Mira: Efficient Folding for Pairing-based Arguments

Josh Beal and Ben Fisch

Yale University
{josh.beal,ben.fisch}@yale.edu

Abstract. Pairing-based arguments offer remarkably small proofs and space-efficient provers, but aggregating such proofs remains costly. Groth16 SNARKs and KZG polynomial commitments are prominent examples of this class of arguments. These arguments are widely deployed in decentralized systems, with millions of proofs generated per day. Recent folding schemes have greatly reduced the cost of proving incremental computations, such as batch proof verification. However, existing constructions require encoding pairing operations in generic constraint systems, leading to high prover overhead. In this work, we introduce *Mira*, a folding scheme that directly supports pairing-based arguments. We construct this folding scheme by generalizing the framework in Protostar to support a broader class of special-sound protocols. We demonstrate the versatility and efficiency of this framework through two key applications: Groth16 proof aggregation and verifiable ML inference. Mira achieves 5.8x faster prover time and 9.7x lower memory usage than the state-of-the-art proof aggregation system while maintaining a constant-size proof. To improve the efficiency of verifiable ML inference, we provide a new lincheck protocol with a verifier degree that is independent of the matrix order. We show that Mira scales effectively to larger models, overcoming the memory bottlenecks of current schemes.

1 Introduction

Succinct non-interactive arguments of knowledge (SNARKs) are powerful cryptographic primitives that enable a prover to convince a verifier of the validity of a claim with a short proof. These primitives enable verifiable computation [BFLS91, GGP10], where a client can outsource a difficult task to an untrusted cloud provider and then check via a proof that the task was performed correctly. While theoretical SNARK constructions have been known for decades [Kil92, Mic94, BCI+13], there have been numerous efforts to minimize the prover complexity and proof size for practical applications.

Pairing-based SNARKs have achieved widespread adoption due to a remarkable property: the proof consists of only a few group elements, regardless of the complexity of the claim. A well-known example is the Groth16 SNARK [Gro16], which can prove the validity of any NP statement. These arguments are widely used in blockchains, with some protocols generating millions of such proofs on a daily basis. However, it is impractical for a smart contract to verify each proof directly due to resource constraints. As a result, proof aggregation techniques are employed to reduce the on-chain verifier's work. As SNARKs have grown in adoption, there have also been efforts to develop efficient *domain-specific SNARKs* for various applications, including zero-knowledge theorem proving [LOB24, LKA+24], regular expression matching [AIM+24], and training and inference of deep neural networks [LXZ21, WK23, CWSK24, SLZ24, APKP24].

In the domain of verifiable ML inference, pairing-based SNARKs have recently achieved impressive results [WK23]. In deep neural networks, the primary operations are matrix-vector multiplication and nonlinear function evaluation. To prove the correctness of a matrix-vector multiplication, a prover may use cqlin, a pairing-based lincheck protocol [EG23a] with constant-size proofs that requires only $O(n)$ prover operations, where n is the matrix order. To prove the correctness of a nonlinear function evaluation, a prover may use a lookup argument for greater efficiency. In this case, all valid input-output pairs for the function are stored in a lookup table, and the prover argues that the selected pairs are present in the table. For example, cq [EFG22] is a pairing-based lookup argument where the prover complexity is independent of the table size. Other pairing-based arguments, such as KZG polynomial commitments [KZG10], have also seen broad adoption in decentralized systems [FK23, TSZ+24] and verifiable machine learning [WK23, SCJ+24].

As SNARKs are applied to increasingly complex computations, it may become infeasible for a single machine to produce a proof with its limited computational resources. To address this issue, there have been efforts to distribute the work of proof generation across multiple machines. One strategy is to partition the work of existing SNARKs across multiple machines. DIZK [WZC+18] developed an

approach of this nature for distributing the workload of the Groth16 prover. Another strategy is to split a single large statement into N smaller statements, which are proved in multiple steps. At each step, the prover generates a proof that: (1) the current statement is valid, and (2) the proof of the previous statement is valid. This strategy has been formalized as *incrementally verifiable computation* (IVC) [Val08]. One can also generalize this strategy to verify any distributed computation over a directed acyclic graph of nodes. In this more general case, the technique is known as *proof-carrying data* (PCD) [CT10]. PCD has been highly impactful in real-world applications, enabling the creation of efficient zero-knowledge virtual machines [BCTV14], verifiable MapReduce computations [CTV15], image authentication systems [NT16], succinct ledgers [BMRS20, CCDW20], and verifiable financial disclosures [BF24]. These techniques are useful not only for coordinating the efforts of multiple independent provers, but also for reducing the resource requirements for a single prover that aims to prove the correctness of a complex local computation. For example, to prove the correctness of inference of an ML model with N layers, a prover could partition the work by layer. Alternatively, the prover could further partition the work, separately proving the correctness of each matrix-vector multiplication or nonlinear function evaluation and then aggregating the resulting proofs.

To achieve concrete efficiency improvements, it is critical to minimize the recursive overhead, i.e., the additional work that the prover performs at each step, beyond proving each statement's validity. In traditional constructions of IVC/PCD [BCTV14, CTV15], the recursive overhead was substantial due to the costs of representing the pairing operations in the step circuit. Halo [BGH19] pioneered a novel approach to minimize the recursive overhead by leveraging the additive property of the polynomial commitment scheme [BDFG21]. Subsequent work further reduced the prover overhead by introducing accumulation/folding schemes [BCMS20, BCL$^+$21, KST22]. These elegant techniques avoid the costly verification of the previous proof at each step, instead performing a lightweight check for the validity of a related claim. There has been tremendous progress in reducing the complexity of this related claim. In recent schemes such as Nova [KST22] and Protostar [BC23], the dominant cost of these checks is just a few group operations, which are far less expensive than pairings in arithmetic circuit format. However, if the main claim involves pairings, folding will still be resource-intensive.

Recently, there have been efforts to generalize the supported relations for folding schemes. For instance, the initial constructions [BCL$^+$21, KST22] of accumulation/folding[1] schemes only supported rank-one constraint systems (R1CS). Subsequently, Protostar [BC23] generalized these techniques to handle a broader set of special-sound protocols with an algebraic verifier, while HyperNova [KS24b] generalized the construction in Nova [KST22] to support Customizable Constraint Systems, which can represent R1CS, Plonkish and AIR arithmetizations without significant overheads. However, none of these works have optimized for the important class of pairing-based protocols.

1.1 Our contributions

We propose a generic accumulation scheme for a broader class of special-sound protocols, building upon the framework in Protostar [BC23] and Protogalaxy [EG23b]. This scheme enables folding of a variety of useful pairing-based arguments without the need to encode the expensive pairing-based checks in the step circuit, thereby improving the folding efficiency for this class of arguments. We implemented this scheme and demonstrated major performance improvements for two key applications:

- The first application is *proof aggregation* for Groth16 SNARKs. Prior work [BMM$^+$21, GMN22] achieves logarithmic-size proofs or incurs high recursive overhead [BCTV14]. We are able to achieve constant-size proofs with low prover complexity via our SnarkStar protocol, a specific instantiation of our scheme that achieves 5.8x faster prover time and 9.7x lower memory usage than the state-of-the-art proof aggregation system. This protocol enables aggregation of proofs with different verification keys, offering a flexible approach to aggregation in prover markets [WZY$^+$24].
- The second application is *verifiable ML inference*. TensorPlonk [WK23] is a recent SNARK for ML inference that uses cq [EFG22], cqlin [EG23a], and KZG polynomial commitments [KZG10] as building blocks. To reduce the prover's space complexity, we apply our folding scheme to these pairing-based arguments. To further improve efficiency, we provide an optimized lincheck protocol with a folding verifier degree that is independent of the matrix order. We assemble these subprotocols to form the TensorStar protocol, which enables fast proof generation for ML inference on resource-constrained devices. We show that our scheme scales to larger models, whereas current solutions are unable to generate proofs for such models due to memory limitations.

[1] As in prior work, we use these terms interchangeably, except when providing the precise technical definitions.

1.2 Technical overview

To address these significant limitations, we present a generic accumulation scheme for a broader class of special-sound protocols. In the class of special-sound protocols that are considered in Protostar, the prover sends messages consisting of field elements, and the verifier performs algebraic checks on these messages and other values in the transcript. In our expanded class of special-sound protocols, the prover messages and the public inputs are encoded with a *linear-only encoding scheme* [BCI+13]. In these schemes, an encoding of a linear combination of elements can be obtained from the encodings of the underlying elements using a homomorphic evaluation algorithm. An algebraic test can be performed on the encoded elements to check an algebraic relation on the underlying elements. The pairing operation is an example of a degree-two algebraic test, where the encoded values are group elements. In this setting, the verifier performs algebraic tests on the encoded values and the challenges.

Two constructions of linear-only encoding schemes are particularly relevant in this setting. The first construction is the trivial identity encoding. This yields a class of protocols where the prover messages are field elements and the verifier checks the output of an algebraic map defined over these field elements. In this case, the Protostar accumulation scheme is directly applicable. The second construction is based on bilinear maps, which allow for degree-two algebraic tests. This yields a class of protocols where the prover messages are group elements, and the verifier performs the algebraic test using pairings. This allows usage of the trivial identity commitment in the accumulation scheme since the prover messages are already succinct due to the choice of the linear-only encoding scheme.

To construct an accumulation scheme for a specific protocol, it suffices to specify the relevant details of the protocol and apply the generic accumulation scheme in a black-box manner. The relevant details include the public inputs, the prover's messages, the verifier's challenges, and the verifier's algebraic tests. We specify these details to evaluate the concrete efficiency and clarify the implementation aspects. However, the security of the accumulation scheme for a given protocol will follow directly from our result that establishes the security for the entire class of protocols.

The degree of the relation has a great impact on the efficiency of the folding scheme, since a higher degree relation requires the prover to compute additional cross terms. The verifier must then check that the linear combination of these cross terms is correct. As a result, it is important to design the special-sound protocol to minimize the complexity of the folding verifier. The lincheck protocol from cqlin [EG23a] has a verifier degree that is linear in the matrix order, which would create prohibitive overhead in folding. We propose a modified protocol that achieves a verifier degree that is independent of the matrix order n, while only requiring $O(\log n)$ additional field elements to be sent by the prover.

2 Preliminaries

2.1 Building blocks

Notation We use $\lambda \in \mathbb{N}$ to denote the security parameter with unary representation 1^λ. We use $\mathsf{negl}(\lambda)$ to denote the class of negligible functions. We use $[l]$ to denote the set of integers $\{1, \ldots, l\}$. We use $\mathbb{F}$ to denote a field of prime order p such that $\log(p) = \Omega(\lambda)$.

Commitment schemes A commitment scheme $\mathcal{C} = (\mathsf{Com}, \mathsf{Vfy})$ is a pair of efficient algorithms defined over a message space $\mathcal{M}$ and a randomness space $\mathcal{R}$ where:

- $\mathsf{cm} \leftarrow \mathsf{Com}(\mathsf{m}; r)$ is a commit algorithm that produces a commitment cm given the message $\mathsf{m} \in \mathcal{M}$ to be committed and the randomness $r \leftarrow\!\!\$\, \mathcal{R}$.
- $b \leftarrow \mathsf{Vfy}(\mathsf{cm}, \mathsf{m}, r)$ is a verification algorithm that checks whether (m, r) is the correct opening of the commitment cm and outputs a bit $b \in \{0, 1\}$ representing accept if $b = 1$ and reject otherwise.

Informally, a commitment scheme is called binding if it is infeasible to open a commitment to a different message. It is called hiding if the commitments of any two messages are indistinguishable. Commitment schemes can be built from collision-resistant hash functions.

Linear-only encoding schemes A linear-only encoding scheme $\mathcal{E} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Vfy}, \mathsf{Add}, \mathsf{QuadTest})$ is a tuple of efficient algorithms defined over an encoding space $\mathbb{C}$ and a domain $\mathbb{F}$ where:

- $\mathbb{C}, \mathsf{pk} \leftarrow \mathsf{Gen}(1^\lambda)$ is a generation algorithm that outputs the encoding space $\mathbb{C}$ and public key pk.

- $c \leftarrow \mathsf{Enc}(\mathsf{pk}, x)$ is an encoding algorithm that outputs the encoded value $c \in \mathbb{C}$ for input $x \in \mathbb{F}$.
- $b \leftarrow \mathsf{Vfy}(\mathsf{pk}, c)$ is a verification algorithm that outputs a bit $b \in \{0,1\}$ representing whether $c \in \mathbb{C}$ is a valid encoding of some element in the domain $\mathbb{F}$.
- $c^* \leftarrow \mathsf{Add}(\mathsf{pk}, c_1, c_2)$ is a homomorphic evaluation algorithm that outputs $c^* = \mathsf{Enc}(\mathsf{pk}, x_1 + x_2)$ for $c_1 = \mathsf{Enc}(\mathsf{pk}, x_1)$ and $c_2 = \mathsf{Enc}(\mathsf{pk}, x_2)$.
- $b \leftarrow \mathsf{QuadTest}(\mathsf{pk}, (\mathbf{c}_1, \mathbf{c}_2, \mathbf{c}_3), \boldsymbol{\alpha})$ is a degree-two algebraic test algorithm that takes a vector $(\mathbf{c}_1, \mathbf{c}_2, \mathbf{c}_3) \in \mathbb{C}^{3n}$ and a vector $\boldsymbol{\alpha} \in \mathbb{F}^n$ as input and outputs a bit $b \in \{0,1\}$ representing whether $c_{ij} = \mathsf{Enc}(\mathsf{pk}, x_{ij})$ and $\langle \boldsymbol{x}_1, \boldsymbol{x}_2 \rangle = \langle \boldsymbol{\alpha}, \boldsymbol{x}_3 \rangle$ for some vectors $\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3 \in \mathbb{F}^n$.

This scheme must satisfy the linear-only property. Informally, this property ensures that if an adversary $\mathcal{A}$ receives encodings $\{c_i\}_{i\in[n]}$ of $\{x_i\}_{i\in[n]}$ and outputs a valid encoding c^*, then c^* must encode a linear combination of $\{x_i\}_{i\in[n]}$. We do not specifically require that the scheme satisfies the one-way property. Informally, this property ensures that it is infeasible to decode the encoding of a random element x. However, the pairing-based construction does satisfy this property.

Linear-only. A linear-only encoding scheme has the linear-only property if for any polynomial-size adversary $\mathcal{A}$ there is a polynomial-size extractor Ext such that for any sufficiently large $\lambda \in \mathbb{N}$, any auxiliary input $z \in \{0,1\}^{\mathrm{poly}(\lambda)}$, and any plaintext generator $\mathcal{M}$, and some $\varPi \in \mathbb{F}^{k\times m}$, $b \in \mathbb{F}^k$:

$$\Pr\left[\begin{array}{l|l} (a'_1,\ldots,a'_k)^\top = \varPi \cdot (a_1,\ldots,a_m)^\top + b & \mathbb{C}, \mathsf{pk} \leftarrow \mathsf{Gen}(1^\lambda) \\ \wedge & (a_1,\ldots a_m) \leftarrow \mathbb{F}^m \\ \exists i \in [k] : \mathsf{Vfy}(\mathsf{pk}, c'_i) = 1, c'_i \notin \mathsf{Enc}(\mathsf{pk}, a'_i) & (c_1,\ldots,c_m) \leftarrow (\mathsf{Enc}(\mathsf{pk}, a_1),\ldots,\mathsf{Enc}(\mathsf{pk}, a_m)) \\ & (c'_1,\ldots,c'_k) \leftarrow \mathcal{A}(\mathsf{pk}, c_1,\ldots,c_m; z) \\ & (\varPi, b) \leftarrow \mathsf{Ext}(\mathsf{pk}, c_1,\ldots,c_m; z) \end{array}\right] \leq \mathrm{negl}(\lambda).$$

For $\mathbf{c} \in \mathbb{C}^n$, $\boldsymbol{x} \in \mathbb{F}^n$, we introduce the notation $\mathbf{c} = \mathsf{Enc}(\mathsf{pk}, \boldsymbol{x})$ if $\mathbf{c}_i = \mathsf{Enc}(\mathsf{pk}, \boldsymbol{x}_i)$ for $i \in [n]$.

Bilinear group structures Pairing-based arguments rely on cryptographic assumptions about bilinear groups. We formalize these assumptions via a bilinear group sampler, which is an efficient algorithm $\mathsf{SampleGrp}$ that given a security parameter λ (represented as 1^λ), returns $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, q, g_1, g_2, e)$ where $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ are groups with order divisible by the prime $q \in \mathbb{N}$, g_1 generates $\mathbb{G}_1$, g_2 generates $\mathbb{G}_2$, and $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is a (non-degenerate) bilinear map. We use the additive bracket notation, writing $[a]_\kappa$ to denote $a \cdot g_\kappa$ where $g_\kappa = [1]_\kappa$ is a fixed generator of $\mathbb{G}_\kappa$ for $\kappa \in \{1, 2, T\}$. A bilinear group structure is of type 3 if there is no efficiently computable homomorphism from $\mathbb{G}_2$ to $\mathbb{G}_1$. We consider pairings of type 3 only. We rely on the q-DLOG assumption [FKL18, EG23a], which states that for any PPT algorithm $\mathcal{A}$:

$$\Pr\left[y = x \;\middle|\; \begin{array}{c} \mathsf{grp} := (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, q, g_1, g_2, e) \leftarrow \mathsf{SampleGrp}(1^\lambda); x \leftarrow\!\!\$\, \mathbb{F} \\ y \leftarrow \mathcal{A}(\mathsf{grp}, [1]_1, [x]_1, \ldots, [x^q]_1, [1]_2, [x]_2, \ldots, [x^q]_2) \end{array}\right] \leq \mathsf{negl}(\lambda).$$

2.2 Interactive proofs

We adopt the definitions of [BCL+21, AFK22, BC23] in the following preliminary sections.

