A Direct Construction for Asymptotically Optimal zkSNARKs

Abhiram Kothapalli [1], Elisaweta Masserova [2], and Bryan Parno [3]

Carnegie Mellon University

Abstract. We present the first direct construction of a zero-knowledge argument system for general computation that features a linear-time prover and a constant-time verifier (after a single linear-time public setup) in terms of the number of field and group operations. Our scheme utilizes a *universal* linear-size structured reference string (SRS) that allows a single trusted setup to be used across all computation instances of a bounded size. Concretely, for computations of size n, our prover's cost is dominated by 35 multi-exponentiations of size n and our verifier's cost is dominated by 34 pairings. To achieve the stated asymptotics, we first construct a nearly-optimal zkSNARK with a logarithmic verifier in the random oracle model. We then show how to achieve a constant-time verifier using proof composition. Along the way we design (1) a new polynomial commitment scheme for evaluation-based representations of polynomials, (2) an asymptotically optimal inner-product argument system, (3) an asymptotically optimal multi-Hadamard-product argument system, and (4) a new constraint system for NP that is particularly well-suited for our bundle of techniques.

1 Introduction

Verifiable computation [44] allows a weak client to outsource a computation and efficiently verify that the returned result is correct. Many recent verifiable computation schemes provide an orthogonal zero-knowledge guarantee, in which the server running the computation can provide a private input to the computation, and still prove correct execution without revealing any information about the input. Such powerful integrity and privacy guarantees have enabled an exciting class of applications, including anonymous credentials [39], image authentication [63], verifiable storage outsourcing [5], blockchain applications [38, 59, 68], verifiable database operation [80, 81], and voting [82].

As shown by Goldwasser et al. [49], this class of interaction can be modeled as a *zero-knowledge argument system.* A zero-knowledge argument system is an interactive protocol in which a *prover* proves a "computational statement" (e.g. "Program P outputs y, on public input x and secret input s") to a *verifier*. Many

[1] akothapa@andrew.cmu.edu

[2] elisawem@andrew.cmu.edu

[3] parno@cmu.edu

classical results address how to model and realize such an interaction [3,4,6,7,9]. These results however are not designed to be practical for the majority of interesting applications, which demand good concrete costs in addition to a sublinear verifier with respect to the size of the original statement (succinctness) and non-interactivity. Modern performance-oriented argument systems that target these requirements and support a broad class of computational statements (such as NP) are typically dubbed *zero-knowledge succinct non-interactive arguments of knowledge* (zkSNARK) [21, 46]. Generally these sorts of argument systems are achieved by putting together the following pieces: (1) a constraint system to represent computational statements as low-level algebraic constraints, (2) mathematical representations (e.g. polynomials) to encode constraints and purported satisfying assignments, (3) efficient algebraic tests (e.g. polynomial equality testing) to check that the encoded assignment satisfies the prescribed algebraic constraints, and (4) cryptographic machinery to prove in zero-knowledge that the prescribed algebraic tests are satisfied.

Many performance-oriented zero-knowledge argument systems have proposed various bundles of techniques to address each of the listed pieces [2,29,33,65,70, 77]. However, all proposals make some combination of the following undesirable compromises: (1) a per-circuit trusted setup (which is especially problematic in settings such as the blockchain where there is no clear authority); (2) a super-linear prover and/or a (super)-logarithmic verifier which hurts practical efficiency and may defeat the purpose of outsourcing computation; (3) a restricted class of computations (such as circuits with repeated structure).

In contrast, our system makes none of these compromises — Instead, by developing new techniques and extending techniques from several previous strands of work, we achieve an efficient zkSNARK for general computation.

1.1 Our Results

We present a time-optimal zkSNARK for general computation that features a linear-time prover and a constant-time verifier (after a single linear-time public setup) in terms of the number of field and group operations. Our argument system utilizes a *universal* linear-size structured reference string (SRS) (i.e., a single trusted setup can be used across all computation instances of a bounded size). In terms of concrete costs, our system is comparable to existing highly optimized zkSNARKs: For computations of size n, our prover's cost is dominated by 35 multi-exponentiations of size n and our verifier's cost is dominated by 34 pairings.

Our core argument is constructed by modifying argument systems for linear algebraic statements (e.g., inner-product, Hadamard-product, polynomial evaluation) to match our desired asymptotics before composing them to create a argument system for general computation. In particular, existing argument systems typically feature either a non-constant-time verifier or a super-linear prover. To achieve a linear-time prover, we design new techniques to avoid super-linear operations (such as polynomial interpolation). Later, to achieve a constant-time verifier, we outsource the verifier's non-constant-time work using a generic argument system. We give a high level overview of our core contributions leading to an argument system with optimal asymptotics:

(i) *Polynomial Commitments for Evaluation-Based Representations:* A polynomial commitment scheme allows a prover to commit to a polynomial and later verifiably evaluate it at a challenge point. We propose a new polynomial commitment scheme based on that of Zhang et al. [81] specifically tailored for evaluation-based representations of polynomials. This allows us to avoid expensive interpolation operations typically found in argument systems that rely on polynomial-based representations.
(ii) *Extended Inner-Product Argument:* An inner-product argument allows a prover to commit to two vectors and later verifiably evaluate their inner-product. We extend the inner-product argument presented by Bünz et al. [31] to operate over commitments to evaluation-based representations of polynomials. Additionally, we modify this argument system to support zero-knowledge.
(iii) *Asymptotically Optimal Multi-Hadamard-Product Argument:* A multi-Hadamard-product argument allows a prover to commit to a list of vectors and later verifiably evaluate the Hadamard product of these vectors. Bayer [8] presents a multi-Hadamard argument that features a linear-time prover and verifier. We show how to compose Bayer's argument with our modified inner-product argument to achieve a constant-time verifier.
(iv) *A New Constraint System to Characterize NP:* We design a new constraint system to capture NP that is particularly well-suited for our bundle of techniques. We refer to the relation defining our constraint system as $\mathcal{R}_{\mathsf{ACS}}$. We show how to encode $\mathcal{R}_{\mathsf{ACS}}$ statements into polynomial representations that can be checked using the techniques described above.

Putting together the listed pieces, we achieve a public-coin argument with a logarithmic number of rounds in the standard model using just a structured reference string. This argument can be made non-interactive with a logarithmic-time verifier in the random oracle model. To outsource the verifier's non-constant-time work, we must instantiate the random oracle with a cryptographic hash function before representing the verifier's checks as a circuit. A similiar approach is also taken by Chiesa et al. [34] and Bowe et al. [27]. Part of the difficulty of outsourcing in our setting is that we must ensure that the outsourced circuit is small enough to preserve our desired asymptotics. In particular, we must design a polylogarithmic sized circuit with a constant-sized verifier input. We carefully construct our suite of techniques for to comply with these constraints.

1.2 Related Work

Theoretical Foundations: Early work in probabilistically checkable proofs (PCPs) [3, 4, 7, 67] considers the setting in which the verifier queries an oracle representing the prover's proof. Kilian [58] provides the first PCP-based construction by utilizing Merkle tree commitments to the prover's proof. A line of work [71–73] initiated by Ishai et al. [55] achieves a more efficient PCP-based proof system by encoding the proof as a linear function, called a linear PCP. Unfortunately many of these foundational works are prohibitively expensive for realistic applications, from both an asymptotic and a practical standpoint.

zkSNARKs for a Limited Class of Computation: In an effort to create practical systems, Goldwasser et al. [48] describe an interactive argument system (over layered arithmetic circuits), which consists of proving statements about each layer of the circuit using the sum-check protocol proposed by Lund et al. [61]. A large line of work [35, 75, 77] refines this approach by considering uniform circuits (i.e., descriptions of the circuit are asymptotically smaller than the circuit itself). Recent works additionally achieve zero-knowledge [77], and an asymptotically linear prover [78, 79]. Unfortunately, systems in this line rely on layered, uniform circuits, in order to achieve a logarithmic verifier, limiting the class of computations which can be efficiently encoded.

zkSNARKs with a Trusted Setup: In a parallel vein, Gennaro et al. [45] achieve a constant-time verifier and a *nearly* linear prover for general computations by making use of a per-instance trusted setup. Core to their work is a new constraint system, Quadratic Arithmetic Programs, which inspires the constraint system designed in this work. Setty et al. [71], followed by Bitansky et al. [24], show that Quadratic Arithmetic Programs can be viewed as a highly optimized linear PCP. Parno et al. optimize Gennaro et al.'s [45] protocol to produce a highly optimized implementation, Pinocchio [65]. A large line of work optimizes and/or applies Pinocchio in various settings [14, 36, 37, 41, 42, 53, 68]. Systems in this line require a *per-circuit* structured reference string (SRS) generated privately by a trusted party, which can be problematic in practice [68]. Additionally, for computations of size n, these systems require $O(n \log n)$ field operations which adds a non-trivial overhead in practice [65].

Efforts To Remove a Trusted Setup: Practical issues with private setup procedures have caused a recent surge in argument systems without a trusted setup, (i.e. *transparent* zkSNARKs). Using only the discrete logarithm assumption, Groth [52] proposes an argument system for statements in NP, by combining zero-knowledge argument systems for linear-algebraic operations such as matrix product, Hadamard product, and inner-product. This system is implemented by Bootle et al. [26] and later refined by Bünz et al. [29]. Systems in this line require the verifier perform linear work in the size of the original computation making them useful only for their zero-knowledge properties, not for outsourcing.

Aurora [16] and zkSTARKs [11] achieve a transparent setup by building upon a line of work by Ben-Sasson et al. [10, 12, 13, 17, 19]. The soundness for both of these systems relies on non-standard assumptions related to Reed-Solomon Codes. Unfortunately, both Aurora and zkSTARKs also feature a linear verifier and a *nearly* linear prover. zkSTARKs achieves a polylogarithmic verifier when considering uniform circuits, but relies on a computational model which can add significant overhead in practice [76].

Ishai et al. construct transparent zero-knowledge argument systems using secure multi-party computation as a fundamental building block [56]. Several works refine this approach [32,47]; however all of these works feature a linear-time verifier. Ames et al. [2] show how to achieve a sublinear verifier by amortizing over multiple instances of the same verification circuit.

Recently, Setty proposed Spartan [70], the first direct construction for a transparent zkSNARK with sublinear verifier without any assumptions about the circuit structure. In more detail, Spartan reduces matrix encodings of arithmetic circuits to a sum-check instance over sparse multivariate polynomials which are verifiably evaluated in zero-knowledge by using the argument system proposed by Wahby et al [77]. Unfortunately for computations of size n, Spartan's verifier still runs in time $O(\sqrt{n})$.

Universal Trusted Setups as an Alternate Solution: The preceding discussion of transparent zkSNARKs indicates that it is unclear how to achieve an asymptotically optimal verifier without the use of a trusted setup. Two recent works, Sonic [62] and Marlin [33], take a middle-ground approach and study the setting where a private trusted setup is performed *only once*, and the resulting SRS can be reused across all circuits that respect a certain size bound (i.e., a *universal* trusted setup). For computations of size n, Sonic achieves an $O(n \log n)$ prover and a constant time verifier, and Marlin achieves an $O(n \log n)$ prover and an $O(\log n)$ verifier (although Marlin is considerably cheaper in practice). In fact, Setty describes a variant of Spartan that utilizes a universal trusted setup to achieve an $O(\log^2 n)$ verifier. Encouraged by these results, we also adopt this setting in our work.

Optimal Asymptotics via Recursive Composition: A zkSNARK supports *recursive composition* if the verifier's execution can be expressed as another computation instance to be proved, thus allowing the prover to write proofs about proofs. Valiant [74] shows how to take any succinct argument system that supports recursive composition and achieve a linear time prover and a (practically) constant-time verifier. Roughly, Valiant's prover breaks down a large circuit into many small circuits and writes a proof of correct execution for each before "folding" all of these proofs into a single (constant-sized) proof using a tree-like structure. Both Bitansky et al. [22] and Ben-Sasson et al. [18] refine this transformation. Bitansky et al.'s transformation can be applied to two recent recursive proof systems, Halo [27] and Fractal [34], to achieve time-optimal zkSNARKs. Unfortunately, recursive composition applied in this manner incurs quite expensive overheads in practice. In contrast, our work achieves an optimal zkSNARK with significantly reduced overhead via a direct construction.

Scheme	**Setup**	**Prover**	**Verifier**	**Proof Size**	**Model**
zkSTARK [11]	public	n polylog n	$\log^2 n$	$\log^2 n$	uniform circuits
Ligero [2]	public	$n \log n$	n	n	arithmetic circuits
Aurora [16]	public	$n \log n$	n	$\log^2 n$	R1CS
Hyrax [77]	public	$d(g + c \log c) + w$	$d \log g + \sqrt{w}$	$\sqrt{n}$	arithmetic circuits
Virgo [79]	public	$n + n \log n$	$d \log n + \log^2 n$	$d \log n + \log^2 n$	uniform circuits
Spartan$_{\text{SQRT}}$ [70]	public	n	$\sqrt{n}$	$\sqrt{n}$	R1CS
Bulletproofs [29]	public	n	n	$\log n$	arithmetic circuits
SuperSonic [30]	private**	$n \log n$	$\log n$	$\log n$	arithmetic circuits
Marlin [33]	private**	$n \log n$	$\log n$	1	R1CS
Libra [78]	private*	n	$d \log n$	$d \log n$	uniform circuits
Spartan$_{\text{KZG}}$ [70]	private*	n	$\log^2 n$	$\log n$	R1CS
GGPR-based [65]	private	$n \log n$	x	1	arithmetic circuits
This Work	private*	n	x	1	$\mathcal{R}_{\mathsf{ACS}}$

Table 1: Asymptotic costs of various zero-knowledge proof systems in terms of field and group operations. n denotes the circuit size. d denotes the circuit depth. x denotes the size of the circuit inputs and outputs. w denotes the size of the provers private input. c denotes the number of repeating identical subcircuits. g denotes the width of the circuit. private* denotes a universal setup. private** denotes a universal and updatable trusted setup. R1CS is an algebraic constraint system based on Quadratic Arithmetic Programs [45].

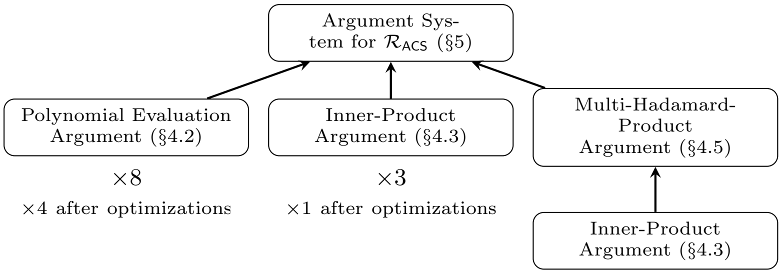
Argument System for $\mathcal{R}_{\mathsf{ACS}}$ (§5)
Polynomial Evaluation Argument (§4.2)
Inner-Product Argument (§4.3)
Multi-Hadamard-Product Argument (§4.5)
×8
×4 after optimizations
×3
×1 after optimizations
Inner-Product Argument (§4.3)

Fig. 1: Overview of the techniques involved to construct our argument system for general computation. We achieve concrete optimizations by batching and instantiating in the Algebraic Group Model [43]. More details are provided in Section 6.

2 Technical Overview

We present a zero-knowledge non-interactive argument system for computational statements in NP that features a linear-time setup, a linear-time prover, and a constant-time verifier in terms of group and field operations. We start by constructing a new linear algebraic constraint system, $\mathcal{R}_{\mathsf{ACS}}$, that resembles Quadratic Arithmetic Programs [45]. Formally, $\mathcal{R}_{\mathsf{ACS}}$ is modeled as a relation that consists of a public *statement* (represented as matrices) and a private *witness* (represented as a vector). In an argument system for $\mathcal{R}_{\mathsf{ACS}}$, the prover shows — in zero-knowledge — that it knows a witness that satisfies the constraints encoded in the statement. We show how to encode both the statement and witness of an $\mathcal{R}_{\mathsf{ACS}}$ instance as polynomials (as part of a single linear-time public setup). To check that the prover's witness polynomial satisfies the given statement polynomials with respect to relation $\mathcal{R}_{\mathsf{ACS}}$, the verifier is tasked with checking that evaluations of a witness-dependent polynomial over a specified set of points sum to 0.

This check can be viewed as a sum-check instance [61], and indeed several recent systems have tackled similiar checks using a generic sum-check protocol [16, 70, 77]. However existing sum-check protocols do not meet our desired asymptotic goals: They either induce a super-linear prover [16] or a non-constant verifier [78]. To get around this, we avoid generic sum-check protocols, and instead carefully tailor our constraint system and the resulting encoding polynomials such that the verifier's task is reduced to (1) checking polynomial equality and (2) checking a sum of the following form:

$$\sigma = \sum_{h \in H} A(h) \cdot B(h) \tag{1}$$

for some predefined set of points H, claimed sum σ, and polynomials $A(X)$, and $B(X)$. Here we realize that the right-hand side of Equation 1 can be evaluated by taking an inner-product over vectors of evaluations of polynomials A and B. This realization motivates us to represent polynomials using their *evaluation-based* representation rather than a coefficient-based representation.

Thus, the verifier can efficiently check Equation 1 by using an argument system for inner-product (Construction 4). We note that the fastest existing inner-product argument [31] still features a logarithmic-time verifier which is incompatible with our asymptotic goals. To get around this, we show how to outsource the verifier's logarithmic work using proof-composition, a technique recently revisited by several argument systems [70, 79].

As for the polynomial equality check, we show how the verifier can reduce this task to another another (simpler) sum-check instance using the Schwartz-Zippel Lemma [69]. The verifier repeats this interaction over several rounds to reduce the original statement to checking the Hadamard-product over vectors generated during the interaction. Thus, in the final round, the verifier engages in an Hadamard-product argument over multiple vectors, which in turn relies on another inner-product argument (Construction 5).

We note that we cannot achieve a sublinear verifier if the prover directly sends the aforementioned polynomials, which are linear in the size of the $\mathcal{R}_{\mathsf{ACS}}$ instance. Instead the prover sends commitments to these polynomials (Construction 1), and later engages in arguments regarding these commitments to convince the verifier that its checks would pass (a technique seen in several recent works [33,70,77,79–81]). Traditionally polynomial commitments, as defined by Kate et al. [57], refer to both the scheme to commit to a vector representing a polynomial and the argument system to evaluate polynomials "under" these commitments. However, in our setting we utilize the same commitment value in multiple contexts: Specifically we treat such commitments as vector commitments when involved in inner product arguments, and as polynomial commitments when involved in polynomial-evaluation arguments. To maintain a cleaner presentation we separate the schemes to commit to a vector representing a polynomial (Construction 1) and the argument system to evaluate committed polynomials (Construction 2).

Throughout the argument, the prover is required to evaluate polynomials (represented as vectors of evaluations) "under" its commitments at challenge points. Our polynomial-evaluation argument modifies that of Zhang et al. [81] which in turn is based on the scheme by Papamanthou et al. [64], both of which achieve a linear-time prover and a constant-time verifier for univariate polynomials represented as coefficients. In order to efficiently evaluate univariate polynomials based on their evaluation representations, we design a structured key which utilizes the Lagrange basis (Definition 7).

Several of the listed techniques require a structured reference string to be generated during a trusted setup phase. We ensure that these setup procedures are not instance dependent, which allows the overall argument system to maintain a universal SRS.

We additionally ensure that all the components used in our argument system for general computation are public-coin (the verifier only sends random challenges) thus ensuring that it can be made non-interactive using the Fiat-Shamir transform [40] in the random-oracle model.

We achieve a constant-time verifier by first instantiating the random oracle with a cryptographic hash function (as done by Chiesa et al. [34] and Bowe et al. [27]) and then outsourcing the verifier's logarithmic work using (another) general-purpose argument system. We summarize the key components of our construction in Figure 1.

2.1 Roadmap

In Section 3 we define argument systems, present several algebraic preliminaries, and define our cryptographic assumptions. In Section 4 we define and provide constructions for vector commitments (§4.1), polynomial-evaluation arguments (§4.2), inner-product arguments (§4.3), and multi-Hadamard arguments (§4.5). In Section 5 we define the $\mathcal{R}_{\mathsf{ACS}}$ constraint system and our argument system for $\mathcal{R}_{\mathsf{ACS}}$. In Section 6 we describe proof-composition techniques to achieve a constant-time verifier, and achieve non-interactivity and zero-knowledge.

3 Preliminaries

3.1 Argument System

An *argument system* is a protocol in which a *prover* proves a "computational statement" to a *verifier*. Formally we capture a computational statement as a ternary relation. For relation $\mathcal{R}$, given public parameters pp, we call w a witness for a statement u if $(\mathsf{pp}, w, u) \in \mathcal{R}$. In this section we define argument systems and their desired properties. We adapt the following notation and definitions from both Chiesa et al. [33] and Bünz et al. [29].

Definition 1 (Argument System). *Let $\mathcal{R} \subset \{0,1\}^* \times \{0,1\}^* \times \{0,1\}^*$ be a polynomial-time-decidable ternary relation. An argument system for relation $\mathcal{R}$ is a tuple of three probabilistic polynomial-time (non-)interactive algorithms $(\mathcal{G}, \mathcal{P}, \mathcal{V})$, denoted the generator, prover, and verifier respectively, with the following structure*

- $\mathcal{G}(\lambda, \mathsf{N}) \to \mathsf{pp}$*: Takes as input security parameter λ and the size bound $\mathsf{N} \in \mathbb{N}$. Outputs public parameters* pp.
- $\mathcal{P}(\mathsf{pp}, u, w)$*: Takes as input public parameters* pp*, statement u, and witness w.*
- $\mathcal{V}(\mathsf{pp}, u) \to 0/1$*: Takes as input public parameters* pp *and statement u. Outputs 0 for* `reject` *and 1 for* `accept`.

