# Spartan: Efficient and general-purpose zkSNARKs without trusted setup

Srinath Setty
*Microsoft Research*

**Abstract**

This paper describes a new public coin, succinct interactive zero-knowledge argument for NP under standard cryptographic hardness assumptions—*without* requiring a trusted setup. In particular, our argument enables a prover to prove the satisfiability of arithmetic circuits over a large finite field (an NP-complete language for which there exist efficient reductions from high-level programs of practical interest) to a verifier. We construct this argument through a novel synthesis of techniques from prior work on short PCPs, MIPs, and doubly-efficient IPs. Specifically, our interactive argument is a succinct variant of the sum-check protocol where the protocol is run with a carefully-constructed low-degree polynomial that encodes a given circuit satisfiability instance. Since our interactive argument is public coin, we make it non-interactive in the random oracle model, thereby obtaining a zero-knowledge succinct non-interactive argument of knowledge (zkSNARK), which we call *Spartan*.

Spartan is the first zkSNARK without trusted setup (i.e., a "transparent" zkSNARK) where verifying a proof incurs sub-linear costs *without* requiring data parallelism (or other homogeneity) in the structure of an arithmetic circuit for which a proof is produced. To achieve this, Spartan introduces a notion of *computation commitments*—a primitive to create a short cryptographic commitment to a mathematical description of an arithmetic circuit. Finally, Spartan is asymptotically efficient with small constants: the prover performs $O(n)$ cryptographic operations to produce a proof of size $O(n^{1/c})$ that can be verified in $O(n^{1-1/c})$ time (after a one-time, public preprocessing of the circuit to create a computation commitment that takes $O(n)$ time), where $n$ denotes the size of an arithmetic circuit and $c \geq 2$ (Spartan can produce $O(\log n)$-sized proofs, but the verifier incurs $O(n)$ costs).

## 1 Introduction

We revisit the problem of designing succinct zero-knowledge arguments for a general class of applications. In particular, we are interested in arguments for the complexity class NP [32, 66, 73]: they enable a computationally-bounded prover to convince the membership of a problem instance in an NP language *without* revealing anything besides the validity of the statement. Furthermore, the communication costs of the protocol and the cost to verify the prover's statement are sub-linear in the size of the statement being proven. We are motivated to design such succinct zero-knowledge arguments because there is significant interest in employing them in many applications including delegation of computation [40, 76, 80–84, 86] and stateful services [33, 39, 41, 47, 79, 94], distributed ledgers and blockchains [15, 31, 36, 67], anonymous credentials [43], and decentralized identities [1, 2].

We are particularly interested in arguments that prove the satisfiability of arithmetic circuits over a large finite field: given an arithmetic circuit instance $\mathcal{C}$, we are interested in a proof that demonstrates the knowledge of a witness $w$ such that $\mathcal{C}(x, w) = y$ where $x$ and $y$ are public inputs and outputs of $\mathcal{C}$ respectively.[1] We focus on the arithmetic circuit satisfiability problem because it is an NP-complete language for which there exists efficient transformations from high-level applications of interest in practice [16, 18, 22, 33, 68, 76, 79, 83, 90].

As we discuss in Section 2, there are many approaches to construct such arguments in the literature, starting with the work of Kilian [66] who provided the first construction of a succinct interactive argument protocol by employing probabilistically checkable proofs (PCPs) [4, 5, 8, 45] in conjunction with Merkle trees [72]. Micali [73] made a similar protocol non-interactive in the random oracle model, thereby obtaining the first zero-knowledge succinct non-interactive argument of knowledge (zkSNARK) [26, 51]. Unfortunately, the underlying PCP machinery is extremely expensive—despite foundational advances to achieve optimal asymptotics [17, 23–25]. Thus, the first works with an explicit motivation to build "practical" systems using probabilistic proofs [40, 80, 82, 83, 86] refine and implement simple interactive protocols of Ishai et al. [60] and Goldwasser et al. [53], which do not invoke asymptotically-efficient PCPs. The principal downside, among others, is that they achieve plausible performance and optimal asymptotics for the prover for only a restricted class of computations (or circuits).

In a breakthrough result, Gennaro, Gentry, Parno, and Raykova (GGPR) