Spartan: Efficient and general-purpose zkSNARKs without trusted setup

Srinath Setty
Microsoft Research

Abstract

This paper introduces Spartan, a new family of zero-knowledge succinct non-interactive arguments of knowledge (zkSNARKs) for the rank-1 constraint satisfiability (R1CS), an NP-complete language that generalizes arithmetic circuit satisfiability. A distinctive feature of Spartan is that it offers the first zkSNARKs without trusted setup (i.e., transparent zkSNARKs) for NP where verifying a proof incurs sub-linear costs—without requiring uniformity in the NP statement's structure. Furthermore, Spartan offers zkSNARKs with a time-optimal prover, a property that has remained elusive for nearly all zkSNARKs in the literature.

To achieve these results, we introduce new techniques that we compose with the sum-check protocol, a seminal interactive proof protocol: (1) *computation commitments*, a primitive to create a succinct commitment to a description of a computation; this technique is crucial for a verifier to achieve sub-linear costs after investing a one-time, public computation to preprocess a given NP statement; (2) SPARK, a cryptographic compiler to transform any existing extractable polynomial commitment scheme for multilinear polynomials to one that efficiently handles *sparse* multilinear polynomials; this technique is critical for achieving a time-optimal prover; and (3) a compact encoding of an R1CS as a low-degree polynomial. The end result is a public-coin succinct interactive argument of knowledge for NP (which can be viewed as *a succinct variant of the sum-check protocol*); we transform it into a zkSNARK using prior techniques. By applying SPARK to different commitment schemes, we obtain four zkSNARKs where the verifier's costs and the proof size range from $O(\log^2 n)$ to $O(\sqrt{n})$ depending on the underlying commitment scheme (n denotes the size of the NP statement). Three of these schemes do not require a trusted setup and one requires a one-time trusted setup that is universal and updateable.

We implement Spartan as a library in about 8,000 lines of Rust. We use the library to build a transparent zkSNARK in the random oracle model where security holds under the discrete logarithm assumption, and compare it with recent zkSNARKs. Among transparent SNARKs, Spartan offers the fastest prover with a speedup of 17.5–115× depending on the baseline, produces proofs that are shorter (except compared to SuperSonic and Bulletproofs) by 2–540×, and incurs verification times that are faster (except compared to Fractal) by 71–240×. When compared to the state-of-the-art zkSNARK with trusted setup, Spartan's prover is 1.5× faster for arbitrary R1CS instances and $>$ 10× faster for data-parallel workloads.

1 Introduction

We revisit the problem of designing zero-knowledge succinct non-interactive arguments of knowledge (zkSNARKs) [27, 63] for a general class of applications (i.e., for the complexity class NP): they enable a computationally-bounded prover to convince the membership of a problem instance in an NP language by producing a proof—*without* revealing anything besides the validity of the statement. Furthermore, the proof size and the verifier's costs are sub-linear in the size of the statement. We are motivated

to design zkSNARKs because there is significant interest in employing them in many applications that involve various forms of delegation of computation for scalability or privacy [16, 33, 35, 37, 39, 44, 48, 50, 51, 53, 58, 78, 90, 93–98, 100, 107, 111].

Specifically, we are interested in zkSNARKs that prove the satisfiability of R1CS instances over a finite field $\mathbb{F}$ (an NP-complete language that generalizes arithmetic circuit satisfiability; see §2.1 for details): given a problem instance $\mathbb{X} = (\mathbb{F}, A, B, C, io, m, n)$, we desire a proof that demonstrates the knowledge of a witness w such that $\mathtt{Sat}_{\mathrm{R1CS}}(\mathbb{X}, w) = 1$.[1] We desire zkSNARKs for R1CS because there exist efficient compiler toolchains to transform high-level applications of interest to R1CS [17, 19, 23, 37, 79, 90, 93, 97, 104].

There are many approaches to construct such arguments in the literature, starting with the work of Kilian [77] who provided the first construction of a succinct interactive argument protocol by employing probabilistically checkable proofs (PCPs) [7, 8, 11, 54, 56, 71] in conjunction with Merkle trees [85]. Micali [86] made a similar protocol non-interactive in the random oracle model, thereby obtaining the first zkSNARK. Unfortunately, the underlying PCP machinery is extremely expensive for the prover and the verifier—despite foundational advances [18, 24–26]. Thus, the first works with an explicit motivation to make proof systems practical [50, 94, 96, 97, 100] refine and implement interactive protocols of Ishai et al. [72] and Goldwasser et al. [64], which do not require asymptotically-efficient PCPs. The principal downside is that they achieve practicality for only a restricted class of NP statements.

In a breakthrough result, Gennaro, Gentry, Parno, and Raykova (GGPR) [61] address the above issue with a new formalism for encoding computations called *quadratic arithmetic programs (QAPs)*. By building on Ishai et al. [72], Groth [66], and Lipmaa [82], GGPR construct a zkSNARK for R1CS in which the prover's running time is $O(n \log n)$, the size of a proof is $O(1)$, and the verifier incurs $O(|io|)$ computation to verify the proof, where n is the size of the statement and io denotes the public input and output. Unfortunately, GGPR's zkSNARK requires a per-statement *trusted* setup that produces a $O(n)$-sized structured common reference string and the trapdoor used in the setup process must be kept secret to ensure soundness. Relying on such a trusted setup is often infeasible in practice, especially for applications that do not have any trusted authorities. There exist several advances atop GGPR, but they retain a trusted setup [19, 23, 29, 67, 69, 90], or require interaction [95].

The above state of affairs has motivated another class of works (called transparent zkSNARKs or zkSNARKs without trusted setup) that aim to eliminate the requirement of a trusted setup—while also providing performance similar to (or even better than) GGPR [61].[2] They prove security in the random oracle model, which is acceptable in practice. There are six works in this class. First, Hyrax [105] extends a line of work [50, 98, 100–103] that refines the doubly-efficient interactive proofs (IPs) of Goldwasser et al. [64]. Second, Ligero [5] builds an interactive PCP [75] using the "MPC in the head" paradigm [73] and then transforms it to a zkSNARK. Third, zkSTARKs [15], Aurora [20], and Fractal [47] build on interactive oracle proofs (IOPs) [21, 91], a generalization of PCPs and IPs. Fourth, Bulletproofs [41] builds on the work of Bootle et al. [34]

[1] Although we use the word "proof", we mean proofs that are computationally sound [36].

[2] The work of Kilian [77] and Micali [86] do not suffer from the trusted setup issue, but as mentioned earlier, they are infeasible to be used in practice.

and leverages a special-purpose argument protocol for inner product operations. Fifth, Virgo [109] builds an IOP-based polynomial commitment scheme for dense polynomials and couples it with a prior system [108] in the line of work that includes Hyrax. Sixth, SuperSonic [38] replaces the polynomial commitment scheme in PLONK [59] (which requires a trusted setup) with one based on groups of unknown order, which can be instantiated without a trusted setup using class groups.

Unfortunately, these works face the following problems. First, the computational model of Hyrax [104] and Virgo [109] is layered arithmetic circuits, and the verifier's costs and the proof size scale linearly in the depth of the circuit; converting an arbitrary circuit into a layered form can increase its size quadratically [64],[3] so Hyrax and Virgo are restricted to low-depth circuits. Second, Hyrax [104] and Virgo [109] achieve sub-linear verification costs only when their layered circuits have uniform structure (e.g., data-parallelism). Third, zkSTARKs [15] require circuits with a sequence of identical sub-circuits. Any circuit can be converted to this form [17, 19], but the transformation increases circuit sizes by 10–1000×, which translates to a similar factor increase in the prover's costs [104]; otherwise, zkSTARKs do not achieve sub-linear verification costs. Fourth, the verifier in Aurora [20], Ligero [5], and Bulletproofs [41] incurs costs that are linear in the size of the statement, so these schemes are NIZK [31] with succinct proofs. Fifth, zkSTARKs [15], Aurora [20], Fractal [47], and Virgo [109] rely on a non-standard conjecture for soundness [15, Appendix B]. Sixth, the prover in Fractal, Aurora, and zkSTARKs is $> 10\times$ slower than Groth16 [67], the state-of-the zkSNARK with trusted setup based on GGPR. Finally, SuperSonic's prover can be orders of magnitude more expensive than other schemes described here.[4]

Our goal is to address the aforementioned problems associated with existing transparent SNARKs. Specifically, we desire zkSNARKs with the following properties.

1. The argument's computational model should be general: the argument should apply to arbitrary R1CS instances without assuming structure (e.g., layering, uniformity).
2. The prover should be *time-optimal* i.e., it should run in $O(n)$ time.
3. Proofs should be succinct i.e., the size of a proof should be sub-linear in n.
4. The cost to verify a proof should be sub-linear in n and linear in $|io|$.
5. The constants in the asymptotics should be small.
6. The argument should not require a trusted setup nor a structured reference string.
7. Security should hold under standard cryptographic assumptions; non-interactivity can rely on random oracles [13] (since non-falsifiable assumptions are inherent [63]).

1.1 Summary of contributions

This paper presents a new family of zkSNARKs, which we call *Spartan*, for proving the satisfiability of NP statements over a large finite field expressed in R1CS. Spartan offers the first transparent zkSNARK that achieves sub-linear verification costs for arbitrary

[3] For a depth-d circuit, converting to a layered form increases the circuit size by a factor of $O(d)$.

[4] SuperSonic does not report the prover's concrete costs. At 128-bit security level, we find that operations over class groups can be $\approx 200\times$ slower than operations over primer-order groups such as `ristretto` [2, 3, 70].

	setup	prover	proof length	verifier	computational model
GGPR [61]	private	$O(n \log n)$	$O(1)$	$O(1)$	R1CS
Libra [108]	private*	$O(n)$	$O(d \log n)$	$O(d \log n)$	uniform circuits
Marlin [46]	private*	$O(n \log n)$	$O(\log n)$	$O(\log n)$	R1CS
PLONK [59]	private*	$O(n \log n)$	$O(1)$	$O(1)$	arithmetic circuits
zkSTARKs [15]	public	$O(n \log^2 n)$	$O(\log^2 n)$	$O(\log^2 n)$	uniform circuits
Hyrax [103]	public	$O(n + m \cdot g)$	$O(m + \sqrt{w})$	$O(m + \sqrt{w})$	data-parallel circuits
Virgo [110]	public	$O(n \log n)$	$O(d \log n)$	$O(d \log n)$	uniform circuits
Ligero [5]	public	$O(n \log n)$	$O(\sqrt{n})$	$O(n)$	arithmetic circuits
Bulletproofs [41]	public	$O(n)$	$O(\log n)$	$O(n)$	arithmetic circuits
SuperSonic [38]	public	$O(n \log n)$	$O(\log n)$	$O(\log n)$	arithmetic circuits
Aurora [20]	public	$O(n \log n)$	$O(\log^2 n)$	$O(n)$	R1CS
Fractal [47]	public	$O(n \log n)$	$O(\log^2 n)$	$O(\log^2 n)$	R1CS
Variants in Spartan's family of zkSNARKs:					
Spartan$_{\text{DL}}$	public	$O(n)$	$O(\sqrt{n})$	$O(\sqrt{n})$	R1CS
Spartan$_{\text{CL}}$	public	$O(n)$	$O(\log^2 n)$	$O(\log^2 n)$	R1CS
Spartan$_{\text{PKE}}$	private*	$O(n)$	$O(\log^2 n)$	$O(\log^2 n)$	R1CS
Spartan$_{\text{CRHF}}$	public	$O(n \log n)$	$O(\log^2 n)$	$O(\log^2 n)$	R1CS

Figure 1—A comparison of zkSNARK schemes, where n denotes the size of the NP statement (e.g., number of gates). For Hyrax [105], we assume a layered arithmetic circuit of depth d, width g, and β copies (i.e., $n = d \cdot g \cdot \beta$); w denotes the size of a witness to $\mathcal{C}$; and $m = d \cdot \log g$. Hyrax and Spartan$_{\text{DL}}$ can achieve sub-sqrt proofs at the cost of increasing $\mathcal{V}$'s time. For Libra and Virgo, we assume a depth-d layered uniform circuit. The verifier incurs $O(|io|)$ additional cost in all schemes where io denotes the public inputs and outputs of the NP relation being proved. Furthermore, all zkSNARKs without trusted setup listed above achieve non-interactivity in the random oracle model using the Fiat-Shamir heuristic [57]. Private* means that the trusted setup is universal and updateable. Ligero, Virgo, zkSTARKs, Aurora, Fractal, and Spartan$_{\text{CRHF}}$ are plausibly post-quantum secure.

NP statements.[5] Spartan also offers zkSNARKs with a time-optimal prover, a property that has remained difficult to achieve in nearly all prior zkSNARKs.

In a nutshell, Spartan introduces a new public-coin succinct interactive argument of knowledge where the verifier incurs sub-linear costs for arbitrary R1CS instances. Our argument makes a black box use of an extractable polynomial commitment scheme in conjunction with an information-theoretic protocol, so its soundness holds under the assumptions needed by the polynomial commitment scheme (note that there exist many polynomial commitment schemes that can be instantiated under standard cryptographic assumptions [38, 105, 110], which can be made non-interactive in the random oracle model). The interactive argument is public-coin, so we add zero-knowledge using existing compilers [105, 108, 112], which themselves build on prior theory [14, 45, 52]. We then make the resulting zero-knowledge argument of knowledge non-interactive in the random oracle model using the Fiat-Shamir transform [57]. Since our interactive argument employs a polynomial commitment scheme as a black box, we obtain a family of zkSNARKs where each variant is built by starting with a different polynomial commitment scheme. Figure 1 compares the asymptotic costs of Spartan-based zkSNARKs with prior schemes. In more detail, Spartan makes the following contributions.

[5] To our knowledge, short PCP-based transparent zkSNARKs [77, 86] do not achieve sub-linear verification costs unless one uses uniform circuits, which is undesirable as noted above.

(1) A new family of public-coin succinct interactive arguments of knowledge. Our core insight is that the sum-check protocol [83], a seminal interactive proof protocol (where soundness holds unconditionally), when applied to a suitably-constructed low-degree polynomial yields a powerful—but *highly inefficient*—interactive proof protocol, but the inefficiency can be tamed with new techniques. Specifically, we introduce three techniques (Figure 2 offers a visual depiction of how these techniques work together):

(i) *Computation commitments*, a primitive for creating succinct cryptographic commitments to a mathematical description of an NP statement, which is critical for achieving sub-linear verification costs.

Achieving sub-linear verification costs appears fundamentally unrealizable because the verifier must process an NP statement for which the proof is produced before it can verify a purported proof. Our observation is that most of this cost can be made sub-linear in the size of an NP statement by introducing a preprocessing step that only looks at the structure of the NP statement, but not public inputs or outputs.

In more detail, our observation is that when verifying a proof under our interactive argument, the verifier must evaluate a low-degree polynomial that encodes the NP statement, which incurs $O(n)$ costs to the verifier. Our primitive, computation commitments, enables verifiably delegating the necessary polynomial evaluations to the prover. Specifically, in Spartan, the verifier reads an R1CS instance (without the *io* component) for which the proof is produced and retains a short cryptographic commitment to a set of sparse multilinear polynomials that encode the R1CS structure. Later, when producing a proof, the prover evaluates the necessary polynomials and proves that the sparse polynomial evaluations are consistent with the commitment retained by the verifier. While the verifier incurs $O(n)$ cost to compute a computation commitment, the cost is amortized over *all* future proofs produced for all R1CS instances with the same structure. This amortization is similar to that of GGPR. However, unlike GGPR's trusted setup, creating a computation commitment does not involve any secret trapdoors. Section 6 provides details.

(ii) SPARK, a cryptographic compiler to transform any existing extractable polynomial commitment scheme for dense multilinear polynomials to one that efficiently handles sparse multilinear polynomials. Using the compiler, we obtain schemes with time-optimal costs for both creating commitments to sparse multilinear polynomials and to produce proofs of evaluations of the committed polynomials. This compiler is crucial for achieving a time-optimal prover in Spartan. In more detail, SPARK employs an existing extractable polynomial commitment scheme as a black box and uses it in conjunction with a special-purpose zkSNARK run on a carefully-constructed circuit (that employs offline memory checking techniques [6, 30, 49, 55, 93]) to efficiently prove evaluations of sparse multilinear polynomials. Section 7 provides details.

(iii) A compact encoding of an R1CS instance as a degree-3 multivariate polynomial that can be decomposed into four multilinear polynomials. The decomposition into multilinear polynomials is critical for achieving a time-optimal prover in the sum-check protocol by employing prior ideas [98, 108]. Section 4 provides details.

The following theorem states our main result informally and the result follows from

our construction in the rest of the paper (Section 2 provides formal definitions). The corollary below follows from prior transformations to achieve zero-knowledge [103, 108, 112] and non-interactivity [57].

Theorem 1.1. *There exists a family of public-coin succinct interactive arguments of knowledge for NP under standard cryptographic hardness assumptions where the prover incurs $O(n)$ costs and the verifier's costs and communication range from $O(\log^2 n)$ to $O(\sqrt{n})$ (depending on the underlying extractable polynomial commitment scheme for multilinear polynomials), where n is size of the NP statement.*

Corollary 1.1. *There exists a family of zkSNARKs for NP in the random oracle model where the prover incurs $O(n)$ costs and the verifier's costs and proof sizes range from $O(\log^2 n)$ to $O(\sqrt{n})$ (depending on the underlying polynomial commitment scheme for multilinear polynomials), where n denotes the size of the NP statement.*

(2) An optimized implementation and experimental evaluation. We implement Spartan as a library in about 8,000 lines of Rust. We use the library to build a transparent zkSNARK that employs an extractable polynomial commitment scheme due to Wahby et al. [105] where soundness holds under the hardness of computing discrete logarithms. We compare Spartan with five state-of-the-art zkSNARKs on the same hardware platform. Among transparent SNARKs, Spartan offers the fastest prover with a speedup of 17.5–115× depending on the baseline, produces proofs that are shorter (except compared to Bulletproofs and SuperSonic) by 2–540×, and incurs verification times that are faster (except compared to Fractal) by 71–240×. When compared to Groth16 [67], the state-of-the-art zkSNARK with trusted setup based on GGPR [61], Spartan's prover is 1.5× faster for arbitrary R1CS instances and $> 10\times$ faster for data-parallel workloads.