Let $\mathsf{tr} \leftarrow \langle \mathcal{P}(\mathsf{pp}, u, w), \mathcal{V}(\mathsf{pp}, u)\rangle$ *denote the transcript* tr *produced by $\mathcal{P}$ and $\mathcal{V}$ on their specified inputs. Let $\langle \mathcal{P}(\mathsf{pp}, u, w), \mathcal{V}(\mathsf{pp}, u)\rangle = 0/1$ denote the verifier's output at the end of the interaction. For relation $\mathcal{R}$, $(\mathcal{G}, \mathcal{P}, \mathcal{V})$ satisfies perfect completeness if*

$$\Pr\left[\begin{matrix}(\mathsf{pp}, u, w) \in \mathcal{R}, \\ \langle \mathcal{P}(\mathsf{pp}, u, w), \mathcal{V}(\mathsf{pp}, u)\rangle = 1\end{matrix} \middle| \mathsf{pp} \leftarrow \mathcal{G}(\lambda, \mathsf{N})\right] = 1$$

and satisfies soundness if for any PPT adversary

$$\Pr\left[\begin{matrix}\langle \mathcal{P}^*(\mathsf{pp}, u, w), \mathcal{V}(\mathsf{pp}, u)\rangle = 1, \\ (\mathsf{pp}, u, w) \notin \mathcal{R}\end{matrix} \middle| \mathsf{pp} \leftarrow \mathcal{G}(\lambda, \mathsf{N})\right] = \mathsf{negl}(\lambda).$$

Definition 2 (Knowledge-Soundness). *Informally, knowledge soundness captures the notion that if the verifier is convinced of a specified statement, then the prover must possess the corresponding witness. Formally, an argument system for relation $\mathcal{R}$, $(\mathcal{G}, \mathcal{P}, \mathcal{P})$, satisfies knowledge-soundness if for any probabilistic polynomial time prover $\mathcal{P}^*$ there exists a probabilistic polynomial time extractor $\mathcal{E}$ such for all inputs u*

$$\Pr\left[\begin{matrix}\langle \mathcal{P}^*(\mathsf{pp}, u, \rho), \mathcal{V}(\mathsf{pp}, u)\rangle = 1, \\ (\mathsf{pp}, u, w) \notin \mathcal{R}\end{matrix} \middle| \begin{matrix}\mathsf{pp} \leftarrow \mathcal{G}(\lambda, \mathsf{N}), \\ w \leftarrow \mathcal{E}(\mathsf{pp}, u, \rho)\end{matrix}\right] = \mathsf{negl}(\lambda)$$

where ρ denotes the input randomness for $\mathcal{P}^*$.

Definition 3 ((Special Honest-Verifier) Zero-Knowledge). *Informally, (Special Honest-Verifier) Zero-Knowledge captures the property that an (honest) verifier gains no additional information after viewing a proof of correct execution. Formally, an interactive argument system $(\mathcal{G}, \mathcal{P}, \mathcal{V})$ satisfies zero-knowledge for relation $\mathcal{R}$ if there exists a PPT simulator $\mathcal{S}$ such that for any PPT adversary $\mathcal{V}^*$, pair of interactive adversaries $\mathcal{A}_1, \mathcal{A}_2$, and auxiliary input z*

$$\left| \begin{array}{l} \Pr\left[\begin{array}{l} (\mathsf{pp}, u, w) \in \mathcal{R}, \\ \mathcal{A}_2(\mathsf{tr}) = 1 \end{array} \middle| \begin{array}{l} \mathsf{pp} \leftarrow \mathcal{G}(\lambda, \mathsf{N}), \\ (u, w, \mathsf{st}) \leftarrow \mathcal{A}_1(\mathsf{pp}, z), \\ \mathsf{tr} \leftarrow \langle \mathcal{P}(\mathsf{pp}, u, w), \mathcal{V}^*(\mathsf{pp}, u; \mathsf{st}) \rangle \end{array} \right] - \\ \Pr\left[\begin{array}{l} (\mathsf{pp}, u, w) \in \mathcal{R}, \\ \mathcal{A}_2(\mathsf{tr}) = 1 \end{array} \middle| \begin{array}{l} (\mathsf{pp}, \mathsf{trap}) \leftarrow \mathcal{S}(\lambda, \mathsf{N}), \\ (u, w) \leftarrow \mathcal{A}_1(\mathsf{pp}, z), \\ \mathsf{tr} \leftarrow \mathcal{S}(\mathsf{pp}, \mathsf{trap}, u) \end{array} \right] \end{array} \right| \leq \mathsf{negl}(\lambda).$$

$(\mathcal{G}, \mathcal{P}, \mathcal{V})$ satisfies special honest-verifier zero-knowledge if $\mathcal{V}^*$ is constrained to be the honest verifier.

Definition 4 (Public Coin). *An argument system $(\mathcal{G}, \mathcal{P}, \mathcal{V})$ is called public coin if all the messages sent from $\mathcal{V}$ to $\mathcal{P}$ are chosen uniformly at random and independently of the prover's messages.*

3.2 Algebraic Preliminaries

The $\mathcal{R}_{\mathsf{ACS}}$ relation involves statements and witnesses represented as a set of polynomials over a field $\mathbb{F}$. $\mathcal{R}_{\mathsf{ACS}}$ efficiently encodes conditions that dictate a valid statement-witness polynomial pair using *vanishing polynomials*, and the *formal derivative of vanishing polynomials*. We borrow both notation and several of the following definitions from Chiesa et al. [33].

Notation 1 (Vectors and Matrices). Throughout this work we denote vectors and matrices using a bold font (i.e. $\boldsymbol{v}$ and $\boldsymbol{M}$). For matrix M we let $M[i, j]$ denote the entry at row i and column j. We let v_i denote element i of vector $\boldsymbol{v}$. We define vectors by their individual components using parenthesis (i.e. $\boldsymbol{v} = (v_1, v_2, \ldots, v_n)$). We denote vector w appended to vector v as $(\boldsymbol{v}, \boldsymbol{w})$. We define $\boldsymbol{v} \cdot \boldsymbol{w}$ to be the inner-product and $\boldsymbol{v} \circ \boldsymbol{w}$ to be the Hadamard product. For vectors $\boldsymbol{g}$ and $\boldsymbol{x}$ of the same length, let $\boldsymbol{g}^{\boldsymbol{x}} = \prod_i g_i^{x_i}$. We let $[n]$ denote the vector $(1, 2, \ldots, n)$ and let $[m, n]$ denote the vector $(m, m+1, \ldots, n)$. Similiarly we let $\{v_i\}_{i \in [n]}$ denote the vector $(v_1, v_2, \ldots, v_n)$.

Notation 2 (Evaluation-Based Polynomial Representation). Throughout our work, we represent various degree n polynomials as vectors of $n + 1$ *evaluations* over a predefined set of points rather than as vectors of coefficients. For a polynomial p we let $\boldsymbol{p}$ denote its evaluation-based vector representation. We treat $\boldsymbol{p}$ as a vector or a polynomial representation interchangably depending on context. For notational conciseness, we let $\boldsymbol{p}(x)$ denote the evaluation $p(x)$. Similiarly for indeterminate X, we let $\boldsymbol{p}(X)$ denote polynomial $p(X)$.

Definition 5 (Vanishing Polynomial [33]). *Consider a finite field $\mathbb{F}$ and a subset $S \subseteq \mathbb{F}$. Let v_S denote the unique, non-zero, monic, polynomial of degree $|S|$ that is zero at every point on S. If S is a multiplicative subgroup, then $v_S(X) = X^{|S|} - 1$, which can be computed in $O(\log |S|)$ field operations.*

Definition 6 (Formal Derivative of the Vanishing Polynomial [15,33]). *Given a finite field $\mathbb{F}$ and a subset $S \subseteq \mathbb{F}$, we define the polynomial*

$$u_S(X,Y) = \frac{v_S(X) - v_S(Y)}{X - Y},$$

where $X, Y \in \mathbb{F}$. Note that u_S is a bivariate polynomial with degree $|S| - 1$ in each variable, because $X - Y$ divides $v_H(X) - v_H(Y)$.

We can compute $u_H(X,Y)$ as follows: If $X \neq Y$ then the term $(v_S(X) - v_S(Y))/(X - Y)$ can be computed directly. If, on the other hand, $X = Y$, then Chiesa et al. [33] show that $u_S(X,X) = |S|X^{|S|-1}$. This property suggests that for all $X, Y \in S$, $u_S(X,Y) \neq 0$ when $X = Y$ and $u_S(X,Y) = 0$ otherwise.

Lemma 1 (Polynomial Decomposition [64]). *Consider degree d polynomial $p(X)$ and arbitrary evaluation point $u \in \mathbb{F}$. Then there exists degree $d-1$ polynomial $q(X)$ such that*

$$\frac{p(X) - p(u)}{X - u} = q(X)$$

Definition 7 (Lagrange Basis). *For evaluation points $x_1, \ldots, x_k$ the Lagrange basis is defined as $\ell(x) = \langle \ell_0(x), \ldots, \ell_k(x) \rangle^\top$ where*

$$\ell_j(x) := \prod_{0 \leq m \leq k, m \neq j} \frac{x - x_m}{x_j - x_m}.$$

Suppose a polynomial P of degree k is defined by points $(x_0, y_0), \ldots, (x_k, y_k)$ Then

$$P(x) = \sum_{j=0}^{k} y_j \ell_j(x).$$

3.3 Cryptographic Assumptions

In order to achieve zero-knowledge we require additional cryptographic machinery overlayed on top of the core interaction. We define our cryptographic assumptions below.

Assumption 1 (Discrete Logarithm Relation [29]). Consider group $\mathbb{G}$. The discrete logarithm assumptoin holds for $\mathbb{G}$ if for all PPT adversaries $\mathcal{A}$ and for all $n \geq 2$

$$\Pr\left[\begin{array}{l} \exists\, a_i \neq 0, \\ \prod_{i=1}^{n} g_i^{a_i} = 1 \end{array} \middle| \begin{array}{l} g_1, \ldots, g_n \xleftarrow{\$} \mathbb{G}, \\ a_1, \ldots, a_n \in \mathbb{Z}_p \leftarrow \mathcal{A}(\mathbb{G}, g_1, \ldots, g_n) \end{array}\right] = \mathsf{negl}(\lambda).$$

Assumption 2 (n-Strong Diffie-Hellman (n-SDH)). Consider group $\mathbb{G}$ of prime order $n = O(2^{\lambda})$ and let $\mathbb{F} = \mathbb{Z}_p^*$. The n-SDH assumption [25] holds for $\mathbb{G}$ if for all PPT adversaries $\mathcal{A}$

$$\Pr\left[\begin{array}{l|l} & g \xleftarrow{\$} \mathbb{G}, \\ c \neq -s, & s \xleftarrow{\$} \mathbb{F}, \\ C = g^{\frac{1}{s+c}} & \sigma = ((\mathbb{F}, \mathbb{G}, \mathbb{G}_{\mathsf{T}}, e), g, g^{s}, \ldots, g^{s^{n}}), \\ & (c, C) \leftarrow \mathcal{A}(\sigma) \end{array}\right] = \mathsf{negl}(\lambda).$$

Assumption 3 (n-Bilinear Strong Diffie-Hellman (n-BSDH)). Consider bilinear group generator $\mathcal{G}$. The n-BSDH assumption [50] holds for $\mathcal{G}$ if for all PPT adversaries $\mathcal{A}$

$$\Pr\left[\begin{array}{l|l} & (\mathbb{F}, \mathbb{G}, \mathbb{G}_{\mathsf{T}}, e) \leftarrow \mathcal{G}(\lambda), \\ c \neq -s, & g \xleftarrow{\$} \mathbb{G}, \\ C = e(g, g)^{\frac{1}{s+c}} & s \xleftarrow{\$} \mathbb{F}, \\ & \sigma = ((\mathbb{F}, \mathbb{G}, \mathbb{G}_{\mathsf{T}}, e), g, g^{s}, \ldots, g^{s^{n}}), \\ & (c, C) \leftarrow \mathcal{A}(\sigma) \end{array}\right] = \mathsf{negl}(\lambda).$$

Assumption 4 (n-Extended Power Knowledge of Exponent (n-EPKE) [81]). Consider bilinear group generator $\mathcal{G}$. The n-Extended Power Knowledge of Exponent holds for $\mathcal{G}$ if for any PPT adversary $\mathcal{A}$ there exists a PPT extractor $\mathcal{E}$ such that

$$\Pr\left[\begin{array}{l|l} & (\mathbb{F}, \mathbb{G}, \mathbb{G}_{\mathsf{T}}, e) \leftarrow \mathcal{G}(\lambda), \\ & \alpha, s, t \xleftarrow{\$} \mathbb{F}, g \xleftarrow{\$} \mathbb{G}, \\ e(A, g^{\alpha}) = e(A', g), & \boldsymbol{u} = (g, g^{s}, \ldots, g^{s^{n}}, g^{t}), \\ A = \left(\prod_{i=0}^{n} g^{s^{i} \cdot a_i}\right) \cdot g^{t \cdot b} & \boldsymbol{v} = (g^{\alpha}, g^{\alpha s}, \ldots, g^{\alpha s^{n}}, g^{\alpha t}), \\ & \sigma = ((\mathbb{F}, \mathbb{G}, \mathbb{G}_{\mathsf{T}}, e), \boldsymbol{u}, \boldsymbol{v}), \\ & (A, A') \leftarrow \mathcal{A}(\lambda, \sigma, z; \rho), \\ & (a_0, \ldots, a_n, b) \leftarrow \mathcal{E}(\lambda, \sigma, z; \rho) \end{array}\right] = 1 - \mathsf{negl}(\lambda)$$

for any benign auxiliary input $z \in \{0,1\}^{\mathsf{poly}(\lambda)}$, and randomness ρ. In this setting we consider input z benign if it is generated independently of α.

Corollary 1 (n-EPKE for a Linearly Independent Basis). *Consider a linearly independent basis of polynomials of degree up to n: $p_0(X), \ldots, p_n(X)$. Then, if the n-EPKE assumption holds for $\mathcal{G}$, then for any PPT adversary $\mathcal{A}$ there exists PPT extractor $\mathcal{E}$ such that*

$$\Pr\left[\begin{array}{l|l} & (\mathbb{F}, \mathbb{G}, \mathbb{G}_{\mathsf{T}}, e) \leftarrow \mathcal{G}(\lambda), \\ & \alpha, s, t \xleftarrow{\$} \mathbb{F}, g \xleftarrow{\$} \mathbb{G}, \\ e(A, g^{\alpha}) = e(A', g), & \boldsymbol{u} = (g, g^{p_0(s)}, \ldots, g^{p_n(s)}, g^{t}), \\ A = \left(\prod_{i=0}^{n} g^{p_i(s) \cdot a_i}\right) \cdot g^{t \cdot b} & \boldsymbol{v} = (g^{\alpha}, g^{\alpha p_0(s)}, \ldots, g^{\alpha p_n(s)}, g^{\alpha t}), \\ & \sigma = ((\mathbb{F}, H, \mathbb{G}, \mathbb{G}_{\mathsf{T}}, e), \boldsymbol{u}, \boldsymbol{v}), \\ & (A, A') \leftarrow \mathcal{A}(\lambda, \sigma, z; \rho), \\ & (a_0, \ldots, a_n, b) \leftarrow \mathcal{E}(\lambda, \sigma, z; \rho) \end{array}\right] = 1 - \mathsf{negl}(\lambda)$$

for any auxiliary input $z \in \{0,1\}^{\mathsf{poly}(\lambda)}$ generated independently of α, and randomness ρ.

Proof. Informally, Corollary 1 follows because any linearly independent basis is a linear combination of the standard monomial basis and visa-versa. A formal proof is provided in supplementary section D.1 □

Remark 1 (Benign Auxiliary Distributions). Boyle et al. [28] and Bitansky et al. [23] show the impossibility of knowledge assumptions with arbitrary auxiliary inputs. To circumvent this issue, we must assume that each of our subprotocols relying on the q-EPKE assumption only receive benign auxiliary inputs. When composing subprotocols we are careful not to introduce any new terms that could break this requirement. Thus when using our final argument system as a subroutine in larger protocols, knowledge-soundness holds so long as the auxiliarly input is sampled benignly.

4 Auxiliary Argument Systems

In this section we define and construct extractible vector commitments (§4.1), an argument system for polynomial evaluation (§4.2), an argument system for inner-product (§4.3), and an argument system for inner-product over the Lagrange basis (§4.4). For simplicity our constructions utilize symmetric bilinear pairings; however, in practice asymmetric bilinear pairings can be used. While we utilize the Lagrange basis, we stress that our auxiliary constructions and corresponding proofs work with any basis by Corollary 1.

4.1 Extractible Vector Commitments

Definition 8 (Vector Commitments). *A vector commitment scheme over $\mathbb{F}^n$ has the following structure*

- *$\mathcal{G}(\lambda, n) \to \mathsf{pp}$: Takes input security parameter λ size bound n. Outputs public parameters* pp.
- $\mathsf{com}(\mathsf{pp}; \boldsymbol{v}; r) \to c$. *Takes input public parameters* pp*, vector $\boldsymbol{v} \in \mathbb{F}^n$ and randomness r. Outputs commitment c.*
- $\mathsf{checkcom}(\mathsf{pp}; c) \to \{0,1\}$. *Takes input public parameters* pp*, and commitment c. Outputs 1 is c is well-formed, 0 otherwise.*

A vector commitment scheme $(\mathcal{G}, \mathsf{com})$ over $\mathbb{F}^n$, with randomness space R*, is said to be computationally binding if for any PPT adversary $\mathcal{A}$*

$$\Pr\left[\begin{matrix}\mathsf{com}(\boldsymbol{v_0}; r_0) = \mathsf{com}(\boldsymbol{v_1}; r_1), \\ \boldsymbol{v_0} \neq \boldsymbol{v_1}\end{matrix}\middle| \begin{matrix}\mathsf{pp} \leftarrow \mathcal{G}(\lambda, n), \\ \boldsymbol{v_0}, \boldsymbol{v_1}, r_0, r_1 \leftarrow \mathcal{A}(\mathsf{pp})\end{matrix}\right] = \mathsf{negl}(\lambda)$$

and is said to be unconditionally hiding if for any PPT adversary $\mathcal{A}$

$$\Pr\left[b = b' \middle| \begin{matrix}\mathsf{pp} \leftarrow \mathcal{G}(\lambda, n), (\boldsymbol{v_0}, \boldsymbol{v_1}) \in \mathbb{F}^n \leftarrow \mathcal{A}(\mathsf{pp}), \\ b \xleftarrow{\$} \{0,1\},\ r \xleftarrow{\$} \mathsf{R}, c \leftarrow \mathsf{com}(\boldsymbol{v_b}; r), b' \leftarrow \mathcal{A}(\mathsf{pp}, c)\end{matrix}\right] = \frac{1}{2}$$

Definition 9 (Extractibility). *We call a vector commitment scheme extractible if for any probabilistic polynomial time adversary $\mathcal{A}$, there exists a probabilistic polynomial time extractor $\mathcal{E}$ such that*

$$\Pr\left[\begin{array}{l}\mathsf{checkcom}(\mathsf{pp};c)=1,\\ c\neq\mathsf{com}(\mathsf{pp};v;r)\end{array}\middle|\begin{array}{l}\mathsf{pp}\leftarrow\mathcal{G}(\lambda,n),\\ c\leftarrow\mathcal{A}(\mathsf{pp},z;\rho),\\ (v,r)\leftarrow\mathcal{E}(\mathsf{pp},z;\rho)\end{array}\right]=\mathsf{negl}(\lambda)$$

for any benign auxiliary input $z\in\{0,1\}^{\mathsf{poly}(\lambda)}$, and randomness ρ.

Definition 10 (Additively Homomorphic Commitment Scheme). *Consider a vector commitment scheme $(\mathcal{G},\mathsf{com})$ over $\mathbb{F}^n$, with abelian groups $(\mathsf{C},+_\mathsf{C})$, $(\mathsf{R},+_\mathsf{R})$ for the commitment space, and randomness space respectively. The commitment scheme is said to be homomorphic if for all $\boldsymbol{v_1},\boldsymbol{v_2}\in\mathbb{F}^n$ and $\boldsymbol{r}_1,\boldsymbol{r}_2\in\mathsf{R}$, we have*

$$\mathsf{com}(\boldsymbol{v_1};r_1)+_\mathsf{C}\mathsf{com}(\boldsymbol{v_2};r_2)=\mathsf{com}(\boldsymbol{v_1}+\boldsymbol{v_2};r_1+_\mathsf{R}r_2).$$

Construction 1 (Structured Polynomial Commitments). We design a scheme to commit to a vector of evaluations representing a polynomial. Similiar to that of Kate et al. [57], our commitment scheme involves evaluating a polynomial on a secret point "in the exponent" using a structured reference string. Using ideas from Zhang et al. [81] (and Chiesa et al. [33]), we can achieve an extractible commitment scheme by enforcing that the prover provides an auxiliary "shifted" commitment, which ensures that the commitments were formed by using a linear combination of terms in the SRS. Our key insight is that we can commit to evaluation-based representations of polynomials by utilizing the Lagrange basis as a part of the structured reference string. We define generator $\mathcal{G}$, com, and $\mathsf{checkcom}$ as follows for vectors of size $\mathbb{Z}_p^n$:

$\underline{\mathsf{Generator}(\lambda,n)\to\mathsf{pp}}$:

1. Generate two groups $\mathbb{G}$ and $\mathbb{G}_\mathsf{T}$ of prime order p (with $p\geq 2^\lambda$) such that there exists a symmetric bilinear pairing $e:\mathbb{G}\times\mathbb{G}\to\mathbb{G}_\mathsf{T}$ where the $(n-1)$-SDH and $(n-1)$-EPKE assumptions hold.
2. Let $H\subseteq\mathbb{F}$ be such that $|H|=n$ and let $\ell_1,\ldots,\ell_n$ be the Lagrange basis basis over evaluation points H.
3. Randomly sample generator $g\in\mathbb{G}$ and $\alpha,s\overset{\$}{\leftarrow}\mathbb{F}$.
4. Compute commitment keys $\boldsymbol{u}=(g^{\ell_1(s)},\ldots,g^{\ell_n(s)})$ and $\boldsymbol{v}=(g^{\alpha\ell_1(s)},\ldots,g^{\alpha\ell_n(s)})$.
5. Sample $h\overset{\$}{\leftarrow}\mathbb{G}$ and output public parameters $\mathsf{pp}=(\mathbb{G},H,\boldsymbol{u},\boldsymbol{v},g,g^\alpha,h,h^\alpha)$.