Public-coin interactive proof An interactive proof $\Pi = (\mathsf{P}, \mathsf{V})$ for a relation $\mathcal{R}$ consists of an interactive protocol whereby the prover P, holding a witness $\mathbf{w}$, interacts with the verifier V on common input pi to convince V that $(\mathsf{pi}, \mathbf{w}) \in \mathcal{R}$. At the end, V outputs a bit for accept/reject. Accordingly, we say that the protocol's transcript (i.e., the set of all messages exchanged in the protocol execution) is accepting or rejecting. The protocol is considered to be public coin if the verifier randomness is public. The verifier messages are referred to as challenges. Π is a $(2k-1)$-move protocol if there are k prover messages and $k-1$ verifier messages.

2.3 Non-interactive arguments in the ROM

Random-Oracle Non-Interactive Argument of Knowledge (RO-NARK) A non-interactive random oracle proof for a relation $\mathcal{R}$ is a pair (P, V) of probabilistic random oracle algorithms such that: Given $(\mathsf{pi}, \mathbf{w}) \in \mathcal{R}$ and access to a random oracle ρ_{NARK}, the prover $\mathsf{P}^{\rho_{\mathsf{NARK}}}(\mathsf{pi}, \mathbf{w})$ outputs a proof π. Given pi, a proof π, and access to the same random oracle ρ_{NARK}, the verifier $\mathsf{V}^{\rho_{\mathsf{NARK}}}(\mathsf{pi}, \pi)$ outputs 0 to accept or any other value to reject. These algorithms have the following security properties:

Perfect completeness. The NARK has perfect completeness if for all $(\mathsf{pi}, \mathbf{w}) \in \mathcal{R}$:

$$\Pr[\mathsf{V}^{\rho_{\mathsf{NARK}}}(\mathsf{pi}, \mathsf{P}^{\rho_{\mathsf{NARK}}}(\mathsf{pi}, \mathbf{w})) = 0] = 1$$

Knowledge soundness. The NARK has adaptive knowledge-soundness with knowledge error $\kappa : \mathbb{N} \times \mathbb{N} \to [0, 1]$ if there exists a knowledge extractor Ext, with the following properties:

The knowledge extractor, given input n, and oracle-access to any polynomial-time Q-query random oracle prover P^* that outputs statements of size n, runs in expected polynomial time in $|\mathsf{pi}| + Q$, and outputs $\{(\mathsf{pi}, \pi, \mathsf{aux}, v; \mathbf{w})\}$ such that a) $\{(\mathsf{pi}, \pi, \mathsf{aux}, v)\}$ is identically distributed to $\{(\mathsf{pi}, \pi, \mathsf{aux}, v) : (\mathsf{pi}, \pi, \mathsf{aux}) \leftarrow \mathsf{P}^{*,\rho_{\mathsf{NARK}}}, v \leftarrow \mathsf{V}^{\rho_{\mathsf{NARK}}}(\mathsf{pi}, \pi)\}$ and b)

$$\Pr\left[\begin{array}{c}(\mathsf{pi}, \mathbf{w}) \in \mathcal{R} \\ \wedge\ \mathsf{V}^{\rho_{\mathsf{NARK}}}(\mathsf{pi}, \pi) = 0\end{array} \;\middle|\; (\mathsf{pi}, \pi, \mathsf{aux}, v; \mathbf{w}) \leftarrow \mathsf{Ext}^{\mathsf{P}^*}\right] \geq \frac{\epsilon(\mathsf{P}^*) - \kappa(n, Q)}{\mathsf{poly}(n)},$$

where $\epsilon(\mathsf{P}^*)$ is P^*'s success probability, i.e., $\epsilon(\mathsf{P}^*) = \Pr[\mathsf{V}^{\rho_{\mathsf{NARK}}}(\mathsf{pi}, \pi) = 0 : \pi, \mathsf{aux}) \leftarrow \mathsf{P}^{*,\rho_{\mathsf{NARK}}}]$. The knowledge extractor Ext can arbitrarily program the random oracle for the prover P^*.

Fiat-Shamir transformation Let $\mathsf{FS}[\Pi]$ denote the Fiat-Shamir transformation of a public-coin interactive proof Π. Specifically, $\mathsf{FS}[\Pi] = (\mathsf{P}_{\mathsf{FS}}, \mathsf{V}_{\mathsf{FS}})$ is a RO-NARK, where $\mathsf{P}^{\rho_{\mathsf{NARK}}}(\mathsf{pi}, \mathbf{w})$ runs $\mathsf{P}(\mathsf{pi}, \mathbf{w})$ but instead of receiving challenge r_i, on message $\mathbf{m}_i$, from the verifier, it computes it as follows: $r_i = \rho_{\mathsf{NARK}}(r_{i-1}, \mathbf{m}_i)$ and $r_0 = \rho_{\mathsf{NARK}}(\mathsf{pi})$. The prover $\mathsf{P}_{\mathsf{FS}}^{\rho_{\mathsf{NARK}}}$ outputs $\pi = (\mathbf{m}_1, \ldots, \mathbf{m}_k)$. The verifier $\mathsf{V}_{\mathsf{FS}}^{\rho_{\mathsf{NARK}}}$ accepts, if V accepts the transcript $(\mathbf{m}_1, r_1, \ldots, \mathbf{m}_k, r_k, \mathbf{m}_{k+1})$ for input pi and the challenges that are computed as specified above. The Fiat-Shamir heuristic states that a random-oracle NARK with negligible knowledge error yields a NARK that has negligible knowledge error in the standard model if the random oracle is replaced with a secure cryptographic hash function H.

2.4 Special-sound protocols

Tree of transcripts Let Π be a $(2k + 1)$-move public-coin interactive proof for a relation $\mathcal{R}$ with challenge spaces $\mathsf{Ch}_1, \ldots, \mathsf{Ch}_k$. For $\mathbf{n} = (n_1, \ldots, n_k) \in \mathbb{N}^k$, we say that $\mathcal{T}$ is an $\mathbf{n}$-tree of transcripts if:

- $\mathcal{T}$ is a tree of depth $k + 1$.
- For each $i \in [k+1]$, each vertex at depth i is labeled with a prover's i-th message $\mathbf{m}_i$, and if $i \leq k$, has exactly n_i outgoing edges to its children with with each edge labeled with a verifier's i-th challenge $r_{i,1}, \ldots, r_{i,n_i}$. We require that the challenges $r_{i,1}, \ldots, r_{i,n_i}$ are distinct. Additionally, the root's label is prepended with the public input pi, so the label becomes $(\mathsf{pi}, \mathbf{m}_1)$.
- The labels on any root-to-leaf path form a valid input-transcript pair $(\mathsf{pi}, \mathsf{tr})$.

We simply write the $\mathbf{n}$-tree of transcripts as an n^k-tree of transcripts when $n = n_1 = n_2 = \cdots = n_k$. We assume for simplicity that the challenge spaces $\mathsf{Ch}_1, \ldots, \mathsf{Ch}_k$ all have the same cardinality N.

Special-soundness Given $k, N \in \mathbb{N}$ and $\mathbf{n} = (n_1, \ldots, n_k) \in \mathbb{N}^k$, a $(2k+1)$-move public-coin interactive proof Π for a relation $\mathcal{R}$ where the verifier samples its challenges from sets of size N is $\mathbf{n}$-out-of-N special-sound if there exists a polynomial time algorithm that, on input pi and any $\mathbf{n}$-tree of transcripts for Π outputs $(\mathsf{pi}, \mathbf{w}) \in \mathcal{R}$. We simply denote the protocol as an n^k-out-of-N (or n^k) special-sound protocol if $n = n_1 = n_2 = \cdots = n_k$.

Knowledge error As shown in [AFK22], the Fiat-Shamir transformation of an $\mathbf{n}$-out-of-N special-sound protocol Π is knowledge sound with knowledge error $\kappa_{\mathsf{FS}}(Q) = (Q + 1)\kappa$, where we consider $\kappa = (1 - \prod_{i=1}^{k}(1 - \frac{n_i}{N}))$ to be the knowledge error of the public-coin interactive proof Π.

2.5 Accumulation schemes

We adopt the definition from Protostar [BC23]. An accumulation scheme for a NARK $(\mathsf{P}_{\mathsf{NARK}}, \mathsf{V}_{\mathsf{NARK}})$ is a triple of algorithms $\mathsf{ACC} = (\mathsf{P}_{\mathsf{acc}}, \mathsf{V}_{\mathsf{acc}}, \mathsf{D})$, all of which have access to the same random oracle ρ_{acc} as well as ρ_{NARK}, the oracle for the NARK. The algorithms have the following syntax and properties:

- $\mathsf{P}_{\mathsf{acc}}(\mathsf{pi}, \pi = (\pi.\mathsf{x}, \pi.\mathsf{w}), \mathsf{acc} = (\mathsf{acc.x}, \mathsf{acc.w})) \to (\mathsf{acc}' = (\mathsf{acc}'.\mathsf{x}, \mathsf{acc}'.\mathsf{w}), \mathsf{pf})$. The accumulation prover $\mathsf{P}_{\mathsf{acc}}$ takes as input the public input pi, the NARK proof π, and the accumulator acc and returns a new accumulator acc' and the correction terms pf.

– $\mathsf{V}_{\mathsf{acc}}(\mathsf{pi}, \pi.\mathsf{x}, \mathsf{acc}, \mathsf{acc}', \mathsf{pf}) \to b$. The accumulation verifier $\mathsf{V}_{\mathsf{acc}}$ takes as input the public input pi, the instance of the NARK proof $\pi.\mathsf{x}$, the old accumulator acc, the new accumulator acc', and the correction terms pf. The verifier 'accepts' by outputting 0 and 'rejects' otherwise.
– $\mathsf{D}(\mathsf{acc}) \to b$. The decider D 'accepts' an accumulator acc by outputting 0 and 'rejects' otherwise.

An accumulation scheme is knowledge sound with knowledge error κ if the RO-NARK $(\mathsf{P}', \mathsf{V}')$ has knowledge error κ for the relation:

$$\mathcal{R}_{\mathsf{acc}}\big((\mathsf{pi}, \pi.\mathsf{x}, \mathsf{acc.x}); (\pi.\mathsf{w}, \mathsf{acc.w})\big) : \big(\mathsf{V}_{\mathsf{NARK}}(\mathsf{pi}, \pi) = 0 \wedge D(\mathsf{acc}) = 0\big)$$

where P' outputs $(\mathsf{acc}', \mathsf{pf})$ and V' on input $((\mathsf{pi}, \pi.\mathsf{x}, \mathsf{acc.x}), (\mathsf{acc}', \mathsf{pf}))$ accepts if $\mathsf{D}(\mathsf{acc}') = 0$ and $\mathsf{V}_{\mathsf{acc}}(\mathsf{pi}, \pi.\mathsf{x}, \mathsf{acc.x}, \mathsf{acc}'.\mathsf{x}, \mathsf{pf}) = 0$. An accumulation scheme is perfectly complete if the RO-NARK $(\mathsf{P}', \mathsf{V}')$ has perfect completeness for the relation $\mathcal{R}_{\mathsf{acc}}$.

2.6 Proof-carrying data

Proof-carrying data (PCD) [CT10] enables a set of parties to carry out an arbitrarily long distributed computation where every step is accompanied by a proof of correctness.

Let $V(\mathsf{G})$ and $E(\mathsf{G})$ denote the vertices and edges of a graph G. A transcript T is a directed acyclic graph where each vertex $u \in V(\mathsf{T})$ is labeled by local data $z_{\mathsf{loc}}^{(u)}$ and each edge $e \in E(\mathsf{T})$ is labeled by a message $z^{(e)} \neq \bot$. The output of a transcript T, denoted $\mathsf{o}(\mathsf{T})$, is $z^{(e')}$ where $e' = (u, v)$ is the lexicographically-first edge such that v is a sink.

A vertex $u \in V(\mathsf{T})$ is φ-compliant for a predicate $\varphi \in \mathsf{F}$ if for all outgoing edges $e = (u, v) \in E(\mathsf{T})$ either: (1) if u has no incoming edges, $\varphi(z^{(e)}, z_{\mathsf{loc}}^{(u)}, \bot, \ldots, \bot)$ evaluates to true or (2) if u has m incoming edges $e_1, ..., e_m$, $\varphi(z^{(e)}, z_{\mathsf{loc}}^{(u)}, z^{(e_1)}, \ldots, z^{(e_m)})$ evaluates to true. A transcript T is φ-compliant if all of its vertices are φ-compliant.

A proof-carrying data system PCD for a class of compliance predicates F consists of a tuple of efficient algorithms $(\mathbb{G}, \mathbb{I}, \mathbb{P}, \mathbb{V})$, known as the generator, indexer, prover, and verifier algorithms, for which the properties of completeness, knowledge soundness, and zero knowledge are defined.

Completeness. PCD has perfect completeness if for every adversary $\mathcal{A}$ the following holds:

$$\Pr\left[\begin{array}{c} \varphi \in \mathsf{F} \\ \wedge\ \varphi(z, z_{\mathsf{loc}}, z_1, \ldots, z_m) = 1 \\ \wedge\ (\forall i, z_i = \bot \vee \forall i, \mathbb{V}(\mathsf{ivk}, z_i, \pi_i) = 1) \\ \Downarrow \\ \mathbb{V}(\mathsf{ivk}, z, \pi) = 1 \end{array} \middle| \begin{array}{c} \mathsf{pp} \leftarrow \mathbb{G}(1^\lambda) \\ (\varphi, z, z_{\mathsf{loc}}, [z_i, \pi_i]_{i=1}^m) \leftarrow \mathcal{A}(\mathsf{pp}) \\ (\mathsf{ipk}, \mathsf{ivk}) \leftarrow \mathbb{I}(\mathsf{pp}, \varphi) \\ \pi \leftarrow \mathbb{P}(\mathsf{ipk}, z, z_{\mathsf{loc}}, [z_i, \pi_i]_{i=1}^m) \end{array}\right] = 1.$$

Knowledge soundness. PCD has knowledge soundness with respect to an auxiliary input distribution $\mathcal{D}$ if for every expected polynomial-time adversary $\tilde{\mathbb{P}}$ there exists an expected polynomial-time extractor $\mathbb{E}_{\tilde{\mathbb{P}}}$ such that for every set Z:

$$\begin{aligned} &\Pr\left[\begin{array}{c} \varphi \in \mathsf{F} \\ \wedge\ (\mathsf{pp}, \mathsf{ai}, \varphi, \mathsf{o}(\mathsf{T}), \mathsf{ao}) \in Z \\ \wedge\ \mathsf{T} \text{ is } \varphi\text{-compliant} \end{array} \middle| \begin{array}{c} \mathsf{pp} \leftarrow \mathbb{G}(1^\lambda) \\ \mathsf{ai} \leftarrow \mathcal{D}(\mathsf{pp}) \\ (\varphi, \mathsf{T}, \mathsf{ao}) \leftarrow \mathbb{E}_{\tilde{\mathbb{P}}}(\mathsf{pp}, \mathsf{ai}) \end{array}\right] \\ \geq &\Pr\left[\begin{array}{c} \varphi \in \mathsf{F} \\ \wedge\ (\mathsf{pp}, \mathsf{ai}, \varphi, \mathsf{o}, \mathsf{ao}) \in Z \\ \wedge\ \mathbb{V}(\mathsf{ivk}, \mathsf{o}, \pi) = 1 \end{array} \middle| \begin{array}{c} \mathsf{pp} \leftarrow \mathbb{G}(1^\lambda) \\ \mathsf{ai} \leftarrow \mathcal{D}(\mathsf{pp}) \\ (\varphi, \mathsf{o}, \pi, \mathsf{ao}) \leftarrow \tilde{\mathbb{P}}(\mathsf{pp}, \mathsf{ai}) \\ (\mathsf{ipk}, \mathsf{ivk}) \leftarrow \mathbb{I}(\mathsf{pp}, \varphi) \end{array}\right] - \mathsf{negl}(\lambda). \end{aligned}$$

Zero knowledge. PCD has (statistical) zero knowledge if there exists a PPT simulator $\mathbb{S}$ such that for every honest adversary $\mathcal{A}$ the distributions below are statistically indistinguishable:

$$\left\{(\mathsf{pp}, \varphi, z, \pi) \;\middle|\; \begin{array}{c} \mathsf{pp} \leftarrow \mathbb{G}(1^\lambda) \\ (\varphi, z, z_{\mathsf{loc}}, [z_i, \pi_i]_{i=1}^m) \leftarrow \mathcal{A}(\mathsf{pp}) \\ (\mathsf{ipk}, \mathsf{ivk}) \leftarrow \mathbb{I}(\mathsf{pp}, \varphi) \\ \pi \leftarrow \mathbb{P}(\mathsf{ipk}, z, z_{\mathsf{loc}}, [z_i, \pi_i]_{i=1}^m) \end{array}\right\} \text{ and } \left\{(\mathsf{pp}, \varphi, z, \pi) \;\middle|\; \begin{array}{c} (\mathsf{pp}, \tau) \leftarrow \mathbb{S}(1^\lambda) \\ (\varphi, z, z_{\mathsf{loc}}, [z_i, \pi_i]_{i=1}^m) \leftarrow \mathcal{A}(\mathsf{pp}) \\ \pi \leftarrow \mathbb{S}(\tau, \varphi, z) \end{array}\right\}$$

An adversary is honest if their output results in the implicant of the completeness condition being satisfied with probability 1, i.e., $\varphi \in \mathsf{F}$, $\varphi(z, z_{\mathsf{loc}}, z_1, \ldots, z_m) = 1$, and either $z_i = \bot$ or $\mathbb{V}(\mathsf{ivk}, z_i, \pi_i) = 1$

for each incoming edge z_i. A proof π has size $\mathsf{poly}(\lambda, |\varphi|)$; that is, the proof size is not allowed to grow with each application of the prover algorithm $\mathbb{P}$.