(3) Connections among different strands of theory. Our interactive argument exposes inter-connections among different lines of work on probabilistic proofs—from the perspective of zkSNARKs—including doubly-efficient IPs, MIPs, and short PCPs. Section 3.2 provides details in context, focusing on how Spartan unifies different strands.

(4) Improvements in zkSNARKs with stronger assumptions. Our principal focus is on transparent zkSNARKs, but Spartan improves on prior zkSNARKs if it can make assumptions similar to theirs.

First, by employing a different polynomial commitment scheme [89, 111], which requires q-type, knowledge of exponent assumptions, in SPARK, Spartan offers an alternative to Libra [108], Marlin [46], and PLONK [59]; we refer to this variant as $\text{Spartan}_{\text{PKE}}$. Unlike Marlin and PLONK, $\text{Spartan}_{\text{PKE}}$ features a linear-time prover. Compared to Libra, $\text{Spartan}_{\text{PKE}}$ supports arbitrary R1CS instances instead of layered arithmetic circuits. Furthermore, unlike Libra, the proof size and the verifier's running time in $\text{Spartan}_{\text{PKE}}$ are independent of the circuit depth. Finally, Libra achieves sub-linear verification costs only for low-depth uniform circuits whereas $\text{Spartan}_{\text{PKE}}$ achieves sub-linear verification costs for arbitrary R1CS instances with preprocessing via computation commitments.

Second, by employing a different polynomial commitment scheme [110], which requires assuming a non-standard conjecture for soundness, in SPARK and by employing their zero-knowledge compiler, we can obtain a zkSNARK that is plausibly post-quantum secure; we refer to this variant as $\text{Spartan}_{\text{CRHF}}$. Improvements of $\text{Spartan}_{\text{CRHF}}$ over Virgo are essentially the same as improvements of $\text{Spartan}_{\text{PKE}}$ over Libra noted above.

1.2 Recent schemes and additional related work

Recent developments. Several schemes mentioned above appeared following a prior version of Spartan. This includes Fractal [47], Marlin [46], PLONK [59], and SuperSonic [38]. We include these schemes in our comparisons for completeness. All these schemes employ a form of computation commitments, a technique introduced in this paper, to preprocess NP statements to achieve sub-linear verification costs. While Marlin and PLONK require a trusted setup, Fractal and SuperSonic do not. Section 8 provides a performance comparison with the latter two systems.

Linear PCPs and QAPs. Ishai, Kushilevitz, and Ostrovsky (IKO) [72] design the first interactive argument protocol without employing short PCPs. Instead, IKO use *linear* PCPs, a type of PCP in which the proof is a linear function [7, 8]. Such PCPs are simpler than short PCPs, but they are of size exponential in n. Thus, a polynomial time prover cannot materialize linear PCPs, a precise issue addressed by IKO: they devise a commitment protocol in which a prover can commit to an exponentially-long PCP without materializing the entire PCP.[6] Because of the specific linear PCP construction that they use, which is based on Hadamard codes [7, 8], the prover's work is $O(n^2)$. Setty et al. [96] strengthen IKO's cryptographic machinery to directly transform linear PCPs to arguments (IKO required the use of MIPs as an intermediate step), which simplifies the overall approach and significantly reduces constants. They also achieve $O(n)$ asymptotics for the prover by designing "tailored" linear PCPs for circuits with regular structure (e.g., matrix multiplication, polynomial evaluation). However, they retain the $O(n^2)$ costs for the prover in the general case.

As discussed in Section 1, GGPR [61] address the prover's asymptotics with QAPs. Bitansky et al. [29] and Setty et al. [95] observe that QAPs can be viewed as linear PCPs. Zaatar [95] leverages this observation to build an interactive argument with $O(n \log n)$ asymptotics for the prover by composing a QAPs-based linear PCP with a refined variant of IKO's machinery [96, 97]. The resulting asymptotics matches that of an argument based on state-of-the-art short PCPs. However, Zaatar retains the rest of IKO's limitations: the argument is interactive, is not publicly verifiable, and achieves succinctness only when proofs for a batch of statements are proved at once. Note that Zaatar and IKO do not support zero-knowledge arguments, but it is not fundamental [72, §3.1].

Pinocchio [90] optimizes and implements GGPR [61] in entirety. It avoids the issues listed above for Zaatar and IKO—at the cost of making q-type, knowledge of exponent assumptions, which are non-standard and non-falsifiable (Zaatar does not need such assumptions, but it is worth noting that such assumptions are inherent for achieving non-interactivity in arguments for NP [63]). BCGTV [19], BCTV [23], and Groth [67] offer algorithmic and concrete performance improvements over Pinocchio's zkSNARK. However, all these works require a trusted setup as detailed earlier. To cope with trusted setup, recent works [46, 59, 69, 84] propose schemes to update the structured reference string after it is generated. However, at least one updating entity must be trusted.

[6] Spartan is analogous to IKO in this aspect: The provers in IKO and Spartan do not materialize a PCP. The difference between these two works is that Spartan employs the polynomially-sized short PCPs of Babai et al. [11] whereas IKO employ the exponentially-sized linear PCPs of Arora et al. [7, 8]. This necessitates different cryptographic machinery.

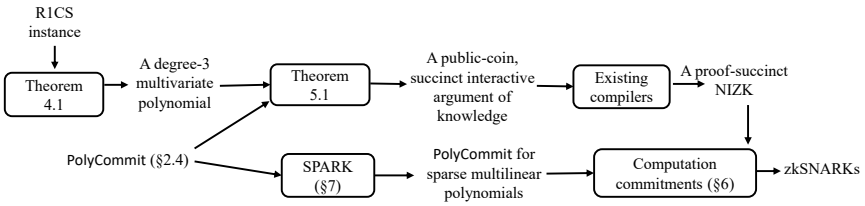

Figure 2—Overview of our techniques for constructing zkSNARKs.

Interactive proofs. While PCP-based arguments make cryptographic hardness assumptions, interactive proofs (IPs) are unconditionally secure. The early works on interactive proofs [10, 65] focus on studying the power of IPs in the context of intractable languages. However, in 2008, Goldwasser, Kalai, and Rothblum (GKR) [64] propose an elegant interactive proof system in the context of delegating computations where both the prover and the verifier are efficient; such IPs are called *doubly-efficient* IPs. Specifically, they construct doubly-efficient IPs for computations that can be expressed as log-space uniform circuits. In their protocol, the prover's running time is a polynomial in the size of the circuit, and the verifier's running time is linear in the depth of the circuit and logarithmic in the size of the circuit. Cormode, Mitzenmacher, and Thaler (CMT) [50] refine GKR's protocol to reduce the prover's work from $O(n^3)$ to $O(n \log n)$ by employing a specific polynomial extension to encode the circuit structure. A series of works [98, 100–103] further refines the approach of GKR and CMT to reduce constants and to improve asymptotics when circuits have structure such as data-parallelism. Recent work builds interactive arguments [111, 113] and zkSNARKs [105, 108, 112] using GKR, CMT, and their refinements. Section 3.2 discusses, in context, how these works relate to Spartan.

2 Preliminaries

We use $\mathbb{F}$ to denote a finite field (e.g., the prime field $\mathbb{F}_p$ for a large prime p) and λ to denote the security parameter. We use $\texttt{negl}(\lambda)$ to denote a negligible function in λ. We use PPT algorithms to refer to probabilistic polynomial time algorithms.

2.1 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 $\texttt{Sat}(\cdot,\cdot)$ such that:

$$\texttt{Sat}_{\mathcal{L}}(\mathbb{X}, w) = \begin{cases} 1 & \text{if } \mathbb{X} \in \mathcal{L} \\ 0 & \text{otherwise} \end{cases}$$

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). As noted earlier, R1CS is a popular target for most compiler toolchains that accept applications expressed in high-level languages. R1CS is implicit in the QAPs formalism of GGPR [61], but it is used with (and without) QAPs in subsequent works [20, 81, 95].

Definition 2.1 (R1CS instance)**.** An R1CS instance is the tuple $(\mathbb{F}, A, B, C, io, m, n)$, where io denotes the public input and output of the instance, $A, B, C \in \mathbb{F}^{m \times m}$, where $m \geq |io| + 1$ and there are at most n non-zero entries in each matrix.

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

Definition 2.2 (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.

Note that R1CS generalizes arithmetic circuit satisfiability because the entries in matrices A, B, C can be used to encode addition and multiplication gates over $\mathbb{F}$. Furthermore, they can be used to encode a class of degree-2 constraints of the form $L(z) \cdot R(z) = O(z)$, where L, R, O are degree-1 polynomials over variables that take values specified by $z = (io, 1, w)$. In other words, R1CS supports arbitrary fan-in addition gates, and multiplication gates verify arbitrary bilinear relations over the entire z.

Definition 2.3. 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 2.4. 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., $\mathbb{X} \in \mathcal{R}_{\mathrm{R1CS}}$) is of size $O(n)$.

2.2 Succinct interactive arguments of knowledge

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

Definition 2.5. 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 for any problem instance $\mathbb{X}$ and for all $w, r \in \{0,1\}^*$, if $\Pr\{\langle \mathcal{A}(pp, w), \mathcal{V}(pp, r) \rangle(\mathbb{X}) = 1\} \geq \mathtt{negl}(\lambda)$, then $\Pr\{\mathtt{Sat}_{\mathcal{L}}(\mathbb{X}, w') = 1 | w' \leftarrow \mathcal{E}^{\mathcal{A}}(pp, \mathbb{X})\} \geq \mathtt{negl}(\lambda)$.

- **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 adapt the following definitions from [105] for our notation.

Definition 2.6 (Witness-extended emulation [68])**.** 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 $\mathtt{negl}(\lambda)$: $\Pr\{pp \leftarrow \mathsf{Setup}(1^\lambda)\ ;\ (\mathbb{X}, z_{\mathcal{P}}) \leftarrow A(pp)\ ;\ t \leftarrow tr\langle \mathcal{P}^*(z_{\mathcal{P}}), \mathcal{V}(z_{\mathcal{V}})\rangle(\mathbb{X}) : A(t) = 1\}$ and $\Pr\{pp \leftarrow \mathsf{Setup}(1^\lambda)\ ;\ (\mathbb{X}, z_{\mathcal{P}}) \leftarrow A(pp)\ ;\ (t, w) \leftarrow E^{\mathcal{P}^*(z_{\mathcal{P}})}(\mathbb{X}) : A(t) = 1 \wedge \text{if } t \text{ is an accepting transcript then } \mathtt{Sat}_{\mathcal{L}}(\mathbb{X}, w) = 1\}$, where tr denotes the random variable that corresponds to the transcript of the interaction between $\mathcal{P}^*$ and $\mathcal{V}$.

Definition 2.7. 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.

2.3 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 2.8 (Multilinear polynomial)**.** A multivariate polynomial is called a multilinear polynomial if the degree of the polynomial in each variable is at most one.

Definition 2.9 (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 $g : \{0,1\}^m \to \mathbb{F}$ is a function that maps m-bit elements into an element of $\mathbb{F}$. A *polynomial extension* of g is a low-degree m-variate polynomial $\widetilde{g}(\cdot)$ such that $\widetilde{g}(x) = g(x)$ for all $x \in \{0,1\}^m$.

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 $Z : \{0,1\}^m \to \mathbb{F}$, the multilinear extension of $Z(\cdot)$ is the unique multilinear polynomial $\widetilde{Z} : \mathbb{F}^m \to \mathbb{F}$. It can be computed as follows.

$$\begin{aligned}\widetilde{Z}(x_1,\ldots,x_m) &= \sum_{e\in\{0,1\}^m} Z(e)\cdot\prod_{i=1}^{m}(x_i\cdot e_i+(1-x_i)\cdot(1-e_i))\\ &= \sum_{e\in\{0,1\}^m} Z(e)\cdot\widetilde{\mathrm{eq}}(x,e)\\ &= \langle (Z(0),\ldots,Z(2^m-1)),(\widetilde{\mathrm{eq}}(x,0),\ldots,\widetilde{\mathrm{eq}}(x,2^m-1)\rangle\end{aligned}$$

Note that $\widetilde{\mathrm{eq}}(x,e) = \prod_{i=1}^{m}(e_i\cdot x_i+(1-e_i)\cdot(1-x_i))$, 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 $r \in \mathbb{F}^m$, $\widetilde{Z}(r)$ can be computed in $O(2^m)$ operations in $\mathbb{F}$ [98, 101].

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 $\mathcal{G}(\cdot) : \mathbb{F}^m \to \mathbb{F}$, it can be represented uniquely by the list of evaluations of $\mathcal{G}(\cdot)$ over the Boolean hypercube $\{0,1\}^m$ (i.e., a function that maps $\{0,1\}^m \to \mathbb{F}$). We denote such a representation of $\mathcal{G}$ as $\texttt{DenseRepr}(\mathcal{G})$.

Lemma 2.1. *If for any $x \in \{0,1\}^m$, $\mathcal{G}(x) = 0$ 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 $(r_1,\ldots,r_m) \in \mathbb{F}^m$: $\mathcal{G}(r_1,\ldots,r_m) = \sum_{x\in\{0,1\}^m}\mathcal{G}(x)\cdot\prod_{i=1}^{m}(r_i\cdot x_i+(1-r_i)\cdot(1-x_i))$. Observe that if for any $x \in \{0,1\}^m$, $\mathcal{G}(x) = 0$, x does not contribute to $\mathcal{G}(r)$ for any $r \in \mathbb{F}^m$. □

Definition 2.10 (Dense and sparse multilinear polynomials)**.** A multilinear polynomial $\mathcal{G} : \mathbb{F}^m \to \mathbb{F}$ is said to be a sparse multilinear polynomial if $|\texttt{DenseRepr}(\mathcal{G})|$ is sub-linear in $O(2^m)$. 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 $O(2^{2s})$.

2.4 A polynomial commitment scheme for multilinear polynomials

We adopt our definitions from Bünz et al. [38] where they generalize the definition of Kate et al. [76] 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 polynomial commitment scheme for multilinear polynomials is a tuple of four protocols $\mathsf{PC} = (\mathsf{Setup}, \mathsf{Commit}, \mathsf{Open}, \mathsf{Eval})$:

- $pp \leftarrow \mathsf{Setup}(1^\lambda, \mu)$: takes as input μ (the number of variables in a multilinear polynomial); produces public parameters pp.
- $(\mathcal{C}; \mathcal{S}) \leftarrow \mathsf{Commit}(pp; \mathcal{G})$: takes as input a μ-variate multilinear polynomial over a finite field $\mathcal{G} \in \mathbb{F}[\mu]$; produces a public commitment $\mathcal{C}$ and a secret opening hint $\mathcal{S}$.
- $b \leftarrow \mathsf{Open}(pp, \mathcal{C}, \mathcal{G}, \mathcal{S})$: verifies the opening of commitment $\mathcal{C}$ to the μ-variate multilinear polynomial $\mathcal{G} \in \mathbb{F}[\mu]$ with the opening hint $\mathcal{S}$; outputs a $b \in \{0,1\}$.
- $b \leftarrow \mathsf{Eval}(pp, \mathcal{C}, r, v, \mu; \mathcal{G}, \mathcal{S})$ is an interactive public-coin protocol between a PPT prover $\mathcal{P}$ and verifier $\mathcal{V}$. Both $\mathcal{V}$ and $\mathcal{P}$ hold a commitment $\mathcal{C}$, the number of variables μ, a scalar $v \in \mathbb{F}$, and $r \in \mathbb{F}^\mu$. $\mathcal{P}$ additionally knows a μ-variate multilinear polynomial $\mathcal{G} \in \mathbb{F}[\mu]$ and its secret opening hint $\mathcal{S}$. $\mathcal{P}$ attempts to convince $\mathcal{V}$ that $\mathcal{G}(r) = v$. At the end of the protocol, $\mathcal{V}$ outputs $b \in \{0,1\}$.

Definition 2.11. A tuple of four protocols $(\mathsf{Setup}, \mathsf{Commit}, \mathsf{Open}, \mathsf{Eval})$ is an extractable polynomial commitment scheme for multilinear polynomials over a finite field $\mathbb{F}$ if the following conditions hold.

- **Completeness.** For any m-variate multilinear polynomial $\mathcal{G} \in \mathbb{F}[\mu]$,

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

- **Binding.** For any PPT adversary $\mathcal{A}$, size parameter $\mu \geq 1$,

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

- **Knowledge soundness.** Eval is a public-coin succinct interactive argument of knowledge with witness-extended emulation (Definition 2.6) for the following NP relation given $pp \leftarrow \mathsf{Setup}(1^\lambda, \mu)$ for a size parameter on the number of variables μ:

$$\mathcal{R}_{\mathsf{Eval}}(pp) = \{\langle (\mathcal{C}, r, v), (\mathcal{G}, \mathcal{S}) \rangle : \mathcal{G} \in \mathbb{F}[\mu] \wedge \mathcal{G}(r) = v \wedge \mathsf{Open}(pp, \mathcal{C}, \mathcal{G}, \mathcal{S}) = 1\}$$

Definition 2.12. An extractable polynomial commitment scheme $(\mathsf{Setup}, \mathsf{Commit}, \mathsf{Open}, \mathsf{Eval})$ provides hiding commitments if for all PPT adversaries $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$:

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

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

Definition 2.13. An extractable polynomial commitment scheme $(\mathsf{Setup}, \mathsf{Commit}, \mathsf{Open}, \mathsf{Eval})$ with hiding commitments (Definition 2.12) is zero-knowledge if Eval is a public-coin succinct interactive argument of knowledge with witness-extended emulation (Definition 2.6) and zero-knowledge (Definition 2.7) for the following NP relation given $pp \leftarrow \mathsf{Setup}(1^\lambda, \mu)$ for a size parameter on the number of variables μ:

$$\mathcal{R}_{\mathsf{Eval}}(pp) = \{\langle (\mathcal{C}, r, v), (\mathcal{G}, \mathcal{S})\rangle : \mathcal{G} \in \mathbb{F}[\mu] \wedge \mathcal{G}(r) = v \wedge \mathsf{Open}(pp, \mathcal{C}, \mathcal{G}, \mathcal{S}) = 1\}$$

3 The sum-check protocol: opportunities and challenges

An interactive proof is an interactive argument where the soundness holds unconditionally. We now describe a seminal interactive proof protocol that we employ in Spartan, called the sum-check protocol [83]. Suppose there is an μ-variate low-degree polynomial, $\mathcal{G} : \mathbb{F}^\mu \to \mathbb{F}$ where the degree of each variable in $\mathcal{G}$ is at most ℓ. Suppose that a verifier $\mathcal{V}_{SC}$ is interested in checking a claim of the following form by an untrusted prover $\mathcal{P}_{SC}$:

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

Of course, given $\mathcal{G}(\cdot)$, $\mathcal{V}_{SC}$ can deterministically evaluate the above sum and verify whether the sum is T. But, this computation takes time exponential in μ.