$\underline{\mathsf{com}(\mathsf{pp};\boldsymbol{p}\in\mathbb{F}^n,r\in\mathbb{F})\to P\in\mathbb{G}^n}$: Interpret $\boldsymbol{p}$ as a vector of polynomial evaluations over H. Output $P=(g^{\boldsymbol{p}(s)}\cdot h^r,g^{\alpha\cdot\boldsymbol{p}(s)}\cdot h^{\alpha\cdot r})$.

$\underline{\mathsf{checkcom}(\mathsf{pp};P\in\mathbb{G}^2)\to\{0,1\}}$: Parse P as (P_1,P_2) and check $e(P_1,g^\alpha)=e(P_2,g)$.

Lemma 2 (Structured Polynomial Commitments). *Construction 1 is a homomorphic vector commitment scheme that satisfies unconditional hiding, computational binding, and extractibility. For polynomials defined by n evaluation points, $\mathcal{G}$ takes time $O(n)$, com takes time $O(n)$, $\mathsf{checkcom}$ takes time $O(1)$.*

Proof. Informally, hiding follows from the blinding terms, binding follows from the $(n-1)$-SDH assumption, and extractibility holds from the $(n-1)$-EPKE assumption. Formally, we prove Lemma 2 in supplementary section D.2. □

4.2 An Argument System for Polynomial Evaluation

In our argument system for general computation, to prove desired properties about the committed witness and subsequent messages (all represented as polynomials), the prover is required to evaluate these polynomials (represented as vectors of evaluations) at challenge points. We modify the polynomial commitment scheme by Zhang et al. [81], which in turn is based on the scheme by Papamanthou et al. [64]. To efficiently evaluate polynomials based on their evaluation representations, we create a structured key which utilizes the Lagrange basis.

Definition 11 (Polynomial Evaluation Relation). *Consider group $\mathbb{G}$ of order q and let $\mathbb{F} = \mathbb{Z}_q$. The polynomial evaluation relation ($\mathcal{R}_{\mathsf{POLY}}$), with respect to vector commitment scheme com, defined over subset $H \subseteq \mathbb{F}$ consists of commitments $P \in \mathbb{G}^2$, $Y \in \mathbb{G}$, evaluation point u, and evaluation result y. A vector $\boldsymbol{p}$ and scalar y satisfies an $\mathcal{R}_{\mathsf{POLY}}$ instance if $\boldsymbol{p}(u) = y$, $Y = \mathsf{com}(y)$, and $P = \mathsf{com}(\boldsymbol{p})$.*

Construction 2 (Argument System for Polynomial Evaluation). We define an argument system for polynomial evaluation (Definition 11) with respect to the structured polynomial commitment scheme (Definition 1)

$\underline{\mathsf{Generator}(\lambda, n) \to \mathsf{pp}}$:

1. Generate two groups $\mathbb{G}$ and $\mathbb{G}_{\mathsf{T}}$ of prime order p (with $p \geq 2^{\lambda}$) such that there exists a symmetric bilinear pairing $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_{\mathsf{T}}$ where the $(n-1)$-BSDH and $(n-1)$-EPKE assumptions hold.
2. Let $H \subseteq \mathbb{F}$ be such that $|H| = n$ and let $\ell_1, \ldots, \ell_n$ be the Lagrange basis over evaluation points H.
3. Randomly sample generator $g \in \mathbb{G}$ and $\alpha, s \xleftarrow{\$} \mathbb{F}$.
4. Compute vectors $\boldsymbol{u} = (g^{\ell_1(s)}, \ldots, g^{\ell_n(s)})$ and $\boldsymbol{v} = (g^{\alpha\ell_1(s)}, \ldots, g^{\alpha\ell_n(s)})$
5. Sample random $h \xleftarrow{\$} \mathbb{G}$ and define the polynomial commitment key $\mathsf{ck}_p = (\mathbb{G}, H, \boldsymbol{u}, \boldsymbol{v}, h, h^{\alpha})$.
6. Sample $\beta \xleftarrow{\$} \mathbb{F}$ and define scalar commitment key $\mathsf{ck}_y = (g, h, g^{\beta}, h^{\beta})$
7. Output public parameters $\mathsf{pp} = ((\mathbb{G}, \mathbb{G}_T, e), \mathsf{ck}_p, \mathsf{ck}_y, g^{\alpha}, g^{s}, g^{\alpha s})$.

$\underline{\langle \mathsf{Prover}, \mathsf{Verifier} \rangle}$:

The prover and verifier are provided with statement $(P \in \mathbb{G}^2, Y \in \mathbb{G}^2, u \in \mathbb{F})$. The prover is additionally provided with witness $\boldsymbol{p} \in \mathbb{F}^n, y, r_p, r_y \in \mathbb{F}$. The prover is tasked with proving that $Y = (g^y \cdot h^{r_y}, g^{\alpha} \cdot h^{\alpha r_y})$, $P = (g^{\boldsymbol{p}(s)} \cdot h^{r_p}, g^{\alpha \boldsymbol{p}(s)} \cdot h^{\alpha r_p})$, and that $y = p(u)$.

1. Using Lemma 1, the prover computes evaluations of polynomial $\boldsymbol{q}$ over H where $\boldsymbol{q}(X) = (\boldsymbol{p}(X) - y)/(X - u)$. Next the prover samples $r_q \overset{\$}{\leftarrow} \mathbb{F}$ and commits to $\boldsymbol{q}$, and the randomness:
$$Q = \mathsf{com}(\boldsymbol{q}; r_q) = (g^{\boldsymbol{q}(s)} \cdot h^{r_q}, g^{\alpha \boldsymbol{q}(s)} \cdot h^{\alpha r_q})$$
$$R = (g^{r_p - r_y - r_q(s-u)}, g^{\alpha \cdot (r_p - r_y - r_q(s-u))}).$$
2. The verifier parses commitments P, Q, R, Y as (P_1, P_2), (Q_1, Q_2), (R_1, R_2) and (Y_1, Y_2) respectively and checks that they are well formed using check-com. Next the verifier checks that the prover was able to compute a valid commitment to $\boldsymbol{q}$: $e(P_1/Y_1, g) \overset{?}{=} e(Q_1, g^{s-u})e(R_1, h)$.

Theorem 1 (Polynomial Evaluation Argument). *Construction 2 satisfies completeness, knowledge soundness, and perfect zero-knowledge. For polynomials defined over n evaluations, the polynomial evaluation argument features an $O(n)$ generator, $O(n)$ prover, and an $O(1)$ verifier.*

Proof. Completeness follows by observation. Informally, knowledge soundness follows from the $(n-1)$-BSDH, and $(n-1)$-EPKE assumptions. Perfect zero-knowledge follows due to the blinding terms. Formally we prove Theorem 1 in supplementary section D.3. □

4.3 An Argument System for Inner-Product

We utilize the argument system for generalized inner-product from Bünz et al. [31], specifically instantiated with the Pedersen-like vector commitment scheme and modified to support zero-knowledge. In Section 4.4, we extend this argument system to handle evaluation based polynomial commitments (Construction 1). Later in Section 6 we show how to achieve a constant-time verifier using proof composition.

Definition 12 (The Inner-Product Relation [31]). *Consider group $\mathbb{G}$ of order p and let $\mathbb{F} = \mathbb{Z}_p$. The inner-product relation ($\mathcal{R}_{\mathsf{IP}}$), characterized by commitment scheme com, consists of hiding and binding commitments $A, B, C \in \mathbb{G}$, and scalar $r \in \mathbb{F} \setminus \{0\}$. Vectors $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{F}^n$ and scalar $c \in \mathbb{F}$ satisfy an $\mathcal{R}_{\mathsf{IP}}$ instance if $c = \boldsymbol{a'} \cdot \boldsymbol{b}$, where $\{\boldsymbol{a}'_i = \boldsymbol{a}_i \cdot r^i\}_{i=0}^{n-1}$, and A, B and C are commitments to $\boldsymbol{a}$, $\boldsymbol{b}$, and c respectively.*

Construction 3 (Argument System for Inner-Product [31]). An argument system for the inner-product relation allows a prover to show that for commitments $A, B, C \in \mathbb{G}$ and scalar $r \in \mathbb{F} \setminus \{0\}$ they know $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{F}^n$ and scalar $c \in \mathbb{F}$ such that A, B, and C are commitments to $\boldsymbol{a}, \boldsymbol{b}$, and c respectively, and that $c = (\boldsymbol{a} \circ \boldsymbol{r}) \cdot \boldsymbol{b}$ where $\boldsymbol{r} = (r^0, r^1, \ldots, r^{n-1})$.

Bünz et al. [31] present a *generalized inner-product argument* which allows a prover to prove the inner-product relation over any binding commitment scheme that is doubly homomorphic (i.e. homomorphic in both the message space and the key space). They additionally show how to achieve an $O(\log n)$ verifier by utilizing a structured reference string and polynomial commitments. We derive an argument system for $\mathcal{R}_{\mathsf{IP}}$ by applying the following commitment scheme to the generalized inner-product argument:

$$\mathsf{com}(\mathsf{ck}; \boldsymbol{a}, \boldsymbol{b}, c; r_a, r_b, r_c) = (\boldsymbol{w}^{\boldsymbol{a}} \cdot h^{r_a}, \boldsymbol{w}^{\boldsymbol{b}} \cdot h^{r_b}, g^c \cdot h^{r_c}), \tag{2}$$

where the commitment key $\mathsf{ck} = (\boldsymbol{w}, (g, h))$ is created by the generator as described below. We further modify the generalized inner-product argument to be zero-knowledge using standard techniques.

$\underline{\mathsf{Generator}(\lambda, n) \to \mathsf{pp}}$:

1. Generate two groups $\mathbb{G}$ and $\mathbb{G}_{\mathsf{T}}$ of prime order p (with $p \geq 2^\lambda$) such that there exists a symmetric bilinear pairing $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_{\mathsf{T}}$ where the $(n-1)$-SDH, and the $(n-1)$-EPKE assumptions hold.
2. Sample generator $g \xleftarrow{\$} \mathbb{G}$, secret $s \xleftarrow{\$} \mathbb{F}$ and define commitment key $\boldsymbol{w} = (g, g^s, \ldots, g^{s^{n-1}})$.
3. Sample $h \xleftarrow{\$} \mathbb{G}$ and output public parameters $\mathsf{pp} = (e, \boldsymbol{w}, h)$.

$\underline{\langle \mathsf{Prover}, \mathsf{Verifier} \rangle}$:

Both the prover and verifier are provided the statement consisting of commitments A, B, and C and scalar r. The prover is additionally provided witness $(\boldsymbol{a}, \boldsymbol{b}, c, r_a, r_b, r_c)$

1. Initially the prover computes $\boldsymbol{r} = (r^0, r^1, \ldots, r^{n-1})$, rescales the commitment key $\boldsymbol{v} = \boldsymbol{w}^{\boldsymbol{r}^{-1}}$, and rescales the corresponding witness vector $\boldsymbol{a} \leftarrow \boldsymbol{a} \circ \boldsymbol{r}$.
2. When $n \geq 2$, the prover defines $\boldsymbol{a_1}$ and $\boldsymbol{a_2}$ to be the first and second half of vector $\boldsymbol{a}$ (similarly for $\boldsymbol{b}$, $\boldsymbol{v}$, and $\boldsymbol{w}$). Next the prover samples randomness $r_{\mathsf{La}}, r_{\mathsf{Ra}}, r_{\mathsf{Lb}}, r_{\mathsf{Rb}}, r_{\mathsf{Lc}}, r_{\mathsf{Rc}} \xleftarrow{\$} \mathbb{F}$ and sets
$$\begin{array}{lll} A_L = h^{r_{\mathsf{La}}} \cdot \boldsymbol{v_1}^{\boldsymbol{a_2}} & B_L = h^{r_{\mathsf{Lb}}} \cdot \boldsymbol{w_2}^{\boldsymbol{b_1}} & C_L = h^{r_{\mathsf{Lc}}} \cdot g^{\boldsymbol{a_2} \cdot \boldsymbol{b_1}} \\ A_R = h^{r_{\mathsf{Ra}}} \cdot \boldsymbol{v_2}^{\boldsymbol{a_1}} & B_R = h^{r_{\mathsf{Rb}}} \cdot \boldsymbol{w_1}^{\boldsymbol{b_2}} & C_R = h^{r_{\mathsf{Rc}}} \cdot g^{\boldsymbol{a_1} \cdot \boldsymbol{b_2}} \end{array}$$
and sends these values to the verifier.
3. The verifier samples $x \xleftarrow{\$} \mathbb{F}$ and sends x to the prover.
4. The prover and verifier each set
$$A' = A_L^x \cdot A \cdot A_R^{x^{-1}} \qquad B' = B_L^x \cdot B \cdot B_R^{x^{-1}} \qquad C' = C_L^x \cdot C \cdot C_R^{x^{-1}}$$
5. The prover additionally folds the commitment keys
$$\boldsymbol{v'} = \boldsymbol{v_1} \circ \boldsymbol{v_2}^{x^{-1}} \qquad\qquad \boldsymbol{w'} = \boldsymbol{w_1} \circ \boldsymbol{w_2}^{x},$$
folds the witness vectors and associated randomness
$$\boldsymbol{a'} = \boldsymbol{a_2} \cdot x + \boldsymbol{a_1} \qquad\qquad \boldsymbol{b'} = \boldsymbol{b_2} \cdot x^{-1} + \boldsymbol{b_1}$$

$$r'_a = r_{\mathsf{La}} \cdot x + r_a + r_{\mathsf{Ra}} \cdot x^{-1} \qquad r'_b = r_{\mathsf{Lb}} \cdot x + r_b + r_{\mathsf{Rb}} \cdot x^{-1},$$

and folds the claimed product and associated randomness

$$c' = (\boldsymbol{a_2} \cdot \boldsymbol{b_1}) \cdot x + c + (\boldsymbol{a_1} \cdot \boldsymbol{b_2}) \cdot x^{-1} \qquad r'_c = r_{\mathsf{Lc}} \cdot x + r_c + r_{\mathsf{Rc}} \cdot x^{-1}.$$

6. Next if $n \geq 2$ the prover and verifier recurse back to step 2 with statement (A', B', C'), witness $(\boldsymbol{a}', \boldsymbol{b}', c', r'_a, r'_b, r'_c)$ and commitment keys $(\boldsymbol{v}', \boldsymbol{w}')$. Otherwise the prover and verifier continue to step 7.
7. In the final round when $n = 1$, the prover sends the final commitment keys $v, w \in \mathbb{G}$. The prover first proves to the verifier that (v, w) have been computed correctly (subprotocol below). Next the prover proves that the product relation holds for commitments A', B', C' with respect to commitment keys v, w (subprotocol below).

In the final round the verifier must check that the commitment keys v and w have been computed correctly. We continue to follow the general approach presented by Bünz et al. [31]. Suppose there were a total of ℓ rounds. Let $x_0, \ldots, x_{\ell-1}$ denote the randomness sent by the verifier in each round. We first define

$$f_v(X) = \prod_{j=0}^{\ell-1} \left(x^{-1}_{(\ell-j)} + (r^{-1}X)^{2^j} \right) \qquad f_w(X) = \prod_{j=0}^{\ell-1} \left(x_{(\ell-1-j)} + X^{2^j} \right).$$

When w and v are computed correctly we have that $v = g^{f_v(s)}$ and $w = g^{f_w(s)}$ [31, Proposition B.1]. Given this observation, the verifier checks the commitment keys by engaging in the following procedure:

Subprotocol to check (v, w):

1. In the setup phase the generator additionally samples $\sigma \xleftarrow{\$} \mathbb{F}$ and outputs keys $\boldsymbol{t} = (g^{\alpha}, g^{\alpha s}, \ldots, g^{\alpha s^{n-1}})$
2. The prover begins the subprotocol by sending claimed evaluations $v, w \in \mathbb{G}$ along with terms $v' = g^{\alpha f_v(s)}$, and $w' = g^{\alpha f_w(s)}$.
3. The verifier responds with challenge $z \xleftarrow{\$} \mathbb{F}$ and computes $Y_v = g^{f_v(z)}$, and $Y_w = g^{f_w(z)}$.
4. Note that (v, v') and (w, w') can be treated as extractible polynomial commitments with respect to the *standard monomial basis* rather than the Lagrange basis (Construction 2). We note that in this setting our polynomial evaluation argument can be viewed as a simplified version of that of Zhang et al. [81]. To check the validity of v, the prover and verifier treat $V = (v, v')$ as a polynomial commitment and engage in an extractible polynomial evaluation argument over the statement (V, Y_v, z) and the provided SRS. Note that the verifier does not need to check the validity of commitment Y_v. Similiarly, to check the validity of w, the prover and verifier treat $W = (w, w')$ as a polynomial commitment and engage in an extractible polynomial evaluation argument over the statement (W, Y_w, z) and the provided SRS.

Additionally, given commitments $A = v^a h^{r_a}$, $B = w^b h^{r_b}$, and $C = g^c h^{r_c}$ the verifier must check $a \cdot b = c$. We cannot use a textbook product argument due to the fact that A, B and C are committed to under different keys. To handle this setting, we use a simplified variant of a product argument presented by Bünz et al. [29]. For completeness we present this protocol in supplementary section A.

Theorem 2 (Inner-Product Argument). *Construction 3 is an argument system for $\mathcal{R}_{\mathsf{IP}}$ that satisfies completeness, knowledge-soundness, and honest-verifier zero-knowledge. For vectors of size n construction 3 features an $O(n)$ generator, $O(n)$ prover, and an $O(\log n)$ verifier.*

Proof. Completeness follows by the completeness of the generalized inner-product argument [26, Theorem 5.4]. Informally, binding of the commitment scheme follows by the $(n-1)$-SDH assumption, and thus knowledge-soundness of the main argument holds by an argument similiar to Bünz et al. [31]. The knowledge soundness of the underlying polynomial commitment scheme holds by the $(n-1)$-EPKE assumption. Zero-knowledge holds due to the blinding terms. We formally prove Theorem 2 in supplementary section D.4. □

4.4 Extending the Inner-Product Argument for the Lagrange Basis

Construction 4 (Argument System for Inner-Product for the Lagrange Basis). For generator $g \in \mathbb{G}$ and random $s \xleftarrow{\$} \mathbb{F}$ recall from Construction 3 that the commitment key has the form $\boldsymbol{w} = (g, g^{s^1}, \ldots, g^{s^{n-1}})$. Construction 3 allows a prover to prove that for vectors $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{F}^n$ and $c \in \mathbb{F}$ that $\boldsymbol{a} \cdot \boldsymbol{b} = c$ specifically when the commitments to $\boldsymbol{a}$, $\boldsymbol{b}$, are of the form

$$A = \boldsymbol{w}^{\boldsymbol{a}} \cdot h^{r_a} \qquad\qquad B = \boldsymbol{w}^{\boldsymbol{b}} \cdot h^{r_b}$$

However, as we show in section 5, we are particularly interested in proving the inner-product of vectors "under" evaluation-based polynomial commitments (Construction 1). In more detail, for random $g \in \mathbb{G}$ and $t \xleftarrow{\$} \mathbb{F}$, and for subset $H \subseteq \mathbb{F}$, consider the vector $\boldsymbol{l} = (g^{\ell_0(t)}, g^{\ell_2(t)}, \ldots, g^{\ell_{n-1}(t)})$, where $\ell_1, \ldots, \ell_n$ are the lagrange basis over evaluation points H (Definition 7). We would like to prove that $\boldsymbol{a} \cdot \boldsymbol{b} = c$ where the commitments to $\boldsymbol{a}$ and $\boldsymbol{b}$ are

$$A' = \boldsymbol{l}^{\boldsymbol{a}} \cdot h^{r'_a} \qquad\qquad B' = \boldsymbol{l}^{\boldsymbol{b}} \cdot h^{r'_b}$$

for randomness r'_a, r'_b. While it is unclear how to directly reason about A', and B' under construction 3, we can use an approach presented by Parno et al. [65] to check that A' and A (similiarly B' and B) commit to the same vector.

<u>$\mathsf{Generator}(\lambda, n) \to \mathsf{pp}$:</u>

1. Generate two groups $\mathbb{G}$ and $\mathbb{G}_{\mathsf{T}}$ of prime order p (with $p \geq 2^\lambda$) such that there exists a symmetric bilinear pairing $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_{\mathsf{T}}$ where the $(n-1)$-SDH, and $(n-1)$-EPKE assumptions hold.