2.7 Algebraic group model

The pairing-based arguments that we consider in this work have been proven secure in the algebraic group model (AGM) [FKL18]. However, the special-sound protocols that we consider do not rely on the AGM. In particular, we define simple interactive proofs where the prover sends group elements and the verifier performs algebraic tests that are directly related to the original pairing-based argument.

3 Construction

In this section, we present the general construction of our accumulation scheme. This general form is necessary to derive the specific applications to Groth16 proof aggregation and verifiable ML inference. We follow the high-level approach of previous folding schemes. Namely, we assume that the verifier has homomorphic commitments to the witnesses and ask the verifier to check a random combination of the witness commitments. The prover must compute the corresponding combination of the witnesses themselves. Additional work is needed to ensure the correctness of the error terms that arise from the random combination. The below formalization is useful for defining the specific error terms and decider checks for this broader class of special-sound protocols, which includes pairing-based arguments.

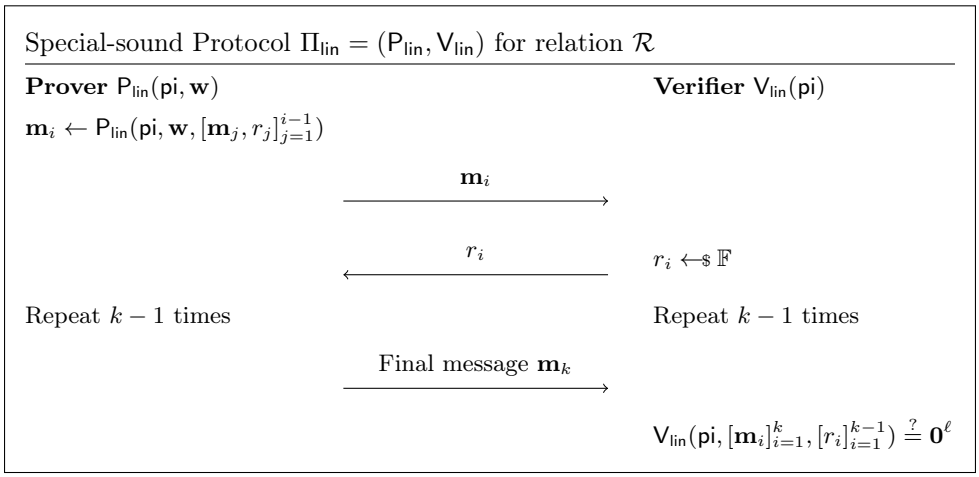
3.1 Special-sound protocols

Protostar considered a specific class of special-sound protocols where the verifier checks algebraic equations. In this case, the protocol Π_{sps} is parametrized by $k, d, \ell \in \mathbb{Z}^+$ such that Π_{sps} is a $(2k+1)$-move protocol with verifier degree d and output length ℓ. The verifier checks ℓ degree-d algebraic equations. In each round i ($1 \leq i \leq k$), the prover $\mathsf{P}_{\mathsf{sps}}(\mathsf{pi}, \mathbf{w}, [\mathbf{m}_j, r_j]_{j=1}^{i-1})$ generates the next message $\mathbf{m}_i$ based on the public input pi, the witness $\mathbf{w}$, and the current transcript $[\mathbf{m}_j, r_j]_{j=1}^{i-1}$, and sends $\mathbf{m}_i$ to the verifier; the verifier replies with a random challenge $r_i \in \mathbb{F}$. After receiving the final message $\mathbf{m}_k$, the verifier computes the algebraic map $\mathsf{V}_{\mathsf{sps}}$ and checks that the output is a zero vector of length ℓ. This algebraic map is defined as follows:

$$\mathsf{V}_{\mathsf{sps}}(\mathsf{pi}, [\mathbf{m}_i]_{i=1}^{k}, [r_i]_{i=1}^{k-1}) := \sum_{j=0}^{d} f_j^{\mathsf{V}_{\mathsf{sps}}}\left(\mathsf{pi}, [\mathbf{m}_i]_{i=1}^{k}, [r_i]_{i=1}^{k-1}\right).$$

In this work, we consider a broader class of special-sound protocols, where the verifier performs algebraic tests. The protocol Π_{lin} is parametrized by $k, d, \ell \in \mathbb{Z}^+$ such that Π_{lin} is a $(2k+1)$-move protocol with a verifier that computes a map $\mathsf{V}_{\mathsf{lin}}$ over encoded values and field elements, resulting in an output vector in $\{0,1\}^\ell$. In each round i ($1 \leq i \leq k$), the prover $\mathsf{P}_{\mathsf{lin}}(\mathsf{pi}, \mathbf{w}, [\mathbf{m}_j, r_j]_{j=1}^{i-1})$ generates the next message $\mathbf{m}_i \in \mathbb{C}^*$ based on the public input $\mathsf{pi} \in \mathbb{C}$, the witness $\mathbf{w} \in \mathbb{C}^*$, and the current transcript $[\mathbf{m}_j, r_j]_{j=1}^{i-1} \in (\mathbb{C}^* \times \mathbb{F})^{i-1}$. The prover sends $\mathbf{m}_i$ to the verifier, and the verifier replies with a challenge $r_i \in \mathbb{F}$. After receiving the final message $\mathbf{m}_k$, the verifier computes $\mathsf{V}_{\mathsf{lin}}$ and checks that the map's output is a zero vector of length ℓ. The map $\mathsf{V}_{\mathsf{lin}}$ performs algebraic tests that directly correspond to degree-d algebraic checks.

To define the map $\mathsf{V}_{\mathsf{lin}}$, we first need to define sets $A_{j,i}$, $B_{j,i}$, and $D_{j,i}$, corresponding to subsets of the set $M = \{\mathbf{m}_i\}_{i=1}^{k} \cup \{\mathsf{pi}\}$, which consists of the prover messages and the public input. Specifically, we let $A_{j,i} = \{\mathbf{m}_{j,i}^{\mathsf{A}}\}$, $B_{j,i} = \{\mathbf{m}_{j,i}^{\mathsf{B}}\}$ and $D_{j,i} = \{\mathbf{m}_{j,i}^{\mathsf{D}}\}$ such that $A_{j,i} \subseteq M$, $B_{j,i} \subseteq M$ and $D_{j,i} \subseteq M$. For each of these sets, we consider the set consisting of the corresponding decoded elements. Specifically, we let $A'_{j,i} = \{\hat{\mathbf{m}}_{j,i}^{\mathsf{A}}\}$, $B'_{j,i} = \{\hat{\mathbf{m}}_{j,i}^{\mathsf{B}}\}$, and $D'_{j,i} = \{\hat{\mathbf{m}}_{j,i}^{\mathsf{D}}\}$ such that $\mathsf{Enc}_i(\mathsf{pk}_i, \hat{\mathbf{m}}_{j,i}^{\mathsf{A}}) = \mathbf{m}_{j,i}^{\mathsf{A}}$, $\mathsf{Enc}_i(\mathsf{pk}_i, \hat{\mathbf{m}}_{j,i}^{\mathsf{B}}) = \mathbf{m}_{j,i}^{\mathsf{B}}$, and $\mathsf{Enc}_i(\mathsf{pk}_i, \hat{\mathbf{m}}_{j,i}^{\mathsf{D}}) = \mathbf{m}_{j,i}^{\mathsf{D}}$. We let $R = \{r_i\}_{i=1}^{k-1}$ denote the set of challenges. We also introduce vectors of constant field elements $(\hat{\mathbf{a}}_{j,i}, \hat{\mathbf{b}}_{j,i}, \hat{\mathbf{d}}_{j,i}) \in \mathbb{F}^{3n-k}$ and $\mathbf{s}_{j,i} \in \mathbb{F}^{n-k+1}$. To simplify the subsequent definitions, we define vectors of field elements $\hat{\boldsymbol{c}}_{1,j,i}, \hat{\boldsymbol{c}}_{2,j,i}, \hat{\boldsymbol{c}}_{3,j,i}, \boldsymbol{\gamma}_{j,i} \in \mathbb{F}^n$ such that $\hat{\boldsymbol{c}}_{1,j,i} = A'_{j,i} \| \hat{\mathbf{a}}_{j,i}$, $\hat{\boldsymbol{c}}_{2,j,i} = B'_{j,i} \| \hat{\mathbf{b}}_{j,i}$, $\hat{\boldsymbol{c}}_{3,j,i} = D'_{j,i} \| \hat{\mathbf{d}}_{j,i}$, and $\boldsymbol{\gamma}_{j,i} = R \| \mathbf{s}_{j,i}$.



We require $f_j^{\mathsf{V}'_{\mathsf{lin}}}$ to be a homogeneous degree-j algebraic map that outputs a vector of ℓ field elements. In an algebraic map, each output element is given by a polynomial over field elements. In this context, homogeneous simply means that the nonzero terms of the polynomial have identical degree, where the degree is defined with respect to the decoded messages and challenges. Specifically, we let $f_j^{\mathsf{V}'_{\mathsf{lin}}}([\hat{\mathbf{m}}_i]_{i=1}^{k}, [r_i]_{i=1}^{k-1}) = [\langle A'_{j,\kappa} \| \hat{\mathbf{a}}_{j,\kappa}, B'_{j,\kappa} \| \hat{\mathbf{b}}_{j,\kappa} \rangle - \langle R \| \mathbf{s}_{j,\kappa}, D'_{j,\kappa} \| \hat{\mathbf{d}}_{j,\kappa} \rangle]_{\kappa=1}^{\ell}$. Equivalently, we have $f_j^{\mathsf{V}'_{\mathsf{lin}}}([\hat{\mathbf{m}}_i]_{i=1}^{k}, [r_i]_{i=1}^{k-1}) = [\langle \hat{\boldsymbol{c}}_{1,j,\kappa}, \hat{\boldsymbol{c}}_{2,j,\kappa} \rangle - \langle \boldsymbol{\gamma}_{j,\kappa}, \hat{\boldsymbol{c}}_{3,j,\kappa} \rangle]_{\kappa=1}^{\ell}$. Recall that $A'_{j,\kappa}$, $B'_{j,\kappa}$, $D'_{j,\kappa}, R$ are defined by the prover messages and verifier challenges, whereas $\hat{\mathbf{a}}_{j,\kappa}, \hat{\mathbf{b}}_{j,\kappa}, \hat{\mathbf{d}}_{j,\kappa}, \mathbf{s}_{j,\kappa}$ are vectors of constant field elements that do not contribute to the degree of the algebraic map $f_j^{\mathsf{V}'_{\mathsf{lin}}}$. From these algebraic maps, we define a degree-d algebraic map $\mathsf{V}'_{\mathsf{lin}}([\hat{\mathbf{m}}_i]_{i=1}^{k}, [r_i]_{i=1}^{k-1}, \mu, \hat{\mathbf{e}}) = \sum_{j=0}^{d} \mu^{d-j} \cdot f_j^{\mathsf{V}'_{\mathsf{lin}}}([\hat{\mathbf{m}}_i]_{i=1}^{k}, [r_i]_{i=1}^{k-1}) - \hat{\mathbf{e}}$.

Next, we define corresponding algebraic tests. First, we define encoded vectors $(\mathbf{c}_{1,j,i}, \mathbf{c}_{2,j,i}, \mathbf{c}_{3,j,i}) \in \mathbb{C}^{3n}$ such that $\mathbf{c}_{1,j,i} = \mathsf{Enc}_i(\mathsf{pk}_i, \mu^{d-j} \cdot \hat{\mathbf{c}}_{1,j,i})$, $\mathbf{c}_{2,j,i} = \mathsf{Enc}_i(\mathsf{pk}_i, \hat{\mathbf{c}}_{2,j,i})$, and $\mathbf{c}_{3,j,i} = \mathsf{Enc}_i(\mathsf{pk}_i, \mu^{d-j} \cdot \hat{\mathbf{c}}_{3,j,i})$. We introduce an error map $h^{\mathsf{V}_{\mathsf{lin}}}(\mathbf{e}) = [\mathsf{QuadTest}_{\kappa}(\mathsf{pk}_{\kappa}, \mathsf{Enc}_{\kappa}(\mathsf{pk}_{\kappa}, \mathbf{e}_{1,\kappa}), \mathsf{Enc}_{\kappa}(\mathsf{pk}_{\kappa}, \mathbf{e}_{2,\kappa})]_{\kappa=0}^{\ell}$ where $\mathbf{e}_{1,\kappa}$ and $\mathbf{e}_{2,\kappa}$ are disjoint subsets of $\hat{\mathbf{e}}$, whose encodings map to $\mathbb{G}_1$ and $\mathbb{G}_2$ elements, respectively, in the main construction. We define the map of algebraic tests $g_j^{\mathsf{V}_{\mathsf{lin}}}(\mathsf{pi}, [\mathbf{m}_i]_{i=1}^{k}, [r_i]_{i=1}^{k-1}, \mu) = [\mathsf{QuadTest}_i(\mathsf{pk}_i, (\mathbf{c}_{1,j,i}, \mathbf{c}_{2,j,i}, \mathbf{c}_{3,j,i}), \boldsymbol{\gamma}_{j,i})]_{i=1}^{\ell}$. We let $\bigcirc$ denote the entry-wise combination of algebraic tests. The verifier's map $\mathsf{V}_{\mathsf{lin}}$ is given by:

$$\mathsf{V}_{\mathsf{lin}}(\mathsf{pi}, [\mathbf{m}_i]_{i=1}^{k}, [r_i]_{i=1}^{k-1}, \mu, \mathbf{e}) = [\bigcirc_{j=0}^{d} g_j^{\mathsf{V}_{\mathsf{lin}}}(\mathsf{pi}, [\mathbf{m}_i]_{i=1}^{k}, [r_i]_{i=1}^{k-1}, \mu)] \bigcirc h^{\mathsf{V}_{\mathsf{lin}}}(\mathbf{e})$$

Recall that $g_j^{\mathsf{V}_{\mathsf{lin}}}$ and $h^{\mathsf{V}_{\mathsf{lin}}}$ are maps consisting of algebraic tests on encoded values and field elements. The output of the map $\mathsf{V}_{\mathsf{lin}}$ is a vector in $\{0,1\}^{\ell}$. By the definitions above, we can see that the output of the map of algebraic tests $\mathsf{V}_{\mathsf{lin}}(\mathsf{pi}, [\mathbf{m}_i]_{i=1}^{k}, [r_i]_{i=1}^{k-1}, \mu, \mathbf{e})$ is equal to the zero vector if the output of the map of direct algebraic checks $\mathsf{V}'_{\mathsf{lin}}([\hat{\mathbf{m}}_i]_{i=1}^{k}, [r_i]_{i=1}^{k-1}, \mu, \hat{\mathbf{e}})$ is equal to the zero vector.

3.2 Protocol transformations

We define a standard commit-and-open transformation for a special-sound protocol. In each round of the protocol, instead of sending the message $\mathbf{m}_i$, the prover sends a commitment to the message given by $C_i = \mathsf{Commit}(\mathsf{ck}, \mathbf{m}_i)$. In some cases, this will be the trivial identity commitment, however in the regular Protostar case, it will be the elliptic curve Pedersen commitment. In the final round, the prover sends openings to the commitments, and the verifier checks these openings, in addition to performing the algebraic tests. We omit the details for brevity, since this is the same as the commit-and-open transformation that is described in Sec. 3.2 of [BC23]. We let $\Pi_{\mathsf{cm}} = \mathsf{cm}[\Pi_{\mathsf{lin}}]$ denote the committed protocol for the special-sound protocol Π_{lin}.

We let $\mathsf{FS}[\Pi_{\mathsf{cm}}]$ denote the Fiat-Shamir transformation of the committed protocol. Per Lemma 1 and Lemma 2 of [BC23], $\mathsf{FS}[\Pi_{\mathsf{cm}}]$ is knowledge sound if Π_{lin} is special-sound.

3.3 Accumulation scheme

In this section, we provide a formal description of the accumulation scheme, which adapts the core approach in Sec. 3.4 of [BC23] to support our broader class of protocols We highlight the key modifications in blue. Let ρ_{acc} be the random oracle for the accumulation scheme. Let ρ_{NARK} be the random oracle for the NARK $\mathsf{FS}[\Pi_{\mathsf{cm}}]$, and let $\mathsf{V}_{\mathsf{NARK}}$ be the NARK verifier. The NARK proof has instance $\pi.\mathsf{x} = [C_i]_{i=1}^k$ and witness $\pi.\mathsf{w} = \{[\mathbf{m}_i]_{i=1}^k\}$.

We define the accumulator format as follows:

- The instance is denoted by:
  - $\mathsf{acc.x} = \{\mathsf{pi}, [C_i]_{i=1}^k, [r_i]_{i=1}^k, E, \mu\}$
- The witness is denoted by:
  - $\mathsf{acc.w} = \{[\mathbf{m}_i]_{i=1}^k, \mathbf{e}\}$

We define the accumulated predicate as follows:

- The challenge derivation is valid:
  - $r_i = \rho_{\mathsf{NARK}}(r_{i-1}, C_i)$ for all $i \in [k-1]$
  - $r_0 = \rho_{\mathsf{NARK}}(\mathsf{pi})$
- The commitment openings are valid:[2]
  - $\mathsf{Commit}(\mathsf{ck}, \mathbf{m}_i) = C_i$ for all $i \in [k]$
- The relaxed algebraic test holds:
  - $\mathsf{V}_{\mathsf{lin}}(\mathsf{pi}, [\mathbf{m}_i]_{i=1}^k, [r_i]_{i=1}^{k-1}, \mu, \mathbf{e}) :=$
    $\left[\bigcirc_{j=0}^d g_j^{\mathsf{V}_{\mathsf{lin}}}(\mathsf{pi}, [\mathbf{m}_i]_{i=1}^k, [r_i]_{i=1}^{k-1}, \mu)\right] \bigcirc h^{\mathsf{V}_{\mathsf{lin}}}(\mathbf{e}) = \mathbf{0}^\ell$

Accumulation prover Given commitment key ck, accumulator acc, and an instance-proof pair (pi, π), the accumulation prover $\mathsf{P}_{\mathsf{acc}}^{\rho_{\mathsf{acc}}, \rho_{\mathsf{NARK}}}$ works as follows:

1. Derive challenges $r_i \leftarrow \rho_{\mathsf{NARK}}(r_{i-1}, C_i)$ for all $i \in [k-1]$ where $r_0 := \rho_{\mathsf{NARK}}(\mathsf{pi})$
2. Compute error terms $[\mathbf{e}_j]_{j=1}^{d-1}$ such that $\mathbf{e} = \mathsf{Enc}(\mathsf{pk}, \hat{\mathbf{e}})$, $\mathbf{e}_j = \mathsf{Enc}(\mathsf{pk}, \hat{\mathbf{e}}_j)$ for $j \in [d-1]$, and
   $\sum_{j=0}^d (X+\mu)^{d-j} \cdot f_j^{\mathsf{V}'_{\mathsf{lin}}}([X \cdot \hat{\mathbf{m}}_i + \hat{\mathbf{m}}'_i]_{i=1}^k, [X \cdot r_i + r'_i]_{i=1}^{k-1})$
   $= \sum_{j=0}^d \mu^{d-j} \cdot f_j^{\mathsf{V}'_{\mathsf{lin}}}([\hat{\mathbf{m}}'_i]_{i=1}^k, [r'_i]_{i=1}^{k-1}) + X^d \cdot \sum_{j=0}^d f_j^{\mathsf{V}'_{\mathsf{lin}}}([\hat{\mathbf{m}}_i]_{i=1}^k, [r_i]_{i=1}^{k-1}) + \sum_{j=1}^{d-1} \hat{\mathbf{e}}_j X^j$
   $= \hat{\mathbf{e}} + \sum_{j=1}^{d-1} \hat{\mathbf{e}}_j X^j$
3. Compute committed error terms:
   - $E_j \leftarrow \mathsf{Commit}(\mathsf{ck}, \mathbf{e}_j)$
4. Derive $\gamma \leftarrow \rho_{\mathsf{acc}}(\mathsf{acc.x}, \mathsf{pi}, \pi.\mathsf{x}, [E_j]_{j=1}^{d-1}) \in \mathbb{F}$
5. Compute random linear combinations:
   - Set vector $\mathbf{v} := (1, \mathsf{pi}, [r_i]_{i=1}^{k-1}, [C_i]_{i=1}^k, [\mathbf{m}_i]_{i=1}^k)$
   - Set vector $\mathbf{v}' := (\mu, \mathsf{pi}', [r'_i]_{i=1}^{k-1}, [C'_i]_{i=1}^k, [\mathbf{m}'_i]_{i=1}^k)$
   - Set vector $\mathbf{v}'' := (\mu', \mathsf{pi}'', [r''_i]_{i=1}^{k-1}, [C''_i]_{i=1}^k, [\mathbf{m}''_i]_{i=1}^k) \leftarrow \gamma \cdot \mathbf{v} + \mathbf{v}'$
   - $E' \leftarrow E + \sum_{j=1}^{d-1} \gamma^j \cdot E_j$
   - $\mathbf{e}' \leftarrow \mathbf{e} + \sum_{j=1}^{d-1} \gamma^j \cdot \mathbf{e}_j$
6. Compute new accumulator acc':
   - Set instance $\mathsf{acc}'.\mathsf{x} = \{\mathsf{pi}'', [C''_i]_{i=1}^k, [r''_i]_{i=1}^k, E', \mu'\}$
   - Set witness $\mathsf{acc}'.\mathsf{w} = \{[\mathbf{m}''_i]_{i=1}^k, \mathbf{e}'\}$
7. Set correction terms: $\mathsf{pf} = [E_j]_{i=1}^{d-1}$

Accumulation verifier Given public input pi, NARK instance $\pi.\mathsf{x} = [C_i]_{i=1}^k$, accumulator instance $\mathsf{acc.x} = \{\mathsf{pi}', [C'_i]_{i=1}^k, [r_i]_{i=1}^k, E, \mu\}$, correction terms pf, and updated accumulator instance $\mathsf{acc}'.\mathsf{x} = \{\mathsf{pi}'', [C''_i]_{i=1}^k, [r'_i]_{i=1}^k, E', \mu'\}$, the accumulation verifier $\mathsf{V}_{\mathsf{acc}}^{\rho_{\mathsf{acc}}, \rho_{\mathsf{NARK}}}$ works as follows:

- Derive challenges $r_i \leftarrow \rho_{\mathsf{NARK}}(r_{i-1}, C_i)$ for all $i \in [k-1]$ where $r_0 := \rho_{\mathsf{NARK}}(\mathsf{pi})$
- Derive $\gamma \leftarrow \rho_{\mathsf{acc}}(\mathsf{acc.x}, \mathsf{pi}, \pi.\mathsf{x}, \mathsf{pf}) \in \mathbb{F}$
- Set vector $\mathbf{v} := (1, \mathsf{pi}, [r_i]_{i=1}^{k-1}, [C_i]_{i=1}^k)$
- Set vector $\mathbf{v}' := (\mu, \mathsf{pi}', [r'_i]_{i=1}^{k-1}, [C'_i]_{i=1}^k)$
- Check $\mathsf{acc}'.\mathsf{x}.(\mu', \mathsf{pi}'', [r''_i]_{i=1}^{k-1}, [C''_i]_{i=1}^k) \stackrel{?}{=} \gamma \cdot \mathbf{v} + \mathbf{v}'$
- Check $\mathsf{acc}'.\mathsf{x}.E' \stackrel{?}{=} \mathsf{acc.x}.E + \sum_{j=1}^{d-1} \gamma^j \cdot E_j$

[2] These checks can be omitted if the trivial identity commitment is used in the accumulation scheme.

Accumulation decider Given accumulator acc with instance $\mathsf{acc.x} = \{\mathsf{pi}, [C_i]_{i=1}^k, [r_i]_{i=1}^k, E, \mu\}$ and witness $\mathsf{acc.w} = \{[\mathbf{m}_i]_{i=1}^k, \mathbf{e}\}$, the accumulation decider $\mathsf{D}_{\mathsf{acc}}$ works as follows:

- Check $C_i \stackrel{?}{=} \mathsf{Commit}(\mathsf{ck}, \mathbf{m}_i)$ for all $i \in [k]$
- Check $\mathsf{V}_{\mathsf{lin}}(\mathsf{pi}, [\mathbf{m}_i]_{i=1}^k, [r_i]_{i=1}^{k-1}, \mu, \mathbf{e}) \stackrel{?}{=} \mathbf{0}^\ell$
- Check $E \stackrel{?}{=} \mathsf{Commit}(\mathsf{ck}, \mathbf{e})$

Error term computation By the bilinearity of the inner product, each coefficient of the error polynomial can be expressed as an inner product of the form $\langle \boldsymbol{x}_1, \boldsymbol{x}_2 \rangle = \langle \boldsymbol{\alpha}, \boldsymbol{x}_3 \rangle$ for some $\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3, \boldsymbol{\alpha} \in \mathbb{F}^n$. An explicit formula for error term calculation is provided in Sec. 3.6 of Protostar [BC23]. The prover may compute error terms using optimized formulas for each application.

Security proof Completeness follows from the homomorphism of the linear-only encoding scheme. Knowledge soundness follows from the linear-only property of the linear-only encoding scheme and the special-soundness of the underlying interactive proof (i.e., the protocol that performs the corresponding algebraic checks). We prove that our scheme satisfies the knowledge soundness property.

We first define an adversary $\mathcal{A}_{\mathsf{enc}}$ for the linear-only encoding scheme $\mathcal{E}$ with a polynomial-size extractor $\mathsf{Ext}_{\mathsf{enc}}$. Suppose we have an accumulation scheme $\mathsf{ACC} = (\mathsf{P}_{\mathsf{acc}}, \mathsf{V}_{\mathsf{acc}}, \mathsf{D})$ for a NARK $(\mathsf{P}_{\mathsf{test}}, \mathsf{V}_{\mathsf{test}})$. Let $\mathsf{ACC}' = (\mathsf{P}'_{\mathsf{acc}}, \mathsf{V}'_{\mathsf{acc}}, \mathsf{D}')$ be an accumulation scheme for the NARK $(\mathsf{P}_{\mathsf{check}}, \mathsf{V}_{\mathsf{check}})$ that performs the corresponding algebraic checks on the underlying field elements.

Recall that the accumulation scheme ACC is knowledge sound with knowledge error κ if the RO-NARK $(\mathsf{P}'_{\mathsf{test}}, \mathsf{V}'_{\mathsf{test}})$ has knowledge error κ for the relation:

$$\mathcal{R}_{\mathsf{acc}}\big((\mathsf{pi}, \pi.\mathsf{x}, \mathsf{acc.x}); (\pi.\mathsf{w}, \mathsf{acc.w})\big) : \big(\mathsf{V}_{\mathsf{test}}(\mathsf{pi}, \pi) = 0 \wedge \mathsf{D}(\mathsf{acc}) = 0\big)$$

where $\mathsf{P}'_{\mathsf{test}}$ outputs $(\mathsf{acc}', \mathsf{pf})$ and $\mathsf{V}'_{\mathsf{test}}$ on input $((\mathsf{pi}, \pi.\mathsf{x}, \mathsf{acc.x}), (\mathsf{acc}', \mathsf{pf}))$ accepts if $\mathsf{D}(\mathsf{acc}') = 0$ and $\mathsf{V}_{\mathsf{acc}}(\mathsf{pi}, \pi.\mathsf{x}, \mathsf{acc.x}, \mathsf{acc}'.\mathsf{x}, \mathsf{pf}) = 0$.

We let Π_{test} denote the underlying $(d+1)$-special-sound protocol. This is the public-coin interactive protocol $\Pi_{\mathsf{test}} = (\mathsf{P}_I(\mathsf{pi}, \pi, \mathsf{acc}), \mathsf{V}_I(\mathsf{pi}, \pi.x, \mathsf{acc}.x))$ where $\mathcal{P}_I$ sends correction terms $\mathsf{pf} = [E_j]_{j=1}^{d-1}$ as computed by $\mathsf{P}_{\mathsf{acc}}$ to V_I. The verifier sends a random challenge $\alpha \in \mathbb{F}$, and the prover P_I responds with acc' as computed by $\mathsf{P}_{\mathrm{acc}}$. V_I accepts if $D_{\mathsf{acc}}(\mathsf{acc}') = 0$ and $V_{\mathsf{acc}}(\pi, \pi.x, \mathsf{acc}.x, \mathsf{pf}, \mathsf{acc}'.x) = 0$ using the random challenge α, instead of a Fiat-Shamir challenge. Similarly, we let Π_{check} denote the underlying special-sound protocol for ACC'. Per Theorem 2 of Protostar [BC23], knowledge soundness of ACC follows from the special-soundness of Π_{test}. Similarly, knowledge soundness of ACC' follows from the special-soundness of Π_{check}.

Let $\mathsf{Ext}_{\mathsf{check}}$ be an extractor for the special-sound protocol Π_{check}. We will construct an extractor $\mathsf{Ext}_{\mathsf{test}}$ that computes a witness for $\mathcal{R}_{\mathsf{acc}}$ given a transcript tree for Π_{check}. The extractor $\mathsf{Ext}_{\mathsf{test}}$ builds a transcript tree for Π_{check} by replacing each encoded group element with its decoded field element by running the extractor $\mathsf{Ext}_{\mathsf{enc}}$ for the linear-only encoding scheme. The extractor $\mathsf{Ext}_{\mathsf{test}}$ then runs the extractor $\mathsf{Ext}_{\mathsf{check}}$ on the transformed transcript tree to obtain the witness. By Lemma 1 of Protostar [BC23], the Fiat-Shamir transformed protocol is knowledge sound if the interactive protocol Π_{check} is special-sound. Hence, the accumulation scheme ACC is knowledge sound.

4 SnarkStar: Efficient Groth16 Proof Aggregation

In this section, we present the SnarkStar protocol for the Groth16 proof verification relation. By applying our general result, we can construct an accumulation scheme for this relation, and thereby construct a PCD scheme for this relation. The resulting scheme is an efficient Groth16 proof aggregation protocol with constant-size proofs.

4.1 Background on Groth16 SNARKs

We recall the basic details of Groth16 SNARKs:

- The verification key $\mathsf{vk} := (P = [\alpha]_1, Q = [\beta]_2, [S_j = [\frac{\beta v_j(s) + \alpha w_j(s) + y_j(s)}{\epsilon}]_1]_{j=0}^t, H = [\epsilon]_2, D = [\delta]_2)$ is generated from a secret $s \in \mathbb{F}$ that is discarded at the end of the setup procedure. The public polynomials $v_j(x), w_j(x), y_j(x)$ define the QAP representation of the computation. To simplify notation, we define $s_j = \frac{\beta v_j(s) + \alpha w_j(s) + y_j(s)}{\epsilon} \in \mathbb{F}$, $s = \sum_{j=0}^t a_j \cdot s_j \in \mathbb{F}$, and $S = \sum_{j=0}^t a_j \cdot S_j \in \mathbb{G}_1$.

– The proof has the form $(A, B, C) \in \mathbb{G}_1 \times \mathbb{G}_2 \times \mathbb{G}_1$ for NP instance $\mathbf{x} = (a_0, \ldots, a_t) \in \mathbb{F}^{t+1}$.
– The verifier checks a pairing equation between the proof elements $\pi = (A, B, C)$ using the verification key vk: $e(A, B) = e(P, Q) \cdot e(S, H) \cdot e(C, D)$.

4.2 Generic proof verification

We first consider the case of proof aggregation for arbitrary circuits, i.e., circuits with different verification keys.

Verification relation Given public input $\mathbf{x} = (P, Q, [S_j]_{j=0}^t, H, D)$, the relation $\mathcal{R}_{\mathsf{GPV}}$ consists of a set of tuples $(\mathbf{x}, \mathbf{w})$ where $\mathbf{w} = (A, B, C) \in \mathbb{G}_1 \times \mathbb{G}_2 \times \mathbb{G}_1$ such that $e(A, B) = e(P, Q) \cdot e(S, H) \cdot e(C, D)$. The decoded public input is a tuple $\hat{\mathbf{x}} = (\alpha, \beta, [a_j \cdot s_j]_{j=0}^t, \epsilon, \delta)$, and the decoded message is $\hat{\mathbf{w}} = (\hat{A}, \hat{B}, \hat{C})$. We obtain the verifier's tests by defining the corresponding algebraic checks:

$$\mathsf{V}'_{\mathsf{GPV}}(\hat{\mathbf{x}}, \hat{\mathbf{w}}, \mu, \hat{\mathbf{e}}) = \mu \cdot (\hat{A} \cdot \hat{B} - \alpha \cdot \beta - \hat{C} \cdot \delta) - (\sum_{j=0}^{t} a_j \cdot s_j) \cdot \epsilon - \hat{\mathbf{e}}_1$$

We define encoded vectors $\mathbf{c}_1 = (\mu \cdot A, -\mu \cdot P, -\mu \cdot C, -a_0 \cdot S_0, \ldots, -a_t \cdot S_t, [1]_1) \in \mathbb{G}_1^{t+5}$ and $\mathbf{c}_2 = (B, Q, D, H, \ldots, H, [1]_2) \in \mathbb{G}_2^{t+5}$. In this case, we have the identity vector $\mathbf{c}_3 = ([1]_2, \ldots, [1]_2) \in \mathbb{G}_2^{t+5}$ and $\boldsymbol{\gamma} = (\mathbf{0}^{t+4}, \hat{\mathbf{e}_1}) \in \mathbb{F}^{t+5}$. The verifier's map $\mathsf{V}_{\mathsf{GPV}}$ is:

$$\mathsf{V}_{\mathsf{GPV}}(\mathbf{x}, \mathbf{w}, \mu, \mathbf{e}) = \mathsf{QuadTest}(\mathsf{pk}, (\mathbf{c}_1, \mathbf{c}_2, \mathbf{c}_3), \boldsymbol{\gamma})$$

We simplify the map $\mathsf{V}_{\mathsf{GPV}}$ to obtain the optimized version of the relaxed pairing checks:

$$e(\mu \cdot A, B) = e(\mu \cdot P, Q) \cdot e(S, H) \cdot e(\mu \cdot C, D) \cdot \mathbf{e}_1$$

4.3 Circuit-specific proof verification

We now consider the case of proof aggregation for a specific circuit. While the generic relation has a degree-3 verifier, the circuit-specific relation has a degree-2 verifier.

