tag{106}$$

$$(\mathbb{i}, \mathbb{x}) = (2H-1, ([f_{\hat{\mathbf{u}}\hat{\mathbf{v}}}(X)], [f_{\hat{\mathbf{y}}}(X)], s)) \in \mathcal{R}_{\text{ip}} \tag{107}$$

and

$$f(u, v) = w \tag{108}$$

$$\sum_{k=0}^{K-1} c_k u^{a_k} v^{b_k} = w \tag{109}$$

$$\mathsf{coeffs}(f_{uv}(X)) \cdot \mathsf{coeffs}(f_c(X)) = w \tag{110}$$

$$\big(\mathsf{coeffs}(f_{u\|v}(X)) \circ \mathsf{coeffs}(f_{u\|v}(X) \cdot X^K)\big)_{[K:2K]} \cdot \mathsf{coeffs}(f_c(X)) = w \tag{111}$$

$$\big(\mathsf{coeffs}(f_{u\|v}(X)) \circ \mathsf{coeffs}(f_{u\|v}(X) \cdot X^K)\big)_{[K:2K]} \cdot (c_k)_{k=0}^{K-1} = w \tag{112}$$

$$\Big((u^{a_k})_{k=0}^{K-1} \circ \big(v^{b_k}\big)_{k=0}^{K-1}\Big) \cdot (c_k)_{k=0}^{K-1} = w \tag{113}$$

Completeness follows from the sequences of implications: a) (100) $\Rightarrow$ (101) $\Leftrightarrow$ (102) $\Leftrightarrow$ (103) $\Rightarrow$ (104) $\Leftrightarrow$ (105) $\Rightarrow$ (106) $\Rightarrow$ (107) and b) (108) $\Leftrightarrow$ (109) $\Rightarrow$ (110) $\Rightarrow$ (111) $\Leftrightarrow$ (112) $\Leftrightarrow$ (113). For soundness, consider when the reverse implications fail.

- (100) $\not\Leftarrow$ (101). This event happens with probability at most $\frac{2K-1}{|\mathbb{F}|-1}$ due to the Schwartz-Zippel lemma.
- (103) $\not\Leftarrow$ (104). This event implies that BiLagrangeVector succeeded despite being run on a false instance. This happens with probability at most equal to the soundness error of that protocol, $\sigma_{\mathsf{BiLagrangeVector}} \leq \frac{7H-5}{|\mathbb{F}|-1}$.
- (105) $\not\Leftarrow$ (106). This event implies that BiVandermondeVector succeeded despite being run on a false instance. This happens with probability at most equal to the soundness error of that protocol, $\sigma_{\mathsf{BiVandermondeVector}} \leq \frac{2H+10K-5}{|\mathbb{F}|-1}$.

– (106) $\not\Leftarrow$ (107). This event implies that InnerProduct succeeded despite being run on a false instance. This happens with probability at most equal to the soundness error of that protocol, $\sigma_{\mathsf{InnerProduct}} \leq \frac{8H-3}{|\mathbb{F}|-1}$.
– (109) $\not\Leftarrow$ (110). This event happens with probability at most equal to the soundness error of InnerProduct, $\sigma_{\mathsf{InnerProduct}} \leq \frac{4K-3}{|\mathbb{F}|-1}$.
– (110) $\not\Leftarrow$ (111). This event happens with probability at most equal to the soundness error of Hadamard, $\sigma_{\mathsf{Hadamard}} \leq \frac{15K-4}{|\mathbb{F}|-1}$.

By the Union Bound, the soundness error of BatchedSparseBiEval is bounded by $\sigma \leq \frac{17H+31K-21}{|\mathbb{F}|-1}$. □

D Call Graphs and Statistics

Table 3. Callgraph Statistics for Hadamard

protocol	preprocessed	input	new	oracles queried	queried in
Hadamard		$f_{\boldsymbol{a}}, f_{\boldsymbol{b}}, f_{\boldsymbol{c}}$		$f_{\boldsymbol{c}}$	α
- InnerProduct		$f_{\boldsymbol{a}}, f_{\boldsymbol{b}}$	$\bar{h}_0$		
-				$\bar{h}_0$	$z_1, \gamma z_1$
-				$f_{\boldsymbol{a}}$	αz_1
-				$f_{\boldsymbol{b}}$	$(z_1)^{-1}$
total:	0	3	1	5	5

Table 4. Callgraph Statistics for DenseMVP

protocol	preprocessed	input	new	oracles queried	queried in
DenseMVP	f_M	$f_{\boldsymbol{a}}, f_{\boldsymbol{b}}$	q	$f_{\boldsymbol{b}}$	α_0
- InnerProduct		$q, f_M, f_{\boldsymbol{a}}$	$\bar{h}_1$		
-				$\bar{h}_1$	$z_2, \gamma z_2$
-				q	z_2
-				l	z_2
-				f_M	z_2
-				$f_{\boldsymbol{a}}$	$(z_2)^{-1}$
total:	1	2	2	6	4

Table 5. Callgraph Statistics for DenseClaymore

protocol	preprocessed	input	new	oracles queried	queried in
DenseClaymore	f_M	$f_{\mathbf{x}}$	f_{wl}, f_{wr}, f_{wo}		
- DenseMVP	f_M	$f_{wl}, f_{wr}, f_{wo}, f_{\mathbf{x}}$	q		
-				$f_{\mathbf{x}}$	α_0
- - InnerProduct		$q, f_M, f_{wl}, f_{wr}, f_{wo}$	$\bar{h}_1$		
- -				$\bar{h}_1$	$z_2, \gamma z_2$
- -				q	z_2
- -				l	z_2
- -				f_M	z_2
- -				f_{wl}	$(z_2)^{-1}$
- -				f_{wr}	$(z_2)^{-1}$
- -				f_{wo}	$(z_2)^{-1}$
- Hadamard		f_{wl}, f_{wr}, f_{wo}			
-				f_{wo}	α
- - InnerProduct		f_{wl}, f_{wr}	$\bar{h}_3$		
- -				$\bar{h}_3$	$z_4, \gamma z_4$
- -				f_{wl}	αz_4
- -				f_{wr}	$(z_4)^{-1}$
total:	1	1	6	13	9
with BatchedInnerProduct:	1	1	5	11	6
and witness concatenation:	1	1	4	10	6