2. Run the generator for the inner-product argument system (Construction 3). In particular, randomly sample generator $g \in \mathbb{G}$ and $s \in \mathbb{F}$ and define inner-product commitment keys over powers of s: $\boldsymbol{w} = (g, g^s, \ldots, g^{s^{n-1}})$.
3. Pick randomness commitment key $h \xleftarrow{\$} \mathbb{G}$.
4. Run the generator for polynomial commitments (Construction 1): In particular, pick random $t, \alpha \xleftarrow{\$} \mathbb{F}$ and create polynomial commitment keys $\boldsymbol{l} = (g^{\ell_0(t)}, g^{\ell_1(t)}, \ldots, g^{\ell_{n-1}(t)})$, and $\boldsymbol{l}' = (g^{\alpha\ell_0(t)}, g^{\alpha\ell_1(t)}, \ldots, g^{\alpha\ell_{n-1}(t)})$.
5. Pick binding randomness $\gamma \xleftarrow{\$} \mathbb{F}$ and create binding keys
$$\boldsymbol{t} = (\boldsymbol{w} \circ \boldsymbol{l})^\gamma = (g^{\gamma(s^0+\ell_0(t))}, g^{\gamma(s^1+\ell_1(t))}, \ldots, g^{\gamma(s^{n-1}+\ell_{n-1}(t))})$$
6. Output public parameters $\mathsf{pp} = (e, \boldsymbol{w}, \boldsymbol{l}, \boldsymbol{l}', \boldsymbol{t}, (g, h), (g^\alpha, h^\alpha), (g^\gamma, h^\gamma))$.

$\langle \mathsf{Prover}, \mathsf{Verifier} \rangle$:
The prover and verifier are provided with the statement consisting of commitments A', B', C and scalar r. The prover is additionally provided witness $(\boldsymbol{a}, \boldsymbol{b}, c, r'_a, r'_b, r_c)$.

1. If computed correctly A' and B' are commitments to $\boldsymbol{a}$ and $\boldsymbol{b}$ respectively under the Lagrange-basis commitment key. That is $A' = (\boldsymbol{l}^{\boldsymbol{a}} \cdot h^{r'_a}, \boldsymbol{l}'^{\boldsymbol{a}} \cdot h^{\alpha r'_a})$ and $B' = (\boldsymbol{l}^{\boldsymbol{b}} \cdot h^{r'_b}, \boldsymbol{l}'^{\boldsymbol{b}} \cdot h^{\alpha r'_b})$. The prover samples $r_a, r_b \xleftarrow{\$} \mathbb{F}$ and sends to the verifier commitments $A, B \in \mathbb{G}$, where A is the claimed commitment to $\boldsymbol{a}$ under inner-product commitment key $\boldsymbol{w}$, and B is the claimed commitment to $\boldsymbol{b}$ under inner-commitment key $\boldsymbol{w}$. That is $A = \boldsymbol{w}^{\boldsymbol{a}} \cdot h^{r_a}$ and $B = \boldsymbol{w}^{\boldsymbol{b}} \cdot h^{r_b}$.
2. To prove that A' and A commit to the same vectors (similiarly B' and B), the prover commits to $\boldsymbol{a}$ and $\boldsymbol{b}$ under the binding keys: $A'' = \boldsymbol{t}^{\boldsymbol{a}} \cdot h^{\gamma \cdot (r_a + r'_a)}$, and $B'' = \boldsymbol{t}^{\boldsymbol{b}} \cdot h^{\gamma \cdot (r_b + r'_b)}$.
3. The verifier first checks that commitments A' and B' are well formed. Next the verifier checks $e(A'', g) \overset{?}{=} e(A \cdot A'_1, g^\gamma)$ and $e(B'', g) \overset{?}{=} e(B \cdot B'_1, g^\gamma)$.
4. If the verifier's check passes, both the prover and verifier engage in an inner-product argument (Construction 3) over statement (A, B, C, r) and witness $(\boldsymbol{a}, \boldsymbol{b}, c, r_a, r_b, r_c)$.

Theorem 3 (Inner-Product Argument for the Lagrange Basis). *Construction 4 is an argument system for $\mathcal{R}_{\mathsf{IP}}$ that satisfies completeness, knowledge soundness, and honest-verifier zero-knowledge. For vectors of size n construction 4 features an $O(n)$ generator, $O(n)$ prover, and an $O(\log n)$ verifier.*

Proof. Completeness follows by observation and the completeness of the underlying inner-product argument. Informally, knowledge soundness follows from the $(n-1)$-EPKE assumption. Zero-knowledge follows due to the blinding terms. We formally prove Theorem 3 in supplementary section D.5. □

4.5 An Argument System for Multi-Hadamard Product

In the final round for our argument system for general computation (Section 5), we require an argument system for a multi-Hadamard product. We achieve a system with our desired asymptotics by composing the multi-Hadamard product argument system presented by Bayer [8] with our argument system for inner-product (Construction 4).

Definition 13 (The Multi-Hadamard Relation). *Consider group $\mathbb{G}$ of order p and let $\mathbb{F} = \mathbb{Z}_p$. The multi-Hadamard relation ($\mathcal{R}_{\mathsf{MHADM}}$) defined over vector size n, and instance size m consists of m commitments $A_1, \ldots, A_m$, and commitment B. Vectors $\boldsymbol{a}_1, \ldots, \boldsymbol{a}_m$ and vector $\boldsymbol{b}$ satisfy the multi-Hadamard relation if $A_i = \mathsf{com}(\boldsymbol{a}_i)$ for all $i \in [m]$, $B = \mathsf{com}(\boldsymbol{b})$, and $\boldsymbol{b} = \boldsymbol{a}_1 \circ \boldsymbol{a}_2 \circ \ldots \circ \boldsymbol{a}_m$.*

Construction 5 (Multi-Hadamard-Product Argument — Sketch). Our construction composes the multi-Hadamard product argument system presented by Bayer [8] with the argument system for inner-product (Construction 4). At a high level, Bayer's argument uses random linear combinations to reduce the original multi-Hadamard-product check into checking that

$$A = \mathsf{com}(\overline{\boldsymbol{a}}, \overline{r}) \qquad B = \mathsf{com}(\overline{\boldsymbol{b}}, \overline{s}) \qquad D = \mathsf{com}((\overline{\boldsymbol{a}} \circ \boldsymbol{y}) \cdot \overline{\boldsymbol{b}}, \overline{t})$$

for commitments A, B, D, vectors $\overline{\boldsymbol{a}}, \overline{\boldsymbol{b}}, \boldsymbol{y}$, and associated randomness $\overline{r}, \overline{s}, \overline{t}$ generated during interaction. Our argument is identical to the one presented by Bayer [8] with the exception that in the final round of Bayer's original argument the prover directly sends $\overline{\boldsymbol{a}}$, $\overline{r}$, $\overline{\boldsymbol{b}}$, $\overline{s}$ and $\overline{t}$ for the verifier to check. In our variant the verifier instead outsources this final check using an argument system for inner-product. For completeness we reproduce Bayer's multi-Hadamard-product argument in supplementary section B, however we stress that the details are not important for understanding our argument system for general computation.

Theorem 4. *Construction 5 is an argument system for $\mathcal{R}_{\mathsf{MHADM}}$ that satisfies completeness, knowledge soundness, and honest-verifier zero-knowledge. For m vectors of size n, Construction 5 features an $O(n)$ generator, $O(nm^2)$ prover, and an $O(\log n + m)$ verifier.*

Proof. We formally prove Theorem 4 in supplementary section B. □

5 An Argument System for $\mathcal{R}_{\mathsf{ACS}}$

We build an interactive argument system for Algebraic Constraint Satisfiability ($\mathcal{R}_{\mathsf{ACS}}$). An $\mathcal{R}_{\mathsf{ACS}}$ instance consists of statements and witnesses represented as matrices and vectors. We first show how to encode an $\mathcal{R}_{\mathsf{ACS}}$ instance as a sum-check instance. The verifier reduces the sum-check instance to checking an inner-product and polynomial equality, which in turn can be reduced into checking another (simpler) sum-check instance. The verifier repeats this interaction over several rounds to reduce the original statement into checking the Hadamard-product over vectors generated during interaction.

Definition 14 (Algebraic Constraint Satisfiability Relation). *The Algebraic Constraint Satisfiability Relation ($\mathcal{R}_{\mathsf{ACS}}$) defined over field $\mathbb{F}$, instance size*

n, witness size m, and constraint size l consists of matrices $\boldsymbol{M}_1, \ldots, \boldsymbol{M}_l$ in $\mathbb{F}^{n \times n}$, and vector $\boldsymbol{x} \in \mathbb{F}^{n-m}$. A witness $\boldsymbol{w} \in \mathbb{F}^m$ satisfies an $\mathcal{R}_{\mathsf{ACS}}$ instance if

$$0 = (\boldsymbol{x}, \boldsymbol{w})\boldsymbol{M}_i(\boldsymbol{x}, \boldsymbol{w})^\top \quad \forall i \in H.$$

We consider an $\mathcal{R}_{\mathsf{ACS}}$ instance sparse if there are $O(n)$ non-zero elements in all matrices $M_1, \ldots, M_l$. We prove in supplementary section C that any relation in NP can be reduced to a sparse $\mathcal{R}_{\mathsf{ACS}}$ instance.

Construction 6 (Argument System for Algebraic Constraint Satisfiability). Consider a sparse $\mathcal{R}_{\mathsf{ACS}}$ instance of size n with witness of size m and constraints indexed by subset $H \subseteq \mathbb{F}$. Let this instance be defined by matrices $\boldsymbol{M}_i \in \mathbb{F}^{n \times n}$ for $i \in H$ and input vector $\boldsymbol{x}$. Let subset $K \subseteq \mathbb{F}$ index the *non-zero* entries in *all* $|H|$ matrices $\{\boldsymbol{M}_i\}_{i \in H}$. For notational simplicity let $N = (1, \ldots, n)$. Suppose a prover would like to prove in zero-knowledge that it possesses a vector $\boldsymbol{w}$ such that

$$0 = (\boldsymbol{x}, \boldsymbol{w})\boldsymbol{M}_i(\boldsymbol{x}, \boldsymbol{w})^\top \quad \forall i \in H. \tag{3}$$

Setup Phase: In order to efficiently check constraints of the above form using standard algebraic techniques, both the prover and verifier first need to encode matrices $\boldsymbol{M}_i$ for $i \in H$ as polynomials: For $k \in K$ let polynomial $\mathsf{A}(k) : K \to H$ return the particular matrix that k is associated with. Similiarly, let $\mathsf{B}(k) : K \to N$ return the particular row that k is associated with and let $\mathsf{C}(k) : K \to N$ return the particular column that k is associated with. Finally, let $\mathsf{V}(k) : K \to \mathbb{F}$ return the value associated with index k. [1] In practice polynomials $\mathsf{A}, \mathsf{B}, \mathsf{C}$, and V only need to be computed and committed to once by a trusted party and can be reused across different input vectors $\boldsymbol{x}$. We additionally stress that these polynomials are represented as vectors of evaluations throughout the argument and therefore do not need to be interpolated. As shown by Chiesa et al. [33], polynomials $\mathsf{A}, \mathsf{B}, \mathsf{C}, \mathsf{V}$ allow us to encode matrices $\boldsymbol{M}_a$ for $a \in H$ as follows

$$\boldsymbol{M}_a[b, c] = \sum_{k \in K} u_H(a, \mathsf{A}(k)) \cdot u_N(b, \mathsf{B}(k)) \cdot u_N(c, \mathsf{C}(k)) \cdot \mathsf{V}(k).$$

Recall that bivariate polynomial $u_H(X, Y) : H \times H \to \mathbb{F}$ returns non-zero if $X = Y$ and 0 otherwise (Definition 6), and can be efficiently computed when H is a multiplicative subgroup. For this reason index sets H, N, and K should be multiplicative subgroups in practice. For notational simplicity we define polynomial $P(k, a, b, c)$ as follows:

$$P(k, a, b, c) \coloneqq u_H(a, \mathsf{A}(k)) \cdot u_N(b, \mathsf{B}(k)) \cdot u_N(c, \mathsf{C}(k)) \cdot \mathsf{V}(k).$$

[1] More precisely polynomial V must account for the non-zero values of $u_H(a, a)$ for $a \in H$ and $u_N(b, b)$ for $b \in N$. These non-zero values can be computed once globally by the generator.

Argument Phase: Let z be the evaluation-based polynomial encoding for vector $(\boldsymbol{x}, \boldsymbol{w})$ (i.e. $z(i) = (\boldsymbol{x}, \boldsymbol{w})_i$ for all $i \in N$) Given the polynomial encodings, we first define

$$Q(a) \coloneqq \sum_{b \in N} \sum_{c \in N} \sum_{k \in K} P(k, a, b, c) z(b) z(c)$$

and observe that equation 3 is true if and only if

$$0 = Q(a) \quad \forall a \in H \tag{4}$$

From the setup phase, the prover and verifier both have access to commitments to $\mathsf{A}, \mathsf{B}, \mathsf{C}, \mathsf{V}$, v_H, v_N, v_K, and $v_{[n-m]}$ represented as vectors of evaluations. The prover additionally has access to the underlying evaluation vectors for polynomials $\mathsf{A}, \mathsf{B}, \mathsf{C}, \mathsf{V}$ generated during the setup phase, and v_H, v_N, v_K, and $v_{[n-m]}$ computed once globally by the generator. To begin the argument the prover sends extractible and hiding evaluation-based commitment to polynomial z. Before checking equation 4 the verifier needs to check that $\boldsymbol{x}$ has been correctly encoded in the prover's commitment. To assist the verifier with this check, the prover additionally sends a commitment to evaluations of "shifted" witness polynomial ([33]), w' such that for all $i \in [n-m+1, n]$

$$w'(i) = \frac{\boldsymbol{w}_i - \boldsymbol{x}_i}{v_{[n-m]}(i)}$$

where $v_{[n-m]}$ is the vanishing polynomial for the range $[n-m]$. We observe that if the prover correctly computes w', we have

$$z(X) = w'(X) v_{[n-m]}(X) + \boldsymbol{x}(X) \tag{5}$$

Additionally, equation 5 ensures that $z(h) = \boldsymbol{x}(h)$ for $h \in [n-m]$, thus ensuring that $\boldsymbol{x}$ has been embedded correctly. Thus the verifier can check that z agrees with w' and $\boldsymbol{x}$ by accepting negligible soundness error, picking random $\tau \in \mathbb{F}$, and checking

$$z(\tau) = w'(\tau) v_{[n-m]}(\tau) + \boldsymbol{x}(\tau)$$

In particular the verifier uses a polynomial evaluation argument to obtain commitments to $z(\tau)$, $w'(\tau)$, and $v_{[n-m]}(\tau)$, and then uses a standard product argument to check that the appropriate relationship holds.

To check equation 4, we first observe that polynomial $P(k, a, b, c)$ is degree $|H| - 1$ in a, which implies that $Q(a)$ is degree $|H| - 1$ in a. Therefore, to check equation 4, it suffices to check that Q is the zero polynomial. To do so, the verifier accepts negligible soundness error, picks random $\alpha \xleftarrow{\$} \mathbb{F}$, and checks

$$0 = Q(\alpha).$$

By definition this requires the verifier check

$$0 = \sum_{b \in N} \sum_{c \in N} \sum_{k \in K} P(k, \alpha, b, c) z(b) z(c). \tag{6}$$

The verifier can rewrite equation 6 as

$$0 = \sum_{b \in N} z(b) \sum_{c \in N} \sum_{k \in K} P(k, \alpha, b, c) z(c). \tag{7}$$

In order to assist the verifier in checking equation 7 the prover can (efficiently [33]) compute and commit to evaluations of degree $|N| - 1$ polynomial

$$P_1(X) = \sum_{c \in N} \sum_{k \in K} P(k, \alpha, X, c) z(c). \tag{8}$$

The verifier is now tasked with checking

$$0 = \sum_{b \in N} z(b) P_1(b) \tag{9}$$

and checking that equation 8 holds. Because both z and P_1 are represented and committed to using their evaluation vectors, we know that the right-hand side of equation 9 is precisely the inner-product of the evaluation vectors. Therefore the verifier can use a proof of inner-product to check equation 9. What remains is for the verifier to check that equation 8 holds.

To do so, the verifier accepts negligible soundness error, picks random $\beta \overset{\$}{\leftarrow} \mathbb{F}$, and reduces the task of checking equation 8 to the task of checking

$$P_1(\beta) = \sum_{c \in N} \sum_{k \in K} P(k, \alpha, \beta, c) z(c). \tag{10}$$

The verifier can rewrite equation 10 as

$$P_1(\beta) = \sum_{c \in N} z(c) \sum_{k \in K} P(k, \alpha, \beta, c). \tag{11}$$

In order to assist the verifier in checking equation 11 the prover can efficiently compute and commit to evaluations of degree $|N| - 1$ polynomial

$$P_2(X) = \sum_{k \in K} P(k, \alpha, \beta, X). \tag{12}$$

The verifier is now tasked with checking

$$P_1(\beta) = \sum_{c \in N} z(c) P_2(c) \tag{13}$$

and checking that equation 12 holds. The verifier can evaluate $P_1(\beta)$ using a polynomial evaluation argument. Then, as observed earlier, the verifier can check equation 13 by checking the inner-product of the evaluation vectors of z and P_2. What remains is for the verifier to check equation 12 holds.

To do so, the verifier accepts negligible soundness error, picks random $\gamma \overset{\$}{\leftarrow} \mathbb{F}$, and reduces the task of checking equation 12 to the task of checking

$$P_2(\gamma) = \sum_{k \in K} P(k, \alpha, \beta, \gamma). \tag{14}$$

While the degree of P in k is $(|H|+|N|)\cdot|K|$, the prover can efficiently compute and commit to evaluations of degree $|K|-1$ polynomial P_3 such that

$$P_3(k) = P(k,\alpha,\beta,\gamma) \quad \forall k \in K. \tag{15}$$

The verifier is now tasked with checking

$$P_2(\gamma) = \sum_{k\in K} P_3(k) \tag{16}$$

and checking that equation 15 holds. The verifier can evaluate $P_2(\gamma)$ using a polynomial evaluation argument. Then the verifier can check equation 16 by checking the dot product of the evaluation vectors P_3 and $\mathbf{1} = (1,1,\dots,1)$. To check equation 15, we observe due to Chiesa et al. [33] that

$$\begin{aligned}
P(k,\alpha,\beta,\gamma) &= u_H(\alpha,\mathsf{A}(k))\cdot u_N(\beta,\mathsf{B}(k))\cdot u_N(\gamma,\mathsf{C}(k))\cdot\mathsf{V}(k)\\
&= \frac{(v_H(\alpha)-v_H(\mathsf{A}(k))\cdot(v_N(\beta)-v_N(\mathsf{B}(k)))\cdot(v_N(\gamma)-v_N(\mathsf{C}(k)))\cdot\mathsf{V}(k)}{(\alpha-\mathsf{A}(k))(\beta-\mathsf{B}(k))(\gamma-\mathsf{C}(k))}\\
&= \frac{v_H(\alpha)v_N(\beta)v_N(\gamma)\mathsf{V}(k)}{(\alpha-\mathsf{A}(k))(\beta-\mathsf{B}(k))(\gamma-\mathsf{C}(k))}
\end{aligned}$$

where the last equality holds because polynomials $\mathsf{A},\mathsf{B},\mathsf{C}$ map elements of K to H and N. Therefore, the verifier can check equation 15 by checking

$$P_3(k)(\alpha-\mathsf{A}(k))(\beta-\mathsf{B}(k))(\gamma-\mathsf{C}(k)) = v_H(\alpha)v_N(\beta)v_N(\gamma)\mathsf{V}(k) \quad \forall k\in K. \tag{17}$$

The prover and verifier can efficiently compute commitments to $\mathsf{A}'(k) = \alpha-\mathsf{A}(k)$, $\mathsf{B}'(k) = \beta-\mathsf{B}(k)$, and $\mathsf{C}'(k) = \gamma-\mathsf{C}(k)$. The verifier can invoke a polynomial evaluation argument to evaluate $v_H(\alpha), v_N(\beta), v_N(\gamma)$, and the prover and verifier can compute the commitment to $\mathsf{V}'(k) = v_H(\alpha)v_N(\beta)v_N(\gamma)\mathsf{V}(k)$. Equation 17 can be rewritten as

$$P_3(k)\mathsf{A}'(k)\mathsf{B}'(k)\mathsf{C}'(k) = \mathsf{V}'(k) \quad \forall k\in K. \tag{18}$$

The verifier can then check equation 18 using a proof of multi-Hadamard-product (construction 5).

Theorem 5. *Construction 6 is an argument system for $\mathcal{R}_{\mathsf{ACS}}$ that satisfies completeness, knowledge-soundness, and honest-verifier zero-knowledge.*

Proof. Informally, completeness follows from the completeness of the underlying argument systems. Knowledge soundness follows from the knowledge soundness of the underlying polynomial commitment scheme. Honest-verifier zero-knowledge holds because the prover only sends perfectly hiding polynomial commitments to the verifier and engages in honest-verifier zero-knowledge arguments regarding these commitments. We formally prove Theorem 5 in supplementary section D.6. □

Lemma 3 (Efficiency). *For a sparse size n $\mathcal{R}_{\mathsf{ACS}}$ instance with input vector size x, construction 6 features an $O(n)$ generator, $O(n)$ prover and $O(\log n + x)$ verifier.*

Proof. The generator must run the generators for all the subarguments; this can be done in $O(n)$ time by Lemma 2, and Theorems 1, 2, 3, and 4. Additionally the generator must compute evaluations of auxiliary polynomials v_H, v_N, v_K, $v_{[n-m]}$, u_H, and u_N, which can be computed in $O(n/\log n)$ exponentiations by Pippenger's algorithm [66] if the associated vanishing domains are multiplicative subgroups (Definition 5).

As claimed in Construction 6 the prover can efficiently compute polynomials P_1, P_2, and P_3 in linear time by constructing a lookup table, as discussed by Chiesa et al. [33]. For completeness, we describe how to construct such a lookup table for polynomial P_1 in supplementary section E. For the remainder of the argument the prover commits to evaluation-based representations of polynomials w', z, P_1, P_2, P_3. By Lemma 2 this is dominated by 10 multi-exponentiations of size n total. Next the prover engages in a polynomial-evaluation argument to verifiably evaluate $z(\tau)$, $w'(\tau)$, $v_{[n-m]}(\tau)$, $P_1(\beta)$, $P_2(\gamma)$, $v_H(\alpha)$, $v_N(\beta)$, and $v_N(\gamma)$. By Theorem 1, this is dominated by 16 multi-exponentiations of size n total. The prover engages in three inner-product arguments over vector pairs (z, P_1), (z, P_2), and $(\mathbf{1}, P_3)$. By Theorem 2 and Theorem 3, this is dominated by 48 multi-exponentiations of size n total. Finally, the prover engages in an multi-Hadamard argument over vectors $P_3, \mathsf{A}', \mathsf{B}', \mathsf{C}'$ and V'. By Theorem 4 and Theorem 3 this is dominated by 20 multi-exponentiations of size n total.

The verifier computes $\boldsymbol{x}(\tau)$ for some random $\tau \xleftarrow{\$} \mathbb{F}$, which takes linear time in the size of the input vector. Next, the verifier checks 8 polynomial-evaluation arguments. By Theorem 1, this can be one in $O(1)$ time. Next the verifier checks 3 inner-product arguments in the main interaction. By Theorem 2, this can be done in $O(\log n)$ time. Finally the verifier checks a single multi-Hadamard argument. By Theorem 4 this can be done in $O(\log n)$ time. We note that the core multi-Hadamard argument features a constant-time verifier, with the underlying inner-product argument incurring the logarithmic overhead. □

6 Optimizations

Construction 7 (A Constant-Time Verifier). The verifier's work for Construction 6 is nearly constant-time, with the exception of the verifier's logarithmic-time work for the inner-product arguments (Construction 3). We achieve a constant-time verifier while maintaining a linear-time prover by outsourcing the verifier's logarithmic work represented as a circuit. Because our main argument (Construction 6) is proven secure in the random oracle model, for outsourcing to be possible, we must heuristically instantiate the random oracle with a cryptographic hash function as seen in Fractal [34] and Halo [27].

At first glance outsourcing seems straightforward: The verifier's work can be outsourced by using an argument system for general computation with a constant-time verifier and a quasi-linear prover such as Pinocchio [65]. However, we must be careful to ensure that the verifier's input to the outsourced verification circuit is also constant sized. In addition, to preserve a linear prover, we need to ensure that the prover's witness is sublinear in the size of the original $\mathcal{R}_{\mathsf{ACS}}$ instance. These constraints make it difficult to apply verification outsourcing to existing systems with a linear prover, which typically rely on complicated multivariate sum-check protocols [70, 78].

Fortunately, we carefully design, Construction 3 for this set of constraints. Recall that the verifier's logarithmic work can be broken down into three distinct tasks: (1) Sample randomness x for a logarithmic number of rounds. (2) Compute commitments A', B', C' over a logarithmic number of rounds. (3) During the subprotocol to check the v and w terms, compute $f_v(z)$ and $f_w(z)$ for some challenge point z.

To ensure that the verifier's input to the verification circuit is constant-sized, we apply the Fiat-Shamir heuristic [40] and have the circuit simulate the verifier's randomness by hashing the prover's messages from the previous round. Theoretically we can use any cryptographic hash function that can be computed by an arithmetic circuit, however in practice we can use an efficiently computable algebraic hash function [1, 51]. Provided the simulated randomness, the circuit will verifiably compute commitments A', B', C' in each round. In the final round the circuit will output the final commitments, $f_v(z)$, and $f_w(z)$, which the verifier can use to finish the inner-product argument. We note that the outsourced circuit is not statement dependent, thus preserving a universal SRS. In more detail, given an $\mathcal{R}_{\mathsf{ACS}}$ instance of size n we describe a size $O(\log^2 n)$ circuit to outsource the verifier's non-constant work:

$O(\log^2 n)$ Circuit to Outsource Verifier's Inner-Product Checks:

1. Let $\ell = \log n$ denote the total number rounds in the inner-product argument. The verifier's input consists of commitments $A, B, C \in \mathbb{G}$ which represent the statement for the inner-product argument. The prover's input consists of all logarithmic number of commitments $A_{Li}, A_{Ri}, B_{Li}, B_{Ri}, C_{Li}, C_{Ri} \in \mathbb{G}$ terms for $i \in \{0, \ldots, \ell - 1\}$ generated in each recursive round, and the terms (v, v'), (w, w') generated in the final round. Initially set $A_0 = A$, $B_0 = B$, and $C_0 = C$.
2. For $i \in \{0, \ldots, \ell{-}1\}$ compute randomness $x_i = \mathsf{hash}(A_{Li}, A_{Ri}, B_{Li}, B_{Ri}, C_{Li}, C_{Ri})$ and compute the resulting commitments
$$A_{i+1} = A_{Li}^{x_i} \cdot A_i \cdot A_{Ri}^{x_i^{-1}}, \quad B_{i+1} = B_{Li}^{x_i} \cdot B_i \cdot B_{Ri}^{x_i^{-1}}, \quad C_{i+1} = C_{Li}^{x_i} \cdot C_i \cdot C_{Ri}^{x_i^{-1}}.$$
3. Compute the verifier's final challenge, $z = \mathsf{hash}(A_\ell, B_\ell, C_\ell, (v, v'), (w, w'))$
4. Compute $f_v(z)$ and $f_w(z)$ as defined in Construction 3 in a logarithmic number of exponentiations, and output A_ℓ, B_ℓ, C_ℓ, $f_v(z)$, $f_w(z)$.

Construction 8 (Zero-Knowledge and Non-Interactivity). Because all the discussed protocols are public-coin and special honest-verifier zero-knowledge, we can apply the Fiat-Shamir transform [40] to achieve zero-knowledge and non-interactivity.

Remark 2 (Additional Optimizations). In supplementary section F, we detail additional minor concrete optimizations. Applying these optimizations, the prover's total cost breaks down as follows: 5 multi-exponentiations of size n (n-MEXP) for polynomial commitments, 4 n-MEXPs for polynomial evaluations, 15 n-MEXPs for the multi-Hadamard-product argument, and 11 n-MEXPs for the inner-product arguments. This puts the prover's total dominating cost at 35 n-MEXPs. The verifier's total dominating costs are 34 pairings.

Acknowledgements

We thank Vipul Goyal and Andrew Miller for insightful comments on earlier versions of this work.

References

[1] Aly, A., Ashur, T., Ben-Sasson, E., Dhooghe, S., Szepieniec, A.: Efficient symmetric primitives for advanced cryptographic protocols (a marvellous contribution). IACR Cryptol. ePrint Arch. **2019**

[2] Ames, S., Hazay, C., Ishai, Y., Venkitasubramaniam, M.: Ligero: Lightweight sublinear arguments without a trusted setup. In: CCS (2017)

[3] Arora, S., Lund, C., Motwani, R., Sudan, M., Szegedy, M.: Proof verification and the hardness of approximation problems. JACM **45**(3) (1998)

[4] Arora, S., Safra, S.: Probabilistic checking of proofs: A new characterization of NP. JACM **45**(1) (1998)

[5] Ateniese, G., Goodrich, M.T., Lekakis, V., Papamanthou, C., Paraskevas, E., Tamassia, R.: Accountable storage. In: ACNS (2017)

[6] Babai, L.: Trading group theory for randomness. In: STOC (1985)

[7] Babai, L., Fortnow, L., Levin, L.A., Szegedy, M.: Checking computations in polylogarithmic time. In: STOC (1991)

[8] Bayer, S.G.M.: Practical Zero-Knowledge Protocols based on the Discrete Logarithm Assumption. Ph.D. thesis, University College London (2014)

[9] Ben-Or, M., Goldreich, O., Goldwasser, S., Håstad, J., Kilian, J., Micali, S., Rogaway, P.: Everything provable is provable in zero-knowledge. In: CRYPTO (1988)

[10] Ben-Sasson, E., Bentov, I., Horesh, Y., Riabzev, M.: Fast Reed-Solomon interactive oracle proofs of proximity. ECCC **24** (2017)

[11] Ben-Sasson, E., Bentov, I., Horesh, Y., Riabzev, M.: Scalable, transparent, and post-quantum secure computational integrity. Cryptology ePrint Archive, Report 2018/046 (2018)

[12] Ben-Sasson, E., Chiesa, A., Gabizon, A., Riabzev, M., Spooner, N.: Interactive oracle proofs with constant rate and query complexity. In: ICALP 2017 (2017)

[13] Ben-Sasson, E., Chiesa, A., Gabizon, A., Virza, M.: Quasi-linear size zero knowledge from linear-algebraic PCPs. In: TCC (2016)

[14] Ben-Sasson, E., Chiesa, A., Genkin, D., Tromer, E., Virza, M.: SNARKs for C: Verifying program executions succinctly and in zero knowledge. In: CRYPTO (2013)

[15] Ben-Sasson, E., Chiesa, A., Goldberg, L., Gur, T., Riabzev, M., Spooner, N.: Linear-size constant-query IOPs for delegating computation. In: TCC (2019)

[16] Ben-Sasson, E., Chiesa, A., Riabzev, M., Spooner, N., Virza, M., Ward, N.P.: Aurora: Transparent succinct arguments for R1CS. In: EUROCRYPT (2019)

[17] Ben-Sasson, E., Chiesa, A., Spooner, N.: Interactive oracle proofs. In: TCC (2016)

[18] Ben-Sasson, E., Chiesa, A., Tromer, E., Virza, M.: Scalable zero knowledge via cycles of elliptic curves. Algorithmica **79**(4) (2017)

[19] Ben-Sasson, E., Sudan, M.: Short PCPs with polylog query complexity. SICOMP **38**(2) (2008)

[20] Berrut, J.P., Trefethen, L.N.: Barycentric Lagrange interpolation. SIREV **46**(3) (2004)

[21] Bitansky, N., Canetti, R., Chiesa, A., Tromer, E.: From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In: ITCS (2012)

[22] Bitansky, N., Canetti, R., Chiesa, A., Tromer, E.: Recursive composition and bootstrapping for snarks and proof-carrying data. In: Proceedings of the forty-fifth annual ACM symposium on Theory of computing. pp. 111–120 (2013)

[23] Bitansky, N., Canetti, R., Paneth, O., Rosen, A.: On the existence of extractable one-way functions. SICOMP **45**(5) (2016)

[24] Bitansky, N., Chiesa, A., Ishai, Y., Paneth, O., Ostrovsky, R.: Succinct non-interactive arguments via linear interactive proofs. In: TCC (2013)

[25] Boneh, D., Boyen, X.: Short signatures without random oracles. In: EUROCRYPT (2004)

[26] Bootle, J., Cerulli, A., Chaidos, P., Groth, J., Petit, C.: Efficient zero-knowledge arguments for arithmetic circuits in the discrete log setting. In: EUROCRYPT (2016)

[27] Bowe, S., Grigg, J., Hopwood, D.: Halo: Recursive proof composition without a trusted setup. IACR Cryptol. ePrint Arch. **2019**

[28] Boyle, E., Pass, R.: Limits of extractability assumptions with distributional auxiliary input. In: ASIACRYPT (2015)

[29] Bünz, B., Bootle, J., Boneh, D., Poelstra, A., Wuille, P., Maxwell, G.: Bulletproofs: Short proofs for confidential transactions and more. In: IEEE S&P (2018)

[30] Bünz, B., Fisch, B., Szepieniec, A.: Transparent SNARKs from DARK compilers. In: EUROCRYPT (2020)

[31] Bünz, B., Maller, M., Mishra, P., Vesely, N.: Proofs for inner pairing products and applications. Cryptology ePrint Archive, Report 2019/1177 (2019)

[32] Chase, M., Derler, D., Goldfeder, S., Orlandi, C., Ramacher, S., Rechberger, C., Slamanig, D., Zaverucha, G.: Post-quantum zero-knowledge and signatures from symmetric-key primitives. In: CCS (2017)

[33] Chiesa, A., Hu, Y., Maller, M., Mishra, P., Vesely, N., Ward, N.: Marlin: Preprocessing zkSNARKs with universal and updatable SRS. In: EUROCRYPT (2020)

[34] Chiesa, A., Ojha, D., Spooner, N.: Fractal: Post-quantum and transparent recursive proofs from holography. In: EUROCRYPT (2020)

[35] Cormode, G., Mitzenmacher, M., Thaler, J.: Practical verified computation with streaming interactive proofs. In: ITCS (2012)

[36] Costello, C., Fournet, C., Howell, J., Kohlweiss, M., Kreuter, B., Naehrig, M., Parno, B., Zahur, S.: Geppetto: Versatile verifiable computation. In: 2015 IEEE Symposium on Security and Privacy (2015)

[37] Danezis, G., Fournet, C., Groth, J., Kohlweiss, M.: Square span programs with applications to succinct NIZK arguments. In: ASIACRYPT (2014)

[38] Danezis, G., Fournet, C., Kohlweiss, M., Parno, B.: Pinocchio coin: Building Zerocoin from a succinct pairing-based proof system. In: Proceedings of the First ACM workshop on Language support for privacy-enhancing technologies (2013)

[39] Delignat-Lavaud, A., Fournet, C., Kohlweiss, M., Parno, B.: Cinderella: Turning shabby X. 509 certificates into elegant anonymous credentials with the magic of verifiable computation. In: IEEE S&P (2016)

[40] Fiat, A., Shamir, A.: How to prove yourself: Practical solutions to identification and signature problems. In: EUROCRYPT (1986)

[41] Fiore, D., Fournet, C., Ghosh, E., Kohlweiss, M., Ohrimenko, O., Parno, B.: Hash first, argue later: Adaptive verifiable computations on outsourced data. In: CCS (2016)

[42] Fiore, D., Gennaro, R., Pastro, V.: Efficiently verifiable computation on encrypted data. In: CCS (2014)

[43] Fuchsbauer, G., Kiltz, E., Loss, J.: The algebraic group model and its applications. In: CRYPTO (2018)

[44] Gennaro, R., Gentry, C., Parno, B.: Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In: CRYPTO (2010)

[45] Gennaro, R., Gentry, C., Parno, B., Raykova, M.: Quadratic span programs and succinct NIZKs without PCPs. In: EUROCRYPT (2013)

[46] Gentry, C., Wichs, D.: Separating succinct non-interactive arguments from all falsifiable assumptions. In: STOC (2011)

[47] Giacomelli, I., Madsen, J., Orlandi, C.: ZKBoo: Faster zero-knowledge for boolean circuits. In: USENIX (2016)

[48] Goldwasser, S., Kalai, Y.T., Rothblum, G.N.: Delegating computation: Interactive proofs for muggles. JACM **62**(4) (2015)

[49] Goldwasser, S., Micali, S., Rackoff, C.: The knowledge complexity of interactive proof systems. SICOMP **18**(1) (1989)

[50] Goyal, V.: Reducing trust in the PKG in identity based cryptosystems. In: CRYPTO (2007)

[51] Grassi, L., Khovratovich, D., Roy, A., Rechberger, C., Schofnegger, M.: Poseidon: A new hash function for zero-knowledge proof systems. In: USENIX (2020)

[52] Groth, J.: Linear algebra with sub-linear zero-knowledge arguments. In: CRYPTO (2009)

[53] Groth, J.: On the size of pairing-based non-interactive arguments. In: EUROCRYPT (2016)

[54] Groth, J., Ishai, Y.: Sub-linear zero-knowledge argument for correctness of a shuffle. In: EUROCRYPT (2008)

[55] Ishai, Y., Kushilevitz, E., Ostrovsky, R.: Efficient arguments without short PCPs. In: CCC (2007)

[56] Ishai, Y., Kushilevitz, E., Ostrovsky, R., Sahai, A.: Zero-knowledge from secure multiparty computation. In: STOC (2007)

[57] Kate, A., Zaverucha, G.M., Goldberg, I.: Constant-size commitments to polynomials and their applications. In: ASIACRYPT (2010)

[58] Kilian, J.: A note on efficient zero-knowledge proofs and arguments. In: STOC (1992)

[59] Kosba, A., Miller, A., Shi, E., Wen, Z., Papamanthou, C.: Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. In: IEEE S&P (2016)

[60] Lindell, Y.: Parallel coin-tossing and constant-round secure two-party computation. J. Cryptology **16**(3) (2003)

[61] Lund, C., Fortnow, L., Karloff, H., Nisan, N.: Algebraic methods for interactive proof systems. JACM **39**(4) (1992)

[62] Maller, M., Bowe, S., Kohlweiss, M., Meiklejohn, S.: Sonic: Zero-knowledge SNARKs from linear-size universal and updatable structured reference strings. In: CCS (2019)

[63] Naveh, A., Tromer, E.: Photoproof: Cryptographic image authentication for any set of permissible transformations. In: IEEE S&P (2016)

[64] Papamanthou, C., Shi, E., Tamassia, R.: Signatures of correct computation. In: TCC (2013)

[65] Parno, B., Howell, J., Gentry, C., Raykova, M.: Pinocchio: Nearly practical verifiable computation. In: IEEE S&P (2013)

[66] Pippenger, N.: On the evaluation of powers and related problems. In: SFCS (1976)

[67] Reingold, O., Rothblum, G.N., Rothblum, R.D.: Constant-round interactive proofs for delegating computation. STOC (2019)

[68] Sasson, E.B., Chiesa, A., Garman, C., Green, M., Miers, I., Tromer, E., Virza, M.: Zerocash: Decentralized anonymous payments from Bitcoin. In: IEEE S&P (2014)

[69] Schwartz, J.T.: Fast probabilistic algorithms for verification of polynomial identities. JACM **27**(4) (1980)

[70] Setty, S.: Spartan: Efficient and general-purpose zkSNARKs without trusted setup. In: CRYPTO (2020)

[71] Setty, S., Braun, B., Vu, V., Blumberg, A.J., Parno, B., Walfish, M.: Resolving the conflict between generality and plausibility in verified computation. In: ACM EuroSys (2013)

[72] Setty, S., Vu, V., Panpalia, N., Braun, B., Blumberg, A.J., Walfish, M.: Making argument systems for outsourced computation practical (sometimes). In: NDSS. vol. 1 (2012)

[73] Setty, S., Vu, V., Panpalia, N., Braun, B., Blumberg, A.J., Walfish, M.: Taking proof-based verified computation a few steps closer to practicality. In: USENIX (2012)

[74] Valiant, P.: Incrementally verifiable computation or proofs of knowledge imply time/space efficiency. In: TCC (2008)

[75] Wahby, R.S., Ji, Y., Blumberg, A.J., Shelat, A., Thaler, J., Walfish, M., Wies, T.: Full accounting for verifiable outsourcing. In: CCS (2017)

[76] Wahby, R.S., Setty, S.T., Ren, Z., Blumberg, A.J., Walfish, M.: Efficient RAM and control flow in verifiable outsourced computation. In: NDSS (2015)

[77] Wahby, R.S., Tzialla, I., Shelat, A., Thaler, J., Walfish, M.: Doubly-efficient zkSNARKs without trusted setup. In: IEEE S&P (2018)

[78] Xie, T., Zhang, J., Zhang, Y., Papamanthou, C., Song, D.: Libra: Succinct zero-knowledge proofs with optimal prover computation. In: CRYPTO (2019)

[79] Zhang, J., Xie, T., Zhang, Y., Song, D.: Transparent polynomial delegation and its applications to zero knowledge proof. In: IEEE S&P (2020)

[80] Zhang, Y., Genkin, D., Katz, J., Papadopoulos, D., Papamanthou, C.: vSQL: Verifying arbitrary SQL queries over dynamic outsourced databases. In: IEEE S&P (2017)

[81] Zhang, Y., Genkin, D., Katz, J., Papadopoulos, D., Papamanthou, C.: A zero-knowledge version of vSQL. Cryptology ePrint Archive, Report 2017/1146 (2017)

[82] Zhao, Z., Chan, T.H.H.: How to vote privately using Bitcoin. In: ICICS (2015)

Supplementary Materials

A An Argument System for Simple Product

Construction 9 (Argument System for Simple Product). Both the prover and verifier are provided with commitment keys v, w, g, h, and commitments A, B, C. The prover is additionally provided witness a, b, c, r_a, r_b, r_c. An argument system for simple product allows a prover to show that

$$\begin{aligned} A &= v^a h^{r_a} \\ B &= w^b h^{r_b} \\ C &= g^c h^{r_c} \end{aligned}$$

and that $a \cdot b = c$. We present a simplified variant of a protocol presented by Bunz et al. [29]:

Subprotocol to check product:
Both the prover and verifier are provided with commitment keys v, w, g, h, and commitments A, B, C. The prover is additionally provided witness a, b, c, r_a, r_b, r_c.

1. The prover samples blinding terms $s_a, s_b \overset{\$}{\leftarrow} \mathbb{F}$, and randomness $\rho \overset{\$}{\leftarrow} \mathbb{F}$ and a commitment to the randomness $S = v^{s_a} \cdot w^{s_b} \cdot h^{\rho}$. Additionally the prover samples randomness $\tau_1, \tau_2 \overset{\$}{\leftarrow} \mathbb{F}$ and computes commitments to the error terms

   $$\begin{aligned} T_1 &= g^{s_a b + a s_b} \cdot h^{\tau_1} \\ T_2 &= g^{s_a s_b} \cdot h^{\tau_2}. \end{aligned}$$

   Finally the prover sends S, T_1, T_2 to the verifier.
2. The verifier responds with challenge $z \overset{\$}{\leftarrow} \mathbb{F}$.
3. The prover computes

   $$\begin{aligned} a' &= a + s_a \cdot z \\ b' &= b + s_b \cdot z c' \qquad\qquad = a' \cdot b' \end{aligned}$$

   Additionally the prover computes aggregated randomness terms

   $$\tau = \tau_2 \cdot z^2 + \tau_1 \cdot z + r_c \qquad\qquad \mu = (r_a + r_b) + \rho \cdot z.$$

   The prover sends τ, μ, a', b', c'.
4. The verifier first checks that c' agrees with C:

   $$g^{c'} h^{\tau} \overset{?}{=} C \cdot T_1^z \cdot T_2^{z^2}.$$

Next the verifier checks that a' and b' agree with A and B:

$$A \cdot B \cdot S^z \stackrel{?}{=} v^{a'} w^{b'} \cdot h^{\mu}.$$

Finally the verifier checks that the product relation holds:

$$c' \stackrel{?}{=} a' \cdot b'$$

B An Argument System for Multi-Hadamard Product

Construction 10 (Argument System for Multi-Hadamard Product). Consider group $\mathbb{G}$ of order p and let $\mathbb{F} = \mathbb{Z}_p$. An argument system for the multi-Hadamard production relation defined over vector size n, and instance size m, allows a prover to show that for commitments $A_1, \ldots, A_m$ and commitment B, it knows vectors $\boldsymbol{a}_1, \ldots, \boldsymbol{a}_m$ and vector $\boldsymbol{b}$ such that

$$A_i = \mathsf{com}(\boldsymbol{a}_i)$$

for all $i \in [m]$,

$$B = \mathsf{com}(\boldsymbol{b}),$$

and $\boldsymbol{b} = \boldsymbol{a}_1 \circ \boldsymbol{a}_2 \circ \ldots \circ \boldsymbol{a}_m$. Our construction composes the multi-Hadamard product argument system presented by Bayer [8] with the argument system for inner-product (Construction 4):

$\underline{\langle \mathsf{Prover}, \mathsf{Verifier} \rangle}$:

The prover and verifier are provided with the statement consisting of a list of commitments $\boldsymbol{A}$, and commitment B. The prover is additionally provided witness $(\{\boldsymbol{a}_i\}_{i \in [m]}, \{r_i\}_{i \in [m]}, \boldsymbol{b}, s)$.