Lund et al. [83] describe an interactive proof, called *the sum-check protocol*, that requires far less computation on $\mathcal{V}_{SC}$'s behalf, but provides a probabilistic guarantee. In the protocol, $\mathcal{V}_{SC}$ interacts with $\mathcal{P}_{SC}$ over a sequence of ℓ rounds. At the end of this interaction, $\mathcal{V}_{SC}$ outputs $b \in \{0,1\}$. The principal cost to $\mathcal{V}_{SC}$ is to evaluate the polynomial $\mathcal{G}$ at a random point in its domain $r \in \mathbb{F}^\mu$. We denote the sum-check protocol as $b \leftarrow \langle \mathcal{P}_{SC}, \mathcal{V}_{SC}(r)\rangle(\mathcal{G}, \mu, \ell, T)$. For any μ-variate polynomial $\mathcal{G}$ with degree at most ℓ in each variable, the following properties hold.

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

An alternate formulation. In the rest of the paper, it is natural to view the sum-check protocol as a mechanism to reduce a claim of the form $\sum_{x \in \{0,1\}^m} \mathcal{G}(x) \stackrel{?}{=} T$ to the claim $\mathcal{G}(r) \stackrel{?}{=} e$. This is because in most cases, $\mathcal{V}$ uses an auxiliary protocol to verify the latter claim, so this formulation makes it easy to describe our end-to-end protocols. Figure 3 depicts the $\mathcal{V}$'s side of the protocol from this perspective. We denote this reduction protocol with $e \leftarrow \langle \mathcal{P}_{SC}(\mathcal{G}), \mathcal{V}_{SC}(r)\rangle(\mu, \ell, T)$.

```
// reduces the claim $\sum_{x\in\{0,1\}^s} \mathcal{G}(x) \stackrel{?}{=} T$ to $\mathcal{G}(r) \stackrel{?}{=} e$
function SumCheckReduce($\mu$, $\ell$, $T$, $r$)
    $(r_1, r_2, \ldots, r_\mu) \leftarrow r$
    $e \leftarrow T$
    for $i = 1, 2, \ldots, \mu$ do
        $\mathcal{G}_i(\cdot) \leftarrow$ ReceiveFromProver() // an honest $\mathcal{P}_{SC}$ returns $\{\mathcal{G}_i(0), \mathcal{G}_i(1), \ldots \mathcal{G}_i(\ell)\}$
        if $\mathcal{G}_i(0) + \mathcal{G}_i(1) \neq e$ then
            return 0
        SendToProver($r_i$)
        $e \leftarrow \mathcal{G}_i(r_i)$ // evaluate $\mathcal{G}_i(r_i)$ using its point-value form received from the prover
    return $e$
```

Figure 3—Description of the sum-check protocol from the perspective of $\mathcal{V}_{SC}$. $\mathcal{V}_{SC}$ checks if the μ-variate polynomial $\mathcal{G}(\cdot)$ sums to T over the Boolean hypercube $\{0,1\}^\mu$ with the assistance of an untrusted prover $\mathcal{P}_{SC}$. The degree of $\mathcal{G}(\cdot)$ in each variable is at most ℓ.

3.1 Challenges with using the sum-check protocol for succinct arguments

To build a succinct interactive argument of knowledge for R1CS, we need an interactive protocol for $\mathcal{V}$ to check if $\mathcal{P}$ knows a witness w to a given R1CS instance $\mathbb{X} = (\mathbb{F}, A, B, C, io, m, n)$ such that $\mathtt{Sat}_{\text{R1CS}}(\mathbb{X}, w) = 1$.

At first glance, the sum-check protocol [83] seems to offer the necessary building block (it is public-coin, incurs succinct communication, etc.). However, to build a succinct interactive argument of knowledge (that can in turn be compiled into a zkSNARK), we must solve the following sub-problems:

1. **Encode R1CS instances as sum-check instances.** For any R1CS instance $\mathbb{X} = (\mathbb{F}, A, B, C, io, m, n)$, we must devise an degree-ℓ, μ-variate polynomial that sums to a specific value T over $\{0,1\}^\mu$ *if and only if* there exists a witness w such that $\mathtt{Sat}_{\text{R1CS}}(\mathbb{X}, w) = 1$, where $\mu = O(\log m)$ and ℓ is a small constant (e.g., 3).
2. **Achieve communication-succinctness.** Although the sum-check protocol offers succinctness (if the first sub-problem is solved with constraints on μ and ℓ noted above), building a succinct interactive argument is non-trivial. This is because after the sum-check reduction, $\mathcal{V}$ must verify $\mathcal{G}(r) \stackrel{?}{=} e$. Unfortunately, $\mathcal{G}(r)$ depends on the $\mathcal{P}$'s witness w to $\mathbb{X}$. Thus, a naive evaluation of $\mathcal{G}(r)$ requires $O(m)$ communication to transmit w. Transmitting w is also incompatible with zero-knowledge.
3. **Achieve verifier-succinctness.** To compile an interactive argument to a zkSNARK, $\mathcal{V}$'s costs must be sub-linear in the size of an NP statement, but evaluating $\mathcal{G}(r)$ requires $O(n)$ computation if the statement has no structure (e.g., data-parallelism). A potential way around this fundamental issue is for $\mathcal{V}$ to preprocess the structure of the R1CS instance to accelerate all future verifications of proofs for different R1CS instances with the same structure. However, to avoid any form of trusted setup, the preprocessing must not involve secret trapdoors.

   The next subsection describes prior solutions to the three sub-problems.

3.2 Prior solutions to solve the above problems (Spartan's closely related works)

Prior literature on probabilistic proofs, starting with Babai et al. [11, 12], offers a low-degree polynomial $\mathcal{G}(\cdot)$ that fits the structure of the sum-check protocol. However, most prior proposals construct such a polynomial for encoding the satisfiability of a Boolean formula [12] (or the correct execution of a program under a pointer machine [11]). These capture a general model of computation, but those representations are orders of magnitude more verbose than R1CS (verbosity translates to constants in the prover's and verifier's costs in the argument protocol). Blumberg et al. [32] offer a low-degree polynomial as part of a multi-prover interactive proof (MIP) protocol [11, 60, 64, 106] for the arithmetic circuit satisfiability (ACS) problem. In theory, this addresses the first sub-problem, but in practice, arithmetic circuits impose overheads that range from small constant factors to orders of magnitude in degenerate cases compared to R1CS. Furthermore, they pose programmability challenges (e.g., ACS must explicitly encode additions whereas R1CS obtains them for free, checking a non-deterministic witness requires additional effort as part of the toolchain or the programmer).

We now discuss how prior work addresses the latter two sub-problems.

MIPs. MIP protocols solve the second sub-problem by employing two (or more) non-colluding provers. For example, in the protocol of Blumberg et al. [32] (which builds on the two-prover protocol of Babai et al. [12]), $\mathcal{V}$ interacts with the first prover to run the sum-check protocol, which requires $\mathcal{V}$ to evaluate $\mathcal{G}(\cdot)$ at a random point (as described above, this would require $O(n)$ computation by $\mathcal{V}$ and $O(|w|)$ communication from $\mathcal{P}$ to $\mathcal{V}$). Instead, $\mathcal{V}$ interacts with a second prover—via *low-degree tests* [9, 87]—to learn the desired evaluation of $\mathcal{G}(\cdot)$. Despite a sophisticated analysis of soundness error, their protocol achieves only 23 bits of security (for $|\mathbb{F}| \approx 2^{300}$). Although MIPs require two (or more) non-colluding provers, Bitansky and Chiesa [28] offer a compiler to transform MIPs to SNARKs [28]. However, their compiler relies on fully-homomorphic encryption (FHE) [62], so it is only of theoretical interest at this point.

If we view Spartan in this light, Spartan is an efficient mechanism—without employing FHE or low-degree tests—to compile the two-prover protocol of Blumberg et al. [32] and Babai et al. [12] (and other similar two-prover IPs) into a public-coin succinct interactive argument of knowledge (and then into a zkSNARK without trusted setup). For non-interactivity, Spartan assumes a random oracle model whereas the compiler of Bitansky and Chiesa [28] requires a non-falsifiable variant of FHE. Furthermore, Spartan achieves a publicly verifiable argument whereas the compiler of Bitansky and Chiesa only yields a *designated* verifier argument (i.e., the proof produced is meant for a specific verifier rather than any verifier). Relatedly, Thaler [99, §3] observes that the MIP of Blumberg et al. [32] can be compiled into a single prover argument using a polynomial commitment scheme. However, the proposal does not solve the third sub-problem to achieve sub-linear verification costs for the verifier, so it does not lead to a zkSNARK.

Short PCPs. Babai et al. [11] devise a short PCP where the PCP includes two components: (1) the prover's responses to all possible $\mathcal{V}$'s challenges in the sum-check protocol (an oracle access to such a PCP allows $\mathcal{V}$ to conduct the sum-check protocol by accessing only a few bits in the PCP) and (2) a low-degree extension (LDE) of a purported witness w to an NP-complete problem (which, with an oracle access, allows $\mathcal{V}$ to evaluate $G(\cdot)$

at a random point as required by the last step of the sum-check protocol). Unfortunately, as discussed in Section 1, constructing a succinct interactive argument of knowledge via such short PCPs—using Kilian's approach [77]—remains highly impractical.

Note that Spartan is a more direct transformation of the short PCP construction of Babai et al. [11] into a succinct interactive argument of knowledge: (1) The prover in Spartan does not write down a low-degree extension of w (but instead cryptographically commits to a low-degree extension of w using w alone); (2) The Spartan prover also does not write down all possible responses to the verifier's challenges in the sum-check protocol; instead, the prover engages in an interactive sum-check protocol with $\mathcal{V}$; (3) Babai et al. [11] avoid multilinear extensions (MLE) of a witness since the resulting PCP string will be super-polynomial in the size of the NP instance; however, since Spartan's prover does not write down the entire PCP, the use of a MLE is not only more efficient than other LDEs but also enables the use of simple cryptographic primitives to commit to such an MLE without ever materializing it. This view of Spartan is reminiscent of ideas in the work of Ishai et al. [72], and the compiler of Bitansky and Chiesa [28].

Doubly-efficient interactive proofs. Doubly-efficient interactive proofs [50, 64, 98, 100–103] solve all the three sub-problems—by restricting themselves to deterministic circuits in a layered form. They apply a sequence of sum-check protocols to recursively reduce a claim about outputs to a claim about inputs of the circuit. As a result, the low-degree polynomial that $\mathcal{V}$ must evaluate as part of the final instance of the sum-check protocol is only over the public inputs to the circuit, which $\mathcal{V}$ can locally compute. Naturally, these works are restricted to low-depth circuits since $\mathcal{V}$'s work is linear in the circuit depth. Furthermore, the circuits in these works cannot take a non-deterministic witness w as an input from $\mathcal{P}$—without incurring $O(|w|)$ communication from $\mathcal{P}$ to $\mathcal{V}$ [101] or using additional machinery (see below). Besides generality, the lack of support for non-determinism necessitates using Boolean circuits instead of arithmetic circuits (Boolean circuits are orders of magnitude more verbose than an equivalent arithmetic circuits for many programs of interest). This is because efficient transformations from high-level programs to ACS (or R1CS) make extensive use of non-determinism for integer and bitwise operations [90, 97], storage [37, 93], and RAM [17, 22, 104].

Zhang et al. [111] extend doubly-efficient IPs to the complexity class NP by employing a polynomial commitment scheme [76, 89] and then transform their interactive argument to a zkSNARK [108, 112] using prior transformations [14, 52]. However, their polynomial commitment scheme requires a trusted setup. In a different work, Wahby et al. [105] transform the Giraffe IP [103] (a doubly-efficient IP in the GKR [64] line of work [50, 98, 102, 103]) into a zkSNARK *without* requiring a trusted setup. They design optimized variants of prior zero-knowledge transformations [14, 52] in conjunction with a new polynomial commitment scheme. However, the zkSNARKs of Wahby et al. [105] and Zhang et al. [112] retain the requirement of layered circuits. To address this, Kalai sketches "squashed GKR" [74]: instead of running the GKR protocol on a layered circuit, it is run on a low-depth circuit that takes as input a witness whose size is proportional to the number of gates in the layered circuit. To avoid the verifier from having to materialize the witness, the proposal employs low-degree tests and a polynomial commitment scheme. However, the scheme only produces designated verifier proofs and it relies on FHE. However, more fundamentally, the proposal does not address

the third sub-problem to achieve sub-linear costs for the verifier.

To achieve sub-linear verification costs, Zhang et al. [111] and Wahby et al. [105] focus on data-parallel computations. Unfortunately, this poses severe restrictions in practice. To mitigate perils of such a requirement, Wahby et al. [105] design an irregular circuit layer, called a *redistribution layer* (RDL), that allows sharing witness elements across different data-parallel units. Naturally, $\mathcal{V}$ incurs linear costs for RDL. Concretely, in two out of their three benchmarks, $\mathcal{V}$'s dominant cost is computation related to RDL.

If seen from the perspective of this line of work, Spartan is a way to eliminate the requirement of layered circuits as well as a way to achieve sub-linear verification costs—without requiring any homogeneity in circuit structure.[7] Specifically, we observe that the sum-check protocol (applied to a suitable low-degree polynomial $\mathcal{G}(\cdot)$), $\mathcal{V}$ can delegate the required evaluation of $\mathcal{G}(\cdot)$ at a random point in its domain to the prover $\mathcal{P}$.

4 An encoding of R1CS instances as low-degree polynomials

This section describes a compact encoding of an R1CS instance as a degree-3 multivariate polynomial. The following theorem summarizes our result, which we prove below.

Theorem 4.1. *For any R1CS instance $\mathbb{X} = (\mathbb{F}, A, B, C, io, m, n)$, there exists a degree-3 $\log m$-variate polynomial $\mathcal{G}$ such that $\sum_{x \in \{0,1\}^{\log m}} \mathcal{G}(x) = 0$ if and only if there exists a witness w such that $\mathtt{Sat}_{R1CS}(\mathbb{X}, w) = 1$ (except for a soundness error that is negligible in λ) under the assumption that $|\mathbb{F}|$ is exponential in λ and $m = O(\lambda)$.*

For a given R1CS instance $\mathbb{X} = (\mathbb{F}, A, B, C, io, m, n)$, let $s = \lceil \log m \rceil$. Thus, we can view matrices $A, B, C \in \mathbb{F}^{m \times m}$ as functions with the following signature: $\{0,1\}^s \times \{0,1\}^s \to \mathbb{F}$. Specifically, any entry in them can be accessed with a $2s$-bit identifier (or two s-bit identifiers). Furthermore, given a purported witness w to $\mathbb{X}$, let $Z = (io, 1, w)$. It is natural to interpret Z as a function with the following signature: $\{0,1\}^s \to \mathbb{F}$, so any element of Z can be accessed with an s-bit identifier.

We now describe a function $F_{io}(\cdot)$ that can be used to encode w such that $F_{io}(\cdot)$ exhibits a desirable behavior *if and only if* $\mathtt{Sat}_{\mathrm{R1CS}}(\mathbb{X}, w) = 1$.

$$F_{io}(x) = \left(\sum_{y \in \{0,1\}^s} A(x,y) \cdot Z(y) \right) \cdot \left(\sum_{y \in \{0,1\}^s} B(x,y) \cdot Z(y) \right) - \sum_{y \in \{0,1\}^s} C(x,y) \cdot Z(y)$$

Lemma 4.1. *$\forall x \in \{0,1\}^s$, $F_{io}(x) = 0$ if and only if $\mathtt{Sat}_{R1CS}(\mathbb{X}, w) = 1$.*

Proof. This follows from the definition of $\mathtt{Sat}_{\mathrm{R1CS}}(\mathbb{X}, w)$ (Section 2.1) and of $Z(\cdot)$. □

Unfortunately $F_{io}(\cdot)$ is a function, *not* a polynomial, so it cannot be directly used in

[7] In theory, Canetti et al [43] propose an alternate approach for achieving sub-linear verification costs: the verifier pre-evaluates the necessary low-degree polynomials at *all* points in their domain (which resembles the prover's effort in the short PCPs of Babai et al. [11]) and builds a Merkle tree. Later, when verifying proof, the verifier (with the root of the Merkle tree) can obtain desired polynomial evaluations from the prover with sub-linear costs. The work to create the Merkle tree is too expensive to be used in practice. In contrast, Spartan offers a more direct approach where the public computation is linear in the size of the NP relation.

the sum-check protocol. But, consider its polynomial extension $\widetilde{F}_{io} : \mathbb{F}^s \to \mathbb{F}$.

$$\widetilde{F}_{io}(x) = \left(\sum_{y\in\{0,1\}^s} \widetilde{A}(x,y)\cdot\widetilde{Z}(y)\right)\cdot\left(\sum_{y\in\{0,1\}^s} \widetilde{B}(x,y)\cdot\widetilde{Z}(y)\right) - \sum_{y\in\{0,1\}^s} \widetilde{C}(x,y)\cdot\widetilde{Z}(y)$$

Lemma 4.2. *$\forall x \in \{0,1\}^s$, $\widetilde{F}_{io}(x) = 0$ if and only if $\mathtt{Sat}_{R1CS}(\mathbb{X}, w) = 1$.*

Proof. For any $x \in \{0,1\}^s$, $\widetilde{F}_{io}(x) = F_{io}(x)$, so the result follows from Lemma 4.1. □

Since $\widetilde{F}_{io}(\cdot)$ is a low-degree multivariate polynomial over $\mathbb{F}$ in s variables, a verifier $\mathcal{V}$ could check if $\sum_{x\in\{0,1\}^s} \widetilde{F}_{io}(x) = 0$ using the sum-check protocol with a prover $\mathcal{P}$. But, this is insufficient: $\sum_{x\in\{0,1\}^s} \widetilde{F}_{io}(x) = 0$ does not imply that $F_{io}(x)$ is zero $\forall x \in \{0,1\}^s$. This is because the 2^s terms in the sum might cancel each other making the final sum zero—even when some of the individual terms are not zero.