Table 6. Callgraph Statistics for VandermondeVector

protocol	preprocessed	input	new	oracles queried	queried in
VandermondeVector	$f_{\varphi(a_k)}, f_{\varphi(a_k)^H}$	$f_{\hat{\mathbf{x}}}$	$f_{\hat{\boldsymbol{\delta}}}$		
				$f_{\hat{\mathbf{x}}}$	δ_0
				$f_{\hat{\boldsymbol{\delta}}}$	z_1
- Hadamard		$f_{\hat{\boldsymbol{\delta}}}, f_{\varphi(a_k)}, f_{\varphi(a_k)^H}$			
-				$f_{\varphi(a_k)^H}$	$\delta_0^H \alpha$
-				f_I	α
- - InnerProduct		$f_{\hat{\boldsymbol{\delta}}}, f_{\varphi(a_k)}$	$\bar{h}_2$		
- -				$\bar{h}_2$	$z_3, \gamma z_3$
- -				$f_{\hat{\boldsymbol{\delta}}}$	αz_3
- -				$f_{\varphi(a_k)}$	$(\delta_0 z_3)^{-1}$
- -				f_I	$(z_3)^{-1}$
total:	2	1	2	7	7

Table 7. Callgraph Statistics for LagrangeVector

protocol	preprocessed	input	new	oracles queried	queried in
LagrangeVector	$f_{\varphi(h)}, f_{\mathcal{H}}, Z_{\mathcal{H}}$	$f_{\hat{\mathbf{x}}}$	$f_{\hat{\boldsymbol{\gamma}}}$		
				$f_{\hat{\mathbf{x}}}$	γ_0
				$f_{\hat{\boldsymbol{\gamma}}}$	z_1
				$Z_{\mathcal{H}}$	γ_0
- Hadamard		$f_{\hat{\boldsymbol{\gamma}}}, f_{\varphi(h)}, f_{\mathcal{H}}$			
-				$f_{\mathcal{H}}$	α
- - InnerProduct		$f_{\hat{\boldsymbol{\gamma}}}, f_{\varphi(h)}$	$\bar{h}_2$		
- -				$\bar{h}_2$	$z_3, \gamma z_3$
- -				$f_{\hat{\boldsymbol{\gamma}}}$	αz_3
- -				f_I	$(z_3)^{-1}$
- -				$f_{\varphi(h)}$	$(z_3)^{-1}$
total:	3	1	2	8	7

Table 8. Callgraph Statistics for SparseMonomialVector

protocol	preprocessed	input	new	oracles queried	queried in
SparseMonomialVector		$f_{\hat{\boldsymbol{x}}}$	$f_{\hat{\boldsymbol{z}}}, f_{\hat{\boldsymbol{y}}}$		
				$f_{\hat{\boldsymbol{x}}}$	x_0
- VandermondeVector	$f_{\varphi(a_k)}, f_{\varphi(a_k)^H}$	$f_{\hat{\boldsymbol{y}}}$	$f_{\hat{\boldsymbol{\delta}}}$		
-				$f_{\hat{\boldsymbol{y}}}$	δ_1
-				$f_{\hat{\boldsymbol{\delta}}}$	z_2
- - Hadamard		$f_{\hat{\boldsymbol{\delta}}}, f_{\varphi(a_k)}, f_{\varphi(a_k)^H}$			
- -				$f_{\varphi(a_k)^H}$	$\delta_1^H \alpha$
- -				f_I	α
- - - InnerProduct		$f_{\hat{\boldsymbol{\delta}}}, f_{\varphi(a_k)}$	h_3		
- - -				$\bar{h}_3$	$z_4, \gamma z_4$
- - -				$f_{\hat{\boldsymbol{\delta}}}$	αz_4
- - -				$f_{\varphi(a_k)}$	$(\delta_1 z_4)^{-1}$
- - -				f_I	$(z_4)^{-1}$
- LagrangeVector	$f_{\varphi(h)}, f_{\mathcal{H}}, Z_{\mathcal{H}}$	$f_{\hat{\boldsymbol{z}}}$	$f_{\hat{\boldsymbol{\gamma}}}$		
-				$f_{\hat{\boldsymbol{z}}}$	γ_5
-				$f_{\hat{\boldsymbol{\gamma}}}$	z_6
-				$Z_{\mathcal{H}}$	γ_5
- - Hadamard		$f_{\hat{\boldsymbol{\gamma}}}, f_{\varphi(h)}, f_{\mathcal{H}}$			
- -				$f_{\mathcal{H}}$	α
- - - InnerProduct		$f_{\hat{\boldsymbol{\gamma}}}, f_{\varphi(h)}$	h_7		
- - -				$\bar{h}_7$	$z_8, \gamma z_8$
- - -				$f_{\hat{\boldsymbol{\gamma}}}$	αz_8
- - -				f_I	$(z_8)^{-1}$
- - -				$f_{\varphi(h)}$	$(z_8)^{-1}$
- InnerProduct		$f_{\hat{\boldsymbol{z}}}, f_{\hat{\boldsymbol{y}}}$	h_9		
-				$\bar{h}_9$	$z_{10}, \gamma z_{10}$
-				$f_{\hat{\boldsymbol{z}}}$	z_{10}
-				$f_{\hat{\boldsymbol{y}}}$	$(z_{10})^{-1}$
total:	5	1	7	20	18