1. Initially the prover computes

$$\boldsymbol{b}_1 = \boldsymbol{a}_1, \boldsymbol{b}_2 = \boldsymbol{a}_1 \circ \boldsymbol{a}_2, \ldots, \boldsymbol{b}_{m-1} = \boldsymbol{a}_1 \circ \cdots \circ \boldsymbol{a}_{m-1}, \boldsymbol{b}_m = \boldsymbol{b}.$$

   Now it is sufficient for the prover to show that for $i \in [m-1]$

$$\boldsymbol{b}_{i+1} = \boldsymbol{a}_{i+1} \circ \boldsymbol{b}_i \tag{19}$$

   so long as $\boldsymbol{b}_1 = \boldsymbol{a}_1$ and $\boldsymbol{b}_m = \boldsymbol{b}$. Next the prover samples $s_2, \ldots, s_{m-1} \stackrel{\$}{\leftarrow} \mathbb{F}$ and sets

$$B_2 = \mathsf{com}(\boldsymbol{b}_2; s_2), \ldots, B_{m-1} = \mathsf{com}(\boldsymbol{b}_{m-1}; s_{m-1}).$$

   The prover (and verifier) ensure that $\boldsymbol{b}_1 = \boldsymbol{a}_1$ and $\boldsymbol{b}_m = \boldsymbol{b}$ by setting $B_1 = A_1$ and $B_m = B$. Finally the prover sends $B_2, \ldots, B_{m-1}$.

2. The verifier samples and sends challenges $x, y \stackrel{\$}{\leftarrow} \mathbb{F}$

3. The prover can use randomness x to simplify the argument by taking a random linear combination of the vectors: In particular, the prover can demonstrate that equation 19 holds with high probability by showing

$$\sum_{i\in[m-1]} x^i \boldsymbol{b}_{i+1} = \sum_{i\in[m-1]} \boldsymbol{a}_{i+1} \circ (x^i \boldsymbol{b}_i). \tag{20}$$

To do so, the prover first computes vectors and associated randomness

$$\boldsymbol{d}_i = x^i \boldsymbol{b}_i \qquad\qquad \boldsymbol{t}_i = x^i s_i \quad \forall i \in [m-1]$$

and

$$\boldsymbol{d} = \sum_{i\in[m-1]} x^i \boldsymbol{b}_{i+1} \qquad\qquad t = \sum_{i\in[m-1]} x^i s_{i+1}$$

4. Next, both the prover and verifier compute the corresponding commitments $D_i = B_i^{x^i}$ for all $i \in [m]$ and $D = \prod_{i\in[m-1]} B_{i+1}^{x^i}$
5. Now equation 20 can be rewritten as

$$\boldsymbol{d} = \sum_{i\in[m-1]} \boldsymbol{a}_{i+1} \circ \boldsymbol{d}_i \tag{21}$$

To check that equation 21 holds with high probability, the verifier can engage in a zero argument (below) to check that

$$0 = \sum_{i\in[m-1]} (\boldsymbol{a}_{i+1} \circ \boldsymbol{y}) \cdot \boldsymbol{d}_i - (\mathbf{1} \circ \boldsymbol{y}) \cdot \boldsymbol{d}.$$

where $\boldsymbol{y} = (y, y^1, \ldots, y^n)$. In particular, the prover and verifier can first compute

$$C_{-1} = \mathsf{com}(-\mathbf{1}; 0)$$

and set the statement to be $(A_2, \ldots, A_m, C_{-1}), (D_1, \ldots, D_{m-1}, D)$.

The prover and verifier complete the multi-Hadamard product argument by engaging in an argument for the zero relation [8]. The zero relation ($\mathcal{R}_{\mathsf{ZERO}}$) defined over vector size n, and instance size m consists of commitments $A_1, \ldots, A_m$ and commitments $B_0, \ldots, B_{m-1}$, and scalar $y \in \mathbb{F}$. Vectors $\boldsymbol{a}_1, \ldots, \boldsymbol{a}_m$ and $\boldsymbol{b}_0, \ldots, \boldsymbol{b}_{m-1}$ satisfy the zero relation if

$$0 = \sum_{i\in[m]} (\boldsymbol{a}_i \circ \boldsymbol{y}) \cdot \boldsymbol{b}_{i-1}$$

where $\boldsymbol{y} = (y, y^2, \ldots, y^m)$, and $A_i = \mathsf{com}(a_i)$ and $B_{i-1} = \mathsf{com}(b_{i-1})$ for all $i \in [m]$.

Using random linear combinations, Bayer's argument for the zero relation reduces checking the original relation to checking that

$$A = \mathsf{com}(\overline{\boldsymbol{a}}, \overline{r}) \tag{22}$$

$$B = \mathsf{com}(\overline{\boldsymbol{b}}, \overline{s}) \tag{23}$$

$$D = \mathsf{com}((\overline{\boldsymbol{a}} \circ \boldsymbol{y}) \cdot \overline{\boldsymbol{b}}, \overline{t}) \tag{24}$$

for commitments A, B, D, vectors $\overline{\boldsymbol{a}}, \overline{\boldsymbol{b}}, \boldsymbol{y}$ and associated randomness $\overline{r}, \overline{s}, \overline{t}$ generated during interaction. Our argument for the zero-relation is identical to the one presented by Bayer [8] with the exception that in the final round of Bayer's original argument the prover directly sends $\overline{\boldsymbol{a}}$, $\overline{r}$, $\overline{\boldsymbol{b}}$, $\overline{s}$ and $\overline{t}$ for the verifier to check. In our variant the verifier instead outsources this final check using an argument system for inner-product:

$\langle\mathsf{Prover}, \mathsf{Verifier}\rangle$: The prover and verifier are provided with the statement consisting of lists of commitments $\boldsymbol{A}, \boldsymbol{B}$ and scalar y. The prover is additionally provided with witness $(\{\boldsymbol{a}_i\}_{i\in[m]}, \{r_i\}_{i\in[m]}, \{\boldsymbol{b}_{i-1}\}_{i\in[m]}, \{s_{i-1}\}_{i\in[m]})$.

1. The prover starts the argument by sampling blinding vectors $\boldsymbol{a}_0, \boldsymbol{b}_m \overset{\$}{\leftarrow} \mathbb{F}^n$ and associated randomness r_0 and s_m. For $k \in \{0, \ldots, 2m\}$, the prover computes
$$d_k = \sum_{\substack{0\le i,j\le m\\ j=(m-k)+i}} (\boldsymbol{a_i} \circ \boldsymbol{y}) \cdot \boldsymbol{b_j}$$
where $\boldsymbol{y} = (y, y^2, \ldots, y^n)$. Next the prover samples randomness $t_0, \ldots, t_{2m+1} \overset{\$}{\leftarrow} \mathbb{F}$ and computes commitments
$$D_0 = \mathsf{com}(d_0; t_0), \ldots, D_{2m} = \mathsf{com}(d_{2m}; t_{2m}).$$
Finally the prover sends A_0, B_m and $D_0, \ldots, D_{2m}$ to the verifier.
2. The verifier responds with challenge $x \overset{\$}{\leftarrow} \mathbb{F}$.
3. Using x the prover computes a random linear combination of the witness vectors:
$$\overline{\boldsymbol{a}} = \sum_{i=0}^{m} x^i \boldsymbol{a}_i \qquad \overline{r} = \sum_{i=0}^{m} x^i r_i$$
$$\overline{\boldsymbol{b}} = \sum_{i=0}^{m} x^i \boldsymbol{b}_i \qquad \overline{s} = \sum_{i=0}^{m} x^i s_i$$
$$\overline{d} = \sum_{i=0}^{2m} x^i d_i \qquad \overline{t} = \sum_{i=0}^{2m} x^i t_i.$$
4. Both the prover and the verifier compute the statement commitments
$$A = \prod_{i=0}^{m} A_i^{x^i} \qquad B = \prod_{i=0}^{m} B_i^{x^i} \qquad D = \prod_{i=0}^{2m} D_i^{x^i}$$
5. The verifier checks that $D_{m+1} = \mathsf{com}(0; 0)$.

6. Finally both the prover and verifier engage in an inner-product argument to convince the verifier that
$$\overline{d} = (\overline{\boldsymbol{a}} \circ \boldsymbol{y}) \cdot \overline{\boldsymbol{b}}$$

Lemma 4. *Construction 10 is an argument system for* $\mathcal{R}_{\mathsf{MHADM}}$ *that satisfies knowledge soundness.*

Proof. By Bayer [8, Theorem 20] the core multi-Hadamard argument is knowledge sound so long as the underlying zero-argument is knowledge sound. [2] We prove the knowledge soundness of our variant of the zero-argument using the knowledge soundness property of Bayer's original zero-argument [8, Theorem 21] and the knowledge soundness property of the underlying inner-product argument.

In particular, given arbitrary prover $\mathcal{P}^*$ we must construct an extractor $\mathcal{E}$ such that for arbitrary statement consisting of vectors of commitments $\boldsymbol{A}$, $\boldsymbol{B}$, and scalar y, if

$$\langle \mathcal{P}^*(\boldsymbol{A}, \boldsymbol{B}, y; \rho), \mathcal{V}(\boldsymbol{A}, \boldsymbol{B}, y) \rangle = 1$$

Then $\mathcal{E}(\boldsymbol{A}, \boldsymbol{B}, y; \rho)$ produces a witness consisting of lists of vectors $\{\boldsymbol{a}_i\}_{i\in[m]}$, $\{\boldsymbol{b}_{i-1}\}_{i\in[m]}$, and associated lists of randomness $\{r_i\}_{i\in[m]}$, and $\{s_{i-1}\}_{i\in[m]}$ such that

$$A_i = \mathsf{com}(\boldsymbol{a}_i, r_i) \quad \forall i \in [m], \tag{25}$$
$$B_{i-1} = \mathsf{com}(\boldsymbol{b}_i, s_i) \quad \forall i \in [m], \tag{26}$$
$$0 = \sum_{i\in[m]} (\boldsymbol{a}_i \circ \boldsymbol{y}) \cdot \boldsymbol{b}_{i-1} \tag{27}$$

with probability $1 - \mathsf{negl}(\lambda)$ Using $\mathcal{P}^*$ we construct a malicious prover for Bayer's zero-argument $\mathcal{P}^*_Z$ which succeeds in convincing the corresponding verifier $\mathcal{V}_Z$ with the same probability. $\mathcal{E}$ can then use $\mathcal{P}^*_Z$ to extract a valid witness. In more detail, $\mathcal{P}^*_Z$ behaves exactly like $\mathcal{P}^*$ with the exception of the final round. In the final round $\mathcal{P}^*$ engages in an inner product argument with $\mathcal{V}$ over the statement (A, B, D, y). If $\mathcal{P}^*$ succeeds, by the knowledge soundness of the inner product argument $\mathcal{P}^*_Z$ can extract $\overline{\boldsymbol{a}}, \overline{r}, \overline{\boldsymbol{b}}, \overline{s}, \overline{t}$ such that equations 22, 23, 24 hold. Thus in the final round $\mathcal{P}^*_Z$ can respond with these extracted terms to successfully convince $\mathcal{V}^*_Z$ that it holds a valid witness to the statement for Bayer's zero-argument. Then by the knowledge-soundness of Bayer's zero-argument, there exists an extractor $\mathcal{E}_Z$ that can extract lists of vectors $\{\boldsymbol{a}_i\}_{i\in[m]}$, $\{\boldsymbol{b}_{i-1}\}_{i\in[m]}$, and associated lists of randomness $\{r_i\}_{i\in[m]}$, and $\{s_{i-1}\}_{i\in[m]}$ such that equations 25, 26, 27 hold with probability $1 - \mathsf{negl}(\lambda)$. Thus $\mathcal{E}$ can use $\mathcal{E}_Z$ to extract a valid witness with probability $1 - \mathsf{negl}(\lambda)$. □

[2] More precisely, Bayer's constructions are proven to be generalized special sound [8]. By the forking lemma [29], this implies witness-extended emulation [54, 60] which trivially implies our notion of knowledge soundness.

Lemma 5. *Construction 10 is an argument system for $\mathcal{R}_{\mathsf{MHADM}}$ that is honest-verifier zero-knowledge*

Proof. This follows by the honest-verifier zero-knowledge property of the multi-Hadamard-product argument by Bayer [8, Theorem 20] and the honest-verifier zero-knowledge property of the inner-product argument for the Lagrange basis (Lemma 16). □

Lemma 6 (Efficiency). *For m vectors of size n, construction 10 features an $O(n)$ generator, $O(nm^2)$ prover, and an $O(\log n + m)$ verifier.*

Proof. This follows by the properties discussed by Bayer and the asymptotics of the inner-product argument system (Construction 4). We provide more detail for the dominating concrete costs:

- *Generator:* The generator's cost consists of creating the structured reference string for the commitment scheme. When instantiated with our polynomial commitment scheme, the generators work is dominated by two multi-exponentiations of size n which can be done in $O(n/\log n)$ exponentiations by Pippenger's algorithm [66].
- *Prover:* To compute $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_m$ the prover will need to compute nm field multiplications. To compute $B_2, \ldots, B_{m-1}$ the prover will need to compute m multi-exponentiations of size n. To compute $\boldsymbol{d}_i$ for $i \in [m-1]$ and d the prover will need to compute $2nm$ field multiplications. In the zero argument, to compute d_k for $i \in \{0, \ldots, 2m\}$ the prover will need to compute $(m+1)^2 \cdot n$ field multiplications. To compute $D_0, \ldots, D_{2m}$ the prover will need to compute $4m$ exponentiations. To compute $\overline{\boldsymbol{a}}$ and $\overline{\boldsymbol{b}}$ the prover will need to compute $2mn$ field multiplications. Finally the prover engages in an inner-product argument over $\overline{\boldsymbol{a}}$, and $\overline{\boldsymbol{b}}$ which by Lemma 14 can be done in $O(n)$ time.
- *Verifier:* In the main argument, to compute D_i for $i \in [m]$ the verifier will need to compute m exponentiations. In the zero argument, to compute A, B and D the verifier will need to compute $4m$ exponentiations total. Finally the verifier engages in an inner-product argument over commitments A, B and D which by Lemma 14 can be done in $O(\log n)$ time.

□

C Reducing Arithmetic Circuit Satisfiability to Algebraic Constraint Satisfiability

Definition 15 (Arithmetic Circuit Satisfiability Relation). *The Arithmetic Circuit Satisfiability Relation ($\mathcal{R}_{\mathsf{CSAT}}$) defined over field $\mathbb{F}$, input size m, witness size q, output size p, and constraint size n, consists of arithmetic circuit $\mathcal{C} : \mathbb{F}^{m+q} \to \mathbb{F}^p$ consisting of n gates, and vectors $\boldsymbol{x} \in \mathbb{F}^m$, $\boldsymbol{y} \in \mathbb{F}^p$. A witness $\boldsymbol{w} \in \mathbb{F}^q$ satisfies an $\mathcal{R}_{\mathsf{CSAT}}$ instance if $\mathcal{C}(\boldsymbol{x}, \boldsymbol{w}) = \boldsymbol{y}$*

Lemma 7. *Any arithmetic circuit satisfiability instance over a circuit $\mathcal{C}$ with n gates can be reduced to a sparse $\mathcal{R}_{\mathsf{ACS}}$ instance of size $n+1$ (Definition 14).*

Proof. Consider arithmetic circuit $\mathcal{C}$ with n canonically ordered gates, with public input vector $\boldsymbol{x}'$, output vector $\boldsymbol{y}'$, and witness vector $\boldsymbol{w}'$. We create an $\mathcal{R}_{\mathsf{ACS}}$ as follows:

1. Let $\boldsymbol{x} = (1, \boldsymbol{x}', \boldsymbol{y}')$.
2. Including $\boldsymbol{w}'$, let $\boldsymbol{w}$ be a canonically vector of all the computed gate values in $\mathcal{C}$.
3. Create $(n+1) \times (n+1)$ matrices M_i for $i \in [n]$ instantiated with zeros.
4. For each gate i let j and k be the indices of its left and right input gates. If gate i is a multiplication gate set $M_i[j,k] = 1$ and $M_i[i,1] = -1$. Otherwise, if gate i is an addition gate set $M_i[j,1] = 1$, $M_i[k,1] = 1$, and $M_i[i,1] = -1$
5. Let the $\mathcal{R}_{\mathsf{ACS}}$ instance be $(\{M_i\}_{i \in n}, \boldsymbol{x}; \boldsymbol{w})$

Intuitively each matrix M_i captures the constraint of gate i. In more detail, letting $\boldsymbol{z} = (\boldsymbol{x}, \boldsymbol{w})$, we observe that if i is a multiplication gate

$$\boldsymbol{z}\boldsymbol{M}_i\boldsymbol{z}^\top = z_j \cdot z_k + (z_i \cdot -1).$$

Similiarly if i is an addition gate

$$\boldsymbol{z}\boldsymbol{M}_i\boldsymbol{z}^\top = z_j \cdot 1 + z_k \cdot 1 + (z_i \cdot -1)$$

Thus $\boldsymbol{w}$ satisfies the constructed $\mathcal{R}_{\mathsf{ACS}}$ instance if and only if it satisfies the provided arithmetic circuit satisfiability instance. By design, each matrix is of size $(n+1) \times (n+1)$ therefore the resulting $\mathcal{R}_{\mathsf{ACS}}$ size is $(n+1)$. Additionally there are a total of n matrices and each matrix M_i contains either 2 non-zero values if i is a multiplication gate, or 3 non-zero values if i is an addition gate. Therefore there are a total of $O(n)$ non-zero values in all n matrices, implying that the constructed $\mathcal{R}_{\mathsf{ACS}}$ instance is sparse. □

D Deferred Proofs

D.1 Proof of Corollary 1 (n-EPKE for a Linearly Independent Basis)

Proof. Suppose we have adversary $\mathcal{A}$ that outputs (A, A') such that $e(A, g^\alpha) = e(A', g)$. To prove Corollary 1 we must a construct a PPT extractor $\mathcal{E}$ that outputs $a_0, \ldots, a_n, b$ such that

$$A = \big(\prod_{i=0}^{n} g^{p_i(s) \cdot a_i}\big) \cdot g^{t \cdot b}$$

with probability $1 - \mathsf{negl}(\lambda)$. In particular, $\mathcal{E}$ initially computes vectors

$$\boldsymbol{u}' = (g, g^s, \ldots, g^{s^n}, g^t)$$

$$\boldsymbol{v}' = (g^{\alpha}, g^{\alpha s}, \ldots, g^{\alpha s^n}, g^{\alpha t}).$$

This can be done efficiently because the monomials $X^0, \ldots, X^n$ are spanned by polynomials $p_0(X), \ldots, p_n(X)$ due to their linear independence, and therefore the terms $s^0, \ldots, s^n$ are a linear combination of the terms $p_0(s), \ldots, p_n(s)$. By the n-EPKE assumption (Assumption 4), there exists PPT extractor $\mathcal{E}'$ that on input $((\mathbb{F}, \mathbb{G}, \mathbb{G}_T, e), \boldsymbol{u}', \boldsymbol{v}')$ outputs $a'_0, \ldots, a'_n, b$ such that

$$A = \big(\prod_{i=0}^{n} g^{s^i \cdot a'_i}\big) \cdot g^{t \cdot b}$$

with probability $1 - \mathsf{negl}(\lambda)$. By interpreting $a'_0, \ldots, a'_n$ as coefficients of degree n polynomial a, we have that

$$A = g^{a(s)} g^{tb}$$

Once again, by the linear independence of polynomials $p_0(X), \ldots, p_n(X)$, degree n polynomial $a(X)$ can be represented as a linear combination of $p_0(X), \ldots, p_n(X)$. Let this linear combination be $a_0, \ldots, a_n$. This means that

$$a(s) = a_0 p_0(s) + \ldots + a_n p_n(s)$$

and therefore

$$A = g^{a_0 p_0(s) + \ldots + a_n p_n(s)} g^{tb}.$$

Thus, $\mathcal{E}$ can return $a_0, \ldots, a_n, b$. □

D.2 Proof of Lemma 2 (Structured Polynomial Commitments)

Proof. We prove the desired properties of our structured polynomial commitment scheme:

Homomorphic: Construction 1 is homomorphic because

$$\begin{aligned}\mathsf{com}(\boldsymbol{p}; r_p) \cdot \mathsf{com}(\boldsymbol{p}; r_p) &= (g^{\boldsymbol{p}(s)} \cdot h^{r_p}, g^{\alpha \boldsymbol{p}(s)} \cdot h^{\alpha r_p}) \cdot (g^{\boldsymbol{q}(s)} \cdot h^{r_q}, g^{\alpha \boldsymbol{q}(s)} \cdot h^{\alpha r_q}) \\ &= (g^{\boldsymbol{p}(s) + \boldsymbol{q}(s)} \cdot h^{r_p + r_q}, g^{\alpha(\boldsymbol{p}(s) + \boldsymbol{q}(s))} \cdot h^{\alpha(r_p + r_q)}) \\ &= \mathsf{com}(\boldsymbol{p} + \boldsymbol{q}; r_p + r_q)\end{aligned}$$

Unconditional Hiding: To prove unconditional hiding we observe that arbitrary P_1 is indistinguishable from a random element $R \xleftarrow{\$} \mathbb{G}$ due to the h^r term. Because $P_2 = (P_1)^{\alpha}$, we have that (P_1, P_2) is indistinguishable from (R, R^{α}).

