Quarks: Quadruple-efficient transparent zkSNARKs

Srinath Setty
Microsoft Research

Jonathan Lee*
Microsoft Research

Abstract

We introduce Xiphos and Kopis, new transparent zero-knowledge succinct non-interactive arguments of knowledge (zkSNARKs) for R1CS. They do not require a trusted setup, and their security relies on the standard SXDH problem. They achieve non-interactivity in the random oracle model using the Fiat-Shamir transform. Unlike prior transparent zkSNARKs, which support either a fast prover, short proofs, or quick verification, our work is the first to simultaneously achieve all three properties (both asymptotically and concretely) and in addition an inexpensive setup phase, thereby providing the first *quadruple-efficient* transparent zkSNARKs (*Quarks*).

Under both schemes, for an R1CS instance of size n and security parameter λ, the prover incurs $O_\lambda(n)$ costs to produce a proof of size $O_\lambda(\log n)$. In Xiphos, verification time is $O_\lambda(\log n)$, and in Kopis it is $O_\lambda(\sqrt{n})$. In terms of concrete efficiency, compared to prior state-of-the-art transparent zkSNARKs, Xiphos offers the fastest verification; its proof sizes are competitive with those of SuperSonic [EUROCRYPT 2020], a prior transparent SNARK with the shortest proofs in the literature. Xiphos's prover is fast: its prover is $\approx 5\times$ of Spartan [CRYPTO 2020], a prior transparent zkSNARK with the fastest prover in the literature, and is $250\times$ faster than SuperSonic. Kopis, at the cost of increased verification time (which is still concretely faster than SuperSonic), shortens Xiphos's proof sizes further, thereby producing proofs shorter than SuperSonic. Xiphos and Kopis incur 10–10,000$\times$ lower preprocessing costs for the verifier in the setup phase depending on the baseline. Finally, a byproduct of Kopis is Lakonia, a NIZK for R1CS with $O_\lambda(\log n)$-sized proofs, which provides an alternative to Bulletproofs [S&P 2018] with over an order of magnitude faster proving and verification times.

1 Introduction

Zero-knowledge SNARKs (zkSNARKs) [21, 40] for NP is a primitive that enables a *prover* to prove to a *verifier* the knowledge of a satisfying witness w to an NP statement by producing a proof π such that the proof is both zero-knowledge [42] and succinct. There are two forms of succinctness: the size of a proof and the time to verify a proof are both sub-linear in the size of the NP statement. Because of these properties, zkSNARKs is a core building block for various forms of delegation of computation for privacy and/or scalability [2, 14, 24, 25, 34, 48, 50, 56, 59, 60]. Given significant interest, constructing zkSNARKs is an active area of research, with a flurry of recent work to improve asymptotic and concrete efficiency.

There are many approaches to construct zkSNARKs, starting with the works of Kilian [47] and Micali [55]. These works rely on short PCPs [9–11, 15, 19, 20], which remain too expensive to be used in practice. A seminal work in this area is GGPR [39],

*Current affiliation: Nanotronics Imaging. Work done while at Microsoft Research.

which provides zkSNARKs for R1CS with near-optimal asymptotics and good constants. A major problem with state-of-the-art zkSNARKs [16, 18, 43, 57] is the requirement of a trusted setup, where a trusted entity (or a group of entities with at least one honest entity) must choose a trapdoor to create public parameters. Furthermore, the trapdoor must be kept secret to ensure soundness.

This problem was recently addressed by Spartan [58],[1] a transparent zkSNARK for R1CS. Unlike its predecessors, Spartan does not give up succinct verification [8, 17, 28] nor sacrifices generality by placing restrictions on the types of NP statements supported [13, 66]. Furthermore, Spartan requires only a transparent setup (e.g., choosing a set of random group elements or a collision-resistant hash function). To achieve succinct verification, the verifier, in a preprocessing step, creates a *computation commitment*, which is a succinct cryptographic commitment to the structure of an NP statement (e.g., the description of a circuit) without requiring secret trapdoors. The preprocessing step incurs time that is at least linear in the size of the statement, but this cost is amortized over all future verification of proofs for statements with the same structure, an amortization property similar to prior zkSNARKs with trusted setup [39, 43]. Following Spartan, Fractal [32] and SuperSonic [26] also employ computation commitments to achieve succinct verification without a trusted setup. Achieving sub-linear verification costs via computation commitments is also referred to as leveraging holography [31, 32].

1.1 Limitations of existing transparent zkSNARKs

Existing transparent zkSNARKs support either a fast prover, short proofs, or quick verification, but not all three properties simultaneously. Also, existing schemes incur high preprocessing costs to create computation commitments. Note that when we refer to "fast", "short", "quick", or "high", we refer to both asymptotic efficiency and concrete efficiency (we make these terms more precise below).

(1) Trade-offs among a fast prover, short proofs, and quick verification.

- Spartan [58] offers the best asymptotics for the prover (Figure 1). Concretely, it provides the fastest prover in the literature (Figure 2). Furthermore, Spartan relies only on the well-studied DLOG problem. Unfortunately, the proofs are $O(\sqrt{n})$ group elements and the verifier must perform $O(\sqrt{n})$ exponentiations, where n is the size of the NP statement. For R1CS statements with 2^{20} constraints, Spartan's proofs are $\approx$142 KB and proof verification takes $\approx$135 ms.
- SuperSonic [26] offers the best asymptotics for the verifier and proof sizes (Figure 1), relying on groups where the Strong RSA assumption [12, 38] and the recently introduced Adaptive Root Assumption [23, 67] hold (e.g., ideal class groups of imaginary quadratic fields). Concretely, for a 2^{20}-sized R1CS statement, the estimated proof sizes are $\approx$48 KB.[2] Unfortunately, the SuperSonic prover must perform $O(n \log n)$ exponentiations in a class group, where each operation is $\approx$800$\times$ more expensive than in a group where DLOG is hard.[3] Thus, SuperSonic's prover is slower than Spar-

[1] PCP-based SNARKs [47, 55] do not require a trusted setup, but they are too expensive to be used. Furthermore, to the best of our knowledge, they require uniform circuits for sub-linear verification.

[2] SuperSonic's authors estimate proof sizes of $\approx$12.3 KB [26, Table 3]. But, this assumes the use of a class group of 1600 bits. Recently, Dobson, Galbraith, and Smith show that this choice only provides 55 bits of security and that one should use class groups of $\approx$6,600 bits to achieve 128 bits of security [35, 36].

[3] We microbenchmark the cost of an exponentiation in a class group with random 128-bit size exponents

	prover	proof size	assistant	encoder	verifier	assumption
$\text{Spartan}_{\text{DL}}$	$n\,\mathbb{G}_1$	$\sqrt{n}\,\mathbb{G}_1$	N/A	$n\,\mathbb{G}_1$	$\sqrt{n}\,\mathbb{G}_1$	DLOG
SuperSonic	$n \log n\,\mathbb{G}_U$	$\log n\,\mathbb{G}_U$	N/A	$n\,\mathbb{G}_U$	$\log n\,\mathbb{G}_U$	sRSA + ARA
Fractal	$n \log n\,\mathbb{F}$	$\log^2 n\,\mathbb{F}$	N/A	$n \log n\,\mathbb{F}$	$\log^2 n\,\mathbb{F}$	CRHF
Spartan++	$n\,\mathbb{G}_1$	$\sqrt{n}\,\mathbb{G}_1$	$n\,\mathbb{G}_1$	$n\,\mathbb{F}$	$\sqrt{n}\,\mathbb{G}_1$	DLOG
Kopis	$n\,\mathbb{G}_1$	$\log n\,\mathbb{G}_T$	$n\,\mathbb{G}_1$	$n\,\mathbb{F}$	$\sqrt{n}\,\mathbb{G}_2$	SXDH
Xiphos	$n\,\mathbb{G}_1$	$\log n\,\mathbb{G}_T$	$n\,\mathbb{G}_1$	$n\,\mathbb{F}$	$\log n\,\mathbb{G}_T$	SXDH

FIGURE 1—Asymptotic efficiency of Kopis and Xiphos. $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$ refers to groups in a bilinear group. Spartan only requires a group where DLOG is hard, so $\mathbb{G}_1$ could be `ristretto255`. $\mathbb{G}_U$ refers to groups where sRSA and ARA hold. We depict the number of exponentations needed in these groups. For $\mathbb{F}$ we depict the number of field multiplications.

	prover (s)	proof size (KB)	assistant (s)	encoder (s)	verifier (ms)
$\text{Spartan}_{\text{DL}}$	47	142	N/A	20	135
SuperSonic	63,700	48	N/A	17,900	2,570
Fractal	864	2,500	N/A	456	220
Spartan++	45	131	24	1.6	97
Kopis	245	39	55	2.2	535
Xiphos	249	61	62	1.8	80

FIGURE 2—Concrete efficiency of Kopis and Xiphos. The reported costs for SuperSonic assume the use of a CRS consisting of n elements of $\mathbb{G}_U$ [27]. Using an $O_\lambda(1)$-sized CRS, as in the original work [26], the encoder runs in time $n \log n\,\mathbb{G}_U$ asymptotically and both the prover and encoder take ≈600,000 s longer for $n = 2^{20}$. The costs for all schemes were measured by running their implementations on the same hardware platform (§9), with one exception. For SuperSonic, we estimate its costs using the cost model provided by the authors augmented with our microbenchmarks of class group operations using the ANTIC library [1].

tan's prover, both asymptotically and concretely. Concretely, for $n = 2^{20}$, SuperSonic is $> 1,700\times$ slower than Spartan.

- Fractal [32] does not offer short proofs nor a fast prover. Concretely, for an R1CS instance with 2^{18} constraints, Fractal's prover is ≈18× slower than Spartan, and it produces proofs of size ≈2.3 MB and takes ≈205 ms to verify.

(2) High preprocessing costs. Besides the above limitations, the verifier in all three prior schemes incurs $\Omega(n)$ cryptographic operations (see the "encoder" column in Figure 1) to create a computation commitment. This cost is unavoidable for R1CS instances without structure: the verifier must at least preprocess the structure of the statement before verifying a proof. But, it is desirable to make the preprocessing concretely fast.

Remark 1.1. Unlike other zkSNARKs discussed above, Fractal offers plausible post-quantum security. Unfortunately, it does not offer short proofs. Proving is memory-intensive and is concretely expensive (§9). Designing concretely-efficient post-quantum transparent zkSNARKs remains an open problem.

Remark 1.2. In the above exposition (and in the rest of the paper), by "Spartan", we refer

using the ANTIC [1] library, which offers fast class groups. Each class group exponentiation costs ≈38 ms. Whereas, an exponentiation on `ristretto255` [4, 45] with the curve25519-dalek library [3] takes ≈45 μs.

	asymptotic efficiency			concrete efficiency ($n = 2^{20}$)		
	prover	proof size	verifier	prover (s)	proof size (KB)	verifier (ms)
Ligero	$n \log n\ \mathbb{F}$	$\sqrt{n}\ \mathbb{F}$	$n\ \mathbb{F}$, $\sqrt{n}\ \mathbb{H}$	69	20,000	31,000
Hyrax	$n\ \mathbb{G}_1$	$\sqrt{n}\ \mathbb{G}_1$	$n\ \mathbb{F}$, $\sqrt{n}\ \mathbb{G}_1$	486	58	7,700
Bulletproofs	$n\ \mathbb{G}_1$	$\log n\ \mathbb{G}_1$	$n\ \mathbb{G}_1$	804	1.5	30,957
Aurora	$n \log n\ \mathbb{F}$	$\log^2 n\ \mathbb{F}$	$n\ \mathbb{F}$, $\log^2 n\ \mathbb{H}$	485	1,600	108,000
STARK	$n \log^2 n\ \mathbb{F}$	$\log^2 n\ \mathbb{F}$	$n\ \mathbb{F}$, $\log^2 n\ \mathbb{H}$	$\geq$4,850	39,384	$\geq$432,000
Spartan$_{\text{DL}}$	$n\ \mathbb{G}_1$	$\sqrt{n}\ \mathbb{G}_1$	$n\ \mathbb{F}$, $\sqrt{n}\ \mathbb{G}_1$	6	48	369
Spartan++	$n\ \mathbb{G}_1$	$\sqrt{n}\ \mathbb{G}_1$	$n\ \mathbb{F}$, $\sqrt{n}\ \mathbb{G}_1$	6	40	347
Lakonia	$n\ \mathbb{G}_1$	$\log n\ \mathbb{G}_{\text{T}}$	$n\ \mathbb{F}$, $\sqrt{n}\ \mathbb{G}_2$	29	12.6	555

FIGURE 3—Asymptotic and concrete efficiency of Lakonia and its baselines. Lakonia produces shortest proofs except compared to Bulletproofs, but Bulletproofs incurs orders of magnitude higher proving and verification costs. The costs for all schemes were measured by running their implementations on the same hardware platform (§9), with two exceptions. For STARK, we provide estimates based on our measurements of Aurora's performance and prior performance reports for STARK in the Aurora paper [17]. For Bulletproofs, we provide estimates based on prior performance reports [28].

to a specific member of the Spartan family of zkSNARKs, called Spartan$_{\text{DL}}$. The family has two additional transparent zkSNARKs that can produce $O_\lambda(\log^2 n)$-sized proofs with $O_\lambda(\log^2 n)$ verification times, but they are not experimentally evaluated. From our estimates, one of them, Spartan$_{\text{CL}}$, incurs prover times analogous to SuperSonic, and another one, Spartan$_{\text{RO}}$, produces proofs as big as Fractal, so they suffer from the limitations listed for SuperSonic and Fractal.

1.2 A new goal: Quadruple-efficient transparent zkSNARKs (Quarks)

To address the aforementioned problems with existing transparent zkSNARKs, we desire zkSNARKs with the following asymptotic and concrete efficiency characteristics. We refer to zkSNARKs that satisfy all the following four properties as Quarks.

1. **A fast prover:** The prover should run in time $O_\lambda(n)$, with a small constant to achieve concrete performance analogous to Spartan.
2. **Short proofs:** The proof length should be $O_\lambda(\log n)$, with a small constant to achieve proof sizes similar to SuperSonic [26].
3. **Quick verification:** The verifier's time to verify a proof should be $O_\lambda(\log n)$, with a small constant to achieve verification times similar to SuperSonic [26].
4. **Low preprocessing costs:** The cost to the verifier to create a computation commitment to an NP statement's structure should be $O(n)$, with small constants such that the concrete cost is only a small constant factor slower than reading the statement.

2 Overview of our work and a summary of our contributions

In this work, we construct two transparent zkSNARKs, namely Xiphos and Kopis. Of these, Xiphos is a Quark, and Kopis supports all but the quick verification property (concrete verification costs of Kopis is still faster than SuperSonic's at R1CS instance sizes we experiment with). Nevertheless, Kopis supports shorter proofs than Xiphos

and SuperSonic. Figure 1 depicts the asymptotic efficiency of Xiphos and Kopis, and compares it with prior transparent zkSNARKs. Similarly, Figure 2 depicts their concrete efficiency for $n = 2^{20}$ R1CS constraints. The security of both schemes relies on the standard SXDH problem [6], and both achieve non-interactivity in the random oracle model using the Fiat-Shamir transform [37]. A byproduct of Kopis is Lakonia, which does not employ computation commitments, so it incurs $O(n)$ verification costs. However, it produces $O_\lambda(\log n)$-sized proofs analogous to Bulletproofs [28]. Figure 3 depicts the asymptotic and concrete efficiency of Lakonia and compares it with its baselines.

Our starting point is Spartan [58], which offers a modular framework for constructing transparent zkSNARKs. It employs a seminal interactive proof protocol, called the sum-check protocol [54], in conjunction with an extractable polynomial commitment scheme [26, 46, 66] for multilinear polynomials. To instantiate computation commitments, Spartan requires a polynomial commitment scheme for sparse multilinear polynomials. A key innovation in Spartan is a cryptographic compiler, called SPARK, that transforms an existing extractable polynomial commitment for dense multilinear polynomials to an extractable polynomial commitment scheme for sparse multilinear polynomials—without introducing undesirable asymptotic or concrete overheads to the prover or the verifier.

To realize Xiphos, Kopis, and Lakonia, this work makes the following contributions.

(1) Polynomial commitments with constant-sized commitments to shorten proofs. Spartan employs the polynomial commitment scheme of Wahby et al. [66], which we call Hyrax-PC, where the size of a commitment to a multilinear polynomial is $\sqrt{m}$ elements of a group $\mathbb{G}_1$ in which DLOG is hard, where $m = 2^\ell$ and ℓ is the number of variables in the committed polynomial.[4] In the context of Spartan, $m = O(n)$, where n is the size of the NP statement. This constitutes a major reason for large proofs in Spartan.

To address this, we design a new polynomial commitment scheme for multilinear polynomials, called Kopis-PC, in which a commitment is a single element of $\mathbb{G}_\mathrm{T}$, where $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_\mathrm{T})$ are groups in a bilinear map in which the SXDH problem is hard, so a polynomial commitment is of size $O_\lambda(1)$. A single element of $\mathbb{G}_\mathrm{T}$ is a commitment to a vector of $\sqrt{m}$ $\mathbb{G}_1$ elements under Hyrax-PC. However, in Hyrax-PC, the verifier locally computes an inner product between a vector of public scalars with a vector of $\sqrt{m}$ elements of $\mathbb{G}_1$ (representing a polynomial commitment under Hyrax-PC). Our polynomial commitment scheme handles this by having the prover compute the desired inner product and produce a proof of correct execution using the generalized inner-product arguments of Bunz et al [27]. Kopis-PC can also be seen as an adaptation of the bivariate polynomial commitment scheme of Bunz et al. [27] to the setting of multilinear polynomials. This observation is however new.

Besides Kopis-PC, we also build on Dory-PC [49], a recent polynomial commitment scheme that employs the same blueprint as Kopis-PC, but in addition exploits the tensor structure in the public vector of scalars. Like Kopis-PC, Dory-PC also produces $O_\lambda(1)$-sized commitments and $O_\lambda(\log m)$-sized polynomial evaluation proofs. The constant associated with Dory-PC's evaluation proof sizes is $\approx 3\times$ larger than in Kopis-PC. In exchange, Dory-PC achieves $O_\lambda(\log m)$ costs to verify a polynomial evaluation proof

[4] Hyrax-PC can provide $O_\lambda(1)$-sized commitments, but it requires $O(m)$ work for the verifier. In the context of SNARKs, we require the verifier's work in the polynomial commitment scheme to be sub-linear in m.

instead of $O_\lambda(\sqrt{m})$ under Kopis-PC.

(2) Sparse polynomial commitments with shorter proofs using Sparkle. Another major component that contributes to proof sizes in Spartan is the SPARK compiler. Even if we replace Hyrax-PC with one of the polynomial commitment schemes that produce constant-sized commitments (Kopis-PC or Dory-PC) in SPARK, the proof sizes are still $O(\log^2 n)$. This is a result of using a $O(\log n)$-depth layered circuit in conjunction with a layered sum-check protocol [33, 41, 63] to prove grand product relations in SPARK. We address this by designing a variant of SPARK, called Sparkle, which reduces proof sizes to $O(\log n)$. In particular, using a combination of the sum-check protocol and polynomial commitments with constant-sized commitments, we design a new special-purpose SNARK for proving grand product relations. However, a naive replacement of the layered sum-check protocol with the special-purpose SNARK increases constants associated with the prover, which is undesirable. To achieve smaller constants for the prover, Sparkle hybridizes the new SNARK with the layered sum-check approach in SPARK, where a constant number of layers of the circuit are proved as before, but the rest of the layers are proved using the special-purpose SNARK, there by achieving $O(\log n)$-sized proofs without incurring large constants for the prover.

(3) An untrusted assistant to accelerate the verifier's preprocessing. Recall that the verifier in Spartan (and other prior transparent zkSNARKs) must run an encoder to preprocess the structure of an NP statement to create a computation commitment, which in turn enables the verifier to achieve sub-linear verification costs. We introduce the notion of an untrusted *assistant* for the encoder. An assistant is an algorithm that can be executed by anyone including the prover. Specifically, both the assistant and the encoder take as input the structure of an NP statement. Both transform the NP statement's structure into a set of polynomials, but only the assistant creates the necessary polynomial commitments, so only the assistant incurs the high preprocessing costs. Furthermore, the encoder checks that the polynomial commitments are correctly created by requiring the assistant to produce a proof of correct evaluation of the underlying polynomials at a random point in their domain, which the encoder checks by evaluating the polynomials it holds (the random point is a public coin, so in the non-interactive version, it is obtained using the Fiat-Shamir transform in the random oracle model). For multilinear polynomials, since the cost of evaluating the necessary polynomials incurs $O(n)$ time and the cost of verifying proofs of evaluations is sub-linear in $O_\lambda(n)$, the encoder incurs $O(n)$ costs with a small constant rather than $O_\lambda(n)$.

(4) An optimized implementation. We implement Kopis, Xiphos, and Lakonia in Rust by extending `libSpartan` [5], a high-speed Rust implementation of Spartan built atop `ristretto255` [3, 4, 45]. This is about 5,000 lines of Rust. Since our polynomial commitment schemes require a pairing-friendly elliptic curve, we use `bls12-381` and employ its implementation from the `relic` toolkit. We implement all of our techniques along with a host of optimizations. For example, instead of producing proofs of correct evaluations of multiple committed polynomials independently, our implementation reduces multiple polynomial evaluation proofs into a single one, which lowers verification costs and proof sizes substantially. Another notable optimization is to the zero-knowledge transformation used by Spartan (§8). Many of these optimizations improve Spartan's

performance and proof sizes (we refer to the improved version of Spartan as Spartan++).

(5) A detailed experimental evaluation. We experimentally evaluate our schemes and compare them with state-of-the-art zkSNARK schemes. We find that Xiphos offers the fastest verification; its proof sizes are competitive with those of SuperSonic, which offers the shortest proofs in the literature. Our evaluation also demonstrates that Xiphos's prover is fast: its prover is $\approx$250$\times$ faster than SuperSonic and is within $\approx$5$\times$ of Spartan, which offers the fastest prover in the literature.[5] Kopis, at the cost of increased verification time (which is still concretely faster than SuperSonic), shortens Xiphos's proof sizes further, thereby producing proofs shorter than SuperSonic. Xiphos and Kopis incur 10–10,000$\times$ lower preprocessing costs for the verifier depending on the baseline. Finally, Lakonia shortens Kopis's proofs further, thereby providing an alternative to Bulletproofs [28] with at least an order of magnitude faster proving and verification costs.

2.1 Roadmap for the rest of the paper

Section 3 describes the basic building blocks we rely on. Section 4 describes Kopis-PC. Section 5 provides a stand-alone description of the special-purpose SNARK for proving grand product relations. Section 6 improves Spartan's SPARK compiler using the special-purpose SNARK. Section 7 describes the use of an untrusted assistant to accelerate the verifier's preprocessing costs. Section 8 describes our improved zero-knowledge transformation. Finally, Section 9 presents an experimental evaluation of Xiphos, Kopis, and Lakonia, and compares them with their baselines.

3 Preliminaries

We adopt preliminaries from Spartan [58], with additional definitions. We use $\mathbb{F}$ to denote a finite field and λ to denote the security parameter. $\texttt{negl}(\lambda)$ denotes a negligible function in λ. "PPT algorithms" refer to probabilistic polynomial time algorithms.

3.1 Problem instances in R1CS

Recall that for any problem instance $\mathbb{x}$, if $\mathbb{x}$ is in an NP language $\mathcal{L}$, there exists a witness w and a deterministic algorithm $\texttt{Sat}$ such that: $\texttt{Sat}_{\mathcal{L}}(\mathbb{x}, w) = 1$ if $\mathbb{x} \in \mathcal{L}$, and 0 otherwise.

Alternatively, the set of tuples of the form $\langle \mathbb{x}, w \rangle$ form a set of NP relations. The subset of those for which $\texttt{Sat}_{\mathcal{L}}(\mathbb{x}, w) = 1$ are called *satisfiable instances*, which we denote as: $\mathcal{R}_{\mathcal{L}} = \{\langle \mathbb{x}, w \rangle : \texttt{Sat}_{\mathcal{L}}(\mathbb{x}, w) = 1\}$.

As an NP-complete language, we focus on the rank-1 constraint satisfiability (R1CS), a popular target for compiler toolchains that accept programs expressed in high-level languages [57, 61, 62, 65]. R1CS is implicit in the QAPs of GGPR [39], but it is used with (and without) QAPs in subsequent works [17, 53, 61].

Definition 3.1 (R1CS instance and structure)**.** An R1CS instance is a tuple $(\mathbb{F}, A, B, C, m, n, io)$, where io denotes the public input and output of the instance, $A, B, C \in \mathbb{F}^{m \times m}$, where

[5] Most of the slowdown of Xiphos relative to Spartan can be attributed to the difference in speed between the cost of an exponentiation in `ristretto255` (used by Spartan) and `bls12-381` (used by our schemes). With a faster implementation of curve arithmetic on `bls12-381`, we believe this gap can be reduced substantially. See Section 9.1 for details.

$m \geq |io| + 1$ and there are at most n non-zero entries in each matrix. The io-independent part of the instance constitutes the structure of an R1CS instance.

Note that matrices A, B, C are defined to be square matrices for conceptual simplicity. Furthermore, WLOG, we assume that $n = O(m)$ throughout the paper.

Below, we use the notation $z = (x, y, z)$, where each of x, y, z is a vector over $\mathbb{F}$, to mean that z is a vector that concatenates the three vectors in a natural way.

Definition 3.2 (R1CS)**.** An R1CS instance $(\mathbb{F}, A, B, C, io, m, n)$ is said to be *satisfiable* if there exists a witness $w \in \mathbb{F}^{m-|io|-1}$ such that $(A \cdot z) \circ (B \cdot z) = (C \cdot z)$, where $z = (io, 1, w)$, $\cdot$ is the matrix-vector product, and $\circ$ is the Hadamard (entry-wise) product.

Definition 3.3. For an R1CS instance $\mathbb{x} = (\mathbb{F}, A, B, C, io, m, n)$ and a purported witness $w \in \mathbb{F}^{m-|io|-1}$, we define:

$$\mathtt{Sat}_{\mathrm{R1CS}}(\mathbb{x}, w) = \begin{cases} 1 & (A \cdot (io, 1, w)) \circ (B \cdot (io, 1, w)) = (C \cdot (io, 1, w)) \\ 0 & \text{otherwise} \end{cases}$$

The set of satisfiable R1CS instances can be denoted as:

$$\mathcal{R}_{\mathrm{R1CS}} = \{\langle (\mathbb{F}, A, B, C, io, m, n), w \rangle : \mathtt{Sat}_{\mathrm{R1CS}}((\mathbb{F}, A, B, C, io, m, n), w) = 1\}$$

Definition 3.4. For a given R1CS instance $\mathbb{x} = (\mathbb{F}, A, B, C, io, m, n)$, the NP statement that $\mathbb{x}$ is satisfiable (i.e., $\langle \mathbb{x}, \cdot \rangle \in \mathcal{R}_{\mathrm{R1CS}}$) is of size $O(n)$.

3.2 Succinct interactive arguments of knowledge

Let $\langle \mathcal{P}, \mathcal{V} \rangle$ denote a pair of PPT interactive algorithms and $\mathtt{Setup}$ denote an algorithm that outputs public parameters pp given as input the security parameter λ.

Definition 3.5. A protocol between a pair of PPT algorithms $\langle \mathcal{P}, \mathcal{V} \rangle$ is called a public-coin succinct interactive argument of knowledge for a language $\mathcal{L}$ if:

- **Completeness.** For any problem instance $\mathbb{x} \in \mathcal{L}$, there exists a witness w such that for all $r \in \{0,1\}^*$, $\Pr\{\langle \mathcal{P}(pp, w), \mathcal{V}(pp, r) \rangle(\mathbb{x}) = 1\} \geq 1 - \mathtt{negl}(\lambda)$.
- **Soundness.** For any non-satisfiable problem instance $\mathbb{x}$, any PPT prover $\mathcal{P}^*$, and for all $w, r \in \{0,1\}^*$, $\Pr\{\langle \mathcal{P}^*(pp, w), \mathcal{V}(pp, r) \rangle(\mathbb{x}) = 1\} \leq \mathtt{negl}(\lambda)$.
- **Knowledge soundness.** For any PPT adversary $\mathcal{A}$, there exists a PPT *extractor* $\mathcal{E}$ such that $\forall \mathbb{x} \in \mathcal{L}, \forall w, r \in \{0,1\}^*$, if $\Pr\{\langle \mathcal{A}(pp, w), \mathcal{V}(pp, r) \rangle(\mathbb{x}) = 1\} \geq \mathtt{negl}(\lambda)$, then $\Pr\{\mathtt{Sat}_{\mathcal{L}}(\mathbb{x}, \mathcal{E}^{\mathcal{A}}(pp, \mathbb{x})) = 1\} \geq \mathtt{negl}(\lambda)$.
- **Succinctness.** The total communication between $\mathcal{P}$ and $\mathcal{V}$ is sub-linear in the size of the NP statement $\mathbb{x} \in \mathcal{L}$.
- **Public coin.** $\mathcal{V}$'s messages are chosen uniformly at random.

We denote the *transcript* of the interaction of two PPTs $\mathcal{P}, \mathcal{V}$ with random tapes $z_{\mathcal{P}}, z_{\mathcal{V}}$ on $\mathbb{x}$ by $tr\langle \mathcal{P}(z_{\mathcal{P}}), \mathcal{V}(z_{\mathcal{V}}) \rangle(\mathbb{x})$

Definition 3.6. A public-coin succinct interactive argument of knowledge is *publicly verifiable* if there is a polynomial time algorithm Accept of the transcript t such that $\mathsf{Accept}(tr\langle\mathcal{P}(z_\mathcal{P}),\mathcal{V}(z_\mathcal{V})\rangle(\mathbb{x}),\mathbb{x}) = \langle\mathcal{P}(z_\mathcal{P}),\mathcal{V}(z_\mathcal{V})\rangle(\mathbb{x})$.

We adapt the following definitions from [66]:

Definition 3.7 (Witness-extended emulation [44])**.** An interactive argument $(\mathsf{Setup},\mathcal{P},\mathcal{V})$ for $\mathcal{L}$ has witness-extended emulation if for all deterministic polynomial time programs $\mathcal{P}^*$ there exists an expected polynomial time emulator E such that for all non-uniform polynomial time adversaries A and all $z_\mathcal{V} \in \{0,1\}^*$, the following probabilities differ by at most $\mathtt{negl}(\lambda)$: $\Pr\{pp \leftarrow \mathsf{Setup}(1^\lambda)\ ;\ (\mathbb{x},z_\mathcal{P}) \leftarrow A(pp)\ ;\ t \leftarrow tr\langle\mathcal{P}^*(z_\mathcal{P}),\mathcal{V}(z_\mathcal{V})\rangle(\mathbb{x}) : \mathcal{A}(t,\mathbb{x}) = 1\}$ and $\Pr\{pp \leftarrow \mathsf{Setup}(1^\lambda)\ ;\ (\mathbb{x},z_\mathcal{P}) \leftarrow A(pp)\ ;\ (t,w) \leftarrow E^{\mathcal{P}^*(z_\mathcal{P})}(\mathbb{x}) : \mathcal{A}(t,\mathbb{x}) = 1 \wedge (\mathsf{Accept}(t) = 1 \Rightarrow \mathtt{Sat}_\mathcal{L}(\mathbb{x},w) = 1)\}$.