Table 9. Callgraph Statistics for SparseBiEval

protocol	preprocessed	input	new	oracles queried	queried in
SparseBiEval	f_c		f_u, f_v, f_{uv}		
- SparseMonomialVector		f_u	$f_{\hat{\boldsymbol{z}}}, f_{\hat{\boldsymbol{y}}}$		
-				f_u	x_0
- - VandermondeVector	$f_{\varphi(a_k)}, f_{\varphi(a_k)^H}$	$f_{\hat{\boldsymbol{y}}}$	$f_{\hat{\boldsymbol{\delta}}}$		
- -				$f_{\hat{\boldsymbol{y}}}$	δ_1
- -				$f_{\hat{\boldsymbol{\delta}}}$	z_2
- - - Hadamard		$f_{\hat{\boldsymbol{\delta}}}, f_{\varphi(a_k)}, f_{\varphi(a_k)^H}$			
- - -				$f_{\varphi(a_k)^H}$	$\delta_1^H \alpha$
- - -				f_I	α
- - - - InnerProduct		$f_{\hat{\boldsymbol{\delta}}}, f_{\varphi(a_k)}$	h_3		
- - - -				$\bar{h}_3$	$z_4, \gamma z_4$
- - - -				$f_{\hat{\boldsymbol{\delta}}}$	αz_4
- - - -				$f_{\varphi(a_k)}$	$(\delta_1 z_4)^{-1}$
- - - -				f_I	$(z_4)^{-1}$
- - LagrangeVector	$f_{\varphi(h)}, f_{\mathcal{H}}, Z_{\mathcal{H}}$	$f_{\hat{\boldsymbol{z}}}$	$f_{\hat{\boldsymbol{\gamma}}}$		
- -				$f_{\hat{\boldsymbol{z}}}$	γ_5
- -				$f_{\hat{\boldsymbol{\gamma}}}$	z_6
- -				$Z_{\mathcal{H}}$	γ_5
- - - Hadamard		$f_{\hat{\boldsymbol{\gamma}}}, f_{\varphi(h)}, f_{\mathcal{H}}$			
- - -				$f_{\mathcal{H}}$	α
- - - - InnerProduct		$f_{\hat{\boldsymbol{\gamma}}}, f_{\varphi(h)}$	h_7		
- - - -				$\bar{h}_7$	$z_8, \gamma z_8$
- - - -				$f_{\hat{\boldsymbol{\gamma}}}$	αz_8
- - - -				f_I	$(z_8)^{-1}$
- - - -				$f_{\varphi(h)}$	$(z_8)^{-1}$
- - InnerProduct		$f_{\hat{\boldsymbol{z}}}, f_{\hat{\boldsymbol{y}}}$	h_9		
- -				$\bar{h}_9$	$z_{10}, \gamma z_{10}$
- -				$f_{\hat{\boldsymbol{z}}}$	z_{10}
- -				$f_{\hat{\boldsymbol{y}}}$	$(z_{10})^{-1}$
- SparseMonomialVector		f_v	$f_{\hat{\boldsymbol{z}}}, f_{\hat{\boldsymbol{y}}}$		
-				f_v	x_{11}
- - VandermondeVector	$f_{\varphi(b_k)}, f_{\varphi(b_k)^H}$	$f_{\hat{\boldsymbol{y}}}$	$f_{\hat{\boldsymbol{\delta}}}$		
- -				$f_{\hat{\boldsymbol{y}}}$	δ_{12}
- -				$f_{\hat{\boldsymbol{\delta}}}$	z_{13}
- - - Hadamard		$f_{\hat{\boldsymbol{\delta}}}, f_{\varphi(b_k)}, f_{\varphi(b_k)^H}$			
- - -				$f_{\varphi(b_k)^H}$	$\delta_{12}^H \alpha$
- - -				f_I	α
- - - - InnerProduct		$f_{\hat{\boldsymbol{\delta}}}, f_{\varphi(b_k)}$	h_{14}		
- - - -				$\bar{h}_{14}$	$z_{15}, \gamma z_{15}$
- - - -				$f_{\hat{\boldsymbol{\delta}}}$	αz_{15}
- - - -				$f_{\varphi(b_k)}$	$(\delta_{12} z_{15})^{-1}$
- - - -				f_I	$(z_{15})^{-1}$
- - LagrangeVector	$f_{\varphi(h)}, f_{\mathcal{H}}, Z_{\mathcal{H}}$	$f_{\hat{\boldsymbol{z}}}$	$f_{\hat{\boldsymbol{\gamma}}}$		
- -				$f_{\hat{\boldsymbol{z}}}$	γ_{16}
- -				$f_{\hat{\boldsymbol{\gamma}}}$	z_{17}
- -				$Z_{\mathcal{H}}$	γ_{16}
- - - Hadamard		$f_{\hat{\boldsymbol{\gamma}}}, f_{\varphi(h)}, f_{\mathcal{H}}$			
- - -				$f_{\mathcal{H}}$	α
- - - - InnerProduct		$f_{\hat{\boldsymbol{\gamma}}}, f_{\varphi(h)}$	h_{18}		
- - - -				$\bar{h}_{18}$	$z_{19}, \gamma z_{19}$
- - - -				$f_{\hat{\boldsymbol{\gamma}}}$	αz_{19}
- - - -				f_I	$(z_{19})^{-1}$
- - - -				$f_{\varphi(h)}$	$(z_{19})^{-1}$
- - InnerProduct		$f_{\hat{\boldsymbol{z}}}, f_{\hat{\boldsymbol{y}}}$	h_{20}		
- -				$\bar{h}_{20}$	$z_{21}, \gamma z_{21}$
- -				$f_{\hat{\boldsymbol{z}}}$	z_{21}
- -				$f_{\hat{\boldsymbol{y}}}$	$(z_{21})^{-1}$
- Hadamard		f_u, f_v, f_{uv}			
-				f_{uv}	α
- - InnerProduct		f_u, f_v	h_{22}		
- -				$\bar{h}_{22}$	$z_{23}, \gamma z_{23}$
- -				f_u	αz_{23}
- -				f_v	$(z_{23})^{-1}$
- InnerProduct		f_{uv}, f_c	h_{24}		
-				$\bar{h}_{24}$	$z_{25}, \gamma z_{25}$
-				f_{uv}	z_{25}
-				f_c	$(z_{25})^{-1}$
total:	8	0	19	49	42

Table 10. Callgraph Statistics for SparseMVP

protocol	preprocessed	input	new	oracles queried	queried in
SparseMVP		$f_{\boldsymbol{a}}, f_{\boldsymbol{b}}$	$f_{\boldsymbol{\alpha}^\intercal M}$		
				$f_{\boldsymbol{b}}$	α_{smvp}
				$f_{\boldsymbol{\alpha}^\intercal M}$	β_{smvp}
- InnerProduct		$f_{\boldsymbol{a}}, f_{\boldsymbol{\alpha}^\intercal M}$	h_0		
-				$\bar{h}_0$	$z_1, \gamma z_1$
-				$f_{\boldsymbol{a}}$	z_1
-				$f_{\boldsymbol{\alpha}^\intercal M}$	$(z_1)^{-1}$
- SparseBiEval	f_c		f_u, f_v, f_{uv}		
- - SparseMonomialVector		f_u	$f_{\hat{\boldsymbol{x}}}, f_{\hat{\boldsymbol{y}}}$		
- -				f_u	x_2
- - - VandermondeVector	$f_{\varphi(a_k)}, f_{\varphi(a_k)^H}$	$f_{\hat{\boldsymbol{y}}}$	$f_{\hat{\boldsymbol{\delta}}}$		
- - -				$f_{\hat{\boldsymbol{y}}}$	δ_3
- - -				$f_{\hat{\boldsymbol{\delta}}}$	z_4
- - - - Hadamard		$f_{\hat{\boldsymbol{\delta}}}, f_{\varphi(a_k)}, f_{\varphi(a_k)^H}$			
- - - -				$f_{\varphi(a_k)^H}$	$\delta_3^H \alpha$
- - - -				f_I	α
- - - - - InnerProduct		$f_{\hat{\boldsymbol{\delta}}}, f_{\varphi(a_k)}$	h_5		
- - - - -				$\bar{h}_5$	$z_6, \gamma z_6$
- - - - -				$f_{\hat{\boldsymbol{\delta}}}$	αz_6
- - - - -				$f_{\varphi(a_k)}$	$(\delta_3 z_6)^{-1}$
- - - - -				f_I	$(z_6)^{-1}$
- - - LagrangeVector	$f_{\varphi(h)}, f_{\mathcal{H}}, Z_{\mathcal{H}}$	$f_{\hat{\boldsymbol{x}}}$	$f_{\hat{\boldsymbol{\gamma}}}$		
- - -				$f_{\hat{\boldsymbol{x}}}$	γ_7
- - -				$f_{\hat{\boldsymbol{\gamma}}}$	z_8
- - -				$Z_{\mathcal{H}}$	γ_7
- - - - Hadamard		$f_{\hat{\boldsymbol{\gamma}}}, f_{\varphi(h)}, f_{\mathcal{H}}$			
- - - -				$f_{\mathcal{H}}$	α
- - - - - InnerProduct		$f_{\hat{\boldsymbol{\gamma}}}, f_{\varphi(h)}$	h_9		
- - - - -				$\bar{h}_9$	$z_{10}, \gamma z_{10}$
- - - - -				$f_{\hat{\boldsymbol{\gamma}}}$	αz_{10}
- - - - -				f_I	$(z_{10})^{-1}$
- - - - -				$f_{\varphi(h)}$	$(z_{10})^{-1}$
- - - InnerProduct		$f_{\hat{\boldsymbol{x}}}, f_{\hat{\boldsymbol{y}}}$	h_{11}		
- - -				$\bar{h}_{11}$	$z_{12}, \gamma z_{12}$
- - -				$f_{\hat{\boldsymbol{x}}}$	z_{12}
- - -				$f_{\hat{\boldsymbol{y}}}$	$(z_{12})^{-1}$
- - SparseMonomialVector		f_v	$f_{\hat{\boldsymbol{x}}}, f_{\hat{\boldsymbol{y}}}$		
- -				f_v	x_{13}
- - - VandermondeVector	$f_{\varphi(b_k)}, f_{\varphi(b_k)^H}$	$f_{\hat{\boldsymbol{y}}}$	$f_{\hat{\boldsymbol{\delta}}}$		
- - -				$f_{\hat{\boldsymbol{y}}}$	δ_{14}
- - -				$f_{\hat{\boldsymbol{\delta}}}$	z_{15}
- - - - Hadamard		$f_{\hat{\boldsymbol{\delta}}}, f_{\varphi(b_k)}, f_{\varphi(b_k)^H}$			
- - - -				$f_{\varphi(b_k)^H}$	$\delta_{14}^H \alpha$
- - - -				f_I	α
- - - - - InnerProduct		$f_{\hat{\boldsymbol{\delta}}}, f_{\varphi(b_k)}$	h_{16}		
- - - - -				$\bar{h}_{16}$	$z_{17}, \gamma z_{17}$
- - - - -				$f_{\hat{\boldsymbol{\delta}}}$	αz_{17}
- - - - -				$f_{\varphi(b_k)}$	$(\delta_{14} z_{17})^{-1}$
- - - - -				f_I	$(z_{17})^{-1}$
- - - LagrangeVector	$f_{\varphi(h)}, f_{\mathcal{H}}, Z_{\mathcal{H}}$	$f_{\hat{\boldsymbol{x}}}$	$f_{\hat{\boldsymbol{\gamma}}}$		
- - -				$f_{\hat{\boldsymbol{x}}}$	γ_{18}
- - -				$f_{\hat{\boldsymbol{\gamma}}}$	z_{19}
- - -				$Z_{\mathcal{H}}$	γ_{18}
- - - - Hadamard		$f_{\hat{\boldsymbol{\gamma}}}, f_{\varphi(h)}, f_{\mathcal{H}}$			
- - - -				$f_{\mathcal{H}}$	α
- - - - - InnerProduct		$f_{\hat{\boldsymbol{\gamma}}}, f_{\varphi(h)}$	h_{20}		
- - - - -				$\bar{h}_{20}$	$z_{21}, \gamma z_{21}$
- - - - -				$f_{\hat{\boldsymbol{\gamma}}}$	αz_{21}
- - - - -				f_I	$(z_{21})^{-1}$
- - - - -				$f_{\varphi(h)}$	$(z_{21})^{-1}$
- - - InnerProduct		$f_{\hat{\boldsymbol{x}}}, f_{\hat{\boldsymbol{y}}}$	h_{22}		
- - -				$\bar{h}_{22}$	$z_{23}, \gamma z_{23}$
- - -				$f_{\hat{\boldsymbol{x}}}$	z_{23}
- - -				$f_{\hat{\boldsymbol{y}}}$	$(z_{23})^{-1}$
- - Hadamard		f_u, f_v, f_{uv}			
- -				f_{uv}	α
- - - InnerProduct		f_u, f_v	h_{24}		
- - -				$\bar{h}_{24}$	$z_{25}, \gamma z_{25}$
- - -				f_u	αz_{25}
- - -				f_v	$(z_{25})^{-1}$
- - InnerProduct		f_{uv}, f_c	h_{26}		
- -				$\bar{h}_{26}$	$z_{27}, \gamma z_{27}$
- -				f_{uv}	z_{27}
- -				f_c	$(z_{27})^{-1}$
total:	8	2	21	55	47