Extractibility: Extractibility follows directly from the $(n-1)$-EPKE assumption and corollary 1.

Computational Binding: To prove computational binding, suppose there exists an adversary $\mathcal{A}$ that outputs two vectors of polynomial evaluations $\boldsymbol{p}$ and $\boldsymbol{q}$ and associated randomness r_p and r_q such that $\boldsymbol{p} \neq \boldsymbol{q}$ but

$$\mathsf{com}(\boldsymbol{p}; r_p) = \mathsf{com}(\boldsymbol{q}; r_q).$$

with non-negligible probability δ.

Then we can construct adversary $\mathcal{B}$ that breaks the $(n-1)$-SDH assumption with non-negligible probability: Suppose $\mathcal{B}$ is provided challenge $(g, g^{\sigma}, \dots, g^{\sigma^{n-1}})$. $\mathcal{B}$ begins by picking random bit $b \xleftarrow{\$} \{0,1\}$ and proceeds as follows:

- If $b = 0$: Let $h = g^{\sigma}$. Sample secret $s \xleftarrow{\$} \mathbb{F}$ and generate the rest of the structured reference string accordingly.
- If $b = 1$: Use terms $(g, g^{\sigma}, \dots, g^{\sigma^{n-1}})$ to efficiently compute commitment keys

  $$\boldsymbol{u} = (g^{\ell_1(\sigma)}, \dots, g^{\ell_n(\sigma)})$$
  $$\boldsymbol{v} = (g^{\alpha\ell_1(\sigma)}, \dots, g^{\alpha\ell_n(\sigma)})$$

  Note that this implicitly sets $s = \sigma$. Generate the rest of the structured reference string accordingly.

$\mathcal{B}$ runs $\mathcal{A}$ with this reference string, receives $(\boldsymbol{p}, \boldsymbol{q}, r_p, r_q)$, and aborts if $\boldsymbol{p} = \boldsymbol{q}$. Because $\mathsf{com}(\boldsymbol{p}; r_p) = \mathsf{com}(\boldsymbol{q}; r_q)$, we have

$$g^{\boldsymbol{p}(s)+\lambda r_p} = g^{\boldsymbol{q}(s)+\lambda r_q} \tag{28}$$

We now have one of two cases: Either $\boldsymbol{p}(s) \neq \boldsymbol{q}(s)$ or $\boldsymbol{p}(s) = \boldsymbol{q}(s)$. We consider both cases.

- Suppose $\boldsymbol{p}(s) \neq \boldsymbol{q}(s)$: If $b = 0$, we have that $g^{\sigma} = h$ by design. Because $\boldsymbol{p}(s) \neq \boldsymbol{q}(s)$ by equation 28, we have that $r_p \neq r_q$. Thus $\mathcal{B}$ can compute

  $$\sigma = \frac{\boldsymbol{q}(s) - \boldsymbol{p}(s)}{r_p - r_q}$$

  and output $(0, e(g,g)^{\frac{1}{\sigma}})$ breaking the $(n-1)$-SDH assumption.
- Suppose $\boldsymbol{p}(s) = \boldsymbol{q}(s)$: If $b = 1$, we have $s = \sigma$ by design. Now consider the polynomial

  $$T(X) = \boldsymbol{p}(X) - \boldsymbol{q}(X).$$

  Because $\boldsymbol{p}(s) = \boldsymbol{q}(s)$, we have that $\sigma = s$ is a root of T. Additionally, T is not the zero polynomial by assumption that $\boldsymbol{p} \neq \boldsymbol{q}$. Thus $\mathcal{B}$ can solve for the roots of T and find σ that agrees with the provided challenge. $\mathcal{B}$ can then output $(0, e(g,g)^{\frac{1}{\sigma}})$ breaking the $(n-1)$-SDH assumption.

We now analyze $\mathcal{B}$'s success probability. Let δ be the probability that $\mathcal{A}$ successfully outputs $(\boldsymbol{p}, \boldsymbol{q}, r_p, r_q)$ such that $\mathsf{com}(\boldsymbol{p}; r_p) = \mathsf{com}(\boldsymbol{q}; r_q)$ but $\boldsymbol{p} \neq \boldsymbol{q}$. Let $\delta = \delta_1 + \delta_2$ where δ_1 is the probability that $\mathcal{A}$ wins with $\boldsymbol{p}(s) \neq \boldsymbol{q}(s)$ and δ_2 is the probability that $\mathcal{A}$ wins with $\boldsymbol{p}(s) = \boldsymbol{q}(s)$. $\mathcal{B}$ succeeds when $\boldsymbol{p}(s) \neq \boldsymbol{q}(s)$ and $b = 0$, or when $\boldsymbol{p}(s) = \boldsymbol{q}(s)$ and $b = 1$. For randomly chosen b, $\Pr[b = 0] = \Pr[b = 1] = 1/2$. Therefore the probability that $\mathcal{B}$ succeeds is $\delta_1/2 + \delta_2/2 = \delta/2$. Therefore if δ is a non-negligible probability, then $\mathcal{A}$ succeeds in breaking the $(n-1)$-SDH assumption with non-negligible probability. □

Lemma 8 (Efficiency). *For polynomials defined by n evaluation points, $\mathcal{G}$ takes time $O(n)$,* com *takes time $O(n)$,* $\mathsf{checkcom}$ *takes time $O(1)$.*

Proof. $\mathcal{G}$ can compute $\ell_1(s), \ldots, \ell_n(s)$ in $O(n)$ time using the Barycentric representation [20]. Next $\mathcal{G}$ can compute $\boldsymbol{u}$ and $\boldsymbol{v}$ using two multi-exponentiations of size n which can be done in $O(n/\log n)$ exponentiations by Pippenger's algorithm [66]. com requires computing P_1 and P_2 which can be done using a multi-exponentiation of size n for each. $\mathsf{checkcom}$ requires two pairings which can be done in $O(1)$ time. □

D.3 Proof of Theorem 1 (Polynomial Evaluation Argument)

Lemma 9. *Construction 2 satisfies knowledge soundness.*

Proof. Given an arbitrary PPT prover $\mathcal{P}^*$, we must construct PPT extractor $\mathcal{E}$ such that for an arbitrary statement $(P \in \mathbb{G}^2, Y \in \mathbb{G}, u \in \mathbb{F})$, if

$$\langle \mathcal{P}^*(P, Y, u; \rho), \mathcal{V}(P, Y, u) \rangle = 1$$

then $\mathcal{E}(P, Y, u; \rho)$ produces a witness $(\boldsymbol{p} \in \mathbb{F}^n, y \in \mathbb{F}, r_p, r_y \in \mathbb{F})$ such that

$$\begin{aligned} P &= \mathsf{com}(\boldsymbol{p}; r_p) \\ Y &= \mathsf{com}(y; r_y) \\ y &= \boldsymbol{p}(u). \end{aligned}$$

Given a successful $\mathcal{P}^*$, we construct $\mathcal{E}$ that extracts a valid witness with probability $1 - \mathsf{negl}(\lambda)$:

Because the all verifier's checks have passed, commitments P, Q, Y, and R are well-formed. Therefore, by Lemma 2, $\mathcal{E}$ can extract evaluation vectors $\boldsymbol{p} = (p_1, \ldots, p_n)$, $\boldsymbol{q} = (q_1, \ldots, q_n)$, y, and $\boldsymbol{r} = (r_1, \ldots, r_n)$ along with associated randomness r_p, r_q, r_y, and r_r such that

$$\begin{aligned} P_1 &= \mathsf{com}(\boldsymbol{p}; r_p) = \boldsymbol{u}^{\boldsymbol{p}} \cdot h^{r_p} = g^{\boldsymbol{p}(s)} h^{r_p} \\ Q_1 &= \mathsf{com}(\boldsymbol{q}; r_q) = \boldsymbol{u}^{\boldsymbol{q}} \cdot h^{r_q} = g^{\boldsymbol{q}(s)} h^{r_q} \\ Y_1 &= \mathsf{com}(y; r_y) = g^y h^{r_y} \\ R_1 &= \mathsf{com}(\boldsymbol{r}; r_r) = \boldsymbol{u}^{\boldsymbol{r}} \cdot h^{r_r} = g^{\boldsymbol{r}(s)} h^{r_r} \end{aligned}$$

with probability $1-\mathsf{negl}(\lambda)$. While the extractor has extracted all of the material required to construct a witness $(\boldsymbol{p}, y, r_p, r_y)$, we must still show that $\boldsymbol{p}(u) = y$.

Suppose, for contradiction, $\boldsymbol{p}(u) \neq y$ with non-negligible probability. Then we can construct adversary $\mathcal{A}$ that uses $\mathcal{P}^*$ and $\mathcal{E}$ to break the $(n-1)$-BSDH assumption. Suppose $\mathcal{A}$ is provided with challenge $(g, g^{\sigma}, \ldots, g^{\sigma^{n-1}})$. $\mathcal{A}$ picks a random bit $b \overset{\$}{\leftarrow} \{0,1\}$ and proceeds as follows:

- If $b = 0$: Let $h = g^{\sigma}$. Sample secret $s \overset{\$}{\leftarrow} \mathbb{F}$ and generate the rest of the structured reference string accordingly.
- If $b = 1$: Use terms $(g, g^{\sigma}, \ldots, g^{\sigma^{n-1}})$ to efficiently compute commitment keys

$$\boldsymbol{u} = (g^{\ell_1(\sigma)}, \ldots, g^{\ell_n(\sigma)})$$
$$\boldsymbol{v} = (g^{\alpha\ell_1(\sigma)}, \ldots, g^{\alpha\ell_n(\sigma)}).$$

  Note that this implicitly sets $s = \sigma$. Generate the rest of the structured reference string accordingly.

Next $\mathcal{A}$ runs $\mathcal{P}^*$ and provided $\mathcal{P}^*$ is successful, runs the extractor $\mathcal{E}$ described above. If $\boldsymbol{p}(u) = y$, abort. Otherwise, because the verifier's final check has passed we have

$$e(P_1/Y, g) = e(Q_1, g^{s-u})e(R_1, h).$$

Letting $g^t = h$, for some unknown t, we have

$$e(g,g)^{\boldsymbol{p}(s)+tr_p-(y+tr_y)} = e(g,g)^{(\boldsymbol{q}(s)+tr_q)(s-u)+t(\boldsymbol{r}(s)+tr_r)}. \tag{29}$$

Aggregating terms with respect to t, we get

$$e(g,g)^{(\boldsymbol{p}(s)-y)+t(r_p-y)} = e(g,g)^{(\boldsymbol{q}(s)(s-u))+t((\boldsymbol{r}(s)+tr_r)+r_q(s-u))}. \tag{30}$$

We now have one of two cases: Either $\boldsymbol{p}(s) - y \neq \boldsymbol{q}(s)(s-u)$ or $\boldsymbol{p}(s) - y = \boldsymbol{q}(s)(s-u)$. We consider both cases.

- Suppose $\boldsymbol{p}(s) - y \neq \boldsymbol{q}(s)(s-u)$: If $b = 0$, we have that $t = \sigma$ by design. Now consider the polynomial

$$T(X) = X((\boldsymbol{r}(s) + Xr_r) + r_q(s-u) - (r_p - y)) + (\boldsymbol{q}(s)(s-u)) - (\boldsymbol{p}(s) - y))$$

  By equation 30 we have that σ is a root of T. Additionally because $\boldsymbol{p}(s)-y \neq \boldsymbol{q}(s)(s-u)$ we have that T is not the zero polynomial. Thus $\mathcal{A}$ can solve for the roots of T and find σ that agrees with the provided challenge. $\mathcal{A}$ can then output $(0, e(g,g)^{\frac{1}{\sigma}})$ breaking the $(n-1)$-BSDH assumption.
- Suppose instead $\boldsymbol{p}(s) - y = \boldsymbol{q}(s)(s-u)$: If $b = 1$, we have $s = \sigma$ by design. Now consider the polynomial

$$T(X) = (\boldsymbol{p}(X) - y) - \boldsymbol{q}(X)(X-u)$$

Because $\boldsymbol{p}(s) - y = \boldsymbol{q}(s)(s-u)$, $\sigma = s$ is a root of T. Additionally, T is not the zero polynomial because then otherwise we would have

$$0 = T(u) = \boldsymbol{p}(u) - y - \boldsymbol{q}(u)(u-u) = \boldsymbol{p}(u) - y$$

which contradicts the assumption that $\boldsymbol{p}(u) \neq y$. Thus $\mathcal{A}$ can solve for the roots of T and find σ that agrees with the provided challenge. $\mathcal{A}$ can then output $(0, e(g,g)^{\frac{1}{\sigma}})$ breaking the $(n-1)$-BSDH assumption.

We now analyze $\mathcal{A}$'s success probability. Let δ be the probability that $\mathcal{P}^*$ successfully convinces the verifier but the extractor $\mathcal{E}$ outputs $(\boldsymbol{p}, y)$ such that $\boldsymbol{p}(u) \neq y$. Let $\delta = \delta_1 + \delta_2$ where δ_1 is the the probability $\mathcal{P}^*$ wins with $\boldsymbol{p}(s) - y \neq \boldsymbol{q}(s)(s-u)$, and δ_2 is the probability $\mathcal{P}^*$ wins with $\boldsymbol{p}(s) - y = \boldsymbol{q}(s)(s-u)$. $\mathcal{A}$ succeeds when $\boldsymbol{p}(s) - y \neq \boldsymbol{q}(s)(s-u)$ and $b = 0$, or when $\boldsymbol{p}(s) - y = \boldsymbol{q}(s)(s-u)$ and $b = 1$. For randomly chosen b, $\Pr[b=0] = \Pr[b=1] = 1/2$. Therefore the probability that $\mathcal{A}$ succeeds is $\delta_1/2 + \delta_2/2 = \delta/2$. Therefore if δ is a non-negligible probability, then $\mathcal{A}$ succeeds in breaking the $(n-1)$-BSDH assumption with non-negligible probability. □

Lemma 10. *Construction 2 is perfect zero-knowledge.*

Proof. To prove perfect zero-knowledge we must construct a PPT simulator $\mathcal{S}$ that can simulate a transcript indistinguishable from one generated by an honest prover for any given statement with a valid witness. We construct $\mathcal{S}$ as follows:

Consider arbitrary statement *with a valid witness* consisting of polynomial commitment P, commitment Y, and evaluation point u. We argue that $\mathcal{S}$ can compute Q and R *exactly* as dictated by the protocol using only P, Y, u, and the trapdoor:

First, $\mathcal{S}$ generates the common reference string, and the associated trapdoor (α, s, t) where $g^t = h$.

Next, $\mathcal{S}$ samples $r'_q \leftarrow \mathbb{F}$ and computes Q_1 as follows:

$$\begin{aligned} Q_1 &= (P_1/Y_1)^{1/(s-u)} \cdot h^{r'_q} \\ &= g^{\frac{p(s)+tr_p-y-tr_y}{(s-u)}+tr'_q} \\ &= g^{q(s)+t(\frac{r_p-r_y}{s-u}+r'_q)} \end{aligned}$$

where the last equality holds because an honest prover computes $q(s) = (p(s) - y)/(s-u)$. We can set $r_q = (r_p - r_y)(s-u) + r'_q$ and rewrite

$$Q_1 = g^{q(s)+tr_q} = g^{q(s)}h^{r_q}.$$

We note that r_q is indistinguishable from a random element in $\mathbb{F}$ in both the real and ideal settings. $\mathcal{S}$ computes $Q_2 = Q_1^{\alpha}$.

Next, $\mathcal{S}$ computes R_1 as follows:

$$R_1 = (P_1/Y_1/Q_1^{(s-u)})^{1/t}$$

$$
\begin{aligned}
&= g^{(1/t)(p(s)+tr_p)-(1/t)(y+tr_y)-(1/t)(q(s)+tr_q)(s-u)} \\
&= g^{(1/t)(p(s)-y-q(s)(s-u))+(r_p-r_y-r_q(s-u))} \\
&= g^{r_p-r_y-r_q(s-u)}
\end{aligned}
$$

where the last equality holds because $p(s) - y - q(s)(s - u) = 0$ due to the fact that $q(s) = \frac{p(s)-y}{(s-u)}$. $\mathcal{S}$ computes $R_2 = R_1^{\alpha}$.

Thus, because Q and R satisfy identical relations in both the real and ideal setting, an unbounded adversary cannot distinguish between real and ideal transcripts.

□

Lemma 11 (Efficiency). *For polynomials defined over n evaluations, the polynomial evaluation argument features an $O(n)$ generator, $O(n)$ prover, and an $O(1)$ verifier.*

Proof. We break down the dominating costs for each of the components:

- *Generator:* $\mathcal{G}$ can compute $\ell_1(s), \ldots, \ell_n(s)$ in $O(n)$ time using the Barycentric representation [20].
- *Prover:* Because $\boldsymbol{p}$ represents a vector of polynomial evaluations, $\mathcal{P}$ can compute the vector of polynomial evaluations $\boldsymbol{q}$ in $O(n)$ time. Additionally, using commitment keys $\boldsymbol{u}$ and $\boldsymbol{v}$, $\mathcal{P}$ can compute Q with two size n multi-exponentiations.
- *Verifier:* The verifier's final check is dominated by computing three pairings which can be done in $O(1)$ time.

□

D.4 Proof of Theorem 2 (Inner-Product Argument)

Lemma 12. *Construction 3 is an argument system for $\mathcal{R}_{\mathsf{IP}}$ (Definition 12) that satisfies knowledge-soundness.*

Proof. Construction 3 is derived by applying commitment scheme com (Equation 2) to the generalized inner-product argument presented by Bünz et al. [31]. To be compatible with the generalized inner-product argument we must show that com is doubly homomorphic (i.e. is homomorphic in both the message space and the key space) and binding. com is doubly homomorphic by observation. Additionally com is binding by the q-SDH assumption and reasoning similiar to Lemma 2. Therefore com is compatible with the generalized inner-product argument presented by Bünz et al. [31]. Thus the main interaction has knowledge soundness due Bünz et al. [31, Theorem 5.4] so long as the subprotocol to convince the verifier that (v, w) are computed correctly is sound and the product relation holds is knowledge sound.

The soundness of the (v, w) argument holds by the Schwartz-Zippel Lemma and the soundness of the polynomial evaluation argument (Construction 2). In particular we consider the case of v (the case for w is symmetric): The soundness of the polynomial evaluation argument ensures that v is of the form $v = g_1^{f(\alpha)}$ for some degree $n-1$ polynomial f. Additionally the the polynomial evaluation argument ensures that $f(z) = f_v(z)$. Because z is a random challenge provided by the verifier, $f = f_v$ with probability $1 - \mathsf{negl}(\lambda)$ by the Schwartz-Zippel lemma [69].

The final product argument a simplified version of a product argument presented by Bünz et al. [29]. Thus the final argument is knowledge sound by an argument similiar to Bünz et al. [29, Theorem 3]. □

Lemma 13. *Construction 3 is an argument system for $\mathcal{R}_{\mathsf{IP}}$ that is honest-verifier zero-knowledge.*

Proof. To prove honest-verifier zero-knowledge we must construct simulator $\mathcal{S}$ that can simulate a transcript indistinguishable from one generated by an honest interaction for any given statement with a valid witness. We construct $\mathcal{S}$ as follows:

Consider arbitrary statement *with a valid witness* consisting of commitments A, B, C and scalar r. The simulator $\mathcal{S}$ sets $\boldsymbol{a}$, $\boldsymbol{b}$ to be $\mathbf{0}$ and simulates an interaction between an honest prover and honest verifier for the main argument. In each round in both the real and ideal setting, the prover's messages $A_L, A_R, B_L, B_R, C_L, C_R$ are indistinguishable from random and independent of each other due to the blinding terms.

Additionally the subprotocol for checking v and w is not witness dependent so can be simulated by running the honest prover.