We addresses the above issue using a prior idea [12, 32, 42]. Consider:

$$Q_{io}(t) = \sum_{x\in\{0,1\}^s} \widetilde{F}_{io}(x)\cdot\widetilde{\mathrm{eq}}(t,x),$$

where $\widetilde{\mathrm{eq}}(t,x) = \prod_{i=1}^{s}(t_i\cdot x_i + (1-t_i)\cdot(1-x_i))$.

Observe that $Q_{io}(\cdot)$ is a multivariate polynomial such that $Q_{io}(t) = \widetilde{F}_{io}(t)$ for all $t \in \{0,1\}^s$. Thus, $Q_{io}(\cdot)$ is a *zero-polynomial* (i.e., it evaluates to zero for all points in its domain) *if and only if* $\widetilde{F}_{io}(\cdot)$ evaluates to zero at all points in the s-dimensional Boolean hypercube (and hence *if and only if* $\widetilde{F}_{io}(\cdot)$ encodes a witness w such that $\mathtt{Sat}_{\mathrm{R1CS}}(\mathbb{X}, w) = 1$). To check if $Q_{io}(\cdot)$ is a zero-polynomial, it suffices to check if $Q_{io}(\tau) = 0$ where $\tau \in_R \mathbb{F}^s$. This introduces a soundness error, which we quantify below.

Lemma 4.3. $\Pr_\tau\{Q_{io}(\tau) = 0 | \exists x \in \{0,1\}^s \text{ s.t. } \widetilde{F}_{io}(x) \neq 0\} \leq \log m/|\mathbb{F}|$

Proof. If $\exists x \in \{0,1\}^s$ such that $\widetilde{F}_{io}(x) \neq 0$, then $Q_{io}(t)$ is not a zero-polynomial. By the Schwartz-Zippel lemma, $Q_{io}(t) = 0$ for at most $d/|\mathbb{F}|$ values of t in the domain of $Q_{io}(\cdot)$, where d is the degree of $Q_{io}(\cdot)$. Here, $d = s = \log m$. □

Proof of Theorem 4.1. For a given R1CS instance $\mathbb{X} = (\mathbb{F}, A, B, C, io, m, n)$, define, $\mathcal{G}_{io,\tau}(x) = \widetilde{F}_{io}(x)\cdot\widetilde{\mathrm{eq}}(\tau, x)$, so $Q_{io}(\tau) = \sum_{x\in\{0,1\}^s} \mathcal{G}_{io,\tau}(x)$. Observe that $\mathcal{G}_{io,\tau}(\cdot)$ is a degree-3 s-variate polynomial if multilinear extensions of A, B, C, and Z are used in $\widetilde{F}_{io}(\cdot)$. In the terminology of the sum-check protocol, $T = 0, \mu = s = \log m$, and $\ell = 3$. Furthermore, if $\tau \in_R \mathbb{F}^s$, $\sum_{x\in\{0,1\}^s} \mathcal{G}_{io,\tau}(x) = 0$ if and only $\widetilde{F_{io}}(x) = 0$ $\forall x \in \{0,1\}^s$—except for soundness error that is negligible in λ under the assumptions noted above (lemma 4.3). This combined with lemma 4.2 implies the desired result.

5 A NIZK with succinct proofs for R1CS

This section describes a new NIZK [31] for R1CS with succinct proofs. We first design an interactive argument with succinct communication costs and then compile it into a proof-succinct NIZK in the random oracle model using prior transformations.

5.1 A new public-coin succinct interactive argument of knowledge

The following theorem summarizes our result in this section.

Theorem 5.1. *Given an extractable polynomial commitment scheme for multilinear polynomials, there exists a public-coin succinct interactive argument of knowledge where security holds under the assumptions needed for the polynomial commitment scheme and assuming $|\mathbb{F}|$ is exponential in λ and the size parameter of R1CS instance $n = O(\lambda)$.*

To prove the above theorem, we first provide a construction of a public-coin succinct interactive argument of knowledge, and then analyze its costs and security. The proof of Theorem 4.1 established that for $\mathcal{V}$ to verify if an R1CS instance $\mathbb{X} = (\mathbb{F}, A, B, C, io, m, n)$ is satisfiable, it can check if $\sum_{x\in\{0,1\}^s} \mathcal{G}_{io,\tau}(x) = 0$. By using the sum-check protocol, we can reduce the claim about the sum to $e_x \stackrel{?}{=} \mathcal{G}_{io,\tau}(r_x)$ where $r_x \in \mathbb{F}^s$, so $\mathcal{V}$ needs a mechanism to evaluate $\mathcal{G}_{io,\tau}(r_x)$—without incurring $O(m)$ communication from $\mathcal{P}$ to $\mathcal{V}$.

Recall that $G_{io,\tau}(x) = \widetilde{F}_{io}(x) \cdot \widetilde{\text{eq}}(\tau, x)$. Thus, to evaluate $G_{io,\tau}(r_x)$, $\mathcal{V}$ must evaluate $\widetilde{F}_{io}(r_x)$ and $\widetilde{eq}(\tau, r_x)$. The latter can be evaluated in $O(\log m)$ time. Furthermore, recall:

$$\widetilde{F}_{io}(r_x) = \left(\sum_{y\in\{0,1\}^s} \widetilde{A}(r_x, y) \cdot \widetilde{Z}(y)\right) \cdot \left(\sum_{y\in\{0,1\}^s} \widetilde{B}(r_x, y) \cdot \widetilde{Z}(y)\right) - \sum_{y\in\{0,1\}^s} \widetilde{C}(r_x, y) \cdot \widetilde{Z}(y)$$

To evaluate $\widetilde{F}_{io}(r_x)$, $\mathcal{V}$ needs to evaluate the following $\forall y \in \{0,1\}^s$: $\widetilde{A}(r_x, y)$, $\widetilde{B}(r_x, y)$, $\widetilde{C}(r_x, y)$, and $\widetilde{Z}(y)$. However, the evaluations of $\widetilde{Z}(y)$ for all $y \in \{0,1\}^s$ is the same as $(io, 1, w)$, so the communication from $\mathcal{P}$ to $\mathcal{V}$ is $\geq O(|w|)$. We now address this issue.

Our solution is a combination of three protocols: the sum-check protocol, a randomized mini protocol, and a polynomial commitment scheme. Our first observation is that the structure of the individual terms in $F_{x,y}(\cdot)$ evaluated at r_x are in a form suitable for the application of a second instance of the sum-check protocol. Specifically, let $\widetilde{F}_{io}(r_x) = \overline{A}(r_x) \cdot \overline{B}(r_x) - \overline{C}(r_x)$, where

$$\begin{aligned}
\overline{A}(r_x) &= \sum_{y\in\{0,1\}^s} \widetilde{A}(r_x, y) \cdot \widetilde{Z}(y) \\
\overline{B}(r_x) &= \sum_{y\in\{0,1\}^s} \widetilde{A}(r_x, y) \cdot \widetilde{Z}(y) \\
\overline{C}(r_x) &= \sum_{y\in\{0,1\}^s} \widetilde{A}(r_x, y) \cdot \widetilde{Z}(y)
\end{aligned}$$

This observation opens up the following solution: the prover can make three separate claims to $\mathcal{V}$, say that $\overline{A}(r_x) = v_A$, $\overline{B}(r_x) = v_B$, and $\overline{C}(r_x) = v_C$. Then, $\mathcal{V}$ can evaluate:

$$\mathcal{G}_{io,\tau}(r_x) = (v_A \cdot v_B - v_C) \cdot \widetilde{\text{eq}}(r_x, \tau),$$

which in turn enables $\mathcal{V}$ to verify $\mathcal{G}_{io,\tau}(r_x) \stackrel{?}{=} e_x$. Of course, $\mathcal{V}$ must still verify three new claims from $\mathcal{P}$: $\overline{A}(r_x) \stackrel{?}{=} v_A$, $\overline{B}(r_x) \stackrel{?}{=} v_B$, and $\overline{C}(r_x) \stackrel{?}{=} v_C$. Of course, $\mathcal{V}$ and $\mathcal{P}$ can run

three independent instances of the sum-check protocol to verify these claims. Instead, we use a prior idea [45, 105] to combine three claims into a single claim:

- $\mathcal{V}$ samples $r_A, r_B, r_C \in_R \mathbb{F}$ and computes $c = r_A \cdot v_A + r_B \cdot v_B + r_C \cdot v_C$.
- $\mathcal{V}$ uses the sum-check protocol with $\mathcal{P}$ to verify $r_A \cdot \overline{A}(r_x) + r_B \cdot \overline{B}(r_x) + r_C \cdot \overline{C}(r_x) \stackrel{?}{=} c$. In more detail, let $L(r_x) = r_A \cdot \overline{A}(r_x) + r_B \cdot \overline{B}(r_x) + r_C \cdot \overline{C}(r_x)$.

$$\begin{aligned} L(r_x) &= \sum_{y \in \{0,1\}^s} r_A \cdot \widetilde{A}(r_x, y) \cdot \widetilde{Z}(y) + r_B \cdot \widetilde{B}(r_x, y) \cdot \widetilde{Z}(y) + r_C \cdot \widetilde{C}(r_x, y) \cdot \widetilde{Z}(y) \\ &= \sum_{y \in \{0,1\}^s} M_{r_x}(y) \end{aligned}$$

  $M_{r_x}(y)$ is an s-variate polynomial with degree at most 2 in each variable. In the terminology of the sum-check protocol, $\mu = s, \ell = 2$, and $T = c$.

Lemma 5.1. $\Pr_{r_A, r_B, r_C}\{r_A \cdot \overline{A}(r_x) + r_B \cdot \overline{B}(r_x) + r_C \cdot \overline{C}(r_x) = c | \overline{A}(r_x) \neq v_A \vee \overline{B}(r_x) \neq v_B \vee \overline{C}(r_x) \neq v_C\} \leq 1/|\mathbb{F}|$, *where* $c = r_A \cdot v_A + r_y \cdot v_B + r_C \cdot v_C$.

Proof. The LHS is a polynomial in r_A, r_B, r_C of total degree 1; the same holds for the RHS. So, the desired result follows from the Schwartz-Zippel lemma. □

$\mathcal{V}$ is not out of the woods. At the end of the second instance of the sum-check protocol, $\mathcal{V}$ must evaluate $M_{r_x}(r_y)$ for $r_y \in \mathbb{F}^s$:

$$\begin{aligned} M_{r_x}(r_y) &= r_A \cdot \widetilde{A}(r_x, r_y) \cdot \widetilde{Z}(r_y) + r_B \cdot \widetilde{B}(r_x, r_y) \cdot \widetilde{Z}(r_y) + r_C \cdot \widetilde{C}(r_x, r_y) \cdot \widetilde{Z}(r_y) \\ &= (r_A \cdot \widetilde{A}(r_x, r_y) + r_B \cdot \widetilde{C}(r_x, r_y) + r_C \cdot \widetilde{C}(r_x, r_y)) \cdot \widetilde{Z}(r_y) \end{aligned}$$

Observe that the only term in $M_{r_x}(r_y)$ that depends on the prover's witness is $\widetilde{Z}(r_y)$. This is because all other terms in the above expression can be computed locally by $\mathcal{V}$ using $\mathbb{X} = (\mathbb{F}, A, B, C, io, m, n)$ in $O(n)$ time (Section 6 discusses how to reduce the cost of those evaluations to be sub-linear in n). Our second observation is that to evaluate $\widetilde{Z}(r_y)$ without incurring $O(|w|)$ communication from $\mathcal{P}$ to $\mathcal{V}$, we can employ an extractable polynomial commitment scheme for multilinear polynomials (§2.4). A similar observation was made by Zhang et al. [111] in a different context (§3.2).

In more detail, $\mathcal{P}$ sends a commitment to $\widetilde{w}(\cdot)$ (i.e., a multilinear extension of its purported witness) to $\mathcal{V}$ *before* the first instance of the sum-check protocol begins using an extractable polynomial commitment scheme for multilinear polynomials. To evaluate $\widetilde{Z}(r_y)$, $\mathcal{V}$ does the following. WLOG, assume $|w| = |io| + 1$. Thus, by the closed form expression of multilinear polynomial evaluations, we have:

$$\widetilde{Z}(r_y) = (1 - r_y[0]) \cdot \widetilde{w}(r_y[1..]) + r_y[0] \cdot \widetilde{(io, 1)}(r_y[1..]),$$

where $r_y[1..]$ refers to a slice of r_y that excludes the the first element.

Putting things together. We now describe our succinct interactive argument of knowledge for R1CS as follows. We assume that there exists an extractable polynomial commitment scheme for multilinear polynomials $\mathsf{PC} = (\mathsf{Setup}, \mathsf{Commit}, \mathsf{Open}, \mathsf{Eval})$. Let m denote the size parameter from the R1CS instance.

- $pp \leftarrow \mathsf{Setup}(1^\lambda)$: Invoke $pp \leftarrow \mathsf{PC.Setup}(1^\lambda, \log m)$; output pp.
- $b \leftarrow \langle \mathcal{P}(w), \mathcal{V}(r) \rangle (\mathbb{F}, A, B, C, io, m, n)$:
  1. $\mathcal{P} : (\mathcal{C}, \mathcal{S}) \leftarrow \mathsf{PC.Commit}(pp, \widetilde{w})$ and send $\mathcal{C}$ to $\mathcal{V}$.
  2. $\mathcal{V} : \tau \in_R \mathbb{F}^{\log m}$ and send τ to $\mathcal{P}$.
  3. Let $T_1 = 0$, $\mu_1 = \log m$, $\ell_1 = 3$.
  4. $\mathcal{V}$: Sample $r_x \in_R \mathbb{F}^{\mu_1}$
  5. **Sum-check#1.** $e_x \leftarrow \langle \mathcal{P}_{SC}(\mathcal{G}_{io,\tau}), \mathcal{V}_{SC}(r_x) \rangle (\mu_1, \ell_1, T_1)$
  6. $\mathcal{P}$: Compute $v_A = \overline{A}(r_x), v_B = \overline{B}(r_x), v_C = \overline{C}(r_x)$; send (v_A, v_B, v_C) to $\mathcal{V}$.
  7. $\mathcal{V}$: Abort with $b = 0$ if $e_x \neq (v_A \cdot v_B - v_C) \cdot \widetilde{\mathrm{eq}}(r_x, \tau)$.
  8. $\mathcal{V}$: Sample $r_A, r_B, r_C \in_R \mathbb{F}$ and send (r_A, r_B, r_C) to $\mathcal{P}$.
  9. Let $T_2 = r_A \cdot v_A + r_B \cdot v_B + r_C \cdot v_C$, $\mu_2 = \log m$, $\ell_2 = 2$.
  10. $\mathcal{V}$: Sample $r_y \in_R \mathbb{F}^{\mu_2}$
  11. **Sum-check#2.** $e_y \leftarrow \langle \mathcal{P}_{SC}(M_{r_x}), \mathcal{V}_{SC}(r_y) \rangle (\mu_2, \ell_2, T_2)$
  12. $\mathcal{P}$: $v \leftarrow \widetilde{w}(r_y[1..])$ and send v to $\mathcal{V}$.
  13. $b_e \leftarrow \langle \mathcal{P}_{\mathsf{PC.Eval}}(\widetilde{w}, \mathcal{S}), \mathcal{V}_{\mathsf{PC.Eval}}(r) \rangle (pp, \mathcal{C}, r_y, v, \mu_2)$
  14. $\mathcal{V}$: Abort with $b = 0$ if $b_e = 0$.
  15. $\mathcal{V} : v_Z = (1 - r_y[0]) \cdot \widetilde{w}(r_y[1..]) + r_y[0] \cdot \widetilde{(io, 1)}(r_y[1..])$
  16. $\mathcal{V} : v_1 \leftarrow \widetilde{A}(r_x, r_y), v_2 \leftarrow \widetilde{B}(r_x, r_y), v_3 \leftarrow \widetilde{C}(r_x, r_y)$
  17. $\mathcal{V}$: Abort with $b = 0$ if $e_y \neq (r_A \cdot v_1 + r_B \cdot v_2 + r_C \cdot v_3) \cdot v_Z$.
  18. $\mathcal{V}$: Output $b = 1$.

Choice of a polynomial commitment scheme. There exist many extractable polynomial commitment schemes for multilinear polynomials [38, 76, 89, 105, 109, 111] that suffice for our purposes. The particular choice impacts the costs of our protocol as well as assumptions, so we review prior commitment schemes' costs and assumptions.

Analysis of costs. By employing prior ideas [98, 103, 108] to implement a linear-time prover for the sum-check protocol, the costs of our interactive argument are as follows.