Verification relation Given configuration $\mathcal{C}_{\mathsf{SPV}} = (P, Q, H, D, [S_j]_{j=0}^t)$ and public input $\mathbf{x} = ([a_j]_{j=0}^t)$, the relation $\mathcal{R}_{\mathsf{SPV}}$ consists of a set of tuples $(\mathbf{x}, \mathbf{w})$ where $\mathbf{w} = (A, B, C) \in \mathbb{G}_1 \times \mathbb{G}_2 \times \mathbb{G}_1$ such that $e(A, B) = e(P, Q) \cdot e(S, H) \cdot e(C, D)$. The decoded public input is a tuple $\hat{\mathbf{x}} = ([a_j]_{j=0}^t)$, and the decoded message is $\hat{\mathbf{w}} = (\hat{A}, \hat{B}, \hat{C})$. We first define the corresponding algebraic checks:

$$\mathsf{V}'_{\mathsf{SPV}}(\hat{\mathbf{x}}, \hat{\mathbf{w}}, \mu, \hat{\mathbf{e}}) = (\hat{A} \cdot \hat{B}) - \mu \cdot (\hat{C} \cdot \delta - (\sum_{j=0}^{t} a_j s_j) \cdot \epsilon) - \mu^2 \cdot \alpha \cdot \beta - \hat{\mathbf{e}}_1$$

We define encoded vectors $\mathbf{c}_1 = ([A, -\mu \cdot C, [-\mu \cdot a_0 s_0]_1, \ldots, [-\mu \cdot a_t s_t]_1, -\mu^2 \cdot P, [1]_1) \in \mathbb{G}_1^{t+5}$ and $\mathbf{c}_2 = (B, D, H, \ldots, H, Q, [1]_2) \in \mathbb{G}_2^{t+5}$. In this case, we simply have the identity vector $\mathbf{c}_3 = ([1]_2, \ldots, [1]_2) \in \mathbb{G}_2^{t+5}$ and $\boldsymbol{\gamma} = (\mathbf{0}^{t+4}, \hat{\mathbf{e}}_1) \in \mathbb{F}^{t+5}$. The verifier's map $\mathsf{V}_{\mathsf{SPV}}$ is:

$$\mathsf{V}_{\mathsf{SPV}}(\mathbf{x}, \mathbf{w}, \mu, \mathbf{e}) = \mathsf{QuadTest}(\mathsf{pk}, (\mathbf{c}_1, \mathbf{c}_2, \mathbf{c}_3), \boldsymbol{\gamma})$$

We simplify the map $\mathsf{V}_{\mathsf{SPV}}$ to obtain the optimized version of the relaxed pairing checks:

$$e(A, B) = e(\mu^2 \cdot P, Q) \cdot e(\mu \cdot S, H) \cdot e(\mu \cdot C, D) \cdot \mathbf{e}_1$$

4.4 Instance aggregation

For a complete proof aggregation protocol, we require a subprotocol for committing to the instances and verification keys. If we directly constructed a PCD scheme using one of the previous subprotocols, we would not preserve the history of instances and structures that were aggregated. The verifier would only know that some set of proofs was correctly aggregated without the ability to check the result with respect to a specific set of instances and verification keys.

We address this issue by updating a vector commitment (VC) at each step i, where the value being inserted at position i is a commitment to the instance and the verification key. This updatable vector commitment may be instantiated with a Merkle tree using a collision-resistant hash function. Since the update can be expressed using Plonkish arithmetization, the Protostar protocol is applicable.

4.5 Main protocol

Below we present the main protocol for the Groth16 proof aggregation relation, which is a composition of the building block protocols that were presented above. We present the case of generic proof aggregation, since the case of circuit-specific aggregation immediately follows.

Aggregation relation Given public I/O $\mathbf{x}_{\mathsf{agg}} = (P, Q, [a_j S_j]_{j=0}^{t}, H, D, \mathsf{pi})$, the proof aggregation relation $\mathcal{R}_{\mathsf{AGG}}$ consists of a set of tuples $(\mathbf{x}_{\mathsf{agg}}, \mathbf{w}_{\mathsf{agg}})$ where $\mathbf{w}_{\mathsf{agg}} = (\mathbf{w}, \mathbf{w}')$, $\mathbf{w} = (A, B, C) \in \mathbb{G}_1 \times \mathbb{G}_2 \times \mathbb{G}_1$ such that $e(A, B) = e(P, Q) \cdot e(S, H) \cdot e(C, D)$, and $(\mathsf{pi} \in \mathbb{F}^{\ell_{\mathsf{in}}}, \mathbf{w}' \in \mathbb{F}^{n-\ell_{\mathsf{in}}})$ is an instance-witness pair in the instance aggregation relation. The decoded public input is a tuple $\hat{\mathbf{x}}_{\mathsf{agg}} = (\alpha, \beta, [a_j s_j]_{j=0}^{t}, \epsilon, \delta, i, \mathsf{cm}, \mathsf{cm}')$. The decoded message is $\hat{\mathbf{w}}_{\mathsf{agg}} = (\hat{\mathbf{w}}, \mathbf{w}')$ where $\hat{\mathbf{w}} = (\hat{A}, \hat{B}, \hat{C})$.

Special-soundness The subprotocols are 1-move protocols where special-soundness follows trivially. Since the main protocol is a composition of special-sound protocols, the main protocol is special-sound.

5 TensorStar: Efficient PCD for Tensor Computations

In this section, we describe the TensorStar accumulation scheme for a language of tensor computations. We support the basic operations of matrix-vector multiplication and vector addition. We additionally support complex nonlinearities through the usage of lookup arguments. Finally, we enable provers to commit to the model weights using KZG commitments to the weight vectors. For simplicity, we do not support non-uniform computations. It suffices to use an identity matrix, identity vector, or identity mapping for the operations that are not relevant at a given step.

5.1 Preliminaries

Let $\mathbb{H} \subset \mathbb{F}$ be a multiplicative subgroup of order n. Let $L_1(X), \ldots, L_n(X)$ be a Lagrange basis for $\mathbb{H}$. For $\omega \in \mathbb{H}$, $L_i(X)$ is the unique element of $\mathbb{F}_{<n}[X]$ with $L_i(\omega^i) = 1$ and $L_i(\omega^j) = 0$ for $i \neq j \in [n]$. Denote $Z(X) := X^{n^2} - 1$, and $Z_H(X) := X^n - 1$. For a polynomial $f(X) \in \mathbb{F}[X]$, define $f|_{\mathbb{H}}$ as the vector $\mathbf{v} \in \mathbb{F}^n$ with $v_i = f(\omega^i)$. For polynomials $f(X), g(X) \in \mathbb{F}[X]$ where $d := \deg(g(X)) > 0$, we denote by $f(x) \mod g(X)$ the unique polynomial $R(X) \in \mathbb{F}_{<d}[X]$ such that for some $Q(X) \in \mathbb{F}[X]$,

$$f(X) = Q(X) \cdot g(X) + R(X).$$

5.2 Matrix-vector multiplication

Standard protocol We recall the definition of the cqlin protocol [EG23a] for verifiable matrix-vector multiplication. This is a pairing-based protocol where the matrix M is fixed in advance. In particular, this yields a 3-move linear special-sound protocol parametrized by the matrix order n.

Setup. Given parameter $n \in \mathbb{N}$ and matrix $M \in \mathbb{F}^n \times \mathbb{F}^n$, the generation algorithm $\mathcal{G}$ outputs a structured reference string srs consisting of group elements in $\mathbb{G}_1$ and $\mathbb{G}_2$.

1. Sample a random element $x \in \mathbb{F}$.
2. Compute and output the elements $\{[x^i]_1\}_{i \in \{0,\ldots,n^2-1\}}$ and $\{[x^i]_2\}_{i \in \{0,\ldots,n^2\}}$.
3. Compute and output the elements $\left\{[x^{n^2-n} \cdot L_i(x)]_1, [L_i(x^n)]_1\right\}_{i \in [n]}$, $\{[L_i(x^n)L_j(x)]_1\}_{i,j \in [n]}$.
4. Compute and output $\mathsf{z} := [Z(x)]_2$.
5. Given the polynomials $\mathsf{R}_i(X) := \sum_{j \in [n]} M_{i,j} L_j(X)$ for $i \in [n]$ and $M(X) := \sum_{i \in [n]} L_i(X^n)\mathsf{R}_i(X)$, compute and output $\mathsf{m} := [M(x)]_2$.
6. For $i \in [n]$, compute and output:
   (a) $\mathsf{r}_i = [R_i(x)]_1$ where $R_i(X) := L_i(X^n) \cdot \mathsf{R}_i(X)$.
   (b) $\mathsf{q}_i = [Q_i(x)]_1$ such that $L_i(X^n) \cdot M(X) = Q_i(X)Z(X) + R_i(X)$.
   (c) $\mathsf{s}_i = [S_i(x)]_1$ such that $(L_i(X^n) - 1/n) \cdot \mathsf{R}_i(X) = X^n S_i(X)$.

Lincheck. A lincheck protocol is an interactive public-coin protocol between a prover $\mathcal{P}$ and a verifier $\mathcal{V}$ where $\mathcal{P}$ has private input $f(X), g(X) \in \mathbb{F}_{<n}[X]$, and both parties have access to $\mathsf{f} = [f(X)]_1$, $\mathsf{g} = [g(X)]_1$, and the structured reference string srs. $\mathcal{P}$ aims to convince $\mathcal{V}$ that $f|_{\mathbb{H}} \cdot M = g|_{\mathbb{H}}$.

The first round of the lincheck protocol involves checking that the matrix-vector multiplication is correct and checking the degree of the polynomial $g(X)$.

1. Let $A(X) := f(X^n) = \sum_{i\in[n]} a_i \cdot L_i(X^n)$. Prover $\mathcal{P}$ computes and sends $\mathsf{a} := [A(x)]_1$.
2. $\mathcal{P}$ computes $\mathsf{r} := [R(x)]_1$ and $\mathsf{q} := [Q(x)]_1$ where $R, Q \in \mathbb{F}_{<n^2}[X]$ are such that

$$A(X) \cdot M(X) = Q(X) \cdot Z(X) + R(X)$$

.
3. $\mathcal{P}$ sends r, q to the verifier $\mathcal{V}$.
4. Verifier $\mathcal{V}$ checks the correctness of r via the pairing check

$$e(\mathsf{a}, \mathsf{m}) = e(\mathsf{q}, \mathsf{z}) \cdot e(\mathsf{r}, [1]_2).$$

5. $\mathcal{P}$ computes $\mathsf{s} := [S(x)]_1$ where $S(X) \in \mathbb{F}_{<n^2}[X]$ is such that

$$R(X) - (1/n) \cdot g(X) = S(X) \cdot X^n.$$

6. $\mathcal{V}$ checks that $R(X) = (1/n) \cdot g(X) \mod X^n$ via the pairing check

$$e(\mathsf{r} - (1/n) \cdot \mathsf{g}, [1]_2) = e(\mathsf{s}, [x^n]_2)$$

7. $\mathcal{P}$ computes $\mathsf{p} := [P(x)]_1$ where $P(X) := g(X) \cdot X^{n^2-n}$.
8. $\mathcal{V}$ checks that $\deg(g) < n$ via the pairing check

$$e(\mathsf{g}, [x^{n^2-n}]_2) = e(\mathsf{p}, [1]_2)$$

The second round of the lincheck protocol involves checking that $A(X) = f(X^n)$ on a random challenge sent by the verifier.

1. Verifier $\mathcal{V}$ sends random challenge $\gamma \in \mathbb{F}$.
2. Let $z := f(\gamma^n) = A(\gamma)$. Prover $\mathcal{P}$ computes $\pi = [h(x)]_1$ and $\pi_1 = [h_1(x)]_1$ where

$$h(X) := \frac{f(X) - z}{X - \gamma^n},$$

$$h_1(X) := \frac{A(X) - z}{X^n - \gamma^n}.$$

3. $\mathcal{V}$ checks that $A(\gamma) = f(\gamma^n)$ via the pairing checks:

$$e(\mathsf{f} - [z]_1 + \gamma^n \cdot \pi, [1]_2) = e(\pi, [x]_2),$$

$$e(\mathsf{a} - [z]_1 + \gamma^n \cdot \pi_1, [1]_2) = e(\pi_1, [x^n]_2).$$

4. $\mathcal{V}$ outputs reject if any of the pairing checks failed, and accept otherwise.

Optimized protocol A key issue with the standard lincheck protocol is the verifier degree, which is $O(n)$ due to the checks in the second round of the protocol. We address this issue by providing an optimized protocol where the verifier degree is independent of the matrix order n at the cost of an additional $O(\log n)$ degree-2 checks. Specifically, the prover compute powers of the challenge, and the verifier can check that these powers are correct. Since these checks involve field elements only, this significantly reduces the verifier costs. In the optimized variant of Round 2 below, we highlight the modifications in blue. We let $k := \log n$.

Round 2 (low-degree variant). The second round of the lincheck protocol involves checking that $A(X) = f(X^n)$.

1. Verifier $\mathcal{V}$ sends random challenge $\gamma \in \mathbb{F}$.
2. Let $z := f(\gamma^n) = A(\gamma)$. Prover $\mathcal{P}$ computes $\pi = [h(x)]_1$ and $\pi_1 = [h_1(x)]_1$ where

$$h(X) := \frac{f(X) - z}{X - \gamma^n},$$

$$h_1(X) := \frac{A(X) - z}{X^n - \gamma^n}.$$

3. Prover $\mathcal{P}$ computes and sends $\beta_i = \gamma^{2^i}$ for $i \in [k]$.
4. Verifier $\mathcal{V}$ checks $\gamma = \beta_0$.
5. Verifier $\mathcal{V}$ checks $\beta_i \cdot \beta_i = \beta_{i+1}$ for $i \in [k]$.
6. $\mathcal{V}$ checks that $A(\gamma) = f(\gamma^n)$ via the pairing checks:

$$e(\mathsf{f} - [z]_1 + \beta_k \cdot \pi, [1]_2) = e(\pi, [x]_2),$$

$$e(\mathsf{a} - [z]_1 + \beta_k \cdot \pi_1, [1]_2) = e(\pi_1, [x^n]_2).$$

7. $\mathcal{V}$ outputs reject if any of the pairing checks failed, and accept otherwise.

Accumulation First, we summarize the key aspects of the optimized protocol. The following constant terms are used in the protocol: $[1]_2 \in \mathbb{G}_2$, $[x^{n^2-n}]_2 \in \mathbb{G}_2$, $[x^n]_2 \in \mathbb{G}_2$, $[x]_2 \in \mathbb{G}_2$ and $[Z(x)]_2 \in \mathbb{G}_2$. The public input $\mathbf{x}$ consists of the group elements $\mathsf{m} = [M(x)]_2$, $\mathsf{f} = [f(x)]_1$, and $\mathsf{g} = [g(x)]_1$. These are commitments to the matrix and the I/O vectors. The prover sends two messages: $\mathbf{m}_1 := (\mathsf{a}, \mathsf{r}, \mathsf{q}, \mathsf{s}, \mathsf{p}) \in \mathbb{G}_1^5$, $\mathbf{m}_2 := ([z_1]_1, \pi, \pi_1, [\beta_i]_{i=0}^k) \in (\mathbb{F} \cup \mathbb{G}_1)^{3+k}$. The verifier issues a random challenge $\gamma \in \mathbb{F}$.

Given the lincheck protocol configuration $\mathcal{C}_{\mathsf{MUL}} = \{n, [1]_1, [1]_2, [x^{n^2-n}]_2, [x^n]_2, [x]_2, [Z(x)]_2\}$, the verification relation $\mathcal{R}_{\mathsf{MUL}}$ consists of a set of $(\mathbf{x}, \mathbf{w})$ where $\mathbf{x} = (\mathsf{m}, \mathsf{f}, \mathsf{g}) \in \mathbb{G}_2 \times \mathbb{G}_1 \times \mathbb{G}_1$ and $\mathbf{w} = (\mathbf{m}_1, \mathbf{m}_2, \gamma)$ such that

$$e(\mathsf{a}, \mathsf{m}) = e(\mathsf{q}, \mathsf{z}) \cdot e(\mathsf{r}, [1]_2)$$

$$e(\mathsf{r} - (1/n) \cdot \mathsf{g}, [1]_2) = e(\mathsf{s}, [x^n]_2)$$

$$e(\mathsf{g}, [x^{n^2-n}]_2) = e(\mathsf{p}, [1]_2)$$

$$e(\mathsf{f} - [z]_1 + \beta_k \cdot \pi, [1]_2) = e(\pi, [x]_2)$$

$$e(\mathsf{a} - [z]_1 + \beta_k \cdot \pi_1, [1]_2) = e(\pi_1, [x^n]_2)$$

The decoded public input is a tuple $\hat{\mathbf{x}} = (M(x), f(x), g(x))$. The decoded protocol messages and challenges are:

$$\hat{\mathbf{w}} = \begin{pmatrix} A(x), R(x), Q(x), S(x), P(x), \\ h(x), h_1(x), \gamma, \beta_1, \beta_2, \ldots, \beta_k \end{pmatrix}$$

The corresponding algebraic checks are:

$$A(x)M(x) - \mu \cdot (Q(x)Z(x) + R(x)) = \hat{\mathbf{e}}_1$$

$$\mu \cdot \left(R(x) - \frac{1}{n} \cdot g(x) - S(x) \cdot x^n \right) = \hat{\mathbf{e}}_2$$

$$\mu \cdot \Big(g(x) \cdot x^{n^2-n} - P(X) \Big) = \hat{\mathbf{e}}_3$$

$$\mu \cdot (f(x) - z - h(x) \cdot x) + \beta_k \cdot h(x) = \hat{\mathbf{e}}_4$$

$$\mu \cdot (A(x) - z - h_1(x) \cdot x^2) + \beta_k \cdot h_1(x) = \hat{\mathbf{e}}_5$$

$$\gamma - \beta_0 = \hat{\mathbf{e}}_6$$

$$\forall i \in [k], \beta_i \cdot \beta_i - \mu \cdot \beta_{i+1} = \hat{\mathbf{e}}_{7+k}$$

The relaxed pairing and algebraic checks are:

$$e(\mathsf{a}, \mathsf{m}) = e(\mu \cdot \mathsf{q}, \mathsf{z}) \cdot e(\mu \cdot \mathsf{r}, [1]_2) \cdot \mathbf{e}_1$$

$$e\left(\mu \cdot \left(\mathsf{r} - \frac{1}{n} \cdot \mathsf{f} \right), [1]_2 \right) = e(\mu \cdot \mathsf{s}, x^n) \cdot \mathbf{e}_2$$

$$e(\mu \cdot \mathsf{g}, x^{n^2-n}) = e(\mu \cdot \mathsf{p}, [1]_2) \cdot \mathbf{e}_3$$

$$e(\mu \cdot (\mathsf{f} - [z]_1) + \beta_k \cdot \pi, [1]_2) = e(\mu \cdot \pi, [x]_2) \cdot \mathbf{e}_4$$

$$e(\mu \cdot (\mathsf{a} - [z]_1) + \beta_k \cdot \pi_1, [1]_2) = e(\mu \cdot \pi_1, [x^2]_2) \cdot \mathbf{e}_5$$

$$\gamma - \beta_0 = \mathbf{e}_6$$

$$\forall i \in [k], \beta_i \cdot \beta_i - \mu \cdot \beta_{i+1} = \mathbf{e}_{7+k}$$

5.3 Vector lookup

Background We recall the definition of the cq protocol [EFG22]. This is a pairing-based lookup argument that supports vector lookups via random linear combinations of homomorphic commitments. Given commitments $\mathsf{cm}_1, \mathsf{cm}_2$ to tables T_1, T_2 with elements $\{a_i\}, \{b_i\}$, and a random challenge α, we check membership in the set $\{a_i + \alpha \cdot b_i\}$ and use $\mathsf{cm} = \mathsf{cm}_1 + \alpha \cdot \mathsf{cm}_2$ as the table commitment.