Table 11. Callgraph Statistics for SparseClaymore

protocol	preprocessed	input	new	oracles queried	queried in
SparseClaymore		$f_{\mathbf{x}}$	f_{wl}, f_{wr}, f_{wo}		
- SparseMVP		$f_{wl}, f_{wr}, f_{wo}, f_{\mathbf{x}}$	$f_{\boldsymbol{\alpha}^\intercal M}$		
-				$f_{\mathbf{x}}$	α_{smvp}
-				$f_{\boldsymbol{\alpha}^\intercal M}$	β_{smvp}
- - InnerProduct		$f_{wl}, f_{wr}, f_{wo}, f_{\boldsymbol{\alpha}^\intercal M}$	h_0		
- -				$\bar{h}_0$	$z_1, \gamma z_1$
- -				f_{wl}	z_1
- -				f_{wr}	z_1
- -				f_{wo}	z_1
- -				$f_{\boldsymbol{\alpha}^\intercal M}$	$(z_1)^{-1}$
- - SparseBiEval	f_c		f_u, f_v, f_{uv}		
- - - SparseMonomialVector		f_u	$f_{\hat{\mathbf{z}}}, f_{\hat{\mathbf{y}}}$		
- - -				f_u	x_2
- - - - VandermondeVector	$f_{\varphi(a_k)}, f_{\varphi(a_k)^H}$	$f_{\hat{\mathbf{y}}}$	$f_{\hat{\boldsymbol{\delta}}}$		
- - - -				$f_{\hat{\mathbf{y}}}$	δ_3
- - - -				$f_{\hat{\boldsymbol{\delta}}}$	z_4
- - - - - Hadamard		$f_{\hat{\boldsymbol{\delta}}}, f_{\varphi(a_k)}, f_{\varphi(a_k)^H}$			
- - - - -				$f_{\varphi(a_k)^H}$	$\delta_3^H \alpha$
- - - - -				f_I	α
- - - - - - InnerProduct		$f_{\hat{\boldsymbol{\delta}}}, f_{\varphi(a_k)}$	h_5		
- - - - - -				$\bar{h}_5$	$z_6, \gamma z_6$
- - - - - -				$f_{\hat{\boldsymbol{\delta}}}$	αz_6
- - - - - -				$f_{\varphi(a_k)}$	$(\delta_3 z_6)^{-1}$
- - - - - -				f_I	$(z_6)^{-1}$
- - - - LagrangeVector	$f_{\varphi(h)}, f_{\mathcal{H}}, Z_{\mathcal{H}}$	$f_{\hat{\mathbf{z}}}$	$f_{\hat{\boldsymbol{\gamma}}}$		
- - - -				$f_{\hat{\mathbf{z}}}$	γ_7
- - - -				$f_{\hat{\boldsymbol{\gamma}}}$	z_8
- - - -				$Z_{\mathcal{H}}$	γ_7
- - - - - Hadamard		$f_{\hat{\boldsymbol{\gamma}}}, f_{\varphi(h)}, f_{\mathcal{H}}$			
- - - - -				$f_{\mathcal{H}}$	α
- - - - - - InnerProduct		$f_{\hat{\boldsymbol{\gamma}}}, f_{\varphi(h)}$	h_9		
- - - - - -				$\bar{h}_9$	$z_{10}, \gamma z_{10}$
- - - - - -				$f_{\hat{\boldsymbol{\gamma}}}$	αz_{10}
- - - - - -				f_I	$(z_{10})^{-1}$
- - - - - -				$f_{\varphi(h)}$	$(z_{10})^{-1}$
- - - - InnerProduct		$f_{\hat{\mathbf{z}}}, f_{\hat{\mathbf{y}}}$	h_{11}		
- - - -				$\bar{h}_{11}$	$z_{12}, \gamma z_{12}$
- - - -				$f_{\hat{\mathbf{z}}}$	z_{12}
- - - -				$f_{\hat{\mathbf{y}}}$	$(z_{12})^{-1}$
- - - SparseMonomialVector		f_v	$f_{\hat{\mathbf{z}}}, f_{\hat{\mathbf{y}}}$		
- - -				f_v	x_{13}
- - - - VandermondeVector	$f_{\varphi(b_k)}, f_{\varphi(b_k)^H}$	$f_{\hat{\mathbf{y}}}$	$f_{\hat{\boldsymbol{\delta}}}$		
- - - -				$f_{\hat{\mathbf{y}}}$	δ_{14}
- - - -				$f_{\hat{\boldsymbol{\delta}}}$	z_{15}
- - - - - Hadamard		$f_{\hat{\boldsymbol{\delta}}}, f_{\varphi(b_k)}, f_{\varphi(b_k)^H}$			
- - - - -				$f_{\varphi(b_k)^H}$	$\delta_{14}^H \alpha$
- - - - -				f_I	α
- - - - - - InnerProduct		$f_{\hat{\boldsymbol{\delta}}}, f_{\varphi(b_k)}$	h_{16}		
- - - - - -				$\bar{h}_{16}$	$z_{17}, \gamma z_{17}$
- - - - - -				$f_{\hat{\boldsymbol{\delta}}}$	αz_{17}
- - - - - -				$f_{\varphi(b_k)}$	$(\delta_{14} z_{17})^{-1}$
- - - - - -				f_I	$(z_{17})^{-1}$
- - - - LagrangeVector	$f_{\varphi(h)}, f_{\mathcal{H}}, Z_{\mathcal{H}}$	$f_{\hat{\mathbf{z}}}$	$f_{\hat{\boldsymbol{\gamma}}}$		
- - - -				$f_{\hat{\mathbf{z}}}$	γ_{18}
- - - -				$f_{\hat{\boldsymbol{\gamma}}}$	z_{19}
- - - -				$Z_{\mathcal{H}}$	γ_{18}
- - - - - Hadamard		$f_{\hat{\boldsymbol{\gamma}}}, f_{\varphi(h)}, f_{\mathcal{H}}$			
- - - - -				$f_{\mathcal{H}}$	α
- - - - - - InnerProduct		$f_{\hat{\boldsymbol{\gamma}}}, f_{\varphi(h)}$	h_{20}		
- - - - - -				$\bar{h}_{20}$	$z_{21}, \gamma z_{21}$
- - - - - -				$f_{\hat{\boldsymbol{\gamma}}}$	αz_{21}
- - - - - -				f_I	$(z_{21})^{-1}$
- - - - - -				$f_{\varphi(h)}$	$(z_{21})^{-1}$
- - - - InnerProduct		$f_{\hat{\mathbf{z}}}, f_{\hat{\mathbf{y}}}$	h_{22}		
- - - -				$\bar{h}_{22}$	$z_{23}, \gamma z_{23}$
- - - -				$f_{\hat{\mathbf{z}}}$	z_{23}
- - - -				$f_{\hat{\mathbf{y}}}$	$(z_{23})^{-1}$
- - - Hadamard		f_u, f_v, f_{uv}			
- - -				f_{uv}	α
- - - - InnerProduct		f_u, f_v	h_{24}		
- - - -				$\bar{h}_{24}$	$z_{25}, \gamma z_{25}$
- - - -				f_u	αz_{25}
- - - -				f_v	$(z_{25})^{-1}$
- - - InnerProduct		f_{uv}, f_c	h_{26}		
- - -				$\bar{h}_{26}$	$z_{27}, \gamma z_{27}$
- - -				f_{uv}	z_{27}
- - -				f_c	$(z_{27})^{-1}$
- Hadamard		f_{wl}, f_{wr}, f_{wo}			
-				f_{wo}	α
- - InnerProduct		f_{wl}, f_{wr}	h_{28}		
- -				$\bar{h}_{28}$	$z_{29}, \gamma z_{29}$
- -				f_{wl}	αz_{29}
- -				f_{wr}	$(z_{29})^{-1}$
total:	8	1	25	62	51
with BatchedInnerProduct:	8	1	16	48	17
and witness concatenation:	8	1	15	46	17