Definition 3.8. An interactive argument $(\mathsf{Setup},\mathcal{P},\mathcal{V})$ for $\mathcal{L}$ is computational zero-knowledge if for every PPT interactive machine $\mathcal{V}^*$, there exists a PPT algorithm S called the simulator, running in time polynomial in the length of its first input such that for every problem instance $\mathbb{x} \in \mathcal{L}$, $w \in \mathcal{R}_\mathbb{x}$, and $z \in \{0,1\}^*$, the following holds when the distinguishing gap is considered as a function of $|\mathbb{x}|$:

$$\mathit{View}(\langle\mathcal{P}(w),\mathcal{V}^*(z)\rangle(\mathbb{x})) \approx_c S(\mathbb{x},z),$$

where $\mathit{View}(\langle\mathcal{P}(w),\mathcal{V}^*(z)\rangle(\mathbb{x}))$ denotes the distribution of the transcript of interaction between $\mathcal{P}$ and $\mathcal{V}^*$, and $\approx_c$ denotes that the two quantities are computationally indistinguishable. If the statistical distance between the two distributions is negligible then the interactive argument is said to be statistical zero-knowledge. If the simulator is allowed to abort with probability at most 1/2, but the distribution of its output conditioned on not aborting is identically distributed to $\mathit{View}(\langle\mathcal{P}(w),\mathcal{V}^*(z)\rangle(\mathbb{x}))$, then the interactive argument is called perfect zero-knowledge.

3.3 Polynomials and low-degree extensions

We recall a few basic facts about polynomials:

- A polynomial $\mathcal{G}$ over $\mathbb{F}$ is an expression consisting of a sum of *monomials* where each monomial is the product of a constant (from $\mathbb{F}$) and powers of one or more variables (which take values from $\mathbb{F}$); all arithmetic is performed over $\mathbb{F}$.
- The degree of a monomial is the sum of the exponents of variables in the monomial; the degree of a polynomial $\mathcal{G}$ is the maximum degree of any monomial in $\mathcal{G}$. Furthermore, the degree of a polynomial $\mathcal{G}$ in a particular variable x_i is the maximum exponent that x_i takes in any of the monomials in $\mathcal{G}$.
- A *multivariate* polynomial is a polynomial with more than one variable; otherwise it is called a *univariate* polynomial.

Definition 3.9 (Multilinear polynomial)**.** A multivariate polynomial is called a multilinear polynomial if the degree of the polynomial in each variable is at most one.

Definition 3.10 (Low-degree polynomial)**.** A multivariate polynomial $\mathcal{G}$ over a finite field $\mathbb{F}$ is called low-degree polynomial if the degree of $\mathcal{G}$ in each variable is exponentially smaller than $|\mathbb{F}|$.

Low-degree extensions (LDEs). Suppose $g : \{0,1\}^\ell \to \mathbb{F}$ is a function that maps ℓ-bit elements into an element of $\mathbb{F}$. A *polynomial extension* of g is a low-degree ℓ-variate polynomial $\widetilde{g}(\cdot)$ such that $\widetilde{g}(x) = g(x)$ for all $x \in \{0,1\}^\ell$.

A *multilinear* polynomial extension (or simply, a multilinear extension, or MLE) is a low-degree polynomial extension where the extension is a multilinear polynomial (i.e., the degree of each variable in $\widetilde{g}(\cdot)$ is at most one). Given a function $Z : \{0,1\}^\ell \to \mathbb{F}$, the multilinear extension of $Z(\cdot)$ is the unique multilinear polynomial $\widetilde{Z} : \mathbb{F}^\ell \to \mathbb{F}$. It can be computed as follows.

$$\begin{aligned}\widetilde{Z}(x_1, \ldots, x_\ell) &= \sum_{e \in \{0,1\}^\ell} Z(e) \cdot \widetilde{\mathrm{eq}}(x, e) \\ &= \langle (Z(0), \ldots, Z(2^\ell - 1)), (\widetilde{\mathrm{eq}}(x, 0), \ldots, \widetilde{\mathrm{eq}}(x, 2^\ell - 1)) \rangle\end{aligned}$$

Note that $\widetilde{\mathrm{eq}}(x, e) = \prod_{i=1}^{\ell} (e_i \cdot x_i + (1 - e_i) \cdot (1 - x_i))$, which is the MLE of the following function:

$$\mathrm{eq}(x, e) = \begin{cases} 1 & \text{if } x = e \\ 0 & \text{otherwise} \end{cases}$$

For any $r \in \mathbb{F}^\ell$, $\widetilde{Z}(r)$ can be computed in $O(2^\ell)$ operations in $\mathbb{F}$ [63, 64].

Dense representation for multilinear polynomials. Since the MLE of a function is unique, it offers the following method to represent any multilinear polynomial. Given a multilinear polynomial $\mathcal{G}(\cdot) : \mathbb{F}^\ell \to \mathbb{F}$, it can be represented uniquely by the list of evaluations of $\mathcal{G}(\cdot)$ over the Boolean hypercube $\{0,1\}^\ell$ (i.e., a function that maps $\{0,1\}^\ell \to \mathbb{F}$). We denote such a representation of $\mathcal{G}$ as $\texttt{DenseRepr}(\mathcal{G})$.

Lemma 3.1. *If for any $x \in \{0,1\}^\ell$, $\mathcal{G}(x) = 0$ then $\texttt{DenseRepr}(\mathcal{G})$ does not have to include an entry for x.*

Proof. Recall the closed-form expression for evaluating $\mathcal{G}(\cdot)$ at $(r_1, \ldots, r_\ell) \in \mathbb{F}^\ell$: $\mathcal{G}(r_1, \ldots, r_\ell) = \sum_{x \in \{0,1\}^\ell} \mathcal{G}(x) \cdot \prod_{i=1}^{\ell} (r_i \cdot x_i + (1 - r_i) \cdot (1 - x_i))$. Observe that if for any $x \in \{0,1\}^\ell$, $\mathcal{G}(x) = 0$, x does not contribute to $\mathcal{G}(r)$ for any $r \in \mathbb{F}^\ell$. □

Definition 3.11. A multilinear polynomial $\mathcal{G} : \mathbb{F}^\ell \to \mathbb{F}$ is a sparse multilinear polynomial if $|\texttt{DenseRepr}(\mathcal{G})|$ is sub-linear in $O(2^\ell)$. Otherwise, it is a dense multilinear polynomial.

As an example, suppose $\mathcal{G} : \mathbb{F}^{2s} \to \mathbb{F}$. Suppose $|\texttt{DenseRepr}(\mathcal{G})| = O(2^s)$, then $\mathcal{G}(\cdot)$ is a sparse multilinear polynomial because $O(2^s)$ is sublinear in $O(2^{2s})$.

3.4 Commitment schemes

We adopt our definitions in this subsection and the next from Bünz et al. [26] where they generalize the definition of Kate et al. [46] to allow interactive evaluation proofs. We also borrow their notation: in a list of arguments or returned tuples, variables before the semicolon are public and the ones after are secret; when there is no secret information, semicolon is omitted.

A commitment scheme for some space of messages $\mathcal{X}$ is a tuple of three protocols $(\mathsf{Setup}, \mathsf{Commit}, \mathsf{Open})$:

- $pp \leftarrow \mathsf{Setup}(1^\lambda)$: produces public parameters pp.
- $(\mathcal{C}; \mathcal{S}) \leftarrow \mathsf{Commit}(pp; x)$: takes as input some $x \in \mathcal{X}$; produces a public commitment $\mathcal{C}$ and a secret opening hint $\mathcal{S}$.
- $b \leftarrow \mathsf{Open}(pp, \mathcal{C}, x, \mathcal{S})$: verifies the opening of commitment $\mathcal{C}$ to $x \in \mathcal{X}$ with the opening hint $\mathcal{S}$; outputs $b \in \{0,1\}$.

Definition 3.12. A tuple of three protocols $(\mathsf{Setup}, \mathsf{Commit}, \mathsf{Open})$ is a binding commitment scheme for $\mathcal{X}$ if:

Binding. For any PPT adversary $\mathcal{A}$,

$$\Pr\left\{\begin{array}{c} pp \leftarrow \mathsf{Setup}(1^\lambda);\ (\mathcal{C}, \mathcal{G}_0, \mathcal{G}_1, \mathcal{S}_0, \mathcal{S}_1) = \mathcal{A}(pp); \\ b_0 \leftarrow \mathsf{Open}(pp, \mathcal{C}, \mathcal{G}_0, \mathcal{S}_0);\ b_1 \leftarrow \mathsf{Open}(pp, \mathcal{C}, \mathcal{G}_1, \mathcal{S}_1): \\ b_0 = b_1 \neq 0 \wedge \mathcal{G}_0 \neq \mathcal{G}_1 \end{array}\right\} \leq \mathtt{negl}(\lambda)$$

Definition 3.13. A commitment scheme $(\mathsf{Setup}, \mathsf{Commit}, \mathsf{Open})$ provides hiding commitments if for all PPT adversaries $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$:

$$\left|1 - 2 \cdot \Pr\left\{\begin{array}{c} b = \bar{b}: \\ pp \leftarrow \mathsf{Setup}(1^\lambda); \\ (\mathcal{G}_0, \mathcal{G}_1, st) = \mathcal{A}_0(pp); \\ b \leftarrow_R \{0,1\}; \\ (\mathcal{C}, \mathcal{S}) \leftarrow \mathsf{Commit}(pp; \mathcal{G}_b); \\ \bar{b} \leftarrow \mathcal{A}_1(st, \mathcal{C}) \end{array}\right\}\right| \leq \mathtt{negl}(\lambda)$$

If the above holds for all algorithms, then the commitment is statistically hiding.

3.5 Polynomial commitments for multilinear polynomials

Suppose that $(\mathsf{Setup}_\mathbb{F}, \mathsf{Commit}_\mathbb{F}, \mathsf{Open}_\mathbb{F})$ is a commitment scheme for $\mathcal{X} = \mathbb{F}$. WLOG, when algorithms below accept as input a multilinear polynomial, they use the dense representation of multilinear polynomials (§3.3).

Definition 3.14. A tuple of four protocols $(\mathsf{Setup}, \mathsf{Commit}, \mathsf{Open}, \mathsf{Eval})$ is a polynomial commitment scheme for ℓ-variate multilinear polynomials over $\mathbb{F}$ if $(\mathsf{Setup}, \mathsf{Commit}, \mathsf{Open})$ is a commitment scheme for ℓ-variate multilinear polynomials over $\mathbb{F}$, and:

- $pp \leftarrow \mathsf{Setup}(1^\lambda)$, $pp_\mathbb{F} \leftarrow \mathsf{Setup}_\mathbb{F}(1^\lambda)$. Both $\mathcal{V}$ and $\mathcal{P}$ hold a commitment $\mathcal{C}_G$ to G.
- $\mathcal{V}$ selects a public coin $r \in_R \mathbb{F}^\ell$; $\mathcal{P}$ then supplies a commitment $\mathcal{C}_v$ to a scalar $v \in \mathbb{F}$.
- $b \leftarrow \mathsf{Eval}(pp, pp_\mathbb{F}, \mathcal{C}_\mathcal{G}, r, \mathcal{C}_v; \mathcal{G}, \mathcal{S}_\mathcal{G}, \mathcal{S}_v)$ is an interactive public-coin protocol between a PPT prover $\mathcal{P}$ and verifier $\mathcal{V}$. $\mathcal{P}$ additionally knows a ℓ-variate multilinear polynomial $\mathcal{G} \in \mathbb{F}[X_1, \ldots, X_\ell]$ and its secret opening hint $\mathcal{S}_\mathcal{G}$, and the scalar $v \in \mathbb{F}$ and its secret opening hint $\mathcal{S}_v$. $\mathcal{P}$ attempts to convince $\mathcal{V}$ that $\mathcal{G}(r) = v$. At the end of the protocol, $\mathcal{V}$ outputs $b \in \{0,1\}$.

Definition 3.15. A polynomial commitment scheme for ℓ-variable multilinear polynomials over $\mathbb{F}$ is *extractable* if:

- **Completeness.** For any ℓ-variate multilinear polynomial $\mathcal{G} \in \mathbb{F}[X_1, \ldots, X_\ell]$,

$$\Pr\left\{ \begin{array}{c} pp \leftarrow \mathsf{Setup}(1^\lambda);\ pp_{\mathbb{F}} \leftarrow \mathsf{Setup}_{\mathbb{F}}(1^\lambda) \\ (\mathcal{C}_{\mathcal{G}}, \mathcal{S}_{\mathcal{G}}) \leftarrow \mathsf{Commit}(pp; \mathcal{G});\ (\mathcal{C}_v, \mathcal{S}_v) \leftarrow \mathsf{Commit}_{\mathbb{F}}(pp_{\mathbb{F}}; v): \\ \mathsf{Eval}(pp, pp_{\mathbb{F}}, \mathcal{C}_{\mathcal{G}}, r, \mathcal{C}_v, ; \mathcal{G}, \mathcal{S}_{\mathcal{G}}, \mathcal{S}_v) = 1 \wedge v = \mathcal{G}(r) \end{array} \right\} \geq 1 - \mathtt{negl}(\lambda)$$

- **Knowledge soundness.** Eval is a public-coin succinct interactive argument of knowledge with witness-extended emulation (Definition 3.7) for the following NP relation given $pp \leftarrow \mathsf{Setup}(1^\lambda)$, $pp_{\mathbb{F}} \leftarrow \mathsf{Setup}_{\mathbb{F}}(1^\lambda)$, and $r \in \mathbb{F}^\ell$ chosen after $\mathcal{C}_{\mathcal{G}}$ is fixed:

$$\mathcal{R}_{\mathsf{Eval}}(pp, pp_{\mathbb{F}}) = \left\{ \begin{array}{c} \langle (\mathcal{C}_{\mathcal{G}}, \mathcal{C}_v), (\mathcal{G}, \mathcal{S}_{\mathcal{G}}, \mathcal{S}_v) \rangle : \\ \mathcal{G} \in \mathbb{F}[X_1, \ldots, X_\ell] \text{ is multilinear } \wedge v \in \mathbb{F} \wedge \mathcal{G}(r) = v \\ \wedge \mathsf{Open}(pp; \mathcal{C}_{\mathcal{G}}, \mathcal{G}, \mathcal{S}_{\mathcal{G}}) = 1 \wedge \mathsf{Open}_{\mathbb{F}}(pp_{\mathbb{F}}; \mathcal{C}_v, v, \mathcal{S}_v) = 1 \end{array} \right\}$$

Definition 3.16. An extractable polynomial commitment scheme $(\mathsf{Setup}, \mathsf{Commit}, \mathsf{Open}, \mathsf{Eval})$ with hiding commitments (Definition 3.13) is zero-knowledge if Eval is a public-coin succinct interactive argument of knowledge with witness-extended emulation (Definition 3.7) and zero-knowledge (Definition 3.8) for the following NP relation given $pp \leftarrow \mathsf{Setup}(1^\lambda)$, $pp_{\mathbb{F}} \leftarrow \mathsf{Setup}_{\mathbb{F}}(1^\lambda)$, and $r \in \mathbb{F}^\ell$ chosen after $\mathcal{C}_{\mathcal{G}}$ is fixed:

$$\mathcal{R}_{\mathsf{Eval}}(pp, pp_{\mathbb{F}}) = \left\{ \begin{array}{c} \langle (\mathcal{C}_{\mathcal{G}}, \mathcal{C}_v), (\mathcal{G}, \mathcal{S}_{\mathcal{G}}, v, \mathcal{S}_v) \rangle : \mathcal{G} \in \mathbb{F}[X_1, \ldots, X_\ell] \text{ is multilinear } \wedge \\ \mathcal{G}(r) = v \wedge \mathsf{Open}(pp; \mathcal{C}_{\mathcal{G}}, \mathcal{G}, \mathcal{S}_{\mathcal{G}}) = 1 \wedge \mathsf{Open}_{\mathbb{F}}(pp_{\mathbb{F}}; \mathcal{C}_v, v, \mathcal{S}_v) = 1 \end{array} \right\}$$

Remark 3.1. Note that in this definition, r is not chosen by the adversary. This weakening is required for the extractability of prior polynomial commitments [27, 49, 66] and Kopis-PC (§4). In our and prior [58, 66] use of these polynomial commitment schemes, $\mathcal{V}$ selects points of evaluation at random. However, for a multilinear polynomial $\mathcal{G}$, if the evaluation point is *not* chosen after the commitment is fixed, one can employ a simple reduction to transform the evaluation claim to a claim about an evaluation at a random point r' where r' is chosen after the polynomial commitment is fixed.