- $\mathcal{P}$ incurs: (1) $O(n)$ costs to participate in the sum-check instances; (2) the cost of $\mathsf{PC.Commit}$ and $\mathsf{PC.Eval}$ for a $\log m$-variate multilinear polynomial $\widetilde{w}(\cdot)$.
- $\mathcal{V}$ incurs: (1) $O(\log m)$ costs for the sum-check instances; (2) the cost of $\mathsf{PC.Eval}$ for a $\log m$-variate multilinear polynomial; and (3) $O(n)$ costs to evaluate $\widetilde{A}(\cdot), \widetilde{B}(\cdot), \widetilde{C}(\cdot)$.
- The amount of communication is: (1) $O(\log m)$ in the sum-check instances; (2) the size of the commitment to $\widetilde{w}(\cdot)$ and the communication in $\mathsf{PC.Eval}$ for $\widetilde{w}(\cdot)$.

prior scheme	setup	$\mathcal{P}_{\mathsf{Eval}}$	$\lvert\mathcal{C}\rvert$	communication	$\mathcal{V}_{\mathsf{Eval}}$	assumption
Hyrax-PC [103]	public	$O(\Gamma)$	$O(\sqrt{\Gamma})$	$O(\log \Gamma)$	$O(\sqrt{\Gamma})$	DLOG
DARK-CL [38]	public	$O(\Gamma)$	$O(\log \Gamma)$	$O(\log \Gamma)$	$O(\log \Gamma)$	$\mathbb{G}$ (unknown order)
vSQL-VPD [111]	private	$O(\Gamma)$	$O(1)$	$O(\log \Gamma)$	$O(\log \Gamma)$	q-PKE
Virgo-VPD [109]	public	$O(\Gamma \log \Gamma)$	$O(1)$	$O(\log^2 \Gamma)$	$O(\log^2 \Gamma)$	CRHF

Figure 4—A comparison of candidate extractable polynomial commitment schemes for multilinear polynomials. Here, $\Gamma = 2^{\mu}$ where μ is the number of variables in the multilinear polynomial. Hyrax-PC refers to the scheme of Wahby et al. [105], which also supports shorter commitments at the cost of increasing the verifier's time. DARK-CL refers to the scheme of Bunz et al [38] instantiated with class groups. vSQL-VPD refers to the zero-knowledge variant [112] of the scheme of Zhang et al. [111]. Virgo-VPD refers to the scheme of Zhang et al. [110]. The communication column refers to the amount of communication required in the interactive argument for PC.Eval.

PC choice	setup	prover	communication	verifier	assumption
Hyrax-PC [103]	public	$O(n)$	$O(\sqrt{m})$	$O(n+\sqrt{m})$	DLOG
DARK-CL [38]	public	$O(n)$	$O(\log m)$	$O(n+\log m)$	$\mathbb{G}$ (unknown order)
vSQL-VPD [111]	private	$O(n)$	$O(\log m)$	$O(n+\log m)$	q-PKE
Virgo-VPD [109]	public	$O(n+m\log m)$	$O(\log^2 m)$	$O(n+\log^2 m)$	CRHF

Figure 5—Costs of our public-coin succinct interactive argument of knowledge instantiated with different polynomial commitment schemes. The depicted costs are for an R1CS instance $\mathbb{X} = (\mathbb{F}, A, B, C, io, m, n)$.

Proof of Theorem 5.1. The desired completeness of our interactive argument of knowledge follows from the completeness of the sum-check protocol and of the underlying polynomial commitment scheme. Furthermore, in all the four candidate constructions for polynomial commitment schemes, the communication from $\mathcal{P}$ to $\mathcal{V}$ is sub-linear in m (Figure 5), which satisfies succinctness. Thus, we are left with proving witness-extended emulation (Definition 2.6), which we prove in Appendix A.

5.2 NIZK with succinct proofs for R1CS

The interactive argument from the prior subsection is public coin, so we add zero-knowledge using prior techniques [14, 52]. There are two prior compilers that are particularly efficient: (1) the compiler employed by Hyrax [105], which relies on a sigma protocol for proving dot-product relationships; and (2) the compiler employed by Libra [108] and Virgo [110], which relies on an extractable polynomial commitment scheme. Both compilers require the polynomial commitment scheme used in the interactive argument to support zero-knowledge (Definition 2.13); all of our candidate schemes in Figure 4 satisfy this requirement. This transformation not change asymptotics of $\mathcal{P}$, $\mathcal{V}$, or of the amount of communication (Figure 5). Finally, since our protocol is public coin, it can be made non-interactive in the random oracle model using the Fiat-Shamir transform [57], thereby obtaining a family of NIZKs with succinct proofs for R1CS.

6 Computation commitments: zkSNARKs for R1CS from NIZK

The previous section constructed a family of NIZKs but not zkSNARKs. This is because the verifier incurs costs linear in the size of the R1CS instance to evaluate $\widetilde{A}, \widetilde{B}, \widetilde{C}$ at (r_x, r_y). We now discuss how to achieve sub-linear verification costs.

At first blush, this appears impossible: The verifier incurs $O(n)$ costs to evaluate $\widetilde{A}, \widetilde{B}, \widetilde{C}$ at (r_x, r_y) (step 16,§5.1), which is time-optimal [98, 101] if $\mathbb{X}$ has no structure

(e.g., uniformity). We get around this impossibility by introducing a preprocessing step for $\mathcal{V}$. In an offline phase, $\mathcal{V}$ with access to non-*io* portions of an R1CS instance $\mathbb{X} = (\mathbb{F}, A, B, C, io, m, n)$ executes the following, where $pp_{cc} \leftarrow \mathsf{PC.Setup}(1^\lambda, 2\log m)$ and PC is an extractable polynomial commitment scheme for multilinear polynomials.

$\underline{\mathsf{Encode}(pp_{cc}, (A, B, C))}$:

- $(\mathcal{C}_A, \mathcal{S}_A) \leftarrow \mathsf{PC.Commit}(pp_{cc}, \widetilde{A})$
- $(\mathcal{C}_B, \mathcal{S}_B) \leftarrow \mathsf{PC.Commit}(pp_{cc}, \widetilde{B})$
- $(\mathcal{C}_C, \mathcal{S}_C) \leftarrow \mathsf{PC.Commit}(pp_{cc}, \widetilde{C})$
- Output $(\mathcal{C}_A, \mathcal{C}_B, \mathcal{C}_C)$

$\mathcal{V}$ retains commitments output by Encode (which need not hide the underlying polynomials, so in practice $\mathcal{S}_A = \mathcal{S}_B = \mathcal{S}_C = \perp$). The interactive argument proceeds as in the prior section except that at step 16, instead of $\mathcal{V}$ evaluating $\widetilde{A}, \widetilde{B}, \widetilde{C}$, we have:

- $\mathcal{P} : v_1 \leftarrow \widetilde{A}(r_x, r_y)$, $v_2 \leftarrow \widetilde{B}(r_x, r_y)$, $v_3 \leftarrow \widetilde{C}(r_x, r_y)$. Send (v_1, v_2, v_3) to $\mathcal{V}$.
- $b_1 \leftarrow \langle \mathcal{P}_{\mathsf{PC.Eval}}(\widetilde{A}, \perp), \mathcal{V}_{\mathsf{PC.Eval}}(r) \rangle (pp_{cc}, \mathcal{C}_A, (r_x, r_y), v_1, 2\log m)$
- $b_2 \leftarrow \langle \mathcal{P}_{\mathsf{PC.Eval}}(\widetilde{B}, \perp), \mathcal{V}_{\mathsf{PC.Eval}}(r) \rangle (pp_{cc}, \mathcal{C}_B, (r_x, r_y), v_2, 2\log m)$
- $b_3 \leftarrow \langle \mathcal{P}_{\mathsf{PC.Eval}}(\widetilde{C}, \perp), \mathcal{V}_{\mathsf{PC.Eval}}(r) \rangle (pp_{cc}, \mathcal{C}_C, (r_x, r_y), v_3, 2\log m)$
- $\mathcal{V}$: Abort with $b = 0$ if $b_1 = 0 \vee b_2 = 0 \vee b_3 = 0$.

Lemma 6.1. *The interactive argument from Section 5.1 where step 16 is replaced with the above protocol is a public-coin succinct interactive argument of knowledge assuming PC is an extractable polynomial commitment scheme for multilinear polynomials.*

Proof. The result follows from the knowledge soundness property satisfied by PC scheme used in the Encode algorithm. □

If $\mathcal{V}$'s costs to verify the three evaluations and the added communication are sublinear in $O(n)$, the modified interactive argument leads to a zkSNARK (if we add zero-knowledge and non-interactivity as before).

Unfortunately, existing polynomial commitment schemes do not satisfy the desired efficiency properties: (1) to participate in Eval for any of $\widetilde{A}, \widetilde{B}, \widetilde{C}$, $\mathcal{P}$ incurs quadratic costs i.e., $O(m^2)$; and (2) in some schemes (e.g., Hyrax-PC), the modified interactive argument does not offer improved asymptotics for the verifier.

Details. Recall that polynomials $\widetilde{A}, \widetilde{B}, \widetilde{C}$ are are multilinear extensions of matrices A, B, C in an R1CS instance $\mathbb{X} = (\mathbb{F}, A, B, C, io, m, n)$ (§4). Furthermore, these multilinear polynomials are defined over $\mu = 2s$ variables, where $s = \log m$. Thus, if we apply any existing polynomial commitment from prior subsection (Figure 4): $\Gamma = 2^\mu = 2^{2\log m}$. Thus, $\mathcal{P}_{\mathsf{Eval}}$ incurs at least $O(m^2)$, which is also quadratic in n since $n = O(m)$. Furthermore, in schemes such as Hyrax-PC, $\mathcal{V}_{\mathsf{Eval}}$ incurs $O(n)$ costs. Neither is desirable for

instantiating computation commitments (§6), which aims to make $\mathcal{V}$'s costs sub-linear in $O(n)$ by delegating evaluations of $\widetilde{A}, \widetilde{B}, \widetilde{C}$ at (r_x, r_y) to $\mathcal{P}$.

The next section describes a scheme that meets our efficiency requirements and leads to asymptotics noted in Figure 1.

7 The SPARK compiler

This section describes SPARK, a new cryptographic compiler to transform an existing extractable polynomial commitment scheme for dense multilinear polynomials to one that can efficiently handle sparse multilinear polynomials. In particular, we describe two compilers: one that offers a standard polynomial commitment scheme and another that is more efficient but requires the Commit algorithm to be run by a trusted entity. This limitation is acceptable in the context of computation commitments because the verifier runs the Commit algorithm as part of Encode (§6). Furthermore, it is worth noting that this does not introduce a trusted setup as there are no secret trapdoors.

Our core observation is that it is possible to build a polynomial commitment scheme that efficiently handles sparse multilinear polynomials by using a zkSNARK that achieves sub-linear verification costs for a restricted class of NP statements. Two attractive candidates here include: Hyrax [105] and the Spartan-based NIZK from Section 5.2 (both can achieve sub-linear verification costs for NP statements with uniform structure).

For ease of exposition, we focus on describing SPARK that applies to $2\log m$-variate sparse polynomials $\widetilde{A}, \widetilde{B}, \widetilde{C}$ (where their dense representation is of size $\leq n$) from Section 5.1, but our result generalizes to other sparse multilinear polynomials. We also conjecture that our solution generalizes to sparse bivariate polynomials of high degree.

7.1 SPARK-naive: A straw-man solution

To present our solution, we describe a straw-man that helps introduce the necessary building blocks as well as articulate difficulties addressed by SPARK. We recall Hyrax [105], a zkSNARK that achieves sub-linear verification costs for uniform circuits, specifically data-parallel circuits. The prover's costs in Hyrax can be made linear in the circuit size using subsequent ideas [108]. Furthermore, the verifier's costs are $O(d \log n + e)$ where d is the depth of the circuit and e is to the verifier to participate in PC.Eval to evaluate a $\log |w|$-variate multilinear polynomial where w is a witness to the circuit.

Details. Let M denote one of $\{A, B, C\}$ and let $s = \log m$, so $\mu = 2s$. Recall the closed-form expression for multilinear polynomial evaluations at $r \in \mathbb{F}^{\mu}$.

$$\widetilde{M}(r) = \sum_{i \in \{0,1\}^{\mu} :: M(i) \neq 0} M(i) \cdot \widetilde{\text{eq}}(i, r) \tag{1}$$

The above sum has at most n terms since $M(i) \neq 0$ for at most n values of i. Also, each entry in the sum can be computed with $\mu + 1$ multiplications. Consider the following circuit to evaluate $\widetilde{M}(r)$.

A $O(\log \mu)$-depth circuit with $O(n \cdot \mu)$ gates that:

- Takes as witness the list of tuples of the form $(i, M(i)) :: M(i) \neq 0$, where each i is represented with a vector of μ elements of $\mathbb{F}$, so each entry in the list is $\mu + 1$ elements

of $\mathbb{F}$ (in other words, the witness is a $\log(n \cdot (\mu+1))$-variate multilinear polynomial whose dense representation is the above list of tuples);

- Takes as public input $r \in \mathbb{F}^\mu$;
- Computes $v \leftarrow \widetilde{M}(r)$ using Equation 1;
- Outputs v

Note that the above circuit is uniform: there are n identical copies of a sub-circuit, where each sub-circuit computes $O(\mu+1)$ multiplications; the outputs of these sub-circuits is fed into a binary tree of addition gates to compute the final sum. Furthermore, there is no sharing of witness elements across data-parallel units, so it truly data-parallel.

Construction. Given an extractable polynomial commitment scheme PC for multilinear polynomials, we build a scheme for sparse multilinear polynomials as follows.

$\mathsf{PC}^{\text{naive}}$:

- $pp \leftarrow \mathsf{Setup}(1^\lambda, \mu, n)$: $\mathsf{PC.Setup}(1^\lambda, \log((\mu+1)\cdot n))$
- $(\mathcal{C}; \mathcal{S}) \leftarrow \mathsf{Commit}(pp; \widetilde{M})$: $\mathsf{PC.Commit}(pp, \mathcal{D})$, where $\mathcal{D}$ is the unique $\log((\mu+1)\cdot n)$-variate multilinear polynomial whose dense representation is the list of tuples $(i, M(i))$ :: $M(i) \neq 0$ and each entry is $(\mu+1)$ elements of $\mathbb{F}$.
- $b \leftarrow \mathsf{Open}(pp, \mathcal{C}, \widetilde{M}, \mathcal{S})$: $\mathsf{PC.Open}(pp, \mathcal{C}, \mathcal{D}, \mathcal{S})$, where $\mathcal{D}$ is defined as above.
- $b \leftarrow \mathsf{Eval}(pp, \mathcal{C}, r, v, \mu, n; \widetilde{M}, \mathcal{S})$: $\mathcal{P}$ and $\mathcal{V}$ use Hyrax to verify the claim that $\widetilde{M}(r) = v$ using the circuit described above.

Analysis of costs. Recall that computing $\widetilde{M}(r)$ for $M \in \{A, B, C\}$ and $r \in \mathbb{F}^\mu$ takes $O(n)$ costs. The costs of $\mathsf{PC}^{\text{naive}}$ are as follows. The principal downside is that the the prover is slower than a direct evaluation of the polynomial by a factor of $O(\mu)$, so it does not lead to time-optimal costs. This slowdown is also significant in practice (§8).

PC choice	setup	$\mathcal{P}_{\mathsf{Eval}}$	$\lvert\mathcal{C}\rvert$	communication	$\mathcal{V}_{\mathsf{Eval}}$
Hyrax-PC [103]	public	$O(\Upsilon)$	$O(\sqrt{\Upsilon})$	$O(\log\Upsilon \cdot \log\log\mu)$	$O(\sqrt{\Upsilon})$
DARK-CL [38]	public	$O(\Upsilon)$	$O(\log\Upsilon)$	$O(\log\Upsilon \cdot \log\log\mu)$	$O(\log\Upsilon \cdot \log\log\mu)$
vSQL-VPD [111]	private*	$O(\Upsilon)$	$O(1)$	$O(\log\Upsilon \cdot \log\log\mu)$	$O(\log\Upsilon \cdot \log\log\mu)$
Virgo-VPD [109]	public	$O(\Upsilon\log\Upsilon)$	$O(1)$	$O(\log^2\Upsilon)$	$O(\log^2\Upsilon)$

Figure 6—Costs of $\mathsf{PC}^{\text{naive}}$ with different choice for PC. Here, $\Upsilon = n\cdot\mu$ where μ is the number of variables in the multilinear polynomial and n is size of the dense representation of a sparse multilinear polynomial.

Lemma 7.1. *PC^{naive} is a polynomial commitment scheme for multilinear polynomials with the costs noted above.*

Proof. Completeness follows from the completeness of PC and Hyrax. Binding follows from the uniqueness of the dense representation of a sparse multilinear polynomial. Knowledge soundness follows from the witness-extended emulation offered by Hyrax

and PC.Eval. The claimed costs follow from the cost model of Hyrax and PC applied to a $(\log n \cdot (\mu+1))$-variate multilinear polynomial. □

7.2 Breaking the $O(n \log m)$ barrier by leveraging memory checking

We now discuss how to improve on the straw-man scheme by a factor of $O(\mu)$ on the prover. Our focus is on realizing a polynomial commitment scheme for the purpose of efficiently realizing computation commitments (§6). For this purpose, the Spartan verifier runs the Commit algorithm as part of the Encode algorithm, so unlike the general setup of polynomial commitments, the sender is not untrusted. The solution below leverages this observation to create additional metadata about the sparse polynomial as part of Commit. Without it, the circuit for evaluating the sparse polynomial will be $O(n \log |\mathbb{F}|)$, which in practice may not be better than the $O(n \log m)$-sized circuit from the prior subsection.

Details. Observe that for $M \in \{A, B, C\}$, $M \in \mathbb{F}^{m \times m}$ and any $r \in \mathbb{F}^{\mu}$, we can rewrite the evaluation of $\widetilde{M}(r)$ as follows. In our context $\mu = 2 \log m$, interpret r as a tuple (r_x, r_y) where $r_x, r_y \in \mathbb{F}^s$ and $s = \log m = \mu/2$. Thus, we can rewrite Equation 1 as:

$$\widetilde{M}(r_x, r_y) = \sum_{(i,j) \in (\{0,1\}^s, \{0,1\}^s) \,::\, M(i,j) \neq 0} M(i,j) \cdot \widetilde{\mathrm{eq}}(i, r_x) \cdot \widetilde{\mathrm{eq}}(j, r_y)$$

In our context, the above sum still contains n terms. Furthermore, computing each entry in the sum still requires $(\mu+1)$ multiplications over $\mathbb{F}$. However, it is possible to compute evaluations of $\widetilde{\mathrm{eq}}(i, r_x)$ for all $i \in \{0,1\}^s$ in $O(2^s) = O(m)$ time. Similarly, we can compute a table of evaluations of $\widetilde{\mathrm{eq}}(j, r_y)$ for all $j \in \{0,1\}^s$ in $O(m)$ time.