Table 12. Callgraph Statistics for BiVandermondeVector

protocol	preprocessed	input	new	oracles queried	queried in
BiVandermondeVector	$f_{\varphi(a_k)}, f_{\varphi(a_k)^H}, f_{\varphi(b_k)}, f_{\varphi(b_k)^H}$	$f_{\hat{\mathbf{x}}}$	$f_{\hat{\boldsymbol{\delta}}}$		
				$f_{\hat{\mathbf{x}}}$	δ_0
				$f_{\hat{\boldsymbol{\delta}}}$	z_1
- Hadamard		$f_{\hat{\boldsymbol{\delta}}}, f_{\varphi(a_k)}, f_{\varphi(b_k)}, f_{\varphi(a_k)^H}, f_{\varphi(b_k)^H}$			
-				$f_{\varphi(a_k)^H}$	α
-				f_I	α
-				$f_{\varphi(b_k)^H}$	α
- - InnerProduct		$f_{\hat{\boldsymbol{\delta}}}, f_{\varphi(a_k)}, f_{\varphi(b_k)}$	h_2		
- -				$\bar{h}_2$	$z_3, \gamma z_3$
- -				$f_{\hat{\boldsymbol{\delta}}}$	αz_3
- -				$f_{\varphi(a_k)}$	$(z_3)^{-1}$
- -				f_I	$(z_3)^{-1}$
- -				$f_{\varphi(b_k)}$	$(z_3)^{-1}$
total:	4	1	2	9	7

Table 13. Callgraph Statistics for BiLagrangeVector

protocol	preprocessed	input	new	oracles queried	queried in
BiLagrangeVector	$f_{\varphi(h)}, f_{\mathcal{H}}, Z_{\mathcal{H}}$	$f_{\hat{\mathbf{x}}}$	$f_{\hat{\boldsymbol{\gamma}}}$		
				$f_{\hat{\mathbf{x}}}$	γ_0
				$f_{\hat{\boldsymbol{\gamma}}}$	z_1
				$Z_{\mathcal{H}}$	γ_0
- Hadamard		$f_{\hat{\boldsymbol{\gamma}}}, f_{\varphi(h)}, f_{\mathcal{H}}$			
-				$f_{\mathcal{H}}$	α
- - InnerProduct		$f_{\hat{\boldsymbol{\gamma}}}, f_{\varphi(h)}$	h_2		
- -				$\bar{h}_2$	$z_3, \gamma z_3$
- -				$f_{\hat{\boldsymbol{\gamma}}}$	αz_3
- -				f_I	$(z_3)^{-1}$
- -				$f_{\varphi(h)}$	$(z_3)^{-1}$
total:	3	1	2	8	7