3.6 Inner product proofs (IPPs)

Suppose that $(\mathsf{Setup}_{\mathbb{F}}, \mathsf{Commit}_{\mathbb{F}}, \mathsf{Open}_{\mathbb{F}})$ denotes a commitment scheme for $\mathcal{X} = \mathbb{F}$.

Definition 3.17. A tuple of four protocols $\mathsf{IPP} = (\mathsf{Setup}, \mathsf{Commit}, \mathsf{Open}, \mathsf{Eval})$ is an inner product proof system for s-length vectors over $\mathbb{F}$ if $(\mathsf{IPP.Setup}, \mathsf{IPP.Commit}, \mathsf{IPP.Open})$ is a commitment scheme for s-length vectors over $\mathbb{F}$, and:

- $b \leftarrow \mathsf{Eval}(pp, pp_{\mathbb{F}}, \mathcal{C}_Z, V, \mathcal{C}_y; Z, \mathcal{S}_Z, \mathcal{S}_y)$ is an interactive public-coin protocol between a PPT prover $\mathcal{P}$ and verifier $\mathcal{V}$. pp refers to an output of $\mathsf{IPP.Setup}(1^\lambda)$ and $pp_{\mathbb{F}}$ refers to an output of $\mathsf{Setup}_{\mathbb{F}}(1^\lambda)$. Both $\mathcal{V}$ and $\mathcal{P}$ hold a commitment $\mathcal{C}_Z$ to a vector $Z \in \mathbb{F}^s$, a commitment $\mathcal{C}_y$ to a scalar $y \in \mathbb{F}$, and $V \in \mathbb{F}^s$. $\mathcal{P}$ additionally knows a a vector $Z \in \mathbb{F}^s$ and its secret opening hint $\mathcal{S}_Z$, and the scalar $y \in \mathbb{F}$ and its secret opening hint $\mathcal{S}_y$. $\mathcal{P}$ attempts to convince $\mathcal{V}$ that $y = \langle Z, V \rangle$. At the end of the protocol, $\mathcal{V}$ outputs $b \in \{0, 1\}$.

Definition 3.18. An inner product proof system for s-length vectors satisfies:

- **Completeness.** For any s-length vector $Z \in \mathbb{F}^s$,

$$\Pr\left\{\begin{array}{c} pp \leftarrow \mathsf{Setup}(1^\lambda);\ pp_{\mathbb{F}} \leftarrow \mathsf{Setup}_{\mathbb{F}}(1^\lambda) \\ (\mathcal{C}_Z, \mathcal{S}_Z) \leftarrow \mathsf{Commit}(pp; Z);\ (\mathcal{C}_y, \mathcal{S}_y) \leftarrow \mathsf{Commit}_{\mathbb{F}}(pp_{\mathbb{F}}; y): \\ \mathsf{Eval}(pp, pp_{\mathbb{F}}, \mathcal{C}_Z, V, \mathcal{C}_y, ; Z, \mathcal{S}_Z, \mathcal{S}_y) = 1 \wedge y = \langle Z, V\rangle \end{array}\right\} \geq 1 - \mathtt{negl}(\lambda)$$

- **Knowledge soundness.** Eval is a public-coin succinct interactive argument of knowledge with witness-extended emulation (Definition 3.7) for the following NP relation given $pp \leftarrow \mathsf{Setup}(1^\lambda)$ and $pp_{\mathbb{F}} \leftarrow \mathsf{Setup}_{\mathbb{F}}(1^\lambda)$:

$$\mathcal{R}_{\mathsf{Eval}}(pp, pp_{\mathbb{F}}) = \left\{\begin{array}{c} \langle(\mathcal{C}_Z, V, \mathcal{C}_y), (Z, \mathcal{S}_Z, \mathcal{S}_y)\rangle : \\ Z \in \mathbb{F}^s \wedge y \in \mathbb{F} \wedge y = \langle Z, V\rangle \wedge \\ \mathsf{Open}(pp; \mathcal{C}_Z, Z, \mathcal{S}_Z) = 1 \wedge \mathsf{Open}_{\mathbb{F}}(pp_{\mathbb{F}}; \mathcal{C}_y, y, \mathcal{S}_y) = 1 \end{array}\right\}$$

Definition 3.19. An inner product proof system for s-length vectors $(\mathsf{Setup}, \mathsf{Commit}, \mathsf{Open}, \mathsf{Eval})$ with hiding commitments (Definition 3.13) is zero-knowledge if Eval is a public-coin succinct interactive argument of knowledge with witness-extended emulation (Definition 3.7) and zero-knowledge (Definition 3.8) for the following NP relation given $pp \leftarrow \mathsf{Setup}(1^\lambda)$ and $pp_{\mathbb{F}} \leftarrow \mathsf{Setup}_{\mathbb{F}}(1^\lambda)$:

$$\mathcal{R}_{\mathsf{Eval}}(pp, pp_{\mathbb{F}}) = \left\{\begin{array}{c} \langle(\mathcal{C}_Z, V, \mathcal{C}_y), (Z, \mathcal{S}_Z, y, \mathcal{S}_y)\rangle : Z \in \mathbb{F}^s \wedge \langle Z, V\rangle = y \wedge \\ \mathsf{Open}(pp; \mathcal{C}_Z, Z, \mathcal{S}_Z) = 1 \wedge \mathsf{Open}_{\mathbb{F}}(pp_{\mathbb{F}}; \mathcal{C}_y, y, \mathcal{S}_y) = 1 \end{array}\right\}$$

3.7 Bilinear inner product proofs (BIPPs)

Definition 3.20. A tuple of four protocols $\mathsf{BIPP} = (\mathsf{Setup}, \mathsf{Commit}, \mathsf{Open}, \mathsf{Eval})$ is a bilinear inner product proof system for s-length vectors over $\mathbb{G}_1$ if $(\mathsf{BIPP.Setup}, \mathsf{BIPP.Commit}, \mathsf{BIPP.Open})$ is a commitment scheme for s-length vectors over $\mathbb{G}_1$, and:

- $b \leftarrow \mathsf{Eval}(pp, \mathcal{C}_Z, V, y; Z, \mathcal{S}_Z)$ is an interactive public-coin protocol between a PPT prover $\mathcal{P}$ and verifier $\mathcal{V}$. pp refers to an output of $\mathsf{BIPP.Setup}(1^\lambda)$. Both $\mathcal{V}$ and $\mathcal{P}$ hold a commitment $\mathcal{C}_Z$ to a vector $Z \in \mathbb{G}_1^s$, a $y \in \mathbb{G}_1$, and $V \in \mathbb{F}^s$. $\mathcal{P}$ additionally knows a a vector $Z \in \mathbb{G}_1^s$ and its secret opening hint $\mathcal{S}_Z$. $\mathcal{P}$ attempts to convince $\mathcal{V}$ that $y = \langle Z, V\rangle$. At the end of the protocol, $\mathcal{V}$ outputs $b \in \{0, 1\}$.

Definition 3.21. A bilinear inner product proof system for s-length vectors satisfies:

- **Completeness.** For any s-length vector $Z \in \mathbb{G}_1^s$,

$$\Pr\left\{\begin{array}{c} pp \leftarrow \mathsf{Setup}(1^\lambda); \\ (\mathcal{C}_Z, \mathcal{S}_Z) \leftarrow \mathsf{Commit}(pp; Z); \\ \mathsf{Eval}(pp, \mathcal{C}_Z, V, y; Z, \mathcal{S}_Z) = 1 \wedge y = \langle Z, V\rangle \end{array}\right\} \geq 1 - \mathtt{negl}(\lambda)$$

- **Knowledge soundness.** Eval is a public-coin succinct interactive argument of knowledge with witness-extended emulation (Definition 3.7) for the following NP relation given $pp \leftarrow \mathsf{Setup}(1^\lambda)$:

$$\mathcal{R}_{\mathsf{Eval}}(pp) = \left\{\begin{array}{c} \langle(\mathcal{C}_Z, V, y), (Z, \mathcal{S}_Z)\rangle : \\ Z \in \mathbb{G}_1^s \wedge y \in \mathbb{G}_1 \wedge y = \langle Z, V\rangle \wedge \\ \mathsf{Open}(pp; \mathcal{C}_Z, Z, \mathcal{S}_Z) = 1 \end{array}\right\}$$

```
// reduces the claim ∑_{x∈{0,1}^s} G(x) ?= T to G(r) ?= e
function SumCheckReduce(μ, ℓ, T, r)
    (r_1, r_2, ..., r_μ) ← r
    e ← T
    for i = 1, 2, ..., μ do
        G_i(·) ← ReceiveFromProver() // an honest P_SC returns {G_i(0), G_i(1), ... G_i(ℓ)}
        if G_i(0) + G_i(1) ≠ e then
            return 0
        SendToProver(r_i)
        e ← G_i(r_i) // evaluate G_i(r_i) using its point-value form received from the prover
    return e
```

FIGURE 4—The sum-check protocol. $\mathcal{V}_{SC}$ checks if a μ-variate polynomial $\mathcal{G}(\cdot)$ sums to T over the Boolean hypercube $\{0,1\}^\mu$ with the assistance of a prover $\mathcal{P}_{SC}$. The degree of $\mathcal{G}(\cdot)$ in each variable is at most ℓ.

3.8 The sum-check protocol

The sum-check protocol is a seminal interactive proof protocol (an interactive argument where soundness holds unconditionally), which we now elaborate.

Suppose that there is a ℓ-variate low-degree polynomial, $\mathcal{G} : \mathbb{F}^\ell \to \mathbb{F}$ where the degree of $\mathcal{G}$ in each variable is $\leq d$. The sum-check protocol enables a prover $\mathcal{P}_{SC}$ to prove to a verifier $\mathcal{V}_{SC}$ claims of the following form, which we call *sum-check instances*:

$$T = \sum_{x_1 \in \{0,1\}} \sum_{x_2 \in \{0,1\}} \cdots \sum_{x_\ell \in \{0,1\}} \mathcal{G}(x_1, x_2, \ldots, x_\ell)$$

Of course, given $\mathcal{G}$, $\mathcal{V}_{SC}$ can deterministically evaluate the above sum and verify whether that the sum is T—without requiring any assistance from $\mathcal{P}_{SC}$. But, $\mathcal{V}_{SC}$ requires computation exponential in ℓ. With the sum-check protocol, $\mathcal{V}_{SC}$ requires far less computation at the cost of a probabilistic soundness guarantee.

In the sum-check protocol, $\mathcal{V}_{SC}$ interacts with $\mathcal{P}_{SC}$ over a sequence of ℓ rounds where in each round $\mathcal{V}_{SC}$ sends a random challenge (i.e., a public coin) and $\mathcal{P}$ responds with a message of size $O(d)$. At the end of this interaction, $\mathcal{V}_{SC}$ outputs $b \in \{0,1\}$. The principal cost to $\mathcal{V}_{SC}$ is to evaluate $\mathcal{G}$ at a random point in its domain $r \in \mathbb{F}^\ell$. We denote the sum-check protocol as $b \leftarrow \langle \mathcal{P}_{SC}, \mathcal{V}_{SC}(r) \rangle (\mathcal{G}, \ell, d, T)$. For any ℓ-variate polynomial $\mathcal{G}$ with degree at most d in each variable, the following properties hold.

- **Completeness.** If $T = \sum_{x \in \{0,1\}^\ell} \mathcal{G}(x)$, then for a correct $\mathcal{P}_{SC}$ and for all $r \in \{0,1\}^*$, $\Pr\{\langle \mathcal{P}_{SC}(\mathcal{G}), \mathcal{V}_{SC}(r) \rangle (\ell, d, T) = 1\} = 1$.
- **Soundness.** If $T \neq \sum_{x \in \{0,1\}^\ell} \mathcal{G}(x)$, then for any $\mathcal{P}^\star_{SC}$ and for all $r \in \{0,1\}^*$, $\Pr_r\{\langle \mathcal{P}^\star_{SC}(\mathcal{G}), \mathcal{V}_{SC}(r) \rangle (\ell, d, T) = 1\} \leq d \cdot \ell / |\mathbb{F}|$.
- **Succinctness.** The communication between $\mathcal{P}_{SC}$ and $\mathcal{V}_{SC}$ is $O(d \cdot \ell)$ elements of $\mathbb{F}$.

An alternate formulation. The sum-check protocol is a mechanism to reduce a claim of the form $\sum_{x \in \{0,1\}^m} \mathcal{G}(x) \stackrel{?}{=} T$ to the claim $\mathcal{G}(r) \stackrel{?}{=} e$. In most cases, $\mathcal{V}_{SC}$ uses an auxiliary protocol to verify the latter claim, so this formulation makes it easy to describe end-to-end protocols. We denote this reduction protocol with $e \leftarrow \langle \mathcal{P}_{SC}(\mathcal{G}), \mathcal{V}_{SC}(r) \rangle (\ell, d, T)$. Figure 4 depicts the sum-check protocol from this perspective.

4 A new commitment scheme for multilinear polynomials

This section describes Kopis-PC, a new polynomial commitment scheme for multilinear polynomials without requiring a trusted setup.

Our scheme can be seen as an extension and generalization of the polynomial commitment scheme of Wahby et al. for multilinear polynomials [66]. Specifically, instead of only relying on singly-homomorphic commitments of Pedersen, our scheme augments the scheme of Wahby et al. [66] with doubly-homomorphic commitments of Abe et al. [7]. Whereas the scheme of Wahby et al. [66] requires only a group where DLOG is hard, our scheme requires a blinear group where SXDH is hard. In exchange, we obtain a substantial improvement in polynomial commitment sizes: for an ℓ-variate multilinear polynomial, the commitment size drops from $O_\lambda(2^{\ell/2})$ to $O_\lambda(1)$. Polynomial evaluation proof sizes increase by a small constant factor ($\approx$6): instead of $O(\ell)$ elements of a group where DLOG is hard, our scheme produces $O(\ell)$ elements of a target group in a bilinear group where SXDH is hard. Nevertheless, in the context of Spartan, we obtain an exponential improvement since it often involves the following three steps: (1) the prover sends one or more polynomial commitments; (2) the prover uses the sum-check protocol to prove certain sum-check instances; and (3) the prover produces polynomial evaluation proofs. For an n-sized R1CS instance, the proof size contribution from steps (1) and (3) drops from $O_\lambda(\sqrt{n})$ to $O_\lambda(\log n)$.