Thus, it suffices to show that the $\mathcal{S}$ can simulate the final proof-of-product protocol with a statement consisting of commitments A, B, C and unknown openings. $\mathcal{S}$ sets $a = b = c = 0$ and randomly samples $r_a, r_b, r_c \overset{\$}{\leftarrow} \mathbb{F}$. Next $\mathcal{S}$ simulates the honest prover and verifier. However instead of computing S and T_1 as dictated by the protocol, $\mathcal{S}$ uses the simulated challenge z to forge these terms to satisfy the verifier's checks:

$$S = (v^{a'} w^{b'} h^{\mu} / A / B)^{\frac{1}{z}}$$
$$T_1 = (g^{c'} h^{\tau} / C / T_2^{z^2})^{\frac{1}{z}}$$

Because $\tau_1, \tau_2, \rho, z, s_a, s_b$ are randomly sampled this implies that the provers message τ, μ, a', b', c' is indistinguishable from random elements in $\mathbb{F}$ in both the real and ideal settings. Additionally this implies that T_2 is indistinguishable from a random element in $\mathbb{G}$ in both the real and the ideal setting. Finally S and T_1 are uniquely fixed by elements which are indistinguishable from random in both the real and the ideal setting. Therefore, an adversary cannot distinguish between the real and ideal transcripts. □

Lemma 14 (Efficiency). *For vectors of size n construction 3 features an $O(n)$ generator, $O(n)$ prover, and an $O(\log n)$ verifier.*

Proof. This follow by properties discussed in Bünz et al. [31]. We provide more detail for the generalized inner-product argument instantiated with the Pedersen commitment:

- *Generator:* $\mathcal{G}$ can compute $\boldsymbol{w}$ with a multi-exponentiation of size n which can be done in $O(n/\log n)$ exponentiations by Pippenger's algorithm [66].
- *Prover:* The prover can compute $\boldsymbol{r}$ and rescale $\boldsymbol{a}$ in $O(n)$ time. In each recursive round the prover's work is dominated by the cost of exponentiations. In total to compute $A_L, A_R, B_L, B_R, \boldsymbol{v'}, \boldsymbol{w'}$ in all rounds the prover incurs $6n$ exponentiations: $3n$ in the first round, $3n/2$ in the second round, and so on.
  In the subprotocol to check (v, w), the prover can evaluate v' and w' using two multi-exponentiations of size n.
  Similiarly the prover can compute $f_v(z)$ and $f_w(z)$ in a logarithmic number of multiplications. The prover can prove the validity of V and W using two polynomial evaluation arguments which incurs $O(n)$ overhead (Lemma 11)
  The subprotocol to check product requires $O(1)$ operations on the prover's end by observation.
- *Verifier:* To compute A', B', C' over all the rounds the verifier performs $6 \log n$ exponentiations.
  In the subprotocol to check (v, w), the verifier can compute $f_v(z)$ and $f_w(z)$ in a logarithmic number of multiplications. Checking the two resulting polynomial evaluation arguments incurs $O(1)$ overhead (Lemma 11)
  By observation, the subprotocol to check product requires $O(1)$ operations on the verifier's end.

□

D.5 Proof of Theorem 3 (Inner-Product Argument for the Lagrange Basis)

Lemma 15. *Construction 4 is an argument system for* $\mathcal{R}_{\mathsf{IP}}$ *(Definition 12) that satisfies knowledge-soundness.*

Proof. Given arbitrary prover $\mathcal{P}^*$ we must construct extractor $\mathcal{E}$ such that for arbitrary statement consisting of commitments A', B', C and scalar r, if

$$\langle \mathcal{P}^*(A', B', C, r; \rho), \mathcal{V}(A', B', C, r)\rangle = 1$$

then $\mathcal{E}(A, B, C, r; \rho)$ produces a witness consisting of vectors $\boldsymbol{a}', \boldsymbol{b}'$ and scalars c, r_a, r_b, r_c such that

$$A' = \mathsf{com}(\boldsymbol{a}', r_a)$$
$$B' = \mathsf{com}(\boldsymbol{b}', r_b)$$
$$C = \mathsf{com}(c, r_c)$$

and

$$c = (\boldsymbol{a}' \circ \boldsymbol{r}) \cdot \boldsymbol{b}'$$

where $\boldsymbol{r} = (r^0, r^1, \ldots, r^{n-1})$. We construct $\mathcal{E}$ that extracts a valid witness with probability $1 - \mathsf{negl}(\lambda)$ as follows:

By assumption, because the verifier accepts, we have that it accepts the inner-product argument over commitments A, B, C (generated during interaction) and r. By the knowledge soundness property of the inner-product argument (Lemma 12) $\mathcal{E}$ can extract the vectors "under" the commitments, namely $\boldsymbol{a}, \boldsymbol{b}$ and scalar c (along with associated randomness r_a, r_b, r_c) such that

$$A = \boldsymbol{w}^{\boldsymbol{a}} \cdot h^{r_a} \tag{31}$$

$$B = \boldsymbol{w}^{\boldsymbol{b}} \cdot h^{r_b} \tag{32}$$

$$C = g^c h^{r_c} \tag{33}$$

and

$$c = (\boldsymbol{a} \circ \boldsymbol{r}) \cdot \boldsymbol{b}$$

What remains to show is that construction 4 additionally enforces that

$$A'_1 = \boldsymbol{l}^{\boldsymbol{a}} \cdot h^{r_a}$$
$$B'_1 = \boldsymbol{l}^{\boldsymbol{b}} \cdot h^{r_b}$$

with probability $1-\mathsf{negl}(\lambda)$. This implies that $(\boldsymbol{a}, \boldsymbol{b}, c, r_a, r_b, r_c)$ is a valid witness to statement (A', B', c, r).

We focus on showing that A and A' must commit to the same vector with probability $1 - \mathsf{negl}(\lambda)$; the case for commitments B and B' is symmetric. Suppose, for contradiction, there exists adversary $\mathcal{A}$ that outputs commitments (A, A'') such that

$$e(A'', g) = e(A \cdot A'_1, g^{\gamma})$$

but there exists no $\boldsymbol{a}$ such that

$$A = \boldsymbol{w}^{\boldsymbol{a}} \cdot h^{r_a}$$
$$A'_1 = \boldsymbol{l}^{\boldsymbol{a}} \cdot h^{r_a}$$

with non-negligible probability. Then we can construct adversary $\mathcal{B}$ that can break the $(n-1)$-SDH assumption. Suppose $\mathcal{B}$ has the following challenge

$$(g, g^{\sigma}, \ldots, g^{\sigma^{n-1}}).$$

Additionally, suppose $\mathcal{B}$ is provided the statement (A', B', C) as auxiliary input. Because (A', B', C) are perfectly hiding commitments, they are independent of s and thus are a valid input for the $(n-1)$-SDH assumption. $\mathcal{B}$ initially picks random $b \xleftarrow{\$} \{0, 1\}$ and proceeds as follows:

- If $b = 0$: Let

$$\boldsymbol{w} = (g, g^{\sigma}, \ldots, g^{\sigma^{n-1}}).$$

  Note that this implicitly sets $s = \sigma$. Sample secret $t \xleftarrow{\$} \mathbb{F}$ and generate the rest of the structured reference string accordingly.

– If $b = 1$: Use terms $(g, g^{\sigma}, \ldots, g^{\sigma^{n-1}})$ to efficiently compute commitment keys

$$\boldsymbol{l} = (g^{\ell_0(\sigma)}, g^{\ell_1(\sigma)}, \ldots, g^{\ell_{n-1}(\sigma)})$$
$$\boldsymbol{l}' = (g^{\alpha\ell_0(\sigma)}, g^{\alpha\ell_1(\sigma)}, \ldots, g^{\alpha\ell_{n-1}(\sigma)})$$

Note that this implicitly sets $t = s$. Sample secret $s \xleftarrow{\$} \mathbb{F}$ and generate the rest of the structured reference string accordingly. Note that $\boldsymbol{l}$ and $\boldsymbol{l}'$ can be computed efficiently by the reasoning in the proof for Corollary 1.

Now $\mathcal{B}$ runs $\mathcal{A}$ on the statement (A', B', C) and public parameters and recieves commitments (A, A''). Because the verifier accepts that commitment A' is well-formed, by Lemma 2, $\mathcal{E}$ can extract vector $\boldsymbol{a}'$ and scalar r'_a such that

$$A'_1 = \boldsymbol{l}^{\boldsymbol{a}'} \cdot h^{r'_a} \tag{34}$$

Next, we observe that in the case that $b = 0$, $\boldsymbol{t} = (\boldsymbol{w} \circ \boldsymbol{l})^{\gamma} = \{g^{U_i(s)}\}_{i \in \{0,\ldots,n-1\}}$, where

$$U_i(X) = \gamma(X^i + \ell_i(t)).$$

Likewise, in the case that $b = 1$, $\boldsymbol{t} = \{g^{V_i(s)}\}_{i \in \{0,\ldots,n-1\}}$, where

$$V_i(X) = \gamma(s^i + \ell_i(X)).$$

Both $\{U_i\}_{i \in \{0,\ldots,n-1\}}$ and $\{V_i\}_{i \in \{0,\ldots,n-1\}}$ define sets of linearly independent polynomials. Therefore, in either case, because

$$e(A'', g) = e(A \cdot A'_1, g^{\gamma})$$

by Corollary 1 which extends the $(n-1)$-EPKE assumption for linearly independent polynomials, $\mathcal{E}$ can extract vector $\boldsymbol{a}''$ and scalar r''_a such that

$$A \cdot A'_1 = \prod_{i=0}^{n-1} g^{(s^i + \ell_i(t)) a''_i} \cdot h^{r''_a} \tag{35}$$

Next observe that

$$r''_a = r_a + r'_a \tag{36}$$

with probability $1 - \mathsf{negl}(\lambda)$ because otherwise we can construct adversary $\mathcal{C}$ that can solve for μ such that $g^{\mu} = h$ breaking the discrete log assumption with non-negligible probability. In more detail, given discrete-logarithm challenge (g, h), adversary $\mathcal{C}$ generates the SRS accordingly with known s and t. Then $\mathcal{C}$ runs $\mathcal{A}$ and $\mathcal{E}$ to extract $\boldsymbol{a}, \boldsymbol{a}', \boldsymbol{a}'', r_a, r'_a, r''_a$ and by equations 31, 34, 35 solves for μ such that $h = g^{\mu}$ as follows:

$$\mu = \frac{\sum_{i=0}^{n-1} s^i a_i + \ell_i(t) a'_i - (s^i + \ell_i(t)) a''_i}{r''_a - (r'_a + r_a)}$$

Returning to adversary $\mathcal{B}$, by assumption, because $\boldsymbol{a} \neq \boldsymbol{a}'$, we must have $\boldsymbol{a} \neq \boldsymbol{a}''$ or $\boldsymbol{a}' \neq \boldsymbol{a}''$. We consider both cases:

- Suppose that $\boldsymbol{a} \neq \boldsymbol{a}''$. If $b \neq 0$ then abort. Otherwise, by equations 31, 34, 35 we have

$$A = \prod_{i=0}^{n-1} g^{s^i a_i} \cdot h^{r_a}$$
$$A'_1 = \prod_{i=0}^{n-1} g^{\ell_i(t) a'_i} \cdot h^{r'_a}$$
$$A \cdot A'_1 = \prod_{i=0}^{n-1} g^{(s^i + \ell_i(t)) a''_i} \cdot h^{r''_a}.$$

Thus we have

$$\prod_{i=0}^{n-1} g^{s^i a_i} \cdot h^{r_a} \cdot \prod_{i=0}^{n-1} g^{\ell_i(t) a'_i} \cdot h^{r'_a} = \prod_{i=0}^{n-1} g^{(s^i + \ell_i(t)) a''_i} \cdot h^{r''_a}$$

By equation 36 we have

$$\prod_{i=0}^{n-1} g^{s^i a_i} \cdot \prod_{i=0}^{n-1} g^{\ell_i(t) a'_i} = \prod_{i=0}^{n-1} g^{(s^i + \ell_i(t)) a''_i} \tag{37}$$
$$\prod_{i=0}^{n-1} g^{(s^i + \ell_i(t)) a''_i - s^i a_i - \ell_i(t) a'_i} = g^0 \tag{38}$$

Now consider polynomial

$$P(X) = \sum_{i=0}^{n-1} (X^i + \ell_i(t)) a''_i - X^i a_i - \ell_i(t) a'_i.$$

Because $b = 0$, we have that $s = \sigma$. Therefore by equation 38 we have that $P(\sigma) = 0$ with probability $1 - \mathsf{negl}(\lambda)$. Additionally, because $\boldsymbol{a}'' \neq \boldsymbol{a}$, we have that P is not the zero polynomial. Therefore $\mathcal{E}$ can solve efficiently for σ by iterating through the roots of P until we find one that satisfies the challenge.

- Suppose instead that $\boldsymbol{a}' \neq \boldsymbol{a}''$. If $b \neq 1$ then abort. Consider polynomial

$$Q(X) = \sum_{i=0}^{n-1} (s^i + \ell_i(X)) a''_i - s^i a_i - \ell_i(X) a'_i$$

By equations 31, 34, 35, and 36, and a similiar argument as in the previous case, we have that $Q(\sigma) = 0$ with probability $1 - \mathsf{negl}(\lambda)$. Additionally, because $\boldsymbol{a}'' \neq \boldsymbol{a}'$, we have that Q is not the zero polynomial. Therefore $\mathcal{E}$ can solve for σ by iterating through the roots of Q until we find one that satisfies the challenge.

Therefore either $\boldsymbol{a} = \boldsymbol{a}'' = \boldsymbol{a}'$ or $\mathcal{B}$ succeeds in breaking the $(n-1)$-SDH assumption with probability $1 - \mathsf{negl}(\lambda)$.

We now analyze the success probability of $\mathcal{B}$. Let δ be the success probability of $\mathcal{A}$. From the above reasoning we have that $\delta = \delta_1 + \delta_2$, where δ_1 is the probability that $\mathcal{A}$ succeeds with $\boldsymbol{a} \neq \boldsymbol{a}''$, and δ_2 is the probability that $\mathcal{A}$ succeeds with $\boldsymbol{a}' \neq \boldsymbol{a}''$. $\mathcal{B}$ is successful when $\boldsymbol{a} \neq \boldsymbol{a}''$ and $b = 0$, or when $\boldsymbol{a}' \neq \boldsymbol{a}''$ and $b = 1$. Thus, because $\Pr[b = 0] = \Pr[b = 1] = 1/2$, the probability that $\mathcal{B}$ succeeds is $\delta_1/2 + \delta_2/2 - \mathsf{negl}(\lambda) = \delta/2 - \mathsf{negl}(\lambda)$. Therefore if δ is a non-negligible probability, $\mathcal{B}$ succeeds in breaking the $(n-1)$-SDH assumption with non-negligible probability.

□

Lemma 16. *Construction 4 is an argument system for $\mathcal{R}_{\mathsf{IP}}$ that is honest-verifier zero-knowledge.*

Proof. To prover honest-verifier zero-knowledge we must construct simulator $\mathcal{S}$ that can simulate a transcript indistinguishable from one generated by an honest interaction for any given statement with a valid witness. We construct $\mathcal{S}$ as follows:

Consider arbitrary statement *with a valid witness* consisting of commitments A', B', C and scalar r. First, $\mathcal{S}$ generates the common reference string and the associated trapdoor δ. Next, $\mathcal{S}$ sets $\boldsymbol{a} = \boldsymbol{b} = \boldsymbol{0}$ and computes commitments A and B as dictated by the protocol. Next, $\mathcal{S}$ uses trapdoor δ to forge A'' and B'' such that the verifier's checks pass:

$$A'' \coloneqq (A \cdot A'_1)^{\frac{1}{\delta}}$$
$$B'' \coloneqq (B \cdot B'_1)^{\frac{1}{\delta}}.$$

In the final round, $\mathcal{S}$ can simulate a transcript indistinguishable from honest inner-product argument due to the honest-verifier zero-knowledge property of the inner product argument (Lemma 13).

In both the real and ideal settings the terms A and B are indistinguishable from random elements in $\mathbb{G}$ due to the blinding terms. Additionally, in both the real and ideal setting, terms A'' and B'' are uniquely fixed by A, B, and the statement under the same relation. Thus an adversary cannot use terms A'' and B'' to distinguish between the real and ideal setting.

□

Lemma 17 (Efficiency). *For vectors of size n construction 4 features an $O(n)$ generator, $O(n)$ prover, and an $O(\log n)$ verifier.*

Proof. At a high level, construction 4 adds an $O(n)$ time overhead to the generator and prover and an $O(1)$ overhead to the verifier. Therefore by the asymptotics of construction 3 (Lemma 14), construction 4 achieves the stated asymptotics. In more detail:

- *Generator:* In addition to running the generator for the inner-product argument, $\mathcal{G}$ needs to compute $\boldsymbol{l}, \boldsymbol{l}', \boldsymbol{t}$ which can be done with 3 multi-exponentiations of size n, which can be done in $O(n/\log n)$ exponentiations by Pippenger's algorithm [66].
- *Prover:* In addition to running the inner-product argument, which takes $O(n)$ time, $\mathcal{P}$ can compute A, B, A'', B'' with 4 multi-exponentiations of size n.
- *Verifier:* In addition to running the inner-product argument which takes $O(\log n)$ time, the verifier needs to perform 4 pairing operations which can be done in $O(1)$ time.

□

D.6 Proof of Theorem 5 (Argument System for $\mathcal{R}_{\mathsf{ACS}}$)

Lemma 18. *Construction 6 satisfies completeness and soundness.*

Proof. Completeness and soundness follow from the description of construction 6 and the correctness and soundness of the underlying protocols □

Lemma 19. *Construction 6 satisfies knowledge soundness.*

Proof. By the soundness of construction 6 (Lemma 18), an accepting verifier implies a valid shifted witness w' with negligible soundness error. By the extractibility of the polynomial commitment scheme (Construction 1), an extractor $\mathcal{E}$ can extract w' from the prover's initial commitment, and return the reconstructed witness $\boldsymbol{w}$. □

Lemma 20. *Construction 6 satisfies honest-verifier zero-knowledge*

Proof. Intuitively, honest-verifier zero-knowledge holds because the prover only sends perfectly hiding polynomial commitments to the verifier and engages in honest-verifier zero-knowledge arguments regarding these commitments. We formally argue honest-verifier zero-knowledge by constructing a simulator:

To prove honest-verifier zero-knowledge we must construct a simulator $\mathcal{S}$ that can simulate a transcript indistinguishable from one generated by an honest interaction for any given statement with a valid witness. We construct $\mathcal{S}$ as follows:

Consider arbitrary $\mathcal{R}_{\mathsf{ACS}}$ statement *with a valid witness.* $\mathcal{S}$ sets the witness $\boldsymbol{w}$ to $\mathbf{0}$ and proceeds to run an interaction between an honest prover and verifier as dictated by the protocol, with the following changes

- When $\mathcal{S}$ needs to simulate a polynomial evaluation argument, it runs the simulator for the polynomial evaluation argument (Lemma 10).
- When $\mathcal{S}$ needs to simulate an inner-product argument, it runs the simulator for the inner-product argument (Lemma 16).
- When $\mathcal{S}$ needs to simulate the multi-Hadamard-product argument, it runs the simulator for the multi-Hadamard argument (Lemma 5)

By the hiding property of the polynomial commitment scheme (Lemma 2) the honest prover's commitments and the simulated commitments are indistinguishable. Next by lemma 10, lemma 16, and lemma 5 the simulator's transcripts for the polynomial evaluation, inner-product, and multi-Hadamard-product arguments are indistinguishable from that of an honest interaction. This implies that the simulator's overall transcript is indistinguishable from that of an honest interaction.

□

E Using a Lookup Table to Efficiently Compute Polynomials P_1, P_2, P_3 in the Main Argument

As claimed in construction 6 polynomials P_1, P_2, and P_3 can be efficiently computed in linear time by constructing a lookup table as discussed by Chiesa et al. [33]. We demonstrate how to compute this lookup table for in order to efficiently compute P_1 (similiar strategies can be used for P_2 and P_3). We first observe that by definition

$$P_1(b) = \sum_{c \in N} \sum_{k \in K} P(k, \alpha, b, c) z(c).$$

Because $u_N(c, \mathsf{C}(k))$ is non-zero only when $c = \mathsf{C}(k)$ we have that

$$P_1(b) = \sum_{k \in K} P(k, \alpha, b, \mathsf{C}(k)) z(\mathsf{C}(k)).$$

Additionally, because $u_N(b, \mathsf{B}(k))$ is non-zero only when $b = \mathsf{B}(k)$ we have that

$$P_1(b) = \sum_{k \text{ s.t. } b = \mathsf{B}(k)} P(k, \alpha, b, \mathsf{C}(k)) z(\mathsf{C}(k)).$$

Thus, to compute $P_1(b)$ for all $b \in N$, the prover first precomputes $u_H(\alpha, a) = v_H(\alpha)/(\alpha - a)$ for all $a \in H$. By design, the prover is provided evaluations of polynomials $\mathsf{A}, \mathsf{B}, \mathsf{C}$ and z. [3] Given the precomputed evaluations, the prover can efficiently compute $P_1(b)$ for all $b \in N$ as follows

1. Initially set $P_1(b) = 0$ for all $b \in N$
2. For all $k \in K$ compute

$$P(k, \alpha, \mathsf{B}(k), \mathsf{C}(k)) z(\mathsf{C}(k))$$

in $O(1)$ time using the precomputations and add the resulting value to $P_1(\mathsf{B}(k))$.

[3] We implicitly assume the prover is provided evaluations $u_N(b, b)$ for all $b \in N$. This can be computed once globally by the generator in $O(n/\log n)$ exponentiations if N is a multiplicative subgroup.

F Additional Concrete Optimizations

Construction 11 (Batched Inner Product). The zero argument presented in construction 5 can be generalized to argue that the sum of inner-products of vectors sums to arbitrary value σ. Multiple arguments for the inner-product can be batched by taking random linear combinations of the vectors in question and applying the zero argument. In our main argument system, this reduces the total number of inner-product arguments from 3 to 1. Note that the multi-Hadamard product argument still requires it's own separate inner-product argument.

Construction 12 (Batched Polynomial Evaluation). As shown by Chiesa et al. [33], multiple polynomial evaluation arguments over polynomials of the same degree over the same evaluation point can be batched by taking random linear combinations of the polynomials and evaluations in question. In our system, the sub-argument to check that (u, v) were computed correctly as part of the inner-product argument (Construction 3) requires two polynomial evaluation arguments that matches this criteria. Therefore in each inner product argument batching enables a single polynomial evaluation argument instead of two.

Construction 13 (Cheaper Polynomial Commitments in the Algebraic Group Model). When proven secure in the Algebraic Group Model [43] in contrast to the plain model, we can modify our polynomial evaluation argument such that commitment and evaluation only costs a single multi-exponentiation of size n using techniques similiar to Chiesa et al. [33].