Given a table $T = \{t_i\}_{i\in[N]}$ of distinct values and lookups $F = \{f_j\}_{j\in[m]}$, we want to show that the lookups are all contained in the table, i.e., $F \subseteq T$. Let V be a subgroup of order N, and H be a subgroup of order m. Let Z_V and Z_H denote the vanishing polynomials on these subgroups, i.e., the low-degree polynomials such that $Z_V(x) = \prod_{v\in V}(x - v)$ and $Z_H(x) = \prod_{h\in H}(x - h)$.

Setup. Given parameter $n \in \mathbb{N}$ and table $T = \{t_i\}_{i\in[N]}$, the generation algorithm $\mathcal{G}$ outputs a structured reference string srs consisting of group elements in $\mathbb{G}_1$ and $\mathbb{G}_2$.

1. Sample a random element $x \in \mathbb{F}$.
2. Compute and output $\{[x^i]_1\}_{i\in\{0,\ldots,N-1\}}$.
3. Compute and output $\{[x^i]_2\}_{i\in\{0,\ldots,N\}}$.
4. Compute and output $[Z_V(x)]_2$.
5. Compute $T(X) = \sum_{i\in[N]} t_i L_i(X)$.
6. Compute and output $[T(x)]_2$.
7. For $i \in [N]$, compute and output:
   (a) $\mathsf{q}_i = [Q_i(x)]_1$ such that

$$L_i(X) \cdot T(X) = t_i \cdot L_i(X) + Z_H(X) \cdot Q_i(X).$$

   (b) $[L_i(x)]_1$.
   (c) $\left[\frac{L_i(x)-L_i(0)}{x}\right]_1$.

Lookup. The lookup protocol is an interactive public-coin protocol between a prover $\mathcal{P}$ and a verifier $\mathcal{V}$ where $\mathcal{P}$ has a private input $f \in \mathbb{F}_{<m}[x]$ corresponding to the lookup values and both parties have access to the table T, the table commitment $\mathsf{cm} = [f(x)]_1$, and the structured reference string srs. The prover aims to convince the verifier that $f|_{\mathbb{H}} \subset T$. We say a lookup protocol is homomorphic if the structured reference strings are homomorphic.

The first round of the lookup protocol involves committing to the multiplicities vector.

1. $\mathcal{P}$ computes the polynomial $M(x) \in \mathbb{F}_{<N}[X]$, which is defined by setting the value M_i to the number of times that t_i appears in $f|_{\mathbb{H}}$.
2. $\mathcal{P}$ sends $\mathsf{m} = [M(x)]_1$.

The second round of the lookup protocol involves interpolating the rational identity at a random β, checking the correctness of A's values, and performing a degree check for B using pairings.

1. $\mathcal{V}$ send random challenge $\beta \in \mathbb{F}$.
2. $\mathcal{P}$ computes $A \in \mathbb{F}_{<N}[X]$ such that for $i \in [N]$, $A_i = m_i/(t_i + \beta)$.
3. $\mathcal{P}$ computes and sends $\mathsf{a} := [A(x)]_1$.
4. $\mathcal{P}$ computes and sends $\mathsf{q_a} := [Q_A(x)]_1$ where $Q_A \in \mathbb{F}_{<N}[X]$ is such that

$$A(X)(T(X) + \beta) - M(x) = Q_A(X) \cdot Z_V(X)$$

5. $\mathcal{P}$ computes $B(X) \in \mathbb{F}_{<m}[X]$ such that for $i \in [m]$, $B_i = 1/(f_i + \beta)$.
6. $\mathcal{P}$ computes $B_0(X) := \frac{B(X)-B(0)}{X} \in \mathbb{F}_{<m-1}[X]$.
7. $\mathcal{P}$ computes and sends $\mathsf{b}_0 := [B_0(x)]_1$.
8. $\mathcal{P}$ computes $Q_B(X)$ such that

$$B(X)(f(x) + \beta) - 1 = Q_B(X) \cdot Z_H(X).$$

9. $\mathcal{P}$ computes and sends $\mathsf{q_b} := [Q_B(x)]_1$.
10. $\mathcal{P}$ computes and sends $\mathsf{p} = [P(x)]_1$ where

$$P(X) := B_0(X) \cdot X^{N-1-(m-2)}.$$

11. $\mathcal{V}$ checks that A encodes the correct values:

$$e(\mathsf{a}, [T(x)]_2) = e(\mathsf{q_a}, [Z_V(x)]_2) \cdot e(\mathsf{m} - \beta \cdot \mathsf{a}, [1]_2)$$

12. $\mathcal{V}$ checks that B_0 has the appropriate degree:

$$e(\mathsf{b}_0, [x^{N-1-(m-2)}]_2) = e(\mathsf{p}, [1]_2).$$

The final round of the lookup protocol involve checking the correctness of B at a random γ. To simplify the accumulation scheme, we do not leverage the batching technique in the final round.

1. $\mathcal{V}$ sends random challenge $\gamma \in \mathbb{F}$.
2. $\mathcal{P}$ sends $b_{0,\gamma} := B_0(\gamma), f_\gamma := f(\gamma)$.
3. $\mathcal{P}$ computes and sends the value $a_0 := A(0)$.
4. $\mathcal{P}$ computes and sends the value $b_0 := (N \cdot a_0)/m$.
5. $\mathcal{V}$ checks that b_0 is computed correctly:

$$b_0 \cdot m = N \cdot a_0$$

6. To check the correctness of the value $b_{0,\gamma}$:
   (a) $\mathcal{P}$ computes and sends the value $\pi = [h(x)]_1$ where

$$h(X) := \frac{B_0(X) - b_{0,\gamma}}{X - \gamma}$$

   (b) $\mathcal{V}$ checks that

$$e([\mathsf{b}_0 - [b_{0,\gamma}]_1 + \gamma \cdot \pi, [1]_2) = e(\pi, [x]_2)$$

7. To check the correctness of the value f_γ:
   (a) $\mathcal{P}$ computes and sends $\pi_1 = [h_1(x)]_1$ where

$$h_1(X) := \frac{f(X) - f_\gamma}{X - \gamma}$$

   (b) $\mathcal{V}$ checks that

$$e(\mathsf{cm} - [f_\gamma]_1 + \gamma \cdot \pi_1, [1]_2) = e(\pi_1, [x]_2)$$

8. To check the correctness of the value $Q_{b,\gamma}$:
   (a) $\mathcal{P}$ and $\mathcal{V}$ separately compute $Z_H(\gamma) = \gamma^m - 1$, $b_\gamma := b_{0,\gamma} \cdot \gamma + b_0$.
   (b) $\mathcal{P}$ computes and sends the value:

$$Q_{b,\gamma} := \frac{b_\gamma \cdot (f_\gamma + \beta) - 1}{Z_H(\gamma)}$$

   (c) $\mathcal{V}$ checks that $Q_{b,\gamma}$ is computed correctly:

$$Q_{b,\gamma} \cdot Z_H(\gamma) = b_\gamma \cdot (f_\gamma + \beta) - 1$$

   (d) $\mathcal{P}$ computes and sends $\pi_2 = [h_2(x)]_1$ where

$$h_2(X) := \frac{Q_B(X) - Q_{b,\gamma}}{X - \gamma}$$

   (e) $\mathcal{V}$ checks that

$$e(\mathsf{q_b} - [Q_{b,\gamma}]_1 + \gamma \cdot \pi_2, [1]_2) = e(\pi_2, [x]_2)$$

9. To check the correctness of the value a_0:
   (a) $\mathcal{P}$ computes and sends $\mathsf{a}_0 := [A_0(x)]_1$ for

$$A_0(x) = \frac{A(X) - a_0}{X}$$

   (b) $\mathcal{V}$ checks that

$$e(\mathsf{a} - [a_0]_1, [1]_2) = e(\mathsf{a}_0, [x]_2)$$

.

Optimized protocol We previously assumed that a single lookup is performed at each step. To extend this to multiple lookups ($m > 1$) while maintaining a low-degree verifier, additional optimizations are necessary. We describe these optimizations in detail for the final round of the lookup protocol. Briefly, the prover computes powers of the challenges, and the verifier checks these powers using additional degree-2 equations, similar to the optimizations for the lincheck protocol in Sec. 5.2.

Round 3 (low-degree variant). The final round of the lookup protocol involve checking the correctness of B at a random γ. To simplify the accumulation scheme, we do not leverage the batching technique in the final round.

1. $\mathcal{V}$ sends random challenge $\gamma \in \mathbb{F}$.
2. $\mathcal{P}$ sends $b_{0,\gamma} := B_0(\gamma), f_\gamma := f(\gamma)$.
3. $\mathcal{P}$ computes and sends the value $a_0 := A(0)$.
4. $\mathcal{P}$ computes and sends the value $b_0 := (N \cdot a_0)/m$.
5. $\mathcal{V}$ checks that b_0 is computed correctly:
$$b_0 \cdot m = N \cdot a_0$$
6. To perform a check for the correctness of the value $b_{0,\gamma}$:
   (a) $\mathcal{P}$ computes and sends the value $\pi = [h(x)]_1$ where
   $$h(X) := \frac{B_0(X) - b_{0,\gamma}}{X - \gamma}$$
   (b) $\mathcal{V}$ checks that
   $$e([\mathsf{b}_0 - [b_{0,\gamma}]_1 + \gamma \cdot \pi, [1]_2) = e(\pi, [x]_2)$$
7. To perform a check for the correctness of the value f_γ:
   (a) $\mathcal{P}$ computes and sends the value $\pi_1 = [h_1(x)]_1$ where
   $$h_1(X) := \frac{f(X) - f_\gamma}{X - \gamma}$$
   (b) $\mathcal{V}$ checks that
   $$e(\mathsf{cm} - [f_\gamma]_1 + \gamma \cdot \pi_1, [1]_2) = e(\pi_1, [x]_2)$$
8. Prover $\mathcal{P}$ computes and sends $\beta_i = \gamma^{2^i}$ for $i \in [k]$.
9. Verifier $\mathcal{V}$ checks $\gamma = \beta_0$.
10. Verifier $\mathcal{V}$ checks $\beta_i \cdot \beta_i = \beta_{i+1}$ for $i \in [k]$.
11. To perform a check for the correctness of the value $Q_{b,\gamma}$:
    (a) $\mathcal{P}$ and $\mathcal{V}$ separately compute $Z_H(\gamma) = \beta_k - 1$, $b_\gamma := b_{0,\gamma} \cdot \gamma + b_0$.
    (b) $\mathcal{P}$ computes and sends the value:
    $$Q_{b,\gamma} := \frac{b_\gamma \cdot (f_\gamma + \beta) - 1}{Z_H(\gamma)}$$
    (c) $\mathcal{V}$ checks that $Q_{b,\gamma}$ is computed correctly:
    $$Q_{b,\gamma} \cdot Z_H(\gamma) = b_\gamma \cdot (f_\gamma + \beta) - 1$$
    (d) $\mathcal{P}$ computes and sends the value $\pi_2 = [h_2(x)]_1$ where
    $$h_2(X) := \frac{Q_B(X) - Q_{b,\gamma}}{X - \gamma}$$
    (e) $\mathcal{V}$ checks that
    $$e(\mathsf{q_b} - [Q_{b,\gamma}]_1 + \gamma \cdot \pi_2, [1]_2) = e(\pi_2, [x]_2)$$
12. To perform a check for the correctness of the value a_0:
    (a) $\mathcal{P}$ computes and sends $\mathsf{a}_0 := [A_0(x)]_1$ for
    $$A_0(x) = \frac{A(X) - a_0}{X}$$
    (b) $\mathcal{V}$ checks that
    $$e(\mathsf{a} - [a_0]_1, [1]_2) = e(\mathsf{a}_0, [x]_2)$$

.

Accumulation For simplicity, we assume that $m = 1$, i.e., a single vector lookup is performed at each step. This simplification reduces the degree of the verifier. If multiple lookups are needed at each step, the prover may compute powers of the challenges, and the verifier can check that these powers are correct with additional degree-2 equations, similar to the lincheck optimizations in Sec. 5.2.

The following constant terms are used in the protocol: $[1]_2 \in \mathbb{G}_2$, $[x]_2 \in \mathbb{G}_2$, $[x^{N-2}]_2 \in \mathbb{G}_2$, and $[Z_V(x)]_2 \in \mathbb{G}_2$. The public input $\mathbf{x}$ consists of the group element $\mathsf{cm} = [T(x)]_2$, which is a commitment to the table elements. The prover sends three messages: $\mathbf{m}_1 := (\mathsf{m}) \in \mathbb{G}_1^1$, $\mathbf{m}_2 := (\mathsf{a}, \mathsf{q_a}, \mathsf{b}_0, \mathsf{q_b}, \mathsf{p}) \in \mathbb{G}_1^5$, $\mathbf{m}_3 = (a_0, b_0, b_{0,\gamma}, \pi, f_\gamma, \pi_1, Q_{b,\gamma}, \pi_2, \mathsf{a}_0) \in (\mathbb{F} \cup \mathbb{G}_1)^9$. The verifier issues random challenges $\beta, \gamma \in \mathbb{F}$.

Given the lookup protocol configuration $\mathcal{C}_{\mathsf{LKP}} = \{N, [1]_1, [1]_2, [x]_2, [x^{N-2}]_2, [Z_V(x)]_2\}$, the relation $\mathcal{R}_{\mathsf{LKP}}$ consists of a set of tuples $(\mathbf{x}, \mathbf{w})$ where $\mathbf{x} = \mathsf{cm} \in \mathbb{G}_2$ and $\mathbf{w} = (\mathbf{m}_1, \mathbf{m}_2, \mathbf{m}_3, \beta, \gamma)$ such that

$$
\begin{gathered}
e(\mathsf{a}, \mathsf{cm}) = e(\mathsf{q_a}, [Z_V(x)]_2) \cdot e(\mathsf{m} - \beta \cdot \mathsf{a}, [1]_2) \\
e(\mathsf{b}_0, [x^{N-2}]_2) = e(\mathsf{p}, [1]_2) \\
b_0 = N \cdot a_0 \\
e(\mathsf{b}_0 - [b_{0,\gamma}]_1 + \gamma \cdot \pi, [1]_2) = e(\pi, [x]_2) \\
e(\mathsf{cm} - [f_\gamma]_1 + \gamma \cdot \pi_1, [1]_2) = e(\pi_1, [x]_2) \\
Q_{b,\gamma} \cdot (\gamma - 1) = (b_{0,\gamma} \cdot \gamma + b_0) \cdot (f_\gamma + \beta) - 1 \\
e(\mathsf{q_b} - [Q_{b,\gamma}]_1 + \gamma \cdot \pi_2, [1]_2) = e(\pi_2, [x]_2) \\
e(\mathsf{a} - [a_0]_1, [1]_2) = e(\mathsf{a}_0, [x]_2)
\end{gathered}
$$

The decoded public input is a tuple $\hat{\mathbf{x}} = T(x)$. The decoded protocol messages and challenges are given by:

$$
\hat{\mathbf{w}} = \begin{pmatrix} M(x), A(x), Q_A(x), B_0(x), Q_B(x), \\ P(x), a_0, b_0, b_{0,\gamma}, h(x), \\ f_\gamma, h_1(x), Q_{b,\gamma}, h_2(x), \\ A_0(x), \beta, \gamma \end{pmatrix}
$$

The corresponding algebraic checks are:

$$
\begin{gathered}
A(x)T(x) - \mu \cdot (Q_A(x)Z_V(x) - M(x)) - \beta \cdot A(x) = \hat{\mathbf{e}}_1 \\
B_0(x) \cdot x^{N-2} - P(x) = \hat{\mathbf{e}}_2 \\
b_0 - N \cdot a_0 = \hat{\mathbf{e}}_3 \\
B_0(x) - b_{0,\gamma} - h(x) \cdot x + \gamma \cdot h(x) = \hat{\mathbf{e}}_4 \\
T(x) - f_\gamma - h_1(x) \cdot x + \gamma \cdot h_1(x) = \hat{\mathbf{e}}_5 \\
-\mu^3 + \mu^2 \cdot Q_{b,\gamma} + \mu \cdot (b_0 \cdot (f_\gamma + \beta) - Q_{b,\gamma} \cdot \gamma) + b_{0,\gamma} \cdot \gamma \cdot (f_\gamma + \beta) = \hat{\mathbf{e}}_6 \\
\mu \cdot (Q_B(x) - Q_{b,\gamma} - h_2(x) \cdot x) + \gamma \cdot h_2(x) = \hat{\mathbf{e}}_7 \\
A(x) - a_0 - A_0(x) \cdot x = \hat{\mathbf{e}}_8
\end{gathered}
$$

The relaxed pairing checks and algebraic checks are:

$$
\begin{gathered}
e(\mathsf{a}, \mathsf{cm}) = e(\mu \cdot \mathsf{q_a}, [Z_V(x)]_2) \cdot e(\mu \cdot \mathsf{m} - \beta \cdot \mathsf{a}, [1]_2) \cdot \mathbf{e}_1 \\
e(\mathsf{b}_0, [x^{N-2}]_2) = e(\mathsf{p}, [1]_2) \cdot \mathbf{e}_2 \\
b_0 - N \cdot a_0 = \mathbf{e}_3 \\
e(\mathsf{b}_0 - [b_{0,\gamma}]_1 + \gamma \cdot \pi, [1]_2) \cdot \mathbf{e}_4 = e(\pi, [x]_2) \\
e(\mathsf{cm} - [f_\gamma]_1 + \gamma \cdot \pi_1, [1]_2) \cdot \mathbf{e}_5 = e(\pi_1, [x]_2) \\
-\mu^3 + \mu^2 \cdot Q_{b,\gamma} + \mu \cdot (b_0 \cdot (f_\gamma + \beta) - Q_{b,\gamma} \cdot \gamma) + b_{0,\gamma} \cdot \gamma \cdot (f_\gamma + \beta) = \mathbf{e}_6 \\
e(\mu \cdot (\mathsf{q_b} - [Q_{b,\gamma}]_1 + \gamma \cdot \pi_2), [1]_2) \cdot \mathbf{e}_7 = e(\mu \cdot \pi_2, [x]_2) \\
e(\mathsf{a} - [a_0]_1, [1]_2) \cdot \mathbf{e}_8 = e(\mathsf{a}_0, [x]_2)
\end{gathered}
$$

5.4 KZG opening proofs

The setup phase generates a KZG commitment to the model's weight and bias vectors and the step at which they are used during proving. This is a vector commitment to values of the form (i, C) where i is the step index and C is a commitment to a weight or bias vector that is generated in the setup.