Table 14. Call graph statistics for BatchedSparseBiEval

protocol	preprocessed	input	new	oracles queried	queried in
BatchedSparseBiEval	f_c		$f_{u\|v}, f_{uv}, f_{\hat{\mathbf{u}}\hat{\mathbf{v}}}, f_{\hat{\mathbf{y}}}$	$f_{u\|v}$	y_8
- BiLagrangeVector	$f_{\varphi(h)}, f_{\mathcal{H}}, Z_{\mathcal{H}}$	$f_{\hat{\mathbf{u}}\hat{\mathbf{v}}}$	$f_{\hat{\boldsymbol{\gamma}}}$		
-				$f_{\hat{\mathbf{u}}\hat{\mathbf{v}}}$	γ_0
-				$f_{\hat{\boldsymbol{\gamma}}}$	z_1
-				$Z_{\mathcal{H}}$	γ_0
- - Hadamard		$f_{\hat{\boldsymbol{\gamma}}}, f_{\varphi(h)}, f_{\mathcal{H}}$			
- -				$f_{\mathcal{H}}$	α
- - - InnerProduct		$f_{\hat{\boldsymbol{\gamma}}}, f_{\varphi(h)}$	h_2		
- - -				$\bar{h}_2$	$z_3, \gamma z_3$
- - -				$f_{\hat{\boldsymbol{\gamma}}}$	αz_3
- - -				f_I	$(z_3)^{-1}$
- - -				$f_{\varphi(h)}$	$(z_3)^{-1}$
- BiVandermondeVector	$f_{\varphi(a_k)}, f_{\varphi(a_k)^H}, f_{\varphi(b_k)}, f_{\varphi(b_k)^H}$	$f_{\hat{\mathbf{y}}}$	$f_{\hat{\boldsymbol{\delta}}}$		
-				$f_{\hat{\mathbf{y}}}$	δ_4
-				$f_{\hat{\boldsymbol{\delta}}}$	z_5
- - Hadamard		$f_{\hat{\boldsymbol{\delta}}}, f_{\varphi(a_k)}, f_{\varphi(b_k)}, f_{\varphi(a_k)^H}, f_{\varphi(b_k)^H}$			
- -				$f_{\varphi(a_k)^H}$	α
- -				f_I	α
- -				$f_{\varphi(b_k)^H}$	α
- - - InnerProduct		$f_{\hat{\boldsymbol{\delta}}}, f_{\varphi(a_k)}, f_{\varphi(b_k)}$	h_6		
- - -				$\bar{h}_6$	$z_7, \gamma z_7$
- - -				$f_{\hat{\boldsymbol{\delta}}}$	αz_7
- - -				$f_{\varphi(a_k)}$	$(z_7)^{-1}$
- - -				f_I	$(z_7)^{-1}$
- - -				$f_{\varphi(b_k)}$	$(z_7)^{-1}$
- InnerProduct		$f_{\hat{\mathbf{u}}\hat{\mathbf{v}}}, f_{\hat{\mathbf{y}}}$	h_9		
-				$\bar{h}_9$	$z_{10}, \gamma z_{10}$
-				$f_{\hat{\mathbf{u}}\hat{\mathbf{v}}}$	z_{10}
-				$f_{\hat{\mathbf{y}}}$	$(z_{10})^{-1}$
- Hadamard		$f_{u\|v}, f_{u\|v}, f_{uv}$			
-				f_{uv}	α
- - InnerProduct		$f_{u\|v}, f_{u\|v}$	$\bar{h}_{11}$		
- -				$\bar{h}_{11}$	$z_{12}, \gamma z_{12}$
- -				$f_{u\|v}$	$(z_{12})^{-1}$
- InnerProduct		f_{uv}, f_c	$\bar{h}_{13}$		
-				$\bar{h}_{13}$	$z_{14}, \gamma z_{14}$
-				f_{uv}	z_{14}
-				f_c	$(z_{14})^{-1}$
total:	8	0	11	30	23

Table 15. Call graph statistics for SparseMVP but with BatchedSparseBiEval

protocol	preprocessed	input	new	oracles queried	queried in
BiSparseMVP		$f_{\boldsymbol{a}}, f_{\boldsymbol{b}}$	$f_{\boldsymbol{\alpha}^\intercal M}$		
				$f_{\boldsymbol{b}}$	α_{smvp}
				$f_{\boldsymbol{\alpha}^\intercal M}$	β_{smvp}
- InnerProduct		$f_{\boldsymbol{a}}, f_{\boldsymbol{\alpha}^\intercal M}$	h_0		
-				$\bar{h}_0$	$z_1, \gamma z_1$
-				$f_{\boldsymbol{a}}$	z_1
-				$f_{\boldsymbol{\alpha}^\intercal M}$	$(z_1)^{-1}$
- BatchedSparseBiEval	f_c		$f_{u\|v}, f_{uv}, f_{\mathbf{uv}}, f_{\hat{\mathbf{y}}}$		
-				$f_{u\|v}$	y_{10}
- - BiLagrangeVector	$f_{\varphi(h)}, f_{\mathcal{H}}, Z_{\mathcal{H}}$	$f_{\mathbf{uv}}$	$f_{\hat{\boldsymbol{\gamma}}}$		
- -				$f_{\mathbf{uv}}$	γ_2
- -				$f_{\hat{\boldsymbol{\gamma}}}$	z_3
- -				$Z_{\mathcal{H}}$	γ_2
- - - Hadamard		$f_{\hat{\boldsymbol{\gamma}}}, f_{\varphi(h)}, f_{\mathcal{H}}$			
- - -				$f_{\mathcal{H}}$	α
- - - - InnerProduct		$f_{\hat{\boldsymbol{\gamma}}}, f_{\varphi(h)}$	h_4		
- - - -				$\bar{h}_4$	$z_5, \gamma z_5$
- - - -				$f_{\hat{\boldsymbol{\gamma}}}$	αz_5
- - - -				f_I	$(z_5)^{-1}$
- - - -				$f_{\varphi(h)}$	$(z_5)^{-1}$
- - BiVandermondeVector	$f_{\varphi(a_k)}, f_{\varphi(a_k)^H}, f_{\varphi(b_k)}, f_{\varphi(b_k)^H}$	$f_{\hat{\mathbf{y}}}$	$f_{\hat{\boldsymbol{\delta}}}$		
- -				$f_{\hat{\mathbf{y}}}$	δ_6
- -				$f_{\hat{\boldsymbol{\delta}}}$	z_7
- - - Hadamard		$f_{\hat{\boldsymbol{\delta}}}, f_{\varphi(a_k)}, f_{\varphi(b_k)}, f_{\varphi(a_k)^H}, f_{\varphi(b_k)^H}$			
- - -				$f_{\varphi(a_k)^H}$	α
- - -				f_I	α
- - -				$f_{\varphi(b_k)^H}$	α
- - - - InnerProduct		$f_{\hat{\boldsymbol{\delta}}}, f_{\varphi(a_k)}, f_{\varphi(b_k)}$	h_8		
- - - -				$\bar{h}_8$	$z_9, \gamma z_9$
- - - -				$f_{\hat{\boldsymbol{\delta}}}$	αz_9
- - - -				$f_{\varphi(a_k)}$	$(z_9)^{-1}$
- - - -				f_I	$(z_9)^{-1}$
- - - -				$f_{\varphi(b_k)}$	$(z_9)^{-1}$
- - InnerProduct		$f_{\mathbf{uv}}, f_{\hat{\mathbf{y}}}$	h_{11}		
- -				$\bar{h}_{11}$	$z_{12}, \gamma z_{12}$
- -				$f_{\mathbf{uv}}$	z_{12}
- -				$f_{\hat{\mathbf{y}}}$	$(z_{12})^{-1}$
- - Hadamard		$f_{u\|v}, f_{u\|v}, f_{uv}$			
- -				f_{uv}	α
- - - InnerProduct		$f_{u\|v}, f_{u\|v}$	h_{13}		
- - -				$\bar{h}_{13}$	$z_{14}, \gamma z_{14}$
- - -				$f_{u\|v}$	$(z_{14})^{-1}$
- - InnerProduct		f_{uv}, f_c	h_{15}		
- -				$\bar{h}_{15}$	$z_{16}, \gamma z_{16}$
- -				f_{uv}	z_{16}
- -				f_c	$(z_{16})^{-1}$
total:	8	2	13	36	28