Our scheme can also be seen as an adaptation of the polynomial commitment scheme for bivariate polynomials in the work of Bunz et al. [27] to the setting of multilinear polynomials. While this may appear straightforward in hindsight, our observation is new. For example, Bunz et al. [27] describe two schemes for bivariate polynomials, but it appears that only one of them can be adapted to multilinear polynomials.

4.1 Details of Kopis-PC

Suppose that $\widetilde{Z}$ is an ℓ-variate multilinear polynomial over $\mathbb{F}$. Recall that $\widetilde{Z}$ can be represented uniquely using a table of its evaluations over the Boolean hypercube $\{0,1\}^\ell$ (§3.3). Conveniently, we denote such a table of evaluations as Z. We will abuse notation and treat Z as a function that maps ℓ-bit strings to elements of $\mathbb{F}$: $Z : \{0,1\}^\ell \to \mathbb{F}$. Naturally, $\forall x \in \{0,1\}^\ell$, $\widetilde{Z}(x) = Z(x)$. Furthermore, recall from Section 3.3 that for $r \in \mathbb{F}^\ell$,

$$\widetilde{Z}(r) = \sum_{i \in \{0,1\}^\ell} \widetilde{\mathrm{eq}}(i,r) \cdot Z(i)$$

WLOG, suppose that ℓ is even. Furthermore, let $s = \ell/2$ and $r = (r_x, r_y)$, where $r_x, r_y \in \mathbb{F}^s$ and (r_x, r_y) denotes a concatenation of two vectors in an obvious fashion. We can rewrite the above equation as follows.

$$\begin{aligned}\widetilde{Z}(r_x, r_y) &= \sum_{(i,j) \in (\{0,1\}^s, \{0,1\}^s)} Z(i,j) \cdot \widetilde{\mathrm{eq}}(i, r_x) \cdot \widetilde{\mathrm{eq}}(j, r_y) \\ &= \sum_{i \in \{0,1\}^s} \widetilde{\mathrm{eq}}(i, r_x) \cdot \sum_{j \in \{0,1\}^s} Z(i,j) \cdot \widetilde{\mathrm{eq}}(j, r_y)\end{aligned}$$

It is also convenient to treat Z as an $s \times s$ matrix with $L(i) = \mathrm{eq}(i, r_x)$ and $R(j) = \mathrm{eq}(j, r_y)$ as vectors of evaluations for all $i, j \in \{0,1\}^s$. With such a formulation, the following holds: $\widetilde{Z}(r) = (L \cdot Z) \cdot R$.

Scheme. We assume that there exists an inner product proof system IPP and a bilinear inner product proof system BIPP. Wahby et al. [66] provide an adaptation of Bulletproofs' inner product argument [28] that serves as our IPP. Bunz et al. [27] provide a generalization of Bulletproofs' inner product argument that serves as our BIPP.

Kopis-PC is identical to the polynomial commitment scheme of Wahby et al. [66] except that they do not use BIPP, so the verifier must compute a weighted sum of group elements locally after receiving $O(2^{\ell/2})$-sized commitment. In Kopis-PC, the verifier receives an $O_\lambda(1)$-sized commitment. Furthermore, BIPP enables the verifier in Kopis-PC to verifiably offload the necessary computation of weighted sum to the prover. Using proofs analogous to the ones in prior work [27, 49, 66], it is straightforward to show that the scheme below is a polynomial commitment scheme for multilinear polynomials. The Eval depicted below is not a zero-knowledge interactive argument, but it can be extended via standard techniques [49].

- $pp \leftarrow \mathsf{Setup}(1^\lambda, \ell)$:
  1. $s \leftarrow 2^{\ell/2}$
  2. $pp_{out} \leftarrow \mathsf{BIPP.Setup}(1^\lambda, s)$
  3. $pp_{in} \leftarrow \mathsf{IPP.Setup}(1^\lambda, s)$
  4. Output (pp_{out}, pp_{in})
- $(\mathcal{C}_\mathcal{G}; \mathcal{S}_\mathcal{G}) \leftarrow \mathsf{Commit}(pp, \mathcal{G})$:
  1. Let Z denote a matrix representation of evaluations of $\mathcal{G}$ over $\{0,1\}^\ell$
  2. $(\mathcal{C}_0, \ldots, \mathcal{C}_{s-1}; \mathcal{S}_0, \ldots, \mathcal{S}_{s-1}) \leftarrow \forall i \in \{0, \ldots, s-1\} :: \mathsf{IPP.Commit}(pp.pp_{in}, Z(i))$, where $Z(i)$ is the ith row of Z with s elements.
  3. $(\mathcal{C}_\mathcal{G}; \mathcal{S}_{out}) \leftarrow \mathsf{BIPP.Commit}(pp.pp_{out}, (\mathcal{C}_0, \ldots, \mathcal{C}_{s-1}))$
  4. $\mathcal{S}_\mathcal{G} \leftarrow (\mathcal{C}_0, \ldots, \mathcal{C}_{s-1}, \mathcal{S}_0, \ldots, \mathcal{S}_{s-1}, \mathcal{S}_{out})$
  5. Output $(\mathcal{C}_\mathcal{G}, \mathcal{S}_\mathcal{G})$
- $b \leftarrow \mathsf{Eval}(pp, pp_\mathbb{F}, \mathcal{C}_\mathcal{G}, r, \mathcal{C}_v; \mathcal{G}, \mathcal{S}_\mathcal{G}, \mathcal{S}_v)$
  1. $\mathcal{V}, \mathcal{P}$: $(r_x, r_y) \leftarrow r$, where $r = (r_x, r_y)$ and $r_x, r_y \in \mathbb{F}^{\ell/2}$
  2. $\mathcal{V}, \mathcal{P}$: $L = \forall i :: \mathrm{eq}(i, r_x)$, so $L \in \mathbb{F}^s$ where $s = 2^{\ell/2}$.
  3. $\mathcal{P}$: $y_{out} \leftarrow \langle (\mathcal{S}_\mathcal{G}.\mathcal{C}_0, \ldots, \mathcal{S}_\mathcal{G}.\mathcal{C}_{s-1}), L \rangle$
  4. $\mathcal{P} \rightarrow \mathcal{V}$: y_{out}
  5. $\mathcal{V}, \mathcal{P}$: $b_{out} \leftarrow \mathsf{BIPP.Eval}(pp.pp_{out}, pp_{\mathbb{G}_1}, \mathcal{C}_\mathcal{G}, L, y_{out}; (\mathcal{S}_\mathcal{G}.\mathcal{C}_0, \ldots, \mathcal{S}_\mathcal{G}.\mathcal{C}_{s-1}))$
  6. $\mathcal{V}$: Abort with $b = 0$ if $b_{out} = 0$
  7. $\mathcal{V}, \mathcal{P}$: $R = \forall j :: \mathrm{eq}(j, r_y)$, so $R \in \mathbb{F}^s$ where $s = 2^{\ell/2}$.
  8. $\mathcal{V}, \mathcal{P}$: $b_{in} \leftarrow \mathsf{IPP.Eval}(pp.pp_{in}, pp_\mathbb{F}, y_{out}, R, \mathcal{C}_v; L \cdot Z, \langle L, (\mathcal{S}.\mathcal{S}_0, \ldots, \mathcal{S}.\mathcal{S}_{s-1}) \rangle, \mathcal{S}_v)$
  9. $\mathcal{V}$: Abort with $b = 0$ if $b_{in} = 0$
  10. $\mathcal{V}$: Output $b = 1$

Analysis of costs. The following table summarizes costs under Kopis-PC and compares with the scheme of Wahby et al. [66], denoted with Hyrax-PC. We also include costs for Dory-PC [49], which follows the same blueprint as Kopis-PC, except that it leverages the

tensor structure in L and R vectors (which are of size $\sqrt{n}$, where $n = 2^\ell$ for an ℓ-variate multilinear polynomial) to avoid materializing them, thereby enabling it to achieve $O_\lambda(\log n)$ verification costs instead of $O_\lambda(\sqrt{n})$ costs under Kopis-PC and Hyrax-PC.

scheme	Commit	$\lvert\mathcal{C}\rvert$	$\mathcal{P}_{\mathsf{Eval}}$	$\lvert\pi_{\mathsf{Eval}}\rvert$	$\mathcal{V}_{\mathsf{Eval}}$	assumption
Hyrax-PC [66]	$n\ \mathbb{G}_1$	$\sqrt{n}\ \mathbb{G}_1$	$n\ \mathbb{F}$	$\log n\ \mathbb{G}_1$	$\sqrt{n}\ \mathbb{G}_1$	DLOG
Dory-PC [49]	$n\ \mathbb{G}_1$	$1\ \mathbb{G}_T$	$n\ \mathbb{F}$	$\log n\ \mathbb{G}_T$	$\log n\ \mathbb{G}_T$	SXDH
Kopis-PC	$n\ \mathbb{G}_1$	$1\ \mathbb{G}_T$	$n\ \mathbb{F}$	$\log n\ \mathbb{G}_T$	$\sqrt{n}\ \mathbb{G}_2$	SXDH

5 A new transparent SNARK for proving grand product relations

This section describes a new transparent SNARK, which may be of independent interest, for proving *grand product* relations:

$$\mathcal{R}_{\text{GP}} = \{(P \in \mathbb{F}, V \in \mathbb{F}^m) : P = \prod_i V_i\}$$

Spartan [58] employs a $O(\log m)$-depth layered circuit [63] for computing such grand products. The layered circuit takes as input a vector V and outputs T. In each layer, the circuit computes the Hadamard product between the left and right halves of the vector output by the previous layer. To construct a SNARK for grand product relations, Spartan applies the sum-check protocol in a layered fashion [33, 41, 63] in conjunction with a polynomial commitment scheme [66] to commit to the input represented as multilinear polynomial. A principal downside of this approach is that it requires $O(\log m)$ invocations of the sum-check protocol, and it produces $O_\lambda(\log^2 m)$-sized proofs—ignoring the size of the commitments and proofs for polynomial commitments. In Spartan [58], which employs the polynomial commitment scheme of Wahby et al. [66], the latter incurs $O_\lambda(\sqrt{m})$ costs and dominates proof sizes both asymptotically and concretely.

We improve these proof sizes to $O_\lambda(\log m)$—including the size of the commitment and polynomial evaluation proofs—by leveraging the constant-sized polynomial commitments and logarithmic polynomial evaluation proofs provided by Kopis-PC and Dory-PC. Specifically, we design a new sum-check instance for grand product relations: a polynomial $\mathcal{G}$ that sums to 0 over a certain Boolean hypercube *if and only if* a given $(P \in \mathbb{F}, V \in \mathbb{F}^m) \in \mathcal{R}_{\text{GP}}$. Given such a sum-check instance, our approach to convert it to an interactive argument (and then into a SNARK in the random oracle model) is the same as in prior work [26, 58]: use the sum-check protocol reduce the sum-check instance into a set of polynomial evaluations, and then use a polynomial commitment scheme to prove the correct evaluations of the polynomials.

Details. Let $m = |V|$. WLOG, assume that m is a power of 2, and let $s = \log m$. Let V denote a table of evaluations of a $\log m$-variate multilinear polynomial $v(x)$ over $\{0,1\}^{\log m}$ in a natural fashion.

Lemma 5.1. *$P = \prod_{x \in \{0,1\}^{\log m}} v(x)$ if and only if there exists a multilinear polynomial f in $\log m + 1$ variables such that $f(1,\ldots,1,0) = P$, and $\forall x \in \{0,1\}^{\log m}$, the following hold: $f(0,x) = v(x)$, $f(1,x) = f(x,0) \cdot f(x,1)$*

Proof. To prove the forward implication, define f to be the MLE of its evaluations on the Boolean hypercube: $f(1,\ldots,1) = 0$ and for all $\ell \in 0,\ldots,\log m$ and $x \in \{0,1\}^{\log m - \ell}$,

$f(1^\ell, 0, x) = \prod_{y \in \{0,1\}^\ell} v(x, y)$. Then taking $\ell = 0$ we have $\forall x \in \{0,1\}^{\log m} : f(0,x) = v(x)$, and taking $\ell = \log m$ we have $f(1,\ldots,1,0) = \prod_{x \in \{0,1\}^{\log m}} a(x) = P$. For $\ell > 0$:

$$\begin{aligned} f(1^\ell, 0, x) &= \prod_{y \in \{0,1\}^\ell} v(x, y) = \prod_{y \in \{0,1\}^{\ell-1}} v(x, 0, y) \cdot \prod_{y \in \{0,1\}^{\ell-1}} v(x, 1, y) \\ &= f(1^{\ell-1}, 0, x, 0) \cdot f(1^{\ell-1}, 0, x, 1) \end{aligned}$$

so $f(1,x) = f(x,0) \cdot f(x,1)$ for all $x \in \{0,1\}^{\log m} \setminus \{1,\ldots,1\}$. In this last case, we have: $f(1,\ldots,1) = 0 = f(1,\ldots,0) \cdot f(1,\ldots,1)$. So a suitable f exists.

To prove the reverse implication, for any f satisfying these conditions, we have by induction on $0 \leq \ell \leq \log m$: $\forall x \in \{0,1\}^{\log m - \ell} : f(1^\ell, 0, x) = \prod_{y \in \{0,1\}^\ell} f(0, x, y)$. Then taking $\ell = \log m$ implies that: $P = f(1,\ldots,1,0) = \prod_{x \in \{0,1\}^{\log m}} v(x)$ □

A sum-check instance for grand products. To check that $\forall x \in \{0,1\}^{\log m} f(1,x) = f(x,0) \cdot f(x,1)$, we use a prior idea [22, 29, 58]. Let g be the MLE of the function $f(1,x) - f(x,0)f(x,1))$. In particular,

$$g(t) = \sum_{x \in \{0,1\}^{\log m}} \widetilde{\mathrm{eq}}(t, x) \cdot (f(1,x) - f(x,0) \cdot f(x,1))$$

By the Schwartz–Zippel lemma, except for a soundness error of $\log m/|\mathbb{F}|$ (which is negligible in λ if $|\mathbb{F}|$ is exponential in λ), $g(\tau) = 0$ for τ uniformly random in $\mathbb{F}^{\log m}$ if and only if $g \equiv 0$, which implies that $f(1,x) - f(x,0) \cdot f(x,1) = 0$ for all $x \in \{0,1\}^{\log m}$. Set $\mathcal{G}(x) = \widetilde{eq}(\tau, x)(f(1,x) - f(x,0) \cdot f(x,1))$, where $\mathcal{V}$ picks a random τ. Similarly, to prove that $v(x) = f(0,x)$ for all $x \in \{0,1\}^\ell$ it suffices to prove that $v(\gamma) = f(0,\gamma)$ for a public coin $\gamma \in \mathbb{F}^\ell$.