Background We recall the details of the KZG polynomial commitment scheme [KZG10]. It is easy to see that KZG can also serve as a vector commitment scheme. In this case, the evaluation points correspond to the vector indices and the evaluation values correspond to the vector elements.

Suppose we have a set of points $U = \{u_0, \ldots, u_t\}$ and want to prove the opening $f(u_i) = v_i$ for $i \in [t]$ for a univariate polynomial $f \in \mathbb{F}^{(<d)}[x]$ of degree at most d. Let $\mathbf{f}$ be a vector consisting of the coefficients of the polynomial $f(x)$. The receiver key consists of $([1]_1, [1]_2, [\tau]_2)$ where $[1]_1$ generates $\mathbb{G}_1$ and $[1]_2$ generates $\mathbb{G}_2$. The committer key consists of a vector $\mathbf{G} := ([1]_1, [\tau]_1, \ldots, [\tau^d]_1)$. These values are generated in a setup procedure using a secret value $\tau \in \mathbb{F}$. A commitment to a polynomial $f(x)$ is formulated as $\mathsf{cm}_f := \langle \mathbf{f}, \mathbf{G} \rangle$, i.e., the inner product of the coefficients and the committer key.

An opening proof π for f at the point u_i consists of a polynomial commitment to the quotient obtained when $f(x) - v_i$ is divided by $x - u_i$. This relies on the fact that the quotient is a polynomial if and only if $f(u_i) = v_i$. When considering multiple points $\{u_0, \ldots, u_t\}$, the quotient of $f(x)$ by a polynomial Z (whose roots are $\{u_i\}$) is used. The remainder, a polynomial $R(x)$, satisfies $R(u_i) = f(u_i)$. Specifically, $f(x) = Q(x)Z(x) + R(x)$. Commitments to Z and R are denoted by $\mathsf{cm}_Z \in \mathbb{G}_2$ and $\mathsf{cm}_R \in \mathbb{G}_1$, respectively. Verification of the proof π works as follows.

For a single point, the verifier checks if:

$$e(\pi, [\tau]_2 - [u]_2) = e(\mathsf{cm}_f - [v]_1, [1]_2)$$

For multiple points, the verifier checks if:

$$e(\pi, \mathsf{cm}_Z) = e(\mathsf{cm}_f - \mathsf{cm}_R, [1]_2)$$

Accumulation for single-point opening proofs Given configuration $\mathcal{C}_{\mathsf{SKZG}} = \{\mathsf{cm}_f, [\tau]_2\}$, the relation $\mathcal{R}_{\mathsf{SZKG}}$ consists of a set of tuples $(\mathbf{x}, \mathbf{w})$ where $\mathbf{x} = ([u]_2, [v]_1) \in \mathbb{G}_2 \times \mathbb{G}_1$ and $\mathbf{w} = \pi \in \mathbb{G}_1$ such that $e(\pi, [\tau]_2 - [u]_2) = e(\mathsf{cm}_f - [v]_1, [1]_2)$. We let $\hat{\pi}, \hat{\mathsf{cm}}_f$ be the decoded values such that $\pi = [\hat{\pi}]_1$ and $\mathsf{cm}_f = [\hat{\mathsf{cm}}_f]_1$. The decoded public input is a tuple $\hat{\mathbf{x}} = (u, v)$, and the decoded protocol message is $\hat{\mathbf{w}} = \hat{\pi}$. We first define the corresponding algebraic checks:

$$\mathsf{V}'_{\mathsf{SKZG}}(\hat{\mathbf{x}}, \hat{\mathbf{w}}, \mu, \hat{\mathbf{e}}) = -\mu^2 \cdot \hat{\mathsf{cm}}_f + \mu \cdot (\hat{\pi} \cdot \tau + v) - \hat{\pi} \cdot u - \hat{\mathbf{e}}_1$$

We define the following encoded vectors: $\mathbf{c}_1 = ([-\mu^2 \cdot \hat{\mathsf{cm}}_f]_1, [\mu \cdot \hat{\pi}]_1, [\mu \cdot v]_1, [-\hat{\pi}]_1, [1]_1) \in \mathbb{G}_1^5$ and $\mathbf{c}_2 = ([1]_2, [\tau]_2, [1]_2, [u]_2, [1]_2) \in \mathbb{G}_2^5$. We additionally have $\mathbf{c}_3 = \mathbf{1}^5 \in \mathbb{G}_2^5$ and $\boldsymbol{\gamma} = (\mathbf{0}^4, \hat{\mathbf{e}}_1) \in \mathbb{F}^5$.

We can express the verifier's map $\mathsf{V}_{\mathsf{SKZG}}$ as

$$\mathsf{V}_{\mathsf{SKZG}}(\mathbf{x}, \mathbf{w}, \mu, \mathbf{e}) = \mathsf{QuadTest}(\mathsf{pk}, (\mathbf{c}_1, \mathbf{c}_2, \mathbf{c}_3), \boldsymbol{\gamma})$$

We simplify the map $\mathsf{V}_{\mathsf{SZKG}}$ to obtain the optimized version of the relaxed pairing checks:

$$e(-\mu^2 \cdot \mathsf{cm}_f + \mu \cdot [v]_1, [1]_2) \cdot e(\mu \cdot \pi, [\tau]_2) \cdot \mathbf{e}_1 = e(\pi, [u]_2)$$

Accumulation for batch opening proofs Given configuration $\mathcal{C}_{\mathsf{BKZG}} = \{\mathsf{cm}_f\}$, the relation $\mathcal{R}_{\mathsf{BZKG}}$ consists of a set of tuples $(\mathbf{x}, \mathbf{w})$ where $\mathbf{x} = (\mathsf{cm}_Z, \mathsf{cm}_R) \in \mathbb{G}_2 \times \mathbb{G}_1$ and $\mathbf{w} = \pi \in \mathbb{G}_1$ such that $e(\pi, \mathsf{cm}_Z) = e(\mathsf{cm}_f - \mathsf{cm}_R, [1]_2)$. We let $\hat{\pi}, \hat{\mathsf{cm}}_f, \hat{\mathsf{cm}}_Z, \hat{\mathsf{cm}}_R$ be the decoded values such that $\pi = [\hat{\pi}]_1$, $\mathsf{cm}_f = [\hat{\mathsf{cm}}_f]_1$, $\mathsf{cm}_Z = [\hat{\mathsf{cm}}_Z]_2$, $\mathsf{cm}_R = [\hat{\mathsf{cm}}_R]_1$. The decoded public input is a tuple $\hat{\mathbf{x}} = (\hat{\mathsf{cm}}_Z, \hat{\mathsf{cm}}_R)$, and the decoded protocol message is $\hat{\mathbf{w}} = \hat{\pi}$. We define the corresponding algebraic checks:

$$\mathsf{V}'_{\mathsf{BKZG}}(\hat{\mathbf{x}}, \hat{\mathbf{w}}, \mu, \hat{\mathbf{e}}) = -\mu^2 \cdot \hat{\mathsf{cm}}_f + \mu \cdot \hat{\mathsf{cm}}_R + \hat{\pi} \cdot \hat{\mathsf{cm}}_Z - \hat{\mathbf{e}}_1$$

We can express the verifier's map $\mathsf{V}_{\mathsf{BKZG}}$ as

$$\mathsf{V}_{\mathsf{BKZG}}(\mathbf{x}, \mathbf{w}, \mu, \mathbf{e}) = \mathsf{QuadTest}(\mathsf{pk}, (\mathbf{c}_1, \mathbf{c}_2, \mathbf{c}_3), \boldsymbol{\gamma})$$

We simplify the map $\mathsf{V}_{\mathsf{BZKG}}$ to obtain the optimized version of the relaxed pairing checks:

$$e(-\mu^2 \cdot \mathsf{cm}_f + \mu \cdot \mathsf{cm}_R, [1]_2) \cdot e(\pi, \mathsf{cm}_Z) = \mathbf{e}_1$$

5.5 Vector addition

To complete the protocol description, we provide a simple subprotocol for vector addition.

Setup. Given parameter $n \in \mathbb{N}$ and vector $b \in \mathbb{F}^n$, the generation algorithm $\mathcal{G}$ outputs a structured reference string srs consisting of $\mathsf{b} := [B(x)]_1$.

Addition. The vector addition protocol is an interactive public coin protocol between a prover $\mathcal{P}$ and a verifier $\mathcal{V}$ where $\mathcal{P}$ has private input $f(X), g(X) \in \mathbb{F}_{<n}[X]$, and both parties have access to $\mathsf{f} = [f(X)]_1$, $\mathsf{g} = [g(X)]_1$, and the structured reference string srs. $\mathcal{P}$ aims to convince $\mathcal{V}$ that $f|_{\mathbb{H}} + b|_{\mathbb{H}} = g|_{\mathbb{H}}$. $\mathcal{P}$ send f, g and $\mathcal{V}$ checks that $\mathsf{f} + \mathsf{b} = \mathsf{g}$.

Accumulation. In the accumulation scheme, the relaxed verifier check is: $\mathsf{f} + \mathsf{b} - \mathsf{g} = \mathbf{e}_1$.

5.6 Main protocol

Given the subprotocols for matrix-vector multiplication, vector lookup, KZG opening proofs, and vector addition, the main protocol is simply a parallel composition of these four subprotocols and the Protostar protocol for Plonkish circuits. A small Plonkish circuit updates the current node index and type at each step. We employ a "universal circuit" model for tensor computations, in contrast to the circuit selection approach of Protostar. An identity matrix is used if no matrix-vector multiplication is performed at the given step. Similarly, an identity vector is used if no vector addition is performed at the given step. We use a single lookup table with vector elements. The first element specifies the node type, indicating the particular nonlinear activation function to be applied. The remaining elements are the identity vectors or the input-output values of the activation function. Since the main protocol is a composition of special-sound protocols, the main protocol is special-sound.

6 Evaluation

In this section, we compare Mira to other techniques for Groth16 proof aggregation and verifiable ML inference. Recall that SnarkStar refers to the instantiation of our scheme for Groth16 proof aggregation and TensorStar refers to the instantiation for verifiable ML inference.

For Groth16 proof aggregation, we compare to Protostar, Nebra UPA[3] and Succinct SP1.[4] Nebra UPA is an implementation of Groth16 proof aggregation based on Halo2, an implementation of Plonk [GWC19] that uses the KZG univariate polynomial commitment scheme [KZG10]. Succinct SP1 is a zero-knowledge virtual machine (zkVM) for Rust programs, which is based on Plonky3, an implementation of FRI-based univariate polynomial commitment schemes [BBHR18]. We use the BN254 pairing-friendly elliptic curve, which is adopted in production systems such as Ethereum.

For verifiable ML inference, we compare to Protostar, EZKL,[5] and Succinct SP1. EZKL is a specialized Halo2-based library for verifiable machine learning. We benchmark a 10-layer feedforward network for MNIST image classification, varying the hidden layer size from 2^5 to 2^{13}.

The Protostar baseline represents the pairing operation as part of the step circuit. Note that this baseline does not require verifying an additional succinct proof as in early constructions of IVC/PCD [BCTV14, CTV15]. However, there remains substantial overhead from encoding the pairing(s) of the target relation.

Our implementation is available open source.[6] We ran all of our benchmarks on Google Cloud Platform using c2-standard-60 instances with 60 vCPUs and 240 GB RAM.

As shown in Figure 1, Mira offers significantly better prover time and memory usage for the application of Groth16 proof aggregation while maintaining a constant-size proof. Mira aggregates 16 Groth16 proofs in 80.3 seconds using 6.9 GB of memory, yielding 5.8x faster prover time and 9.7x lower memory usage than Nebra UPA, a state-of-the-art Groth16 proof aggregation system. Furthermore, Mira achieves 3.5x faster prover time and 2.9x lower memory usage than Protostar.

We note that SnarkPack [GMN22] reports faster performance in the case of circuit-specific proof aggregation (i.e., aggregating 8192 proofs in 8.7 seconds). However, this scheme does not support aggregation of proofs with different verification keys, and distributed proving is not supported.

[3] `https://github.com/NebraZKP/upa`
[4] `https://github.com/succinctlabs/sp1`
[5] `https://github.com/zkonduit/ezkl`
[6] `https://github.com/joshbeal/mira`

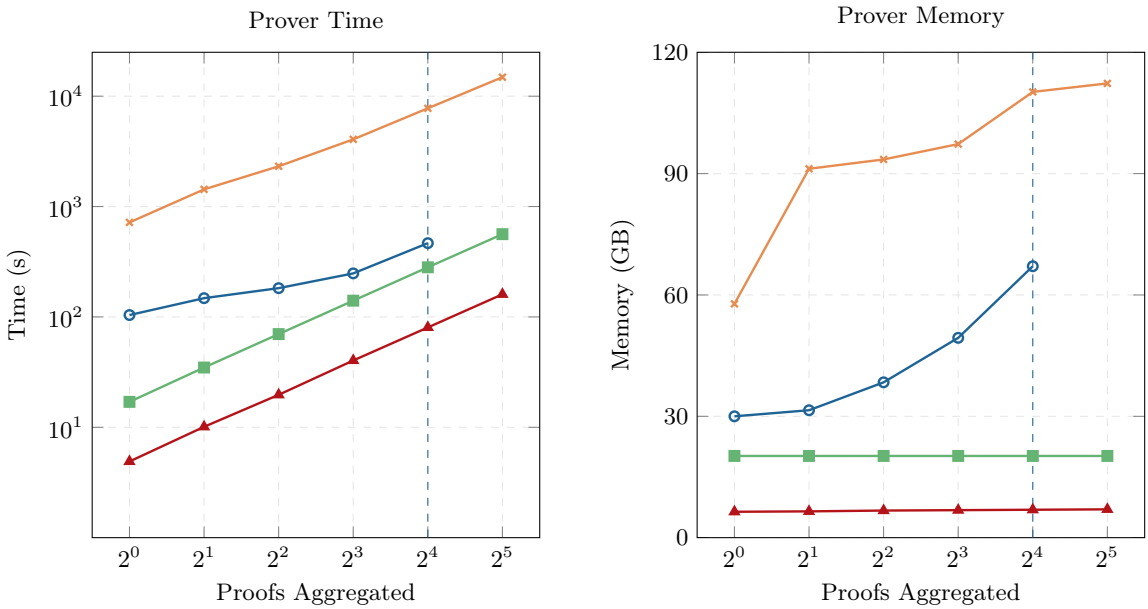


Fig. 1: Comparison of prover time/space complexity for Groth16 proof aggregation. We evaluated Mira - SnarkStar (▲), Protostar (■), Nebra UPA (○), and Succinct SP1 (x). Nebra UPA encountered out-of-memory errors at 32 proofs aggregated. Succinct SP1 has precompiles for BN254 group operations.

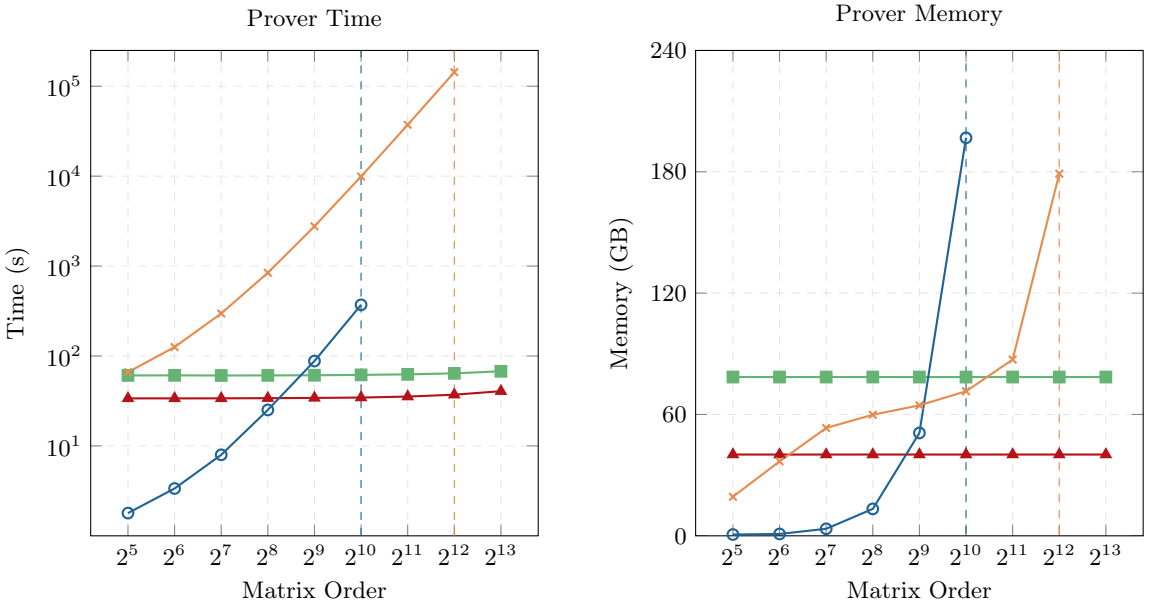


Fig. 2: Comparison of prover time/space complexity for verifiable ML inference. We evaluated Mira - TensorStar (▲), Protostar (■), EZKL (○), and Succinct SP1 (x). EZKL encountered out-of-memory errors at matrix orders 2^{11} and higher. Succinct SP1 encountered such errors at matrix order 2^{13}.