Table 16. Call graph statistics for SparseClaymore but with batched BatchedSparseBiEval

protocol	preprocessed	input	new	oracles queried	queried in
BiSparseClaymore		$f_{\mathbf{x}}$	f_{wl}, f_{wr}, f_{wo}		
- BiSparseMVP		$f_{wl}, f_{wr}, f_{wo}, f_{\mathbf{x}}$	$f_{\boldsymbol{\alpha}^\intercal M}$		
-				$f_{\mathbf{x}}$	α_{smvp}
-				$f_{\boldsymbol{\alpha}^\intercal M}$	β_{smvp}
- - InnerProduct		$f_{wl}, f_{wr}, f_{wo}, f_{\boldsymbol{\alpha}^\intercal M}$	h_0		
- -				$\bar{h}_0$	$z_1, \gamma z_1$
- -				f_{wl}	z_1
- -				f_{wr}	z_1
- -				f_{wo}	z_1
- -				$f_{\boldsymbol{\alpha}^\intercal M}$	$(z_1)^{-1}$
- - BatchedSparseBiEval	f_c		$f_{u\|v}, f_{uv}, f_{\mathbf{uv}}, f_{\hat{\mathbf{y}}}$		
- -				$f_{u\|v}$	y_{10}
- - - BiLagrangeVector	$f_{\varphi(h)}, f_{\mathcal{H}}, Z_{\mathcal{H}}$	$f_{\mathbf{uv}}$	$f_{\hat{\boldsymbol{\gamma}}}$		
- - -				$f_{\mathbf{uv}}$	γ_2
- - -				$f_{\hat{\boldsymbol{\gamma}}}$	z_3
- - -				$Z_{\mathcal{H}}$	γ_2
- - - - Hadamard		$f_{\hat{\boldsymbol{\gamma}}}, f_{\varphi(h)}, f_{\mathcal{H}}$			
- - - -				$f_{\mathcal{H}}$	α
- - - - - InnerProduct		$f_{\hat{\boldsymbol{\gamma}}}, f_{\varphi(h)}$	h_4		
- - - - -				$\bar{h}_4$	$z_5, \gamma z_5$
- - - - -				$f_{\hat{\boldsymbol{\gamma}}}$	αz_5
- - - - -				f_I	$(z_5)^{-1}$
- - - - -				$f_{\varphi(h)}$	$(z_5)^{-1}$
- - - BiVandermondeVector	$f_{\varphi(a_k)}, f_{\varphi(a_k)^H}, f_{\varphi(b_k)}, f_{\varphi(b_k)^H}$	$f_{\hat{\mathbf{y}}}$	$f_{\hat{\boldsymbol{\delta}}}$		
- - -				$f_{\hat{\mathbf{y}}}$	δ_6
- - -				$f_{\hat{\boldsymbol{\delta}}}$	z_7
- - - - Hadamard		$f_{\hat{\boldsymbol{\delta}}}, f_{\varphi(a_k)}, f_{\varphi(b_k)}, f_{\varphi(a_k)^H}, f_{\varphi(b_k)^H}$			
- - - -				$f_{\varphi(a_k)^H}$	α
- - - -				f_I	α
- - - -				$f_{\varphi(b_k)^H}$	α
- - - - - InnerProduct		$f_{\hat{\boldsymbol{\delta}}}, f_{\varphi(a_k)}, f_{\varphi(b_k)}$	h_8		
- - - - -				$\bar{h}_8$	$z_9, \gamma z_9$
- - - - -				$f_{\hat{\boldsymbol{\delta}}}$	αz_9
- - - - -				$f_{\varphi(a_k)}$	$(z_9)^{-1}$
- - - - -				f_I	$(z_9)^{-1}$
- - - - -				$f_{\varphi(b_k)}$	$(z_9)^{-1}$
- - - InnerProduct		$f_{\mathbf{uv}}, f_{\hat{\mathbf{y}}}$	h_{11}		
- - -				$\bar{h}_{11}$	$z_{12}, \gamma z_{12}$
- - -				$f_{\mathbf{uv}}$	z_{12}
- - -				$f_{\hat{\mathbf{y}}}$	$(z_{12})^{-1}$
- - - Hadamard		$f_{u\|v}, f_{u\|v}, f_{uv}$			
- - -				f_{uv}	α
- - - - InnerProduct		$f_{u\|v}, f_{u\|v}$	h_{13}		
- - - -				$\bar{h}_{13}$	$z_{14}, \gamma z_{14}$
- - - -				$f_{u\|v}$	$(z_{14})^{-1}$
- - - InnerProduct		f_{uv}, f_c	h_{15}		
- - -				$\bar{h}_{15}$	$z_{16}, \gamma z_{16}$
- - -				f_{uv}	z_{16}
- - -				f_c	$(z_{16})^{-1}$
- Hadamard		f_{wl}, f_{wr}, f_{wo}			
-				f_{wo}	α
- - InnerProduct		f_{wl}, f_{wr}	h_{17}		
- -				$\bar{h}_{17}$	$z_{18}, \gamma z_{18}$
- -				f_{wl}	αz_{18}
- -				f_{wr}	$(z_{18})^{-1}$
total:	8	1	17	43	32
with BatchedInnerProduct:	8	1	11	31	10
and witness concatenation:	8	1	10	30	10