Unfortunately, this observation is insufficient: even though these tables can be computed in $O(m)$ time, the sum is taken over the list of $(i,j) \in (\{0,1\}^s, \{0,1\}^s)$ where $M(i,j) \neq 0$ and for an arbitrary $2s$-variate sparse multilinear polynomial, such a list of (i,j) has no structure, so computing the sum requires n random accesses into the two tables, each of size m. We could attempt to build a circuit that supports RAM operations. Unfortunately, existing techniques to encode RAM in circuits incur a logarithmic blowup or constants that in practice are larger than a logarithmic blowup.

For m RAM operations over a memory of size m,

- Pantry [37], using Merkle trees, trees [30, 85], offers a circuit of size $O(m \log m)$.
- Buffet [104], using permutation networks [17], offers a circuit of size $O(m \log m)$ with constants smaller than the ones in Pantry.
- vRAM [113] offers a $O(m)$-sized circuit with a constant of $\log |\mathbb{F}|$ (to encode consistency checks over a memory transcript), so, in practice, this does not improve on the straw-man. Other downsides: (1) it only supports 32-bit sized memory cells, whereas we need a memory over elements of $\mathbb{F}$; (2) nearly all of the circuit's non-deterministic witness must be committed by $\mathcal{P}$ during circuit evaluation.

Our solution specializes and improves upon a recent implementation of offline memory checking techniques [30] in Spice [93], which builds circuits to encode operations on persistent storage with serializable transactions. The storage abstraction can be used as a memory abstraction where for m operations, the circuit is of size $O(m)$, but the constants are worse than those of VRAM: ≥ 1000 (to encode an elliptic-curve based

multiset collision-resistant hash function for each memory operation). We get around this issue by designing a new randomized check. Furthermore, unlike a vRAM-based solution, most of the non-deterministic witness needed by the circuit can be created by PC.Commit (i.e., by the Encode algorithm in computation commitments).

Encoding sparse polynomials. Given a sparse polynomial $\widetilde{M}$ (e.g., $\widetilde{M} \in \{\widetilde{A}, \widetilde{B}, \widetilde{C}\}$), we encode it using three lists of size n as follows. Since $\widetilde{M}$ is represented by n tuples of the form $(i, j, M(i,j))$, where each tuple has 3 elements of $\mathbb{F}$ (this differs from the straw-man where each i and j were encoded using a vector of s elements of $\mathbb{F}$), such that $M(i,j) \neq 0$. In some canonical order, let row, col, val be three lists that encode the above n tuples such that for $k \in [0, n-1]$ $row(k) = i, col(k) = j, val(k) = M(i,j)$.

"Memory in the head". We now capture additional metadata about $\widetilde{M}$ that is necessary for memory checking during the evaluation of $\widetilde{M}(r)$. Note that computing this additional metadata only needs the following parameters: memory size (which is determined by $2^s = m$) and the sequence of addresses at which the memory is accessed (which are provided by *row* and *col*).

Let $\textit{read-ts}_{row} \in \mathbb{F}^n, \textit{write-ts}_{row} \in \mathbb{F}^n$ denote the timestamps associated with read and write operations and $\textit{audit-ts}_{row} \in \mathbb{F}^m$ denote the final timestamps of memory cells in the offline memory checking primitive [30, §4.1] for the address sequence specified by *row* over a memory of size $m = O(2^s)$. Similarly, let $\textit{read-ts}_{col} \in \mathbb{F}^n, \textit{write-ts}_{col} \in \mathbb{F}^n$ denote the timestamps associated with read and write operations and $\textit{audit-ts}_{col} \in \mathbb{F}^m$ denote the final timestamps of memory cells in the offline memory checking primitive [30, §4.1] for the address sequence specified by *col* over a memory of size $m = O(2^s)$. The following pseudocode summarizes how these timestamps are computed (*vec*! uses Rust notation).

MemoryInTheHead($m, n, addrs$):

- $\textit{read-ts} \leftarrow \textit{vec}![n\ ;\ 0]$; $\textit{write-ts} \leftarrow \textit{vec}![n\ ;\ 0]$; $\textit{audit-ts} \leftarrow \textit{vec}![m\ ;\ 0]$; $ts \leftarrow 0$
- for i in $(0..\textit{addrs.len}())$:
  - $addr \leftarrow addrs[i]$
  - $\textit{r-ts} \leftarrow \textit{audit-ts}[i]$
  - $ts \leftarrow max(ts, \textit{r-ts}) + 1$
  - $\textit{read-ts}[i] \leftarrow \textit{r-ts}$
  - $\textit{write-ts}[i] \leftarrow ts$
  - $\textit{audit-ts}[addr] \leftarrow ts$
- return $(\textit{read-ts}, \textit{write-ts}, \textit{audit-ts})$

An $O(n)$-sized circuit. We now describe an $O(n)$-sized circuit to compute an evaluation of $\widetilde{M}$. We prove that the circuit indeed computes the correct evaluation of the sparse polynomial in lemma 7.5. In the description of the circuit, we assume hash functions H and $\mathcal{H}$, which are defined after the description of the circuit.

An $O(n)$-sized, $O(\log n)$-depth circuit ($Circuit_{\text{eval-opt}}$).

- Takes as witness the following lists (Hyrax can accept witness in separate lists).
  1. a succinct description of $\widetilde{M}$: three lists *row*, *col*, *val*, where each list has n entries.
  2. two lists e_{row}, e_{col}, where each list contains n elements of $\mathbb{F}$.
  3. six lists: $read\text{-}ts_{row}$,$read\text{-}ts_{col}$,$write\text{-}ts_{row}$, $write\text{-}ts_{col}$,$audit\text{-}ts_{row}$, and $audit\text{-}ts_{col}$. The first four are of size n and the last two are of size m; each entry is an element of $\mathbb{F}$.
  4. two challenges $\gamma_1, \gamma_2 \in \mathbb{F}$.
- Takes as public input $r = (r_x, r_y) \in \mathbb{F}^{\mu}$;
- Output $\widetilde{M}(r)$ using $v \leftarrow \sum_{k=0}^{n-1} val[k] \cdot e_{row}[k] \cdot e_{col}[k]$.
- Memory checking for e_{row}:
  - $mem_{row} \leftarrow [\widetilde{\text{eq}}(0, r_x), \ldots, \widetilde{\text{eq}}(m-1, r_x)] \in \mathbb{F}^m$
  - $Init_{row} \leftarrow H_{\gamma_1}([0, \ldots, m-1], mem_{row}, [0, \ldots, 0]) \in \mathbb{F}^m$
  - $RS_{row} \leftarrow H_{\gamma_1}(row, e_{row}, read\text{-}ts_{row}) \in \mathbb{F}^n$
  - $WS_{row} \leftarrow H_{\gamma_1}(row, e_{row}, write\text{-}ts_{row}) \in \mathbb{F}^n$
  - $Audit_{row} \leftarrow H_{\gamma_1}([0, \ldots, m-1], mem_{row}, audit\text{-}ts_{row}) \in \mathbb{F}^m$
  - Assert $\mathcal{H}_{\gamma_2}(Init_{row}) \cdot \mathcal{H}_{\gamma_2}(WS_{row}) = \mathcal{H}_{\gamma_2}(RS_{row}) \cdot \mathcal{H}_{\gamma_2}(Audit_{row})$
- Memory checking for e_{col}:
  - $mem_{col} \leftarrow [\widetilde{\text{eq}}(0, r_y), \ldots, \widetilde{\text{eq}}(m-1, r_y)] \in \mathbb{F}^m$
  - Let $Init_{col} \leftarrow H_{\gamma_1}([0, \ldots, m-1], mem_{col}, [0, \ldots, 0]) \in \mathbb{F}^m$
  - Let $RS_{col} \leftarrow H_{\gamma_1}(col, e_{col}, read\text{-}ts_{col}) \in \mathbb{F}^n$
  - Let $WS_{col} \leftarrow H_{\gamma_1}(col, e_{col}, write\text{-}ts_{col}) \in \mathbb{F}^n$
  - Let $Audit_{col} \leftarrow H_{\gamma_1}([0, \ldots, m-1], mem_{col}, audit\text{-}ts_{col}) \in \mathbb{F}^m$
  - Assert $\mathcal{H}_{\gamma_2}(Init_{col}) \cdot \mathcal{H}_{\gamma_2}(WS_{col}) = \mathcal{H}_{\gamma_2}(RS_{col}) \cdot \mathcal{H}_{\gamma_2}(Audit_{col})$

Definitions for H and $\mathcal{H}$. The above description of the circuit uses H and $\mathcal{H}$, which we now define. Unlike ECC-based multiset hash functions in Spice [93], we leverage the public-coin nature of Hyrax to verify the multiset relationship at a random point chosen using public coins. Specifically, we define two hash functions: (1) $h_\gamma : \mathbb{F}^3 \to \mathbb{F}$; and (2) $\mathcal{H}_\gamma : \mathbb{F}^* \to \mathbb{F}$, where $\mathbb{F}^*$ denotes a multiset with elements from $\mathbb{F}$ and $\gamma \in_R \mathbb{F}$.

$$h_\gamma(a, v, t) = a \cdot \gamma^2 + v \cdot \gamma + t$$

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

Given $(A, V, T) \in (\mathbb{F}^\ell, \mathbb{F}^\ell, \mathbb{F}^\ell)$ for $\ell > 0$, we define a map $H_\gamma : (\mathbb{F}^\ell, \mathbb{F}^\ell, \mathbb{F}^\ell) \to \mathbb{F}^\ell$:

$$H_\gamma(A, V, T) = [h_\gamma(A[0], V[0], T[0]), \ldots, h_\gamma(A[\ell-1], V[\ell-1], T[\ell-1])]$$

Lemma 7.2. *For any two pairs $(a_1, v_1, t_1) \in \mathbb{F}^3$ and $(a_2, v_2, t_2) \in \mathbb{F}^3$, $\Pr_\gamma\{h_\gamma(a_1, v_1, t_1) = h_\gamma(a_2, v_2, t_2) | (a_1, v_1, t_1) \neq (a_2, v_2, t_2)\} \leq 3/|\mathbb{F}|$.*

Proof. This follows from the Schwartz-Zippel lemma. □

Lemma 7.3. *For any $\ell > 0$, $(A_1, V_1, T_1) \in (\mathbb{F}^\ell, \mathbb{F}^\ell, \mathbb{F}^\ell)$ and $(A_2, V_2, T_2) \in (\mathbb{F}^\ell, \mathbb{F}^\ell, \mathbb{F}^\ell)$* $\Pr_\gamma\{\exists i :: H_\gamma(A_1, V_1, T_1)[i] = H_\gamma(A_2, V_2, T_2)[i] | (A_1, V_1, T_1) \neq (A_2, V_2, T_2)\} \leq 3 \cdot \ell/|\mathbb{F}|$.

Proof. This follows from a standard union bound with the result of the lemma 7.2. □

Lemma 7.4. *For any two multisets $\mathcal{M}_1, \mathcal{M}_2$ of size ℓ over $\mathbb{F}$,*

$$\Pr_\gamma\{\mathcal{H}_\gamma(\mathcal{M}_1) = \mathcal{H}_\gamma(\mathcal{M}_2) | \mathcal{M}_1 \neq \mathcal{M}_2\} \leq \ell/|\mathbb{F}|$$

Proof. This follows from the Schwartz-Zippel lemma. □

Lemma 7.5. *Assuming that $|\mathbb{F}|$ is exponential in λ and $n = O(\lambda)$, for any $2\log m$-variate multilinear polynomial $\widetilde{M}$ whose dense representation is of size at most n and for any given $e_{row}, e_{col} \in \mathbb{F}^n$,*

$$\Pr_{\gamma_1,\gamma_2}\{Circuit_{eval\text{-}opt}(w, (\gamma_1, \gamma_2), r) = v | \widetilde{M}(r) \neq v\} \leq \mathtt{negl}(\lambda),$$

where $w = (row, col, val, e_{row}, e_{col}, \mathsf{MemoryInTheHead}(m, n, row), \mathsf{MemoryInTheHead}(m, n, col))$ and (row, col, val) denotes the dense representation of $\widetilde{M}$.

Proof. This follows from the soundness of the memory checking primitive [30] and the collision-resistance of the underlying hash functions used (lemmas 7.4 and 7.3). □

Construction. Given an extractable polynomial commitment scheme PC for multilinear polynomials, we build a scheme for sparse multilinear polynomials as follows.

$\underline{\mathsf{PC}^{\mathrm{SPARK}}}$:

- $pp \leftarrow \mathsf{Setup}(1^\lambda, \mu, n)$: $(\mathsf{PC.Setup}(1^\lambda, \mu)), \mathsf{PC.Setup}(1^\lambda, \log(n)))$
- $(\mathcal{C}; \mathcal{S}) \leftarrow \mathsf{Commit}(pp; \widetilde{M})$:
  - Let $(pp_m, pp_n) \leftarrow pp$
  - Let (row, col, val) denote the dense representation of $\widetilde{M}$ as described in text.
    - $(\mathcal{C}_{row}, \mathcal{S}_{row}) \leftarrow \mathsf{PC.Commit}(pp_n, \widetilde{row})$
    - $(\mathcal{C}_{col}, \mathcal{S}_{col}) \leftarrow \mathsf{PC.Commit}(pp_n, \widetilde{col})$
    - $(\mathcal{C}_{val}, \mathcal{S}_{val}) \leftarrow \mathsf{PC.Commit}(pp_n, \widetilde{val})$
  - Let $(read\text{-}ts_{row}, write\text{-}ts_{row}, audit\text{-}ts_{row}) \leftarrow \mathsf{MemoryInTheHead}(2^{\mu/2}, n, row)$
    - $(\mathcal{C}_{read\text{-}ts_{row}}, \mathcal{S}_{read\text{-}ts_{row}}) \leftarrow \mathsf{PC.Commit}(pp_n, \widetilde{read\text{-}ts_{row}})$
    - $(\mathcal{C}_{write\text{-}ts_{row}}, \mathcal{S}_{write\text{-}ts_{row}}) \leftarrow \mathsf{PC.Commit}(pp_n, \widetilde{write\text{-}ts_{row}})$
    - $(\mathcal{C}_{audit\text{-}ts_{row}}, \mathcal{S}_{audit\text{-}ts_{row}}) \leftarrow \mathsf{PC.Commit}(pp_m, \widetilde{audit\text{-}ts_{row}})$
  - Let $(read\text{-}ts_{col}, write\text{-}ts_{col}, audit\text{-}ts_{col}) \leftarrow \mathsf{MemoryInTheHead}(2^{\mu/2}, n, col)$
    - $(\mathcal{C}_{read\text{-}ts_{col}}, \mathcal{S}_{read\text{-}ts_{col}}) \leftarrow \mathsf{PC.Commit}(pp_n, \widetilde{read\text{-}ts_{col}})$

- $(\mathcal{C}_{\text{write-ts}_{col}}, \mathcal{S}_{\text{write-ts}_{col}}) \leftarrow \mathsf{PC.Commit}(pp_n, \widetilde{\text{write-ts}_{col}})$
- $(\mathcal{C}_{\text{audit-ts}_{col}}, \mathcal{S}_{\text{audit-ts}_{col}}) \leftarrow \mathsf{PC.Commit}(pp_m, \widetilde{\text{audit-ts}_{col}})$

- Let $\mathcal{C} \leftarrow (\mathcal{C}_{row}, \mathcal{C}_{col}, \mathcal{C}_{val}, \mathcal{C}_{\text{read-ts}_{row}}, \mathcal{C}_{\text{write-ts}_{row}}, \mathcal{C}_{\text{audit-ts}_{row}}, \mathcal{C}_{\text{read-ts}_{col}}, \mathcal{C}_{\text{write-ts}_{col}}, \mathcal{C}_{\text{audit-ts}_{col}})$
- Let $\mathcal{S} \leftarrow (\mathcal{S}_{row}, \mathcal{S}_{col}, \mathcal{S}_{val}, \mathcal{S}_{\text{read-ts}_{row}}, \mathcal{S}_{\text{write-ts}_{row}}, \mathcal{S}_{\text{audit-ts}_{row}}, \mathcal{S}_{\text{read-ts}_{col}}, \mathcal{S}_{\text{write-ts}_{col}}, \mathcal{S}_{\text{audit-ts}_{col}})$
- Output $(\mathcal{C}, \mathcal{S})$

- $b \leftarrow \mathsf{Open}(pp, \mathcal{C}, \widetilde{M}, \mathcal{S})$:
  - Let $(pp_m, pp_n) \leftarrow pp$.
  - Let *row*, *col*, *val* denote dense representation of $\widetilde{M}$ as defined above.
  - Output $\mathsf{PC.Open}(pp_n, \mathcal{C}.\mathcal{C}_{row}, \widetilde{row}, \mathcal{S}.\mathcal{S}_{row}) \wedge \mathsf{PC.Open}(pp_n, \mathcal{C}.\mathcal{C}_{col}, \widetilde{col}, \mathcal{S}.\mathcal{S}_{col}) \wedge \mathsf{PC.Open}(pp_n, \mathcal{C}, \mathcal{C}_{val}, \widetilde{val}, \mathcal{S}.\mathcal{S}_{val})$
- $b \leftarrow \mathsf{Eval}(pp, \mathcal{C}, r, v, \mu, n; \widetilde{M}, \mathcal{S})$:
  - Let $(pp_m, pp_n) \leftarrow pp$ and let $(r_x, r_y) = r$, where $r_x, r_y \in \mathbb{F}^{\mu/2}$.
  - Let *row*, *col*, *val* denote dense representation of $\widetilde{M}$ as defined above.
  - $\mathcal{P}$:
    - Compute e_{row} and e_{col} with $2n$ lookups over a table of size $m = 2^{\mu/2}$. That is, $e_{row} = [\widetilde{\text{eq}}(row(0), r_x), \ldots, \widetilde{\text{eq}}(row(n-1), r_x)]$; let $e_{col} = [\widetilde{\text{eq}}(col(0), r_y), \ldots, \widetilde{\text{eq}}(col(n-1), r_y)]$.
    - $(\mathcal{C}_{e_{row}}, \mathcal{S}_{e_{row}}) \leftarrow \mathsf{PC.Commit}(pp_n, \widetilde{e_{row}})$; send $\mathcal{C}_{e_{row}}$ to $\mathcal{V}$.
    - $(\mathcal{C}_{e_{col}}, \mathcal{S}_{e_{col}}) \leftarrow \mathsf{PC.Commit}(pp_n, \widetilde{e_{col}})$; send $\mathcal{C}_{e_{col}}$ to $\mathcal{V}$.
  - $\mathcal{V} : (\gamma_1, \gamma_2) \in_R \mathbb{F}^2$. Send (γ_1, γ_2) to $\mathcal{P}$.
  - $\mathcal{P}$ and $\mathcal{V}$ use Hyrax (with PC as the extractable polynomial commitment scheme) to verify the claim that $\widetilde{M}(r) = v$ using $Circuit_{\text{eval-opt}}$.