Thus, to prove the existence of f and hence the grand product relationship, it suffices to prove, for some verifier selected random $\tau, \gamma \in_R \mathbb{F}^\ell$, that:

- $0 = \sum_{x \in \{0,1\}^{\log m}} \widetilde{\mathrm{eq}}(x, \tau) \cdot (f(1,x) - f(x,0) \cdot f(x,1))$
- $f(0,\gamma) = v(\gamma)$
- $f(1,\ldots,1,0) = P$.

SNARKs from combining the sum-check protocol with polynomial commitments. As in Spartan [58] and the compiler of Bunz et al. [26], to build an interactive argument for grand products, $\mathcal{P}$ sends to $\mathcal{V}$ commitments to polynomials v, f. $\mathcal{P}$ and $\mathcal{V}$ run the sum-check reduction to reduce the first claim in the list above to an evaluation of $\mathcal{G}$ at some point r, and $\mathcal{P}$ uses Eval to convince $\mathcal{V}$ of the correctness of commitments to $f(0,r), f(1,r), f(r,0), f(r,1)$ and $f(1,\ldots,1,0)$. This interactive argument is compiled to a SNARK by the Fiat-Shamir transform [37] in the random-oracle model.

6 Sparkle compiler: More efficient sparse polynomial commitments

Spartan provides a compiler, called SPARK, to compile an existing polynomial commitment scheme for dense multilinear polynomials to ones that efficiently handle sparse multilinear polynomials. We now describe a modification of SPARK, which we call Sparkle, that reduces polynomial evaluation proof sizes—without substantially increasing the prover's costs.

To evaluate a sparse multilinear polynomial whose dense representation is of size m (§3.3), SPARK employs $O(m)$-sized circuit with $O(\log m)$ depth. SPARK-derived polynomial commitment schemes implement Eval using a layered sum-check protocol [33, 41, 63] in conjunction with a polynomial commitment scheme [66]. The layered sum-check protocol alone produces $O(\log^2 m)$-sized proofs for sparse polynomial evaluations. Inspecting [58, §7.2.1], The only portion of SPARK's circuit that requires a non-constant depth, is the evaluation of an element of a universal multiset hash function family, requiring the computation of a grand product over elements in a multiset $\mathcal{M}$, where for each in $e \in \mathcal{M}$, $e \in \mathbb{F}$:

$$\mathcal{H}_\gamma(\mathcal{M}) = \prod_{e \in \mathcal{M}} (e - \gamma)$$

Of course, we can employ our special-purpose SNARK for proving grand product relations (§5) instead of using the layered sum-check protocol with $O(\log m)$-depth circuit, bringing proof sizes from $O_\lambda(\log^2 m)$ to $O_\lambda(\log m)$.

dense PC choice	setup	$\mathcal{P}_{\mathsf{Eval}}$	$\lvert\mathcal{C}\rvert$	communication	$\mathcal{V}_{\mathsf{Eval}}$
With SPARK:					
Hyrax-PC [66]	public	$O_\lambda(m)$	$O_\lambda(\sqrt{m})$	$O_\lambda(\log^2 m)$	$O_\lambda(\sqrt{m})$
vSQL-VPD [70]	private	$O_\lambda(m)$	$O_\lambda(1)$	$O_\lambda(\log^2 m)$	$O_\lambda(\log^2 m)$
Virgo-VPD [69]	public	$O_\lambda(m \log m)$	$O_\lambda(1)$	$O_\lambda(\log^2 m)$	$O_\lambda(\log^2 m)$
Kopis-PC	public	$O_\lambda(m)$	$O_\lambda(1)$	$O_\lambda(\log^2 m)$	$O_\lambda(\sqrt{m})$
Dory-PC [49]	public	$O_\lambda(m)$	$O_\lambda(1)$	$O_\lambda(\log^2 m)$	$O_\lambda(\log m)$
With Sparkle:					
vSQL-VPD [70]	private	$O_\lambda(m)$	$O_\lambda(1)$	$O_\lambda(\log m)$	$O_\lambda(\log m)$
Kopis-PC	public	$O_\lambda(m)$	$O_\lambda(1)$	$O_\lambda(\log m)$	$O_\lambda(\sqrt{m})$
Dory-PC [49]	public	$O_\lambda(m)$	$O_\lambda(1)$	$O_\lambda(\log m)$	$O_\lambda(\log m)$

FIGURE 5—Costs of sparse polynomial commitments with different choices for dense PC. Here, m is number of entries in the dense representation of the multilinear polynomial. Applying Sparkle to Hyrax-PC or Virgo-VPD does not improve proof sizes given their commitment sizes and proof sizes respectively.

Unfortunately, the special-purpose SNARK requires the prover to compute commitments to polynomials that encode the intermediate state of the grand product computation. Whereas, with layered circuit approach, most of the commitments that are required are created as part of creating a computation commitment in a preprocessing step. Furthermore, the layered sum-check requires no cryptographic operations since claims about the outputs of each layer i are reduced to claims about outputs of the previous layer of i. Thus, if we naively apply the special-purpose SNARK, the prover's costs increase by $\geq 10\times$ compared to a prover that uses the layered sum-check approach.

To address this problem, we observe that in Spartan, grand products are computed over vectors of size $\approx 16n$, where n is the size of the R1CS instance. Furthermore, we devise a hybrid scheme where we use a constant-depth layered circuit (in conjunction with a layered sum-check) to reduce the grand product instance size to $\approx n$ (instead of $16n$) i.e., we apply a depth-4 layered sum-check before employing the special-purpose SNARK for grand product relations. The result is that the prover's costs increase by $\approx 20\%$, which is reasonable, while providing asymptotic and concrete proof size improvements.

Figure 5 depicts the asymptotic improvements of Sparkle-derived sparse polynomial commitment schemes compared to SPARK-derived schemes.

7 Accelerating the encoder with an untrusted assistant

Prior work [26, 32, 58] employs a preprocessing phase where the verifier creates a commitment to the structure of an R1CS instance. For example, in Spartan, given the structure of an R1CS instance, $(\mathbb{F}, A, B, C, m, n)$, and some public parameters pp, the verifier creates commitments to three sparse multilinear polynomials: $\widetilde{A}, \widetilde{B}, \widetilde{C}$. Using SPARK (or *Sparkle*), this requires $O(1)$ commitments to dense multilinear polynomials in $O(\log n)$ variables. Since $\mathcal{V}$ relies on the correctness of the commitments, in prior work, $\mathcal{V}$ computes them directly. For example, in Spartan, $\mathcal{V}$ incurs $O(n)$ group exponentiations. A similar cost is incurred under both SuperSonic and Fractal to create such commitments. The linear cost is unavoidable, but we introduce a mechanism that enables the verifier to employ an untrusted *assistant*, which can be run by anyone including the prover. In the context of Spartan, this reduces reduces the cost of creating a computation commitment to be $O(n)$ multiplications over $\mathbb{F}$ ($\mathcal{V}$ also incurs exponentiations that are sub-linear in $O(n)$). The improvement is substantial in practice (§9). This technique is general and applies to other schemes including SuperSonic and Fractal.

Details. Suppose that we have an extractable polynomial commitment scheme for multilinear polynomials PC. $\mathcal{V}$ holds $pp, pp_{\mathbb{F}}$, which are public parameters for PC and a commitment scheme for $\mathbb{F}$. To assist $\mathcal{V}$ in computing a commitment $\mathcal{C}_{\mathcal{G}}$ to a dense multilinear polynomial $\mathcal{G}$, we have an untrusted assistant compute a commitment $\mathcal{C}$ with some opening hint $\mathcal{S}$. The assistant $\mathcal{A}$ and $\mathcal{V}$ then engage in an interactive protocol to convince $\mathcal{V}$ that $\mathcal{C}$ was computed correctly. Given $\mathcal{C}, \mathcal{G}$ an ℓ-variate multilinear polynomial shared between $\mathcal{A}$ and $\mathcal{V}$:

1. $\mathcal{A} \to \mathcal{V}$: $(\mathcal{C}_v\ ;\ \mathcal{S}_v) \leftarrow \mathsf{Commit}_{\mathbb{F}}(pp_{\mathbb{F}}\ ;\ v)$
2. $\mathcal{V} \to \mathcal{A}$: $r \leftarrow_{\$} \mathbb{F}^{\ell}$
3. $\mathcal{A}, \mathcal{V}$: $b_{poly} = \mathsf{PC.Eval}(pp, pp_{\mathbb{F}}, \mathcal{C}, r, \mathcal{C}_v\ ;\ \mathcal{G}, \mathcal{S}, \mathcal{S}_v)$
4. $\mathcal{V}$: $v \leftarrow \mathcal{G}(r)$
5. $\mathcal{A}, \mathcal{V}$: $b_{eval} = \mathsf{Open}_{\mathbb{F}}(pp_{\mathbb{F}}, \mathcal{C}_v, v, \mathcal{S}_v)$
6. $\mathcal{V}$: Output $b = b_{poly} \wedge b_{eval}$

Lemma 7.1. *The above protocol is a public-coin succinct interactive argument of knowledge for the language:* $\{\langle(\mathcal{C}_{\mathcal{G}}, \mathcal{G}), (\mathcal{S}_{\mathcal{G}})\rangle : \mathsf{Open}(pp, \mathcal{C}_{\mathcal{G}}, \mathcal{G}, \mathcal{S}_{\mathcal{G}}) = 1\}$*, assuming the* $|\mathbb{F}|$ *is exponential in the security parameter* λ*.*

Proof. Completeness, succinctness, and public coin follow from the same properties of $\mathsf{PC.Eval}$ and $\mathsf{Open}_{\mathbb{F}}$. Since PC is extractable, there is some multilinear $\mathcal{G}'$ underlying $\mathcal{C}$ such that $\mathcal{C}_v$ is a commitment to $\mathcal{G}'(r)$. Since the commitment to $v \in \mathbb{F}$ is binding, $\mathcal{G}'(r) = v = \mathcal{G}(r)$. So $\mathcal{G}$ and $\mathcal{G}'$ are equal at a randomly chosen r, and so by the Schwartz-Zippel lemma $\mathcal{G} = \mathcal{G}'$, except for a soundness error of $O(\log m/|\mathbb{F}|) \approx \mathtt{negl}(\lambda)$. □

8 A more efficient zero-knowledge transformation

Like Spartan [58], Xiphos, Kopis, and Lakonia require a *zero-knowledge sum-check protocol*: given a commitment $\mathcal{C}_F$ to a ℓ-variate polynomial $F(x)$ of degree d in each variable, and a commitment $\mathcal{C}_y$ to $y \in \mathbb{F}$, we reduce a claim of the form $y = \sum_{x \in \{0,1\}^\ell} F(x)$ to another commitment $C_{y'}$ to $y' \in \mathbb{F}$ and a claim that $y' = F(r)$, where $r \in \mathbb{F}^\ell$.

Recall that the non-hiding sum-check proceeds as follows (§3.8). After i rounds, $\mathcal{P}$ and $\mathcal{V}$ share some $r_1, \ldots, r_i \in \mathbb{F}$, and some target scalar $s \in \mathbb{F}$ which is initialized to y. They then follow the following round of the protocol:

1. $\mathcal{P} \to \mathcal{V}$: $f_i(X) = \sum_{x \in \{0,1\}^{\ell-i-1}} F(r_1, \ldots, r_i, X, x)$
2. $\mathcal{V}$: Check that $f_i(0) + f_i(1) = s$
3. $\mathcal{V} \to \mathcal{P}$: $r_{i+1} \leftarrow_\$ \mathbb{F}$.
4. $\mathcal{P}, \mathcal{V}$: $s = f_i(r_{i+1})$

y' is the value of s at the last round. Additionally, $\mathcal{P}$ must prove to $\mathcal{V}$ that $F(r) = s$, which is performed with an auxiliary protocol (e.g., polynomial commitments).

In Spartan, the sum-check protocol is made zero-knowledge with techniques from Hyrax [65]. The core observation is that $\mathcal{V}$ only computes linear functions of the polynomials that $\mathcal{P}$ sends. So $\mathcal{P}$ can send linearly homomorphic commitments to the evaluations (or coefficients) of these polynomials, and $\mathcal{V}$ can manipulate the commitments to obtain $C_{y'}$ with some known $\mathcal{S}_{y'}$. Unfortunately, this requires that $\mathcal{P}$ send ℓ commitments to vectors of $O(d)$ scalars, and later prove knowledge of their openings. For k sum-checks this contributes $O(k\ell + d)$ group elements to the proof and exponentiations to verification, which is concretely expensive.

We take a different approach, conceptually closer to the zero-knowledge sum-check of from Chiesa et al. [30] and follow-up adaptations [68, 69]. This allows us to replace these $O(k\ell + d)$ costs with an $O(kd + \ell)$ costs, which is concretely smaller. The idea in this case is that $\mathcal{P}$ will choose a suitably random polynomial G and send an extractable commitment $\mathcal{C}_G$ to it to the verifier, along with a claimed value in $z \in \mathbb{F}$ for $\sum_{x \in \{0,1\}^\ell} F(x) + G(x)$. A non-hiding sum-check will then be performed on $F + G$ to obtain a claim $z' = F(r) + G(r)$; analysis of the randomness of G will show that the transcript of this sum-check is independent of F. $\mathcal{P}$ and $\mathcal{V}$ will then use Eval and a standard sigma protocol to prove the consistency of commitments $C_y, C_{y'}$, commitments to evaluations of G, and z, z'.

Definition 8.1. We call a multilinear polynomial in ℓ variables of form:

$$g(X) = b_0 \prod_{i=1}^{\ell} (1 - X_i) + \sum_{i=1}^{\ell} b_i (2X_i - 1) \prod_{j=1, j \neq i}^{\ell} (1 - X_j)$$

a *low-weight* polynomial.

Lemma 8.1. *Low-weight polynomials are exactly ℓ-variable multilinear polynomials whose support on $\{0,1\}^\ell$ is contained in $\{(0, \ldots, 0), e_1, \ldots, e_\ell\}$*

Proof. For g a low-weight polynomial as above, $g(0,\ldots,0) = b_0 - \sum_{i=1}^{\ell} b_i$ and for all $i \in \{1,\ldots,\ell\}$ and $g(e_i) = b_i$; for all other points on the Boolean hypercube at least 2 of the X_i are 1 and so every term vanishes. Conversely, let f be a multilinear polynomial whose support on the Boolean hypercube is contained in $\{(0,\ldots,0), e_1, \ldots, e_\ell\}$. Then let $b_i = f(e_i)$ and $b_0 = f(0,\ldots,0) + \sum_i f(e_i)$, and define g as above. Then g and f are now two multilinear polynomials that agree on $\{0,1\}^\ell$ and so $g = f$. □

Lemma 8.2. *For a low-weight polynomial g, the polynomial in the first variable obtained by summing over the hypercube: $\sum_{x\in\{0,1\}^{\ell-1}} g(X,x) = (b_0 - b_1) + (2b_1 - b_0)X$, which is independent of $b_2,\ldots,b_i$.*

Lemma 8.3. *When a variable of a low-weight polynomial is bound, the resulting polynomial is still low-weight:*

$$g(r,X) = [b_0(1-r) + (2r-1)b_1] \prod_{i=1}^{\ell-1}(1-X_i) + \sum_{i=1}^{\ell-1} [b_{i+1}(1-r)]\,(2X_i - 1) \prod_{j=1, j\neq i}^{\ell-1} (1-X_j)$$

Lemma 8.4. *Let g be a uniformly random low-weight polynomial, and for some $r \in \mathbb{F}^\ell$ define for $i \in 0\ldots\ell-1$:*

$$g_i(X) = \sum_{x\in\{0,1\}^{\ell-i-1}} g(r_1,\ldots,r_i,X,x).$$

Then if $\forall i : r_i \neq 1$, the g_i are a sequence of independent, uniformly random linear polynomials, subject to the constraint $\forall i > 0 : g_{i-1}(r_i) = g_i(0) + g_i(1)$.