For realistic model sizes with matrix order of 2^9 or greater, Mira offers a faster prover with lower memory usage for verifiable ML inference, as illustrated in Figure 2. These results demonstrate Mira's scalability for larger models, whereas EZKL and Succinct were unable to generate proofs for larger models due to memory limitations.

In this work, we are interested in constructing a space-efficient scheme that supports distributed proving by low-resource nodes. While we evaluate a selection of monolithic proof systems for comparison, we do not evaluate zkLLM [SLZ24], which requires powerful GPU hardware.

7 Discussion

Further performance optimizations are possible using the techniques in CycleFold [KS24b]. Folding schemes are typically instantiated over a two-cycle of elliptic curves [BCTV14, NBS23]. In a further optimized scheme, we would use a circuit on the second curve in the cycle to represent each group operation. The prover would fold multiple invocations of these smaller circuits on the first curve in the cycle using Protogalaxy [EG23b]. This would avoid encoding multiple group operations in the step circuit. An alternative strategy from [KTW24] that defers the group operations is also applicable in this setting. By leveraging a space-efficient realization of the KZG polynomial commitment scheme [BCHO22], the prover memory usage could be further reduced. We leave implementing and evaluating these potential performance optimizations to future work.

8 Related work

Proof-carrying data. Our scheme generalizes the construction in Protostar [BC23] and leverages the IVC/PCD compiler of [BCL$^+$21]. Other PCD constructions are described in [BCCT13], [BCTV14], Halo [BGH19], [BCMS20], Fractal [COS20], Halo Infinite [BDFG21], Nova [KST22] and HyperNova [KS24b] and NeutronNova [KS24a]. Recently, lattice-based [BC24a] and hash-based [BMNW24a, BMNW24b] folding schemes have been proposed. These schemes are particularly effective for relations defined over small fields. However, pairing-based arguments use elliptic curves over large fields.

Distributed proving. DIZK [WZC$^+$18] proposes a method for distributing the work of Groth16 proof generation across many machines. Several follow-on works have developed similar techniques for other proof systems [OB22, GGJ$^+$23, LXZ$^+$24], while also improving the privacy and malicious security guarantees. Our work focuses on aggregation of the resulting proofs and is complementary to such techniques.

Proof aggregation. Other works have proposed specialized protocols for Groth16 proof aggregation with notable limitations. TIPP [BMM$^+$21] and SnarkPack [GMN22] obtain a logarithmic-size proof, while SnarkFold [LGZX23] and FLIP-and-prove R1CS [NPR24] can only aggregate proofs for the same R1CS structure. Mira enables aggregation of proofs for multiple structures while achieving a constant-size proof. Furthermore, we offer a generic framework that extends to new pairing-based arguments, such as the Garuda and Pari SNARKs [DMS24].

Verifiable machine learning. Several works have proposed specialized monolithic proof systems for ML inference [LXZ21, WYX$^+$21, WK23, CWSK24, SLZ24, LVA$^+$24]. In concurrent work, aggregation schemes for sumcheck-based proofs were developed for verifiable ML training [APKP24, BC24b], which also enables bounded-depth PCD for ML inference. On the other hand, our work has no bound on the computation depth.

References

AFK22. T. Attema, S. Fehr, and M. Klooß. Fiat-shamir transformation of multi-round interactive proofs. In *TCC 2022, Part I*, *LNCS* 13747, pages 113–142. Springer, Cham, November 2022.

AIM$^+$24. S. Angel, E. Ioannidis, E. Margolin, S. T. V. Setty, and J. Woods. Reef: Fast succinct non-interactive zero-knowledge regex proofs. In *USENIX Security 2024*. USENIX Association, August 2024.

APKP24. K. Abbaszadeh, C. Pappas, J. Katz, and D. Papadopoulos. Zero-knowledge proofs of training for deep neural networks. In *ACM CCS 2024*, pages 4316–4330. ACM Press, October 2024.

BBHR18. E. Ben-Sasson, I. Bentov, Y. Horesh, and M. Riabzev. Fast reed-solomon interactive oracle proofs of proximity. In *ICALP 2018*, *LIPIcs* 107, pages 14:1–14:17. Schloss Dagstuhl, July 2018.

BC23. B. Bünz and B. Chen. Protostar: Generic efficient accumulation/folding for special-sound protocols. In *ASIACRYPT 2023, Part II*, *LNCS* 14439, pages 77–110. Springer, Singapore, December 2023.

BC24a. D. Boneh and B. Chen. LatticeFold: A lattice-based folding scheme and its applications to succinct proof systems. Cryptology ePrint Archive, Report 2024/257, 2024.

BC24b. B. Bünz and J. Chen. Proofs for deep thought: Accumulation for large memories and deterministic computations. Cryptology ePrint Archive, Report 2024/325, 2024.

BCCT13. N. Bitansky, R. Canetti, A. Chiesa, and E. Tromer. Recursive composition and bootstrapping for SNARKS and proof-carrying data. In *45th ACM STOC*, pages 111–120. ACM Press, June 2013.

BCHO22. J. Bootle, A. Chiesa, Y. Hu, and M. Orrù. Gemini: Elastic SNARKs for diverse environments. In *EUROCRYPT 2022, Part II*, *LNCS* 13276, pages 427–457. Springer, Cham, May / June 2022.

BCI+13. N. Bitansky, A. Chiesa, Y. Ishai, R. Ostrovsky, and O. Paneth. Succinct non-interactive arguments via linear interactive proofs. In *TCC 2013*, *LNCS* 7785, pages 315–333. Springer, Berlin, Heidelberg, March 2013.

BCL+21. B. Bünz, A. Chiesa, W. Lin, P. Mishra, and N. Spooner. Proof-carrying data without succinct arguments. In *CRYPTO 2021, Part I*, *LNCS* 12825, pages 681–710, Virtual Event, August 2021. Springer, Cham.

BCMS20. B. Bünz, A. Chiesa, P. Mishra, and N. Spooner. Recursive proof composition from accumulation schemes. In *TCC 2020, Part II*, *LNCS* 12551, pages 1–18. Springer, Cham, November 2020.

BCTV14. E. Ben-Sasson, A. Chiesa, E. Tromer, and M. Virza. Scalable zero knowledge via cycles of elliptic curves. In *CRYPTO 2014, Part II*, *LNCS* 8617, pages 276–294. Springer, Berlin, Heidelberg, August 2014.

BDFG21. D. Boneh, J. Drake, B. Fisch, and A. Gabizon. Halo infinite: Proof-carrying data from additive polynomial commitments. In *CRYPTO 2021, Part I*, *LNCS* 12825, pages 649–680, Virtual Event, August 2021. Springer, Cham.

BF24. J. Beal and B. Fisch. Derecho: Privacy pools with proof-carrying disclosures. In *ACM CCS 2024*, pages 3197–3211. ACM Press, October 2024.

BFLS91. L. Babai, L. Fortnow, L. A. Levin, and M. Szegedy. Checking computations in polylogarithmic time. In *23rd ACM STOC*, pages 21–31. ACM Press, May 1991.

BGH19. S. Bowe, J. Grigg, and D. Hopwood. Halo: Recursive proof composition without a trusted setup. Cryptology ePrint Archive, Report 2019/1021, 2019.

BMM+21. B. Bünz, M. Maller, P. Mishra, N. Tyagi, and P. Vesely. Proofs for inner pairing products and applications. In *ASIACRYPT 2021, Part III*, *LNCS* 13092, pages 65–97. Springer, Cham, December 2021.

BMNW24a. B. Bünz, P. Mishra, W. Nguyen, and W. Wang. Accumulation without homomorphism. Cryptology ePrint Archive, Report 2024/474, 2024.

BMNW24b. B. Bünz, P. Mishra, W. Nguyen, and W. Wang. Arc: Accumulation for reed–solomon codes. Cryptology ePrint Archive, Report 2024/1731, 2024.

BMRS20. J. Bonneau, I. Meckler, V. Rao, and E. Shapiro. Coda: Decentralized cryptocurrency at scale. Cryptology ePrint Archive, Report 2020/352, 2020.

CCDW20. W. Chen, A. Chiesa, E. Dauterman, and N. P. Ward. Reducing participation costs via incremental verification for ledger systems. Cryptology ePrint Archive, Report 2020/1522, 2020.

COS20. A. Chiesa, D. Ojha, and N. Spooner. Fractal: Post-quantum and transparent recursive proofs from holography. In *EUROCRYPT 2020, Part I*, *LNCS* 12105, pages 769–793. Springer, Cham, May 2020.

CT10. A. Chiesa and E. Tromer. Proof-carrying data and hearsay arguments from signature cards. In *ICS 2010*, pages 310–331. Tsinghua University Press, January 2010.

CTV15. A. Chiesa, E. Tromer, and M. Virza. Cluster computing in zero knowledge. In *EUROCRYPT 2015, Part II*, *LNCS* 9057, pages 371–403. Springer, Berlin, Heidelberg, April 2015.

CWSK24. B. J. Chen, S. Waiwitlikhit, I. Stoica, and D. Kang. Zkml: An optimizing system for ml inference in zero-knowledge proofs. In *19th European Conference on Computer Systems, EuroSys 2024*, pages 560–574. Association for Computing Machinery, 2024.

DMS24. M. Dellepere, P. Mishra, and A. Shirzad. Garuda and pari: Faster and smaller SNARKs via equifficient polynomial commitments. Cryptology ePrint Archive, Report 2024/1245, 2024.

EFG22. L. Eagen, D. Fiore, and A. Gabizon. cq: Cached quotients for fast lookups. Cryptology ePrint Archive, Report 2022/1763, 2022.

EG23a. L. Eagen and A. Gabizon. cqlin: Efficient linear operations on KZG commitments with cached quotients. Cryptology ePrint Archive, Report 2023/393, 2023.

EG23b. L. Eagen and A. Gabizon. ProtoGalaxy: Efficient ProtoStar-style folding of multiple instances. Cryptology ePrint Archive, Report 2023/1106, 2023.

FK23. D. Feist and D. Khovratovich. Fast amortized KZG proofs. Cryptology ePrint Archive, Report 2023/033, 2023.

FKL18. G. Fuchsbauer, E. Kiltz, and J. Loss. The algebraic group model and its applications. In *CRYPTO 2018, Part II*, *LNCS* 10992, pages 33–62. Springer, Cham, August 2018.

GGJ+23. S. Garg, A. Goel, A. Jain, G.-V. Policharla, and S. Sekar. zkSaaS: Zero-knowledge SNARKs as a service. In *USENIX Security 2023*, pages 4427–4444. USENIX Association, August 2023.

GGP10. R. Gennaro, C. Gentry, and B. Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In *CRYPTO 2010*, *LNCS* 6223, pages 465–482. Springer, Berlin, Heidelberg, August 2010.

GMN22. N. Gailly, M. Maller, and A. Nitulescu. SnarkPack: Practical SNARK aggregation. In *FC 2022*, *LNCS* 13411, pages 203–229. Springer, Cham, May 2022.

Gro16. J. Groth. On the size of pairing-based non-interactive arguments. In *EUROCRYPT 2016, Part II*, *LNCS* 9666, pages 305–326. Springer, Berlin, Heidelberg, May 2016.

GWC19. A. Gabizon, Z. J. Williamson, and O. Ciobotaru. PLONK: Permutations over Lagrange-bases for oecumenical noninteractive arguments of knowledge. Cryptology ePrint Archive, Report 2019/953, 2019.

Kil92. J. Kilian. A note on efficient zero-knowledge proofs and arguments (extended abstract). In *24th ACM STOC*, pages 723–732. ACM Press, May 1992.

KS24a. A. Kothapalli and S. Setty. NeutronNova: Folding everything that reduces to zero-check. Cryptology ePrint Archive, Report 2024/1606, 2024.

KS24b. A. Kothapalli and S. T. V. Setty. HyperNova: Recursive arguments for customizable constraint systems. In *CRYPTO 2024, Part X*, *LNCS* 14929, pages 345–379. Springer, Cham, August 2024.

KST22. A. Kothapalli, S. Setty, and I. Tzialla. Nova: Recursive zero-knowledge arguments from folding schemes. In *CRYPTO 2022, Part IV*, *LNCS* 13510, pages 359–388. Springer, Cham, August 2022.

KTW24. T. Kohrita, P. Towa, and Z. J. Williamson. One-shot native proofs of non-native operations in incrementally verifiable computations. Cryptology ePrint Archive, Report 2024/1651, 2024.

KZG10. A. Kate, G. M. Zaverucha, and I. Goldberg. Constant-size commitments to polynomials and their applications. In *ASIACRYPT 2010*, *LNCS* 6477, pages 177–194. Springer, Berlin, Heidelberg, December 2010.

LGZX23. X. Liu, S. Gao, T. Zheng, and B. Xiao. SnarkFold: Efficient SNARK proof aggregation from split incrementally verifiable computation. Cryptology ePrint Archive, Report 2023/1946, 2023.

LKA+24. D. Luick, J. C. Kolesar, T. Antonopoulos, W. R. Harris, J. Parker, R. Piskac, E. Tromer, X. Wang, and N. Luo. ZKSMT: A VM for proving SMT theorems in zero knowledge. In *Proceedings of the 33rd USENIX Security Symposium (USENIX Security 2024)*, pages 3837–3845, 2024.

LOB24. E. Laufer, A. Ozdemir, and D. Boneh. zkPi: Proving lean theorems in zero-knowledge. Cryptology ePrint Archive, Report 2024/267, 2024.

LVA+24. H. Lycklama, A. Viand, N. Avramov, N. Küchler, and A. Hithnawi. Artemis: Efficient commit-and-prove snarks for zkml. *arXiv preprint arXiv:2409.12055*, 2024.

LXZ21. T. Liu, X. Xie, and Y. Zhang. zkCNN: Zero knowledge proofs for convolutional neural network predictions and accuracy. In *ACM CCS 2021*, pages 2968–2985. ACM Press, November 2021.

LXZ+24. T. Liu, T. Xie, J. Zhang, D. Song, and Y. Zhang. Pianist: Scalable zkRollups via fully distributed zero-knowledge proofs. In *2024 IEEE Symposium on Security and Privacy*, pages 1777–1793. IEEE Computer Society Press, May 2024.

Mic94. S. Micali. CS proofs (extended abstracts). In *35th FOCS*, pages 436–453. IEEE Computer Society Press, November 1994.

NBS23. W. Nguyen, D. Boneh, and S. Setty. Revisiting the nova proof system on a cycle of curves. Cryptology ePrint Archive, Report 2023/969, 2023.

NPR24. A. Nitulescu1, N. Paslis, and C. Ràfols. Flip-and-prove r1cs. Cryptology ePrint Archive, Report 2024/1364, 2024.

NT16. A. Naveh and E. Tromer. PhotoProof: Cryptographic image authentication for any set of permissible transformations. In *2016 IEEE Symposium on Security and Privacy*, pages 255–271. IEEE Computer Society Press, May 2016.

OB22. A. Ozdemir and D. Boneh. Experimenting with collaborative zk-SNARKs: Zero-knowledge proofs for distributed secrets. In *USENIX Security 2022*, pages 4291–4308. USENIX Association, August 2022.

SCJ+24. T. South, A. Camuto, S. Jain, S. Nguyen, R. Mahari, C. Paquin, J. Morton, and A. S. Pentland. Verifiable evaluations of machine learning models using zksnarks. *arXiv preprint arXiv:2402.02675*, 2024.

SLZ24. H. Sun, J. Li, and H. Zhang. zkLLM: Zero knowledge proofs for large language models. In *ACM CCS 2024*, pages 4405–4419. ACM Press, October 2024.

TSZ+24. E. N. Tas, I. A. Seres, Y. Zhang, M. Melczer, M. Kelkar, J. Bonneau, and V. Nikolaenko. Atomic and fair data exchange via blockchain. Cryptology ePrint Archive, Report 2024/418, 2024.

Val08. P. Valiant. Incrementally verifiable computation or proofs of knowledge imply time/space efficiency. In *TCC 2008*, *LNCS* 4948, pages 1–18. Springer, Berlin, Heidelberg, March 2008.

WK23. S. Waiwitlikhit and D. Kang. Tensorplonk: A "gpu" for zkml, delivering 1,000x speedups. 2023. `https://medium.com/@danieldkang/tensorplonk-a-gpu-for-zkml-delivering-1-000x-speedups-d1ab0ad27e1c`.

WYX+21. C. Weng, K. Yang, X. Xie, J. Katz, and X. Wang. Mystique: Efficient conversions for zero-knowledge proofs with applications to machine learning. In *USENIX Security 2021*, pages 501–518. USENIX Association, August 2021.

WZC+18. H. Wu, W. Zheng, A. Chiesa, R. A. Popa, and I. Stoica. DIZK: A distributed zero knowledge proof system. In *USENIX Security 2018*, pages 675–692. USENIX Association, August 2018.

WZY+24. W. Wang, L. Zhou, A. Yaish, F. Zhang, B. Fisch, and B. Livshits. Mechanism design for zk-rollup prover markets. *arXiv preprint arXiv:2404.06495*, 2024.