Analysis of costs. $Circuit_{\text{eval-opt}}$ is uniform because computing $\mathcal{H}$ using a binary tree of multiplications [98] constitutes nearly all of the work in the above circuit. Figure 7 depicts the costs of $\mathsf{PC}^{\text{SPARK}}$ with different choices for PC.

PC choice	setup	$\mathcal{P}_{\text{Eval}}$	$\lvert\mathcal{C}\rvert$	communication	$\mathcal{V}_{\text{Eval}}$
Hyrax-PC [103]	public	$O(n)$	$O(\sqrt{n})$	$O(\log^2 n)$	$O(\sqrt{n})$
DARK-CL [38]	public	$O(n)$	$O(\log n)$	$O(\log^2 n)$	$O(\log^2 n)$
vSQL-VPD [111]	private*	$O(n)$	$O(1)$	$O(\log^2 n)$	$O(\log^2 n)$
Virgo-VPD [109]	public	$O(n \log n)$	$O(1)$	$O(\log^2 n)$	$O(\log^2 n)$

Figure 7—Costs of $\mathsf{PC}^{\text{SPARK}}$ with different choices for PC. Here, n is number of entries in the dense representation of the multilinear polynomial.

Lemma 7.6. *Assuming that $\mathsf{PC}^{\text{SPARK}}.\mathsf{Commit}$ is run by an honest entity, then $\mathsf{PC}^{\text{SPARK}}$ is a polynomial commitment scheme for multilinear polynomials with the costs noted.*

Proof. Completeness follows from the completeness of PC, Hyrax, and $Circuit_{\text{eval-opt}}$. Binding follows from the uniqueness of the dense representation of the sparse multi-

linear polynomial as (row, col, val). Knowledge soundness follows from the witness-extended emulation offered by Hyrax and PC, and from the negligible soundness error of $Circuit_{\text{eval-opt}}$ (lemma 7.5). Finally, the claimed costs follow from the cost model of Hyrax and of PC applied to a constant number of $O(\log n)$-variate multilinear polynomials. □

7.3 Optimizations

We now describe many optimizations to SPARK to reduce constants.

1. Instead of using Hyrax as a black box, we tailor it for $Circuit_{\text{eval-opt}}$ using prior ideas [98]. This reduces overall costs significantly. We also do not need Hyrax's zero-knowledge compiler for computation commitments.
2. For computation commitments, we build a single circuit that produces evaluations of $\widetilde{A}, \widetilde{B}, \widetilde{C}$ at (r_x, r_y). This enables reusing parts of the memory checking circuit (related to the state of the memory) across evaluations.
3. In our particular context, we can set $\forall 0 \leq i < n$: $\textit{write-ts}_{row}[i] = \textit{read-ts}_{row}[i] + 1$ and $\textit{write-ts}_{col}[i] = \textit{read-ts}_{read}[i] + 1$. This is because unlike the traditional setting of offline memory checking, the read timestamps are not untrusted. This avoids having to commit to $\widetilde{\textit{write-ts}_{row}}$ and $\widetilde{\textit{write-ts}_{col}}$.
4. During $\mathsf{PC}^{\text{SPARK}}.\mathsf{Eval}$, at the witness layer in Hyrax, $\mathcal{V}$ needs to evaluate a number of multilinear polynomials at either $r_{row}, r_{col} \in \mathbb{F}^{\log n}$ or $r_{mem} \in \mathbb{F}^{\log m}$. We avoid having to commit to them by leveraging their succinct representations.
   - $\mathcal{V}$ can compute $\widetilde{mem}_{row}(r_{row})$ and $\widetilde{mem}_{col}(r_{col})$ in $O(\log m)$ as follows:

     $$\widetilde{mem}_{row}(r_{row}) \leftarrow \widetilde{\text{eq}}(r_{row}, r_x)$$

     $$\widetilde{mem}_{col}(r_{col}) \leftarrow \widetilde{\text{eq}}(r_{col}, r_y)$$

     This avoids $\mathcal{P}$ having to commit to $\widetilde{mem}_{row}$ and $\widetilde{mem}_{col}$. It also avoids $\mathcal{V}$ having to verify if the commitments correctly represent $\widetilde{\text{eq}}(i, r_x)$ and $\widetilde{\text{eq}}(i, r_y)$ $\forall 0 \leq i < m$, which in turn would require verifying the satisfiability of another circuit.
   - We leverage the following facts: (1) $\widetilde{(0, 1, \ldots, m-1)}(r_{mem}) = \sum_{i=0}^{\log m} 2^i \cdot r_{mem}[i]$; (2) $\widetilde{(0, 0, \ldots, 0)}(r_{mem}) = 0$.
5. It is possible to combine k μ-variate multilinear polynomials into a single multilinear polynomial over $\mu + \log k$ variables. We employ this technique to reduce the number of committed multilinear polynomials from 23 to 3.

8 Implementation and experimental evaluation

8.1 Implementation details

We implement Spartan as a modular library in about 8,000 lines of Rust including many optimizations listed throughout the paper as well as optimizations to the sum-check protocol from prior work [98, 101, 103, 105, 108]. Our codebase can leverage multiple CPU cores using `rayon`.[8] We also implement SPARK-naive to demonstrate benefits of

[8] https://github.com/rayon-rs/rayon

SPARK (in our experiments we find SPARK to be faster by about $10\times$ than SPARK-naive for R1CS instances with 2^{20} constraints).

Below, we present results from SPARK instantiated with Hyrax-PC [105], that is, we evaluate a zkSNARK whose security holds under the discrete logarithm problem (Spartan$_{\text{DL}}$). For curve arithmetic, we use `curve25519-dalek` [2], which offers an efficient implementation of a prime-order Ristretto group [3, 70] called `ristretto255`. The scalar arithmetic in the library is however slow since it represents the underlying scalar elements as byte strings (to facilitate fast curve arithmetic). To cope with this, we optimize the underlying scalar arithmetic by $\approx 10\times$ by adapting code from `bls12-381`,[9] which offers optimized 256-bit arithmetic for the field of scalars of the BLS12-381 curve, to the field of scalars of `ristretto255`.

We report results from Spartan$_{\text{DL}}$ rather than Spartan$_{\text{CL}}$ because the latter requires operations over class groups, which are orders of magnitude slower than operations over a prime-order elliptic curve group needed by Hyrax-PC [105]. As we show below, despite $O(\sqrt{n})$-sized proofs, Spartan$_{\text{DL}}$ offers proofs (and verification times) that are shorter (and smaller) than nearly all other schemes.

8.2 Metrics, methodology, and testbed

Our principal evaluation metrics are: (1) $\mathcal{P}$'s costs to produce a proof; (2) $\mathcal{V}$'s costs to preprocess an R1CS instance; (3) $\mathcal{V}$'s costs to verify a proof; and (4) the size of a proof. We measure $\mathcal{P}$'s and $\mathcal{V}$'s costs using a real-time clock and the size of proofs in bytes by serializing proof data structures using `serde`.[10]

We experiment with Spartan and several baselines (listed below) on a machine with the following configuration: Intel Xeon CPU E3-1280 v5 3.70 GHz with 64 GB RAM running Ubuntu 18.04 (on Windows 10). We report results from a single-threaded configuration since not all our baselines leverage multiple cores. As with prior work [20], 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. In all Spartan experiments $|io| = 10$.

Baselines. We compare Spartan with five state-of-the-art zkSNARKs.

1. Groth16 [67], the most efficient zkSNARK with trusted setup based on GGPR [61].
2. Ligero [5], a prior transparent zkSNARK with a light-weight prover.
3. Aurora [20], a prior transparent zkSNARK with short proofs.
4. Fractal [47], a recent transparent zkSNARK that uses a technique analogous to computation commitments to achieve sub-linear verification costs.
5. Hyrax [105], a prior transparent zkSNARK that achieves sub-linear verification costs for data-parallel computations.

For Groth16, we benchmark its implementation from `libsnark` with `bn128` curve.[11] For Hyrax, we use its reference implementation with curve25519.[12] For Ligero, Aurora, and Fractal, we use their implementations from `libiop` with a prime field of size

[9] `https://github.com/zkcrypto/bls12_381`
[10] https://github.com/serde-rs/serde
[11] https://github.com/scipr-lab/libsnark/
[12] https://github.com/hyraxZK/fennel

$\approx 2^{256}$.[13] We choose these fields and/or curves to offer an apples-to-apples comparison with Spartan (in terms of the expressiveness of the underlying computational model), which uses the Ristretto group where the field size is $\approx 2^{256}$.

Additional baselines. There are other transparent zkSNARKs in the literature (§1). We now provide a rough comparison with Spartan and leave it to future work to run them on the same hardware. Bulletproofs [41] offers shorter proofs ($\approx$ 1.5 KB) than all transparent zkSNARKs discussed here, but it incurs orders of magnitude higher prover and verifier costs than Spartan and other schemes discussed here [108, Table 1]. SuperSonic [38] estimates that its proof size is $\approx$ 10 KB and verification time is $\approx$100 ms for circuits with 2^{20} gates, but as discussed earlier, there are no reports of prover's concrete costs and our benchmarks suggest that the prover's costs can be $\approx 200\times$ higher than Spartan$_{\text{DL}}$. zkSTARKs [15] perform worse than Aurora on all our performance aspects for arbitrary R1CS instances: prover time, verifier time, and proof sizes [15, §11.2]. Virgo [109] is specialized to layered arithmetic circuits over $\mathbb{F}_p$ where $p = 2^{61} - 1$, so it precludes many computations including efficient representations of ECC-based digital signatures (or other cryptographic primitives) that operate over fields of size $\approx 2^{256}$ [1, 39, 40, 80, 88, 107]. Switching to a 256-bit prime field increases Virgo's prover's costs by at least 17×, which is much slower than Spartan.

A note about comparison with Hyrax. Hyrax's model of computation is layered arithmetic circuits and its codebase only supports three circuits. To offer an approximate comparison with Spartan, we take the following approach. We use the matrix multiplication benchmark, which allows the prover to prove that it knows two secret $\ell \times \ell$ matrices such that their product is a public matrix. We choose this benchmark because the R1CS instances that we generate for other baselines perform n multiplications using n constraints. Given that $\ell \times \ell$ matrix multiplication requires ℓ^3 constraints in R1CS [4], when we experiment with other systems for a given R1CS instance with n constraints, we set the dimension of the matrix for Hyrax as $\ell = n^{1/3}$. This is optimistic for Hyrax's prover: in all other systems, the prover's cryptographic operations is $O(n)$ whereas for this benchmark the Hyrax prover only performs cryptographic operations proportional to the size of the witness to the layered circuit i.e., $O(n^{2/3})$ operations for this benchmark. On the other hand, this is somewhat pessimistic for Hyrax's verifier: the public *io* in the case of Hyrax is ℓ^2 elements of $\mathbb{F}$ whereas for other systems, it is nearly 0. At 2^{20} constraints, we estimate that this discrepancy underestimates Spartan's verifier's performance by 5 ms of CPU-time (which is $< 3.8\%$ of Spartan's verifier's time).

Spartan variants and amortization. For Spartan, we report results from two variants: the SNARK (Spartan-snark) and NIZK (Spartan-nizk). The former incurs sub-linear verification costs and the latter incurs linear verification costs. We report the performance of the NIZK variant for two reasons: (1) Two of our baselines (Aurora and Ligero) offer only a linear-time verifier (so this helps offer an apples-to-apples comparison); and (2) for data-parallel workloads, the NIZK variant depicts the performance that Spartan-snark can achieve for the prover and proof sizes (this is because Spartan-snark can amortize the costs of computation commitments across different data-parallel units to approach the performance of Spartan-nizk for the prover and proof sizes).

[13] https://github.com/scipr-lab/libiop

8.3 Performance results

Prover. Figure 8 depicts the prover's costs under Spartan and its baselines. As expected, Spartan outperforms all its baselines. Specifically, Spartan-snark is faster than its baselines by 1.5×–17.6×, and Spartan-nizk by 12.4×–115×.

When compared to the most related system, Spartan-snark is 17.5× faster than Fractal at 2^{19} constraints.[14] Even compared to Ligero, a transparent SNARK with an efficient prover, Spartan-snark is up to 3× faster. When we compare Ligero, Aurora, and Hyrax with Spartan-nizk (since all of them are proof-succinct NIZKs),[15] Spartan-nizk is up to 24× faster than Ligero, 115× faster than Aurora, and 82× faster than Hyrax. Finally, compared to Groth16 (the state-of-the-art in SNARKs with trusted setup), Spartan's prover is 1.5× faster and Spartan-nizk's prover is 12.4× faster.

	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}	2^{18}	2^{19}	2^{20}
Groth16	0.2	0.3	0.5	1	1.6	3.1	5.8	11.3	21.4	42	81
Aurora	0.5	1.1	2.2	4.4	9.0	19.6	40.5	84.3	175.3	363.4	752.6
Ligero	0.3	0.6	1.1	2.0	4.0	6.6	11.7	21.5	40.4	79.0	156.7
Fractal	0.6	1.2	2.5	5.3	11.8	25.0	51.4	105.8	218.9	447.2	–
Hyrax	0.7	0.8	1.6	3.2	7.6	14.6	26.2	55.4	118.5	246	534.6
Spartan-nizk	0.01	0.02	0.04	0.07	0.14	0.24	0.52	0.95	1.73	3.18	6.5
Spartan-snark	0.1	0.2	0.3	0.6	1.1	2.0	3.7	7.2	12.9	25.4	52

Figure 8—Prover's performance (in seconds) for varying R1CS instance sizes under different schemes.

Proof sizes. Figure 9 depicts proof sizes under Spartan and its baselines. Although Spartan-snark's proofs are asymptotically larger than Fractal (Figure 1), Spartan-snark offers up to 2× shorter proofs. When we compare the proof-succinct NIZKs, Spartan-nizk offers proofs that are 1.8–540× shorter than Hyrax, Aurora, and Ligero. All transparent zkSNARKs produce orders of magnitude longer proofs than Groth16.

	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}	2^{18}	2^{19}	2^{20}
Aurora	52.1	57.4	70.0	74.6	82.7	95.7	101.7	108.7	119.3	131.1	141
Ligero	550	627	1M	1.2M	2M	3M	5.6M	5.8M	10M	10.6M	20M
Fractal	125	136	149	162	171	191	202	216	232	249	–
Hyrax	23.3	25.1	26.4	33.8	36.9	37.7	46.5	49.9	52.2	61.2	65.9
Spartan-nizk	3.3	3.6	4.7	4.9	7.2	7.4	11.7	11.9	20.2	20.5	37.0
Spartan-snark	29.9	35.5	40	47.9	53.9	66.3	74.6	95.6	108.4	146.2	167.6

Figure 9—Proof sizes in KBs for various zkSNARKs. Entries with "M" are in megabytes. The proof sizes under Groth16 [67] is 128 bytes for all instance sizes.

Verifier. Figure 10 depicts the verifier times under different schemes. Groth16 offers the fastest verifier, but it requires a trusted setup. Among transparent zkSNARKs, Fractal offers the fastest verifier, which is 7.2× faster than Spartan-snark at 2^{19} constraints. However, arguably, Spartan-snark's verification times are concretely efficient for many

[14] Unfortunately, we could not run the Fractal prover at 2^{20} constraints.

[15] Hyrax incurs linear-time verification costs if the computation has no data-parallelism.

applications. Furthermore, Spartan-snark's verifier is 71× faster than Aurora, 240× faster than Ligero, and 60× faster than Hyrax. This type of performance is expected because Aurora, Ligero, and Hyrax incur linear costs for the verifier whereas Spartan-snark (and Fractal) incur sub-linear verification costs due to the use of computation commitments, which requires preprocessing the non-*io* component of an R1CS instance. We quantify the costs of that process below. Finally, note that Spartan-snark beats Spartan-nizk for verification costs at roughly 2^{18} constraints since the former incurs $O(\sqrt{n})$ costs whereas the latter incurs $O(n)$ costs, where n is the size of the R1CS instance.

	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}	2^{18}	2^{19}	2^{20}
Aurora	12.1	20	35.7	68.5	130	262	542	1.1s	2.2s	4.5s	9.5s
Ligero	45.6	90	161	335.3	627.5	1.0s	1.9s	3.6s	7.1s	14.5s	32s
Fractal	7.5	7.7	9.0	9.0	9.9	10.2	10.8	11.7	11.8	13.5	–
Hyrax	285	303	331	507	653	737	963	1.5s	2.3s	4.2s	8.1s
Spartan-nizk	1.7	2.1	3.3	4.7	8.4	14.4	27.4	45.9	90.0	201.1	439.5
Spartan-snark	12.0	12.5	15.6	19.5	25.6	33.0	44.0	56.8	73.8	97.7	133.0

Figure 10—Verifier's performance (in ms) under different schemes. Entries with "s" are in seconds. The verifier under Groth16 [67] takes ≈ 2 ms at all instance sizes.