Proof. That $g_{i-1}(r_i) = g_i(0) + g_i(1)$ is clear from the definition of the g_i.

Since g is uniformly random, we have $b \leftarrow_\$ \mathbb{F}^{\ell+1}$. So g_0 is uniformly random as b_0, b_1 are uniformly random and independent. Note that $g_0,\ldots,g_{i-1}$ are independent of b_{i+1}, whilst g_i has a contribution $b_{i+1}\prod_{j\leq i}(1-r_j)(2X-1)$. Since $g_i(0) + g_i(1)$ is fixed, g_i has one degree of freedom, and so if $\forall j \leq i : r_j \neq 1$ we have g_i uniformly random and independent of b_j for $j \leq i$. □

In particular, the Prover samples d random low-weight polynomials $g^1,\ldots,g^d$ uniformly at random, and writing $X^j = (X_1^j,\ldots,X_\ell^j)$, sets:

$$G(X) = \sum_{i\in 1\ldots d} g^j(X^j).$$

The commitment to G will be a vector of hiding, blinding commitments to the multilinear polynomials g^i.

Corollary 8.1. *For $\mathrm{ord}_{\mathbb{F}}(r_i) > d$, and g^j sampled uniformly at random, the polynomials $G_i(X) = \sum_{x\in\{0,1\}^{\ell-i-1}} G(r[1\ldots i], X, x)$ are independent, uniformly random polynomials of degree d subject to the condition that: $\forall i > 0 : G_{i-1}(r_i) = G_i(0) + G_i(1)$.*

Proof. Since $\mathrm{ord}_{\mathbb{F}}(r_i) > d$, we have $r_i^j \neq 1$ for any $j \leq d$. So by the previous lemma, $g^j(X^j)$ contributes an independent, uniformly random linear combination of in $1, X^j$ to G_i, subject to the constraint that $g_{i-1}^j(r_i^j) = g_i(0) + g_i(1)$. Since the g^j are independent, G_i has independent, uniformly random coefficients in X^j for all $j > 0$ and satisfies $G_{i-1}(r_i) = G_i(0) + G_i(1)$. So the G_i are independent and uniformly random polynomials of degree d subject to this condition. □

It remains to relate the claims that $z = \sum_{x \in \{0,1\}} F(x) + G(x)$, $z' = F(r) + G(r)$ to commitments $\mathcal{C}_y, \mathcal{C}_{y'}, \mathcal{C}_F, \mathcal{C}_G$. In this protocol, we make use of 2^{-1} and assume that $\mathbb{F}$ is not of characteristic 2.

Recall that the commitments to elements of $\mathbb{F}$ are Pedersen commitments with generators $pp_{\mathbb{F}} = (P_G, P_H)$, i.e. that $\mathsf{Commit}_{\mathbb{F}}(x) = (pp_{\mathbb{F}}\ ; xP_G + rP_H\ ; r)$ for $r \leftarrow_{\$} \mathbb{F}$.

ZK-sumcheck-reduce(C_y)

$\mathcal{P}$ **witness:** $y = \sum_{x \in \{0,1\}^\ell} F(x)$, opening hint for C_y.

$\mathcal{P}$**:** $\forall i \in 1 \ldots \ell : g^i \leftarrow_{\$} \{$ *low-weight* polynomials in ℓ variables$\}$

$$z \leftarrow y + \sum_i \sum_{x \in \{0,1\}^\ell} g^i(x)$$

$\mathcal{P}$**:** $(\mathcal{C}_{g^i}; \mathcal{S}_{g^i}) \leftarrow \mathsf{Commit}(pp; g^i)$ for $i \in 1, \ldots, \ell$

$\mathcal{P} \to \mathcal{V}$**:** $\{C_{g^i} : i \in [1, \ldots, \ell]\}, z$

$\mathcal{P}, \mathcal{V}$**:** $(r, z') \leftarrow$ Sumcheck-reduce(z).

$\mathcal{P}$**:** $(\mathcal{C}_{h^i}; \mathcal{S}_{h^i}) \leftarrow \mathsf{Commit}_{\mathbb{F}}(pp_{\mathbb{F}}; g^i(r^i))$ for $i \in 1, \ldots, \ell$

$\mathcal{P} \to \mathcal{V}$**:** $\{C_{h^i} : i \in [1, \ldots, \ell]\}$

$\mathcal{P}, \mathcal{V}$**:** $C_{\sum g^i} = \sum_i C_{g_i}$,

$C_{\mathbb{E}(\sum g^i)} = 2^{-\ell}(zP_G - C_y)$,

$\mathsf{Assert}(\mathsf{Eval}(pp, pp_{\mathbb{F}}; C_{\sum g^i}, C_{\mathbb{E}(\sum g^i)}), (2^{-1}, \ldots, 2^{-1}))$

$\forall i : \mathsf{Assert}(\mathsf{Eval}(pp, pp_{\mathbb{F}}; C_{g^i}, C_{h^i}, r^i))$

$\mathcal{P}, \mathcal{V}$**:** $C_{y'} \leftarrow z'P_G - \sum_i C_{h^i}$

$\mathcal{V}$**:** Return $(r, C_{y'})$

Theorem 8.1. *The above protocol is complete, computationally sound, and zero-knowledge with respect to F, y.*

Proof. Completeness is immediate; $\mathcal{P}$ uses $F(X) + \sum g^i(X^i)$ in the Sumcheck-reduce, and can open $C_{y'}$ to $y' = z' - \sum_i h^i = F(r)$.

We will show computational soundness assuming that Eval and the Pedersen commitments to elements of $\mathbb{F}$ are sound. From the soundness of Eval, the second set of

checks imply that C_{h^i} are commitments to evaluations of g^i at r^i. Since Pedersen commitments are linearly homomorphic, their sum is a commitment to $G(r)$. So if $C_{y'}$ is a commitment to $F(r)$ then z' must equal $F(r) + G(r)$. Sumcheck-reduce ensures that if $z' = F(r) + G(r)$ with non-negligible probability and F, G are of low degree, then $z = \sum_{x \in \{0,1\}} F(x) + G(x)$ Note that for any multilinear polynomial p in ℓ variables and $i > 0$:

$$\sum_{x \in \{0,1\}^\ell} p(x^i) = \sum_{x \in \{0,1\}^\ell} p(x) = 2^\ell p(2^{-1}, \ldots, 2^{-1})).$$

So since Eval is sound, the first check on $C_{\sum g^i}$ proves that $C_{\mathbb{E}(\sum g^i)}$ is a commitment to $\sum_{x \in \{0,1\}} G(x)$. Then since Pedersen commitments are linearly homomorphic, C_y must be a commitment to $\sum_{x \in \{0,1\}} F(x)$.

So this protocol reduces a claim that C_y is a commitment to the sum of F on the cube to a claim that $C_{y'}$ is a commitment to $F(r)$.

To see zero-knowledge with respect to F, y, we will show that $\mathcal{P}$'s messages are independent of F and y. Initially, Prover sends

$$z = y + \sum_{i=1 \ldots \ell} \sum_{x \in \{0,1\}^\ell} g^i(x),$$

which is plainly independent of F. The remaining messages from $\mathcal{P}$ to $\mathcal{V}$ outside of the interior, non-hiding sumcheck are all independent hiding commitments. So it suffices to show that the $\mathcal{P} \to \mathcal{V}$ messages in a non-hiding sumcheck on $F + G$ are independent of F given the randomness of G and conditional on z.

The remaining messages in the sum-check are a series of $\ell - 1$ degree-d polynomials $p_i(X)$ such that $p_0(0) + p_0(1) = z$ and for all $i > 0$, $p_i(0) + p_i(1) = p_{i-1}(r_i)$ For a prover following the protocol, we have $p_i(X) = F_i(X) + G_i$, where by Corollary 8.1 the G_i are uniformly random and independent, subject to $G_{i-1}(r_i) = G_i(0) + G_i(1)$. Since the p_i must obey this constraint, they are independent of F. □

Implementation Any extractable polynomial commitment scheme can be used as a black box for the low-weight polynomials. However, as each is a linear function of the $\ell + 1$ values b_i, it is concretely efficient to commit to them with Pedersen commitments to their vectors b. These commitments have the necessary linearity properties, and Eval is implemented with a linear-time (i.e. $O(\ell)$) naive inner-product proof.

9 Experimental evaluation

This section experimentally evaluates our implementations of Kopis, Xiphos, and Lakonia, and compares them with a set of baselines.

Metrics and methodology. Our evaluation metrics are: (1) the prover's costs to produce a proof; (2) the verifier's costs to preprocess the structure of an R1CS instance; (3) the verifier's costs to verify a proof; and (4) the size of a proof. We measure CPU costs using a real-time clock; we measure proof sizes by serializing proof data structures to byte strings. For our schemes, we employ `cargo bench` to measure performance, and for baselines, we use the profilers provided with their open source code.

We run our experiments on an Azure Standard F16s_v2 virtual machine (16 vCPUs, 32 GB memory) with Ubuntu 20.10. We report results from a single-threaded configuration since not all our baselines leverage multiple cores. As with prior work [17, 32, 58], we vary the size of the R1CS instance by varying the number of constraints and variables m and maintain the ratio n/m to approximately 1.

Baselines. For Kopis and Xiphos, the baselines are: (1) Spartan [58], (2) Fractal [32], and (3) SuperSonic [26]. For Spartan, we use its open-source implementation [5]; we also report its performance with our optimizations such as batched polynomial evaluations, which we refer to as Spartan++. For Fractal, we use its open-source implementations from `libiop` [52], configured to provide provable security.

Finally, since there does not exist a prior implementation of SuperSonic, we estimate its performance using the authors' cost models and microbenchmarks. We microbenchmark the cost of an exponentiation in a class group with random 128-bit size exponents using the ANTIC library [1], which offers a fast class group implementation. We find that each class group exponentiation costs $\approx$38 ms. In our estimates of SuperSonic, we ignore the costs of scalar arithmetic (in their information-theoretic proof system) and count only the costs incurred by their polynomial commitment scheme (this is optimistic for SuperSonic and pessimistic to our schemes). Furthermore, our estimates assume standard optimizations such as the Pippenger's algorithm for multiexponentiation.

For Lakonia, the baselines are: (1) Ligero [8], (2) Hyrax [66], and (3) Aurora [17]. For Ligero and Aurora, we use their open-source implementations from `libiop` [52], configured to provide provable security, and for Hyrax, we use its reference implementation [51].[6] Additional baselines for Lakonia include STARK [13] and Bulletproofs [28]. Given the lack of a standard implementation, we report their performance from prior measurements in Figure 3.

9.1 Performance results of Kopis and Xiphos

Prover. Figure 6 depicts the prover's costs under Kopis, Xiphos, and their baselines. At 2^{20} constraints, Xiphos and Kopis are $\approx 5\times$ more expensive than Spartan, which features the fastest prover in the literature. Most of this slowdown can be attributed to the difference in speed between the cost of an exponentiation on `ristretto2555` (used by Spartan) and on $\mathbb{G}_1$ of `bls12-381` (used by our schemes); this is $\approx 5\times$. Furthermore, Spartan's underlying library for curve arithmetic [3] features an advanced implementation that leverages `avx2` instructions to achieve up to $2\times$ higher speed. With a faster implementation of curve arithmetic on `bls12-381`, we believe this gap can be reduced substantially. Compared to SuperSonic (which offers the shortest proofs in the literature), Kopis and Xiphos are up to $250\times$ faster. Finally, compared to Fractal, Kopis and Xiphos are $\approx 3\times$ faster at 2^{18} constraints (we could not run Fractal beyond 2^{18} constraints as it runs out of memory).

Proof sizes. Figure 7 depicts the proof sizes under Kopis, Xiphos, and their baselines. It is easy to see that Xiphos offers proof sizes competitive with SuperSonic.[7] Furthermore,

[6] To compare Lakonia with Hyrax, as before [58], we transform R1CS instances to arithmetic circuits where the circuit evaluates constraints in the R1CS instance, and outputs a vector of zeros if the constraints are satisfied. For an arbitrary R1CS instance, the circuit has no structure, so Hyrax incurs linear verification costs.

[7] Xiphos's and Kopis's proof sizes are missing an optimization that reduces proof sizes by an additional 15%.

	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}	2^{18}	2^{19}	2^{20}
SuperSonic	86	163	311	599	1160	2240	4360	8500	16600	32500	63800
Fractal	0.8	1.5	2.9	5.9	12	25	51	104	216	–	–
Spartan	0.1	0.2	0.3	0.6	1	1.9	3.5	6.8	12	24	47
Spartan++	0.08	0.15	0.26	0.5	0.9	1.8	3.3	6.5	12	24	45
Kopis	1.2	1.6	2.7	4.2	7.3	12	21	36	69	123	245
Xiphos	1.5	2.5	3.2	5.7	8.2	14.5	23	43	74	132	249

FIGURE 6—Prover's performance (in seconds) for varying R1CS instance sizes under different schemes. Fractal's prover runs out of memory at 2^{18} constraints and beyond.

Kopis offers the shortest proofs, both concretely and asymptotically. Proof sizes under our schemes are orders of magnitude shorter than those produced by Fractal. Although Spartan produces proofs shorter than Xiphos at small instance sizes, Xiphos's superior asymptotics are visible around 2^{13} constraints. Finally, Spartan++ features modest improvements in proof sizes over Spartan.

	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}	2^{18}	2^{19}	2^{20}
SuperSonic	31	33	34	36	38	40	41	43	45	47	49
Fractal	1M	1.2M	1.4M	1.5M	1.7M	1.8M	2M	2.1M	2.3M	–	–
Spartan	32	37	41.7	48	54	63	72	85	98	120	142
Spartan++	27	31	36	41	47	55	64	76	89	110	131
Kopis	25	26	27	29	30	32	33	34	36	37	39
Xiphos	40	44	45	48	49	51	53	55	57	59	61

FIGURE 7—Proof sizes in KBs. Entries with "M" suffix are in MBs.

Verifier. Figure 8 depicts verifier's costs to verify a proof under Kopis, Xiphos, and their baselines. As we can see, Xiphos offers a verifier that is faster than SuperSonic—despite sharing the same asymptotics. Xiphos overtakes Spartan at roughly 2^{18} constraints despite Spartan using an advanced implementation of curve arithmetic—because of Xiphos's better asymptotics. Kopis is slower than Xiphos and Spartan, but is concretely faster than SuperSonic at all instance sizes we measured. Finally, Spartan++ features modest improvements in verification times over Spartan.

	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}	2^{18}	2^{19}	2^{20}
SuperSonic	1.4s	1.5s	1.6s	1.7s	1.9s	2s	2.1s	2.2s	2.3s	2.5s	2.6s
Fractal	148	120	163	168	141	184	188	165	205	–	–
Spartan	14	17	20	24	29	36	47	58	77	99	135
Spartan++	8	9	11	14	18	22	30	38	53	68	97
Kopis	71	79	98	108	135	156	203	238	318	384	535
Xiphos	64	66	67	70	70	74	74	77	78	80	80

FIGURE 8—Verifier's performance (in ms) under different schemes. Entries with "s" are in seconds.

Verifier's preprocessing (encoder). Figure 9 depicts the verifier's preprocessing costs to create a computation commitment to the structure of an R1CS instance. For Kopis, Xiphos, and Spartan++, we depict the costs of an untrusted assistant in addition to

reporting the cost of an encoder. It is easy to see that the use of an untrusted assistant improves preprocessing costs substantially under Xiphos, Kopis, and Spartan++, with speedups of 10–10,000× depending on the baseline. Furthermore, the assistant under Xiphos (and Kopis) is substantially cheaper than the encoders of SuperSonic and Fractal, and is ≈3× of the encoder under Spartan.

	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}	2^{18}	2^{19}	2^{20}
SuperSonic	35	64	117	216	400	747	1.4k	2.6k	4.9k	9.4k	17.9k
Fractal	0.3	0.6	1.2	2.5	5.4	11.5	24	51	107	227	–
Spartan	0.06	0.1	0.2	0.3	0.6	1.1	2.2	3.3	6.5	9.9	20
Spartan++ (A)	0.06	0.1	0.2	0.3	0.6	1.1	2.4	3.7	7.4	12	24
Spartan++ (E)	0.005	0.007	0.01	0.016	0.03	0.05	0.13	0.23	0.44	0.8	1.6
Kopis (A)	0.6	0.8	1.3	1.6	2.9	3.8	6.9	9.9	18.4	28	55
Kopis (E)	0.04	0.05	0.07	0.09	0.12	0.17	0.3	0.4	0.7	1.2	2.2
Xiphos (A)	1	2	2	3	4	7	9	16	22	40	62
Xiphos (E)	0.04	0.04	0.05	0.05	0.06	0.09	0.17	0.28	0.5	0.9	1.8

FIGURE 9—Encoder's performance (in seconds) for varying R1CS instance sizes under different schemes. Entries with suffix "k" are in thousands. For Kopis, Xiphos, and Spartan++, we depict two rows each. Rows with "A" denote the cost of the untrusted assistant and rows with "E" denote the cost of the encoder with advice from an untrusted assistant.

9.2 Performance of Lakonia

Lakonia and its baselines do not require the verifier to incur any preprocessing costs, so we focus on reporting the prover's costs, the verifier's costs, and proof sizes.

Prover. Figure 10 depicts the performance of the prover under Lakonia and its baselines. Lakonia is faster than all its baselines except the NIZK variant of Spartan (for 2^{13} constraints and beyond). The slowdown relative to Spartan is analogous to slowdown of Kopis and Xiphos relative to Spartan. Nevertheless, at 2^{20} constraints, Lakonia is ≈2.3× faster than Ligero, and ≈16× faster than Aurora and Hyrax.

	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}	2^{18}	2^{19}	2^{20}
Ligero	0.1	0.2	0.4	0.8	1.6	2	4	8	17	35	69
Hyrax	1	1.7	2.8	5	9	18	36	61	117	244	486
Aurora	0.5	0.8	1.6	3.2	6.5	13.3	27	56	116	236	485
Spartan	0.02	0.03	0.05	0.09	0.16	0.27	0.6	0.9	1.7	3	6
Spartan++	0.01	0.02	0.04	0.07	0.14	0.25	0.5	0.8	1.7	3	6
Lakonia	0.2	0.3	0.5	0.6	1.1	1.6	3	5	9	15	29

FIGURE 10—Prover's performance (in seconds) for varying R1CS instance sizes under different schemes.

Proof sizes. Figure 11 depicts proof sizes under Lakonia and its baselines. Bulletproofs (not depicted) offers the shortest proof sizes: ≈1.5 KB for 2^{20} constraints. As reported earlier (Figure 3), Bulletproofs incurs orders of magnitude higher proving and verification costs than Lakonia. Besides Bulletproofs, Lakonia offers the shortest proof sizes, which are substantially shorter than most baseline proof systems. Thus, we believe Lakonia offers a new point in the design space of concretely-efficient proof systems.

	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}	2^{18}	2^{19}	2^{20}
Ligero	546	628	1M	1.2M	2M	3M	5M	5M	10M	10M	20M
Hyrax	14	16	17	20	21	26	28	37	38	56	58
Aurora	447	510	610	717	810	931	1M	1.1M	1.3M	1.5M	1.6M
Spartan	9	10	12	13	15	16	21	22	30	31	48
Spartan++	6	6	7	8	10	10	15	15	23	24	40
Lakonia	8	8	9	9	10	10	11	11	12	12	12.6

FIGURE 11—Proof sizes in KBs for Lakonia and its baselines. Entries with "M" are in megabytes.

Verifier. Figure 12 depicts the costs of the verifier under Lakonia and its baselines. Despite sharing the same asymptotics, Lakonia's verifier is orders of magnitude faster than all its baselines. The only exception is Spartan where, at 2^{20} constraints, Lakonia is $\approx$50% slower than Spartan.

	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}	2^{18}	2^{19}	2^{20}
Ligero	49	96	172	357	680	976	1.9s	3.7s	7.3s	15s	31s
Hyrax	195	229	262	317	388	502	510	1.2s	1.9s	3.5s	7.7s
Aurora	186	316	574	933	1.8s	3.5s	6.7s	13s	27s	54s	108s
Spartan	7	8	10	12	16	22	33	55	98	194	369
Spartan++	3	4	5	6	9	15	25	44	87	166	347
Lakonia	29	32	38	44	56	70	100	133	203	320	555

FIGURE 12—Verifier's performance (in ms) under different schemes. Entries with "s" are in seconds.

References

[1] Antic – algebraic number theory in c. https://github.com/wbhart/antic.

[2] Ethereum Roadmap. ZK-Rollups. https://docs.ethhub.io/ethereum-roadmap/layer-2-scaling/zk-rollups/.

[3] A pure-Rust implementation of group operations on Ristretto and Curve25519. https://github.com/dalek-cryptography/curve25519-dalek.

[4] The Ristretto group. https://ristretto.group/.

[5] Spartan: High-speed zkSNARKs without trusted setup. https://github.com/Microsoft/Spartan.

[6] M. Abe, G. Fuchsbauer, J. Groth, K. Haralambiev, and M. Ohkubo. Structure-preserving signatures and commitments to group elements. In *CRYPTO*, pages 209–236, 2010.

[7] M. Abe, J. Groth, M. Kohlweiss, M. Ohkubo, and M. Tibouchi. Efficient fully structure-preserving signatures and shrinking commitments. *Journal of Cryptology*, 32(3):973–1025, July 2019.

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

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

[10] S. Arora and S. Safra. Probabilistic checking of proofs: A new characterization of NP. *J. ACM*, 45(1):70–122, Jan. 1998.

[11] L. Babai, L. Fortnow, L. A. Levin, and M. Szegedy. Checking computations in polylogarithmic time. In *STOC*, 1991.

[12] N. Barić and B. Pfitzmann. Collision-free accumulators and fail-stop signature schemes without trees. In *EUROCRYPT*, pages 480–494, 1997.

[13] E. Ben-Sasson, I. Bentov, Y. Horesh, and M. Riabzev. Scalable, transparent, and post-quantum secure computational integrity. ePrint Report 2018/046, 2018.

[14] E. Ben-Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza. Zerocash: Decentralized anonymous payments from Bitcoin. In *S&P*, 2014.

[15] E. Ben-Sasson, A. Chiesa, D. Genkin, and E. Tromer. On the concrete efficiency of probabilistically-checkable proofs. In *STOC*, pages 585–594, 2013.

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

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

[18] E. Ben-Sasson, A. Chiesa, E. Tromer, and M. Virza. Succinct non-interactive zero knowledge for a von Neumann architecture. In *USENIX Security*, 2014.

[19] E. Ben-Sasson, O. Goldreich, P. Harsha, M. Sudan, and S. Vadhan. Short PCPs verifiable in polylogarithmic time. In *Computational Complexity*, 2005.

[20] E. Ben-Sasson and M. Sudan. Short PCPs with polylog query complexity. *SIAM J. Comput.*, 38(2):551–607, May 2008.

[21] N. Bitansky, A. Chiesa, Y. Ishai, O. Paneth, and R. Ostrovsky. Succinct non-interactive arguments via linear interactive proofs. In *TCC*, 2013.

[22] A. J. Blumberg, J. Thaler, V. Vu, and M. Walfish. Verifiable computation using multiple provers. ePrint Report 2014/846, 2014.

[23] D. Boneh, B. Bünz, and B. Fisch. A survey of two verifiable delay functions. Cryptology ePrint Archive, Report 2018/712, 2018.

[24] S. Bowe, A. Chiesa, M. Green, I. Miers, P. Mishra, and H. Wu. Zexe: Enabling decentralized private computation. ePrint Report 2018/962, 2018.

[25] B. Braun, A. J. Feldman, Z. Ren, S. Setty, A. J. Blumberg, and M. Walfish. Verifying

computations with state. In *SOSP*, 2013.
[26] B. Bunz, B. Fisch, and A. Szepieniec. Transparent SNARKs from DARK compilers. ePrint Report 2019/1229, 2019.
[27] B. Bunz, M. Maller, P. Mishra, and N. Vesely. Proofs for inner pairing products and applications. Cryptology ePrint Archive, Report 2019/1177, 2019.
[28] B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *S&P*, 2018.
[29] M. Campanelli, D. Fiore, and A. Querol. LegoSNARK: modular design and composition of succinct zero-knowledge proofs. ePrint Report 2019/142, 2019.
[30] A. Chiesa, M. A. Forbes, and N. Spooner. A zero knowledge sumcheck and its applications. *CoRR*, abs/1704.02086, 2017.
[31] A. Chiesa, Y. Hu, M. Maller, P. Mishra, N. Vesely, and N. Ward. Marlin: Preprocessing zkSNARKs with universal and updatable SRS. ePrint Report 2019/1047, 2019.
[32] A. Chiesa, D. Ojha, and N. Spooner. Fractal: Post-quantum and transparent recursive proofs from holography. ePrint Report 2019/1076, 2019.
[33] G. Cormode, M. Mitzenmacher, and J. Thaler. Practical verified computation with streaming interactive proofs. In *ITCS*, 2012.
[34] A. Delignat-Lavaud, C. Fournet, M. Kohlweiss, and B. Parno. Cinderella: Turning shabby X.509 certificates into elegant anonymous credentials with the magic of verifiable computation. In *S&P*, 2016.
[35] S. Dobson, S. D. Galbraith, and B. Smith. Trustless construction of groups of unknown order with hyperelliptic curves. `https://www.math.auckland.ac.nz/~sgal018/ANTS/posters/Dobson-Galbraith-Smith.pdf`, 2020.
[36] S. Dobson, S. D. Galbraith, and B. Smith. Trustless groups of unknown order with hyperelliptic curves. Cryptology ePrint Archive, Report 2020/196, 2020.
[37] A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *CRYPTO*, pages 186–194, 1986.
[38] E. Fujisaki and T. Okamoto. Statistical zero knowledge protocols to prove modular polynomial relations. In *CRYPTO*, pages 16–30, 1997.
[39] R. Gennaro, C. Gentry, B. Parno, and M. Raykova. Quadratic span programs and succinct NIZKs without PCPs. In *EUROCRYPT*, 2013.
[40] C. Gentry and D. Wichs. Separating succinct non-interactive arguments from all falsifiable assumptions. In *STOC*, pages 99–108, 2011.
[41] S. Goldwasser, Y. T. Kalai, and G. N. Rothblum. Delegating computation: Interactive proofs for muggles. In *STOC*, 2008.
[42] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof-systems. In *STOC*, 1985.
[43] J. Groth. On the size of pairing-based non-interactive arguments. In *EUROCRYPT*, 2016.
[44] J. Groth and Y. Ishai. Sub-linear zero-knowledge argument for correctness of a shuffle. In *EUROCRYPT*, 2008.
[45] M. Hamburg. Decaf: Eliminating cofactors through point compression. In *CRYPTO*, 2015.
[46] A. Kate, G. M. Zaverucha, and I. Goldberg. Constant-size commitments to polynomials and their applications. In *ASIACRYPT*, pages 177–194, 2010.
[47] J. Kilian. A note on efficient zero-knowledge proofs and arguments (extended abstract). In *STOC*, 1992.
[48] A. Kosba, A. Miller, E. Shi, Z. Wen, and C. Papamanthou. Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. In *S&P*, 2016.
[49] J. Lee. Dory: Efficient, transparent arguments for generalised inner products and polynomial commitments. Cryptology ePrint Archive, Report 2020/xxx, 2020.

[50] J. Lee, K. Nikitin, and S. Setty. Replicated state machines without replicated execution. In *S&P*, 2020.
[51] libfennel. Hyrax reference implementation. `https://github.com/hyraxZK/fennel`.
[52] libiop. A C++ library for IOP-based zkSNARK. `https://github.com/scipr-lab/libiop`.
[53] libsnark. A C++ library for zkSNARK proofs. `https://github.com/scipr-lab/libsnark`.
[54] C. Lund, L. Fortnow, H. Karloff, and N. Nisan. Algebraic methods for interactive proof systems. In *FOCS*, Oct. 1990.
[55] S. Micali. CS proofs. In *FOCS*, 1994.
[56] A. Ozdemir, R. S. Wahby, and D. Boneh. Scaling verifiable computation using efficient set accumulators. Cryptology ePrint Archive, Report 2019/1494, 2019.
[57] B. Parno, C. Gentry, J. Howell, and M. Raykova. Pinocchio: Nearly practical verifiable computation. In *S&P*, May 2013.
[58] S. Setty. Spartan: Efficient and general-purpose zkSNARKs without trusted setup. ePrint Report 2019/550, 2019.
[59] S. Setty, S. Angel, T. Gupta, and J. Lee. Proving the correct execution of concurrent services in zero-knowledge. In *OSDI*, Oct. 2018.
[60] S. Setty, S. Angel, and J. Lee. Verifiable state machines: Proofs that untrusted services operate correctly. *ACM SIGOPS Operating Systems Review*, 54(1):40–46, Aug. 2020.
[61] S. Setty, B. Braun, V. Vu, A. J. Blumberg, B. Parno, and M. Walfish. Resolving the conflict between generality and plausibility in verified computation. In *EuroSys*, Apr. 2013.
[62] S. Setty, V. Vu, N. Panpalia, B. Braun, A. J. Blumberg, and M. Walfish. Taking proof-based verified computation a few steps closer to practicality. In *USENIX Security*, Aug. 2012.
[63] J. Thaler. Time-optimal interactive proofs for circuit evaluation. In *CRYPTO*, 2013.
[64] V. Vu, S. Setty, A. J. Blumberg, and M. Walfish. A hybrid architecture for verifiable computation. In *S&P*, 2013.
[65] R. S. Wahby, S. Setty, Z. Ren, A. J. Blumberg, and M. Walfish. Efficient RAM and control flow in verifiable outsourced computation. In *NDSS*, 2015.
[66] R. S. Wahby, I. Tzialla, A. Shelat, J. Thaler, and M. Walfish. Doubly-efficient zkSNARKs without trusted setup. In *S&P*, 2018.
[67] B. Wesolowski. Efficient verifiable delay functions. In *EUROCRYPT*, pages 379–407, 2019.
[68] T. Xie, J. Zhang, Y. Zhang, C. Papamanthou, and D. Song. Libra: Succinct zero-knowledge proofs with optimal prover computation. ePrint Report 2019/317, 2019.
[69] J. Zhang, T. Xie, Y. Zhang, and D. Song. Transparent polynomial delegation and its applications to zero knowledge proof. In *S&P*, 2020.
[70] Y. Zhang, D. Genkin, J. Katz, D. Papadopoulos, and C. Papamanthou. vSQL: Verifying arbitrary SQL queries over dynamic outsourced databases. In *S&P*, 2017.