Encoder. Figure 11 depicts the cost to the verifier to preprocess an R1CS instance (without the *io* component) under Spartan-snark, Fractal [47], and Groth16 [67]. We do not depict other baselines because they do not require any preprocessing. Spartan-snark's encoder is up to 26× faster than Fractal's encoder and about 3.5× faster than the trusted setup for Groth16 at the largest instance sizes.

	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}	2^{18}	2^{19}	2^{20}
Groth16	0.14	0.25	0.5	0.8	1.6	3.1	5.9	11	20	39.7	74
Fractal	0.25	0.55	1.2	2.6	5.8	12.7	27.3	56.6	117.2	242.1	562.2
Spartan-snark	0.08	0.1	0.18	0.31	0.61	1.1	2.13	3.3	6.7	10.3	20.8

Figure 11—Encoder's performance (in seconds) for varying R1CS instance sizes under different schemes. For Groth16 [67], we depict the cost of trusted setup, which preprocesses a given R1CS instance to generate keys for the prover and the verifier. For Spartan and Fractal, the preprocessing is untrusted and public computation.

Acknowledgments

Comments from Sebastian Angel, Melissa Chase, Ben Fisch, Esha Ghosh, Jonathan Lee, Satya Lokam, and Bryan Parno helped improve this draft. We thank Justin Thaler, Riad Wahby, and Michael Walfish for their detailed attention and thorough comments, which helped clarify several aspects of this work. Special thanks to Jonathan Lee for providing insights into the concrete performance of operations over class groups.

References

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

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

[3] The Ristretto group. `https://ristretto.group/`.

[4] Pequin: An end-to-end toolchain for verifiable computation, SNARKs, and probabilistic proofs. `https://github.com/pepper-project/pequin`, 2016.

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

[6] A. Arasu, K. Eguro, R. Kaushik, D. Kossmann, P. Meng, V. Pandey, and R. Ramamurthy. Concerto: A high concurrency key-value store with integrity. 2017.

[7] 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.

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

[9] S. Arora and M. Sudan. Improved low-degree testing and its applications. *Combinatorica*, 23(3):365–426, 2003.

[10] L. Babai. Trading group theory for randomness. In *STOC*, pages 421–429, 1985.

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

[12] L. Babai, L. Fortnow, and C. Lund. Non-deterministic exponential time has two-prover interactive protocols. *Computational Complexity*, 2(4), Dec. 1992.

[13] M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *CCS*, pages 62–73, 1993.

[14] M. Ben-Or, O. Goldreich, S. Goldwasser, J. Håstad, J. Kilian, S. Micali, and P. Rogaway. Everything provable is provable in zero-knowledge. In *CRYPTO*, pages 37–56, 1988.

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

[16] 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.

[17] E. Ben-Sasson, A. Chiesa, D. Genkin, and E. Tromer. Fast reductions from RAMs to delegatable succinct constraint satisfaction problems: Extended abstract. In *ITCS*, 2013.

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

[19] 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.

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

[21] E. Ben-Sasson, A. Chiesa, and N. Spooner. Interactive Oracle Proofs. In *TCC*, 2016.

[22] E. Ben-Sasson, A. Chiesa, E. Tromer, and M. Virza. Scalable zero knowledge via cycles of elliptic curves. In *CRYPTO*, 2014.

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

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

[25] E. Ben-Sasson and M. Sudan. Simple PCPs with poly-log rate and query complexity. In *STOC*, pages 266–275, 2005.

[26] E. Ben-Sasson and M. Sudan. Short PCPs with polylog query complexity. *SIAM J.*

Comput., 38(2):551–607, May 2008.

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

[28] N. Bitansky and A. Chiesa. Succinct arguments from multi-prover interactive proofs and their efficiency benefits. In *CRYPTO*, 2012.

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

[30] M. Blum, W. Evans, P. Gemmell, S. Kannan, and M. Naor. Checking the correctness of memories. In *FOCS*, 1991.

[31] M. Blum, P. Feldman, and S. Micali. Non-interactive zero-knowledge and its applications (extended abstract). In *STOC*, pages 103–112, 1988.

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

[33] D. Boneh, E. Boyle, H. Corrigan-Gibbs, N. Gilboa, and Y. Ishai. Zero-knowledge proofs on secret-shared data via fully linear PCPs. ePrint Report 2019/188, 2019.

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

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

[36] G. Brassard, D. Chaum, and C. Crépeau. Minimum disclosure proofs of knowledge. *J. Comput. Syst. Sci.*, 37(2):156–189, Oct. 1988.

[37] B. Braun, A. J. Feldman, Z. Ren, S. Setty, A. J. Blumberg, and M. Walfish. Verifying computations with state. In *SOSP*, 2013.

[38] B. Bunz, B. Fisch, and A. Szepieniec. Transparent SNARKs from DARK compilers. ePrint Report 2019/1229, 2019.

[39] V. Buterin. On-chain scaling to potentially 500 tx/sec through mass tx validation. `https://ethresear.ch/t/on-chain-scaling-to-potentially-500-tx-sec-through-mass-tx-validation/3477`, Sept. 2018.

[40] V. Buterin. The dawn of hybrid layer 2 protocols. `https://vitalik.ca/general/2019/08/28/hybrid_layer_2.html`, Aug. 2019.

[41] 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.

[42] M. Campanelli, D. Fiore, and A. Querol. Legosnark: Modular design and composition of succinct zero-knowledge proofs. ePrint Report 2019/142, 2019.

[43] R. Canetti, B. Riva, and G. N. Rothblum. Two protocols for delegation of computation. In *ICITS*, pages 37–61, 2012.

[44] J. P. M. Chase. ZSL Proof of Concept. `https://github.com/jpmorganchase/quorum/wiki/ZSL`, 2017.

[45] A. Chiesa, M. A. Forbes, and N. Spooner. A zero knowledge sumcheck and its applications. *CoRR*, abs/1704.02086, 2017.

[46] 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.

[47] A. Chiesa, D. Ojha, and N. Spooner. Fractal: Post-quantum and transparent recursive proofs from holography. ePrint Report 2019/1076, 2019.

[48] A. Chiesa, E. Tromer, and M. Virza. Cluster computing in zero knowledge. In *EUROCRYPT*, 2015.

[49] D. Clarke, S. Devadas, M. V. Dijk, B. Gassend, G. Edward, and S. Mit. Incremental multiset hash functions and their application to memory integrity checking. In *ASIACRYPT*, 2003.

[50] G. Cormode, M. Mitzenmacher, and J. Thaler. Practical verified computation with

streaming interactive proofs. In *ITCS*, 2012.
[51] C. Costello, C. Fournet, J. Howell, M. Kohlweiss, B. Kreuter, M. Naehrig, B. Parno, and S. Zahur. Geppetto: Versatile verifiable computation. In *S&P*, May 2015.
[52] R. Cramer and I. Damgård. Zero-knowledge proofs for finite field arithmetic, or: Can zero-knowledge be for free? In *CRYPTO*, pages 424–441, 1998.
[53] 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.
[54] I. Dinur. The PCP theorem by gap amplification. *J. ACM*, 54(3), June 2007.
[55] C. Dwork, M. Naor, G. N. Rothblum, and V. Vaikuntanathan. How efficient can memory checking be? In *TCC*, 2009.
[56] U. Feige, S. Goldwasser, L. Lovász, S. Safra, and M. Szegedy. Interactive proofs and the hardness of approximating cliques. *J. ACM*, 43(2):268–292, Mar. 1996.
[57] A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *CRYPTO*, pages 186–194, 1986.
[58] D. Fiore, C. Fournet, E. Ghosh, M. Kohlweiss, O. Ohrimenko, and B. Parno. Hash first, argue later: Adaptive verifiable computations on outsourced data. In *CCS*, 2016.
[59] A. Gabizon, Z. J. Williamson, and O. Ciobotaru. Plonk: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. ePrint Report 2019/953, 2019.
[60] R. Gennaro, C. Gentry, and B. Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In *CRYPTO*, 2010.
[61] R. Gennaro, C. Gentry, B. Parno, and M. Raykova. Quadratic span programs and succinct NIZKs without PCPs. In *EUROCRYPT*, 2013.
[62] C. Gentry. *A fully homomorphic encryption scheme*. PhD thesis, Stanford University, 2009.
[63] C. Gentry and D. Wichs. Separating succinct non-interactive arguments from all falsifiable assumptions. In *STOC*, pages 99–108, 2011.
[64] S. Goldwasser, Y. T. Kalai, and G. N. Rothblum. Delegating computation: Interactive proofs for muggles. In *STOC*, 2008.
[65] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof-systems. In *STOC*, 1985.
[66] J. Groth. Short pairing-based non-interactive zero-knowledge arguments. In *ASIACRYPT*, 2010.
[67] J. Groth. On the size of pairing-based non-interactive arguments. In *EUROCRYPT*, 2016.
[68] J. Groth and Y. Ishai. Sub-linear zero-knowledge argument for correctness of a shuffle. In *EUROCRYPT*, 2008.
[69] J. Groth, M. Kohlweiss, M. Maller, S. Meiklejohn, and I. Miers. Updatable and universal common reference strings with applications to zk-SNARKs. In *CRYPTO*, 2018.
[70] M. Hamburg. Decaf: Eliminating cofactors through point compression. In *CRYPTO*, 2015.
[71] J. Håstad. Some optimal inapproximability results. In *STOC*, pages 1–10, 1997.
[72] Y. Ishai, E. Kushilevitz, and R. Ostrovsky. Efficient arguments without short PCPs. In *Computational Complexity*, 2007.
[73] Y. Ishai, E. Kushilevitz, R. Ostrovsky, and A. Sahai. Zero-knowledge from secure multiparty computation. In *STOC*, pages 21–30, 2007.
[74] Y. T. Kalai. Delegating computation: A new perspective. A workshop at STOC 2017 on Probabilistically checkable and interactive proofs (PCP/IP): Between theory and practice, June 2017.
[75] Y. T. Kalai and R. Raz. Interactive PCP. In *ICALP*, pages 536–547, 2008.
[76] A. Kate, G. M. Zaverucha, and I. Goldberg. Constant-size commitments to polynomials and their applications. In *ASIACRYPT*, pages 177–194, 2010.
[77] J. Kilian. A note on efficient zero-knowledge proofs and arguments (extended abstract). In

STOC, 1992.

[78] 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.

[79] A. Kosba, C. Papamanthou, and E. Shi. xJsnark: A framework for efficient verifiable computation. In *S&P*, 2018.

[80] J. Lee, K. Nikitin, and S. Setty. Replicated state machines without replicated execution. In *S&P*, 2020.

[81] libsnark. A C++ library for zkSNARK proofs. `https://github.com/scipr-lab/libsnark`, 2012.

[82] H. Lipmaa. Progression-free sets and sublinear pairing-based non-interactive zero-knowledge arguments. In *TCC*, 2012.

[83] C. Lund, L. Fortnow, H. Karloff, and N. Nisan. Algebraic methods for interactive proof systems. In *FOCS*, Oct. 1990.

[84] M. Maller, S. Bowe, M. Kohlweiss, and S. Meiklejohn. Sonic: Zero-knowledge SNARKs from linear-size universal and updateable structured reference strings. ePrint Report 2019/099, 2019.

[85] R. C. Merkle. A digital signature based on a conventional encryption function. In *CRYPTO*, 1988.

[86] S. Micali. CS proofs. In *FOCS*, 1994.

[87] D. Moshkovitz and R. Raz. Sub-constant error low degree test of almost-linear size. *SIAM J. Comput.*, 38(1):140–180, 2008.

[88] A. Ozdemir, R. S. Wahby, and D. Boneh. Scaling verifiable computation using efficient set accumulators. Cryptology ePrint Archive, Report 2019/1494, 2019.

[89] C. Papamanthou, E. Shi, and R. Tamassia. Signatures of correct computation. In *TCC*, 2013.

[90] B. Parno, C. Gentry, J. Howell, and M. Raykova. Pinocchio: Nearly practical verifiable computation. In *S&P*, May 2013.

[91] O. Reingold, G. N. Rothblum, and R. D. Rothblum. Constant-round interactive proofs for delegating computation. In *STOC*, pages 49–62, 2016.

[92] G. Rothblum. *Delegating Computation Reliably: Paradigms and Constructions*. PhD thesis, MIT, 2009.

[93] S. Setty, S. Angel, T. Gupta, and J. Lee. Proving the correct execution of concurrent services in zero-knowledge. In *OSDI*, Oct. 2018.

[94] S. Setty, A. J. Blumberg, and M. Walfish. Toward practical and unconditional verification of remote computations. In *HotOS*, May 2011.

[95] 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.

[96] S. Setty, R. McPherson, A. J. Blumberg, and M. Walfish. Making argument systems for outsourced computation practical (sometimes). In *NDSS*, Feb. 2012.

[97] 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.

[98] J. Thaler. Time-optimal interactive proofs for circuit evaluation. In *CRYPTO*, 2013.

[99] J. Thaler. A state of the art MIP for circuit satisfiability (lecture 14, "COSC 544 – probabilistic proof systems"). `http://people.cs.georgetown.edu/jthaler/SecondMIPLecture.pdf`, Oct. 2017.

[100] J. Thaler, M. Roberts, M. Mitzenmacher, and H. Pfister. Verifiable computation with massively parallel interactive proofs. In *HotCloud*, 2012.

[101] V. Vu, S. Setty, A. J. Blumberg, and M. Walfish. A hybrid architecture for verifiable computation. In *S&P*, 2013.

[102] R. S. Wahby, M. Howald, S. Garg, A. Shelat, and M. Walfish. Verifiable ASICs. In *S&P*,

2016.

[103] R. S. Wahby, Y. Ji, A. J. Blumberg, A. Shelat, J. Thaler, M. Walfish, and T. Wies. Full accounting for verifiable outsourcing. In *CCS*, 2017.

[104] 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.

[105] R. S. Wahby, I. Tzialla, A. Shelat, J. Thaler, and M. Walfish. Doubly-efficient zkSNARKs without trusted setup. In *S&P*, 2018.

[106] M. Walfish and A. J. Blumberg. Verifying computations without reexecuting them: From theoretical possibility to near practicality. *Commun. ACM*, 58(2), Jan. 2015.

[107] B. WhiteHat, A. Gluchowski, HarryR, Y. Fu, and P. Castonguay. Roll_up / roll_back snark side chain ~17000 tps. `https://ethresear.ch/t/roll-up-roll-back-snark-side-chain-17000-tps/3675`, Oct. 2018.

[108] 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.

[109] J. Zhang, T. Xie, Y. Zhang, and D. Song. Transparent polynomial delegation and its applications to zero knowledge proof. ePrint Report 2019/1482, 2019.

[110] J. Zhang, T. Xie, Y. Zhang, and D. Song. Transparent polynomial delegation and its applications to zero knowledge proof. In *S&P*, 2020.

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

[112] Y. Zhang, D. Genkin, J. Katz, D. Papadopoulos, and C. Papamanthou. A zero-knowledge version of vSQL. ePrint Report 2017/1146, 2017.

[113] Y. Zhang, D. Genkin, J. Katz, D. Papadopoulos, and C. Papamanthou. vRAM: Faster verifiable RAM with program-independent preprocessing. In *S&P*, 2018.

A Proof of witness-extended emulation for the protocol in §5.1

To simplify the proof, we define an information-theoretic protocol (which we denote as $\langle \mathcal{P}_{IP}, \mathcal{V}_{IP} \rangle$ and refer to it as *Spartan-core*) that is identical to the interactive argument in Section 5.1 except for the following:

- At step 1, instead of a commitment $\mathcal{C}$, $\mathcal{V}$ receives $\widetilde{w}'$, which is a $\log m$-variate multilinear polynomial, as an auxiliary input.
- At steps 13 and 14, $\mathcal{V}$ simply evaluates $v \leftarrow \widetilde{w}'(r_x)$

Spartan-core is similar to the Gir++ doubly-efficient interactive proof, except that Spartan-core only invokes two instances of the sum-check protocol whereas Gir++ invokes $O(d)$ sum-check instances for a depth-d circuit.

Lemma A.1. *For any non-satisfiable R1CS instance $\mathbb{X}$, any PPT prover $\mathcal{P}^*_{IP}$, and for all $w, r \in \{0,1\}^*$, $\Pr\{\langle \mathcal{P}^*_{IP}(w), \mathcal{V}_{IP}(r) \rangle(\mathbb{X}) = 1\} \leq (6 \log m + 1)/|\mathbb{F}|$*

Proof. If $\mathbb{X}$ is not satisfiable, then $Q_{io}(t)$ is not a zero-polynomial. By the Schwartz-Zippel lemma, $Q_{io}(t) = 0$ for at most $d/|\mathbb{F}|$ values of $t \in \mathbb{F}^s$, where d is the degree of Q_{io}. In our context, $d = s = \log m$. There are two cases to consider.

First, if $\mathcal{V}$ chooses a $\tau \in \mathbb{F}^s$ where $Q_{io}(\tau) = 0$, then the verifier in Spartan-Core may incorrectly output $b = 1$.

Second, If $\mathcal{V}$ chooses a $\tau \in \mathbb{F}^s$ where $Q_{io}(\tau) \neq 0$, then a malicious prover begins with a false claim in the sum-check protocol. By the soundness of the sum-check protocol (§3 and a standard analysis of soundness error for a sequence of sum-checks [92, §3.3.3]) and the soundness error of the random linear combination (lemma 5.1), the malicious prover succeeds with probability at most $(\ell_1 \cdot \mu_1 + \ell_2 \cdot \mu_2 + 1)/|\mathbb{F}|$, where $\mu_1 = \mu_2 = \log m$, $\ell_1 = 3$, $\ell_2 = 2$, and the probability is over $\mathcal{V}$'s randomness.

Applying a standard union bound establishes the desired result. □

Theorem A.1. *Given an extractable polynomial commitment scheme for multilinear polynomials PC that satisfies witness-extended emulation for PC.Eval, the protocol in Section 5.1 has witness-extended emulation.*

Proof. The proof is identical to Hyrax's except where their proof invokes properties of Gir++, we invoke properties of Spartan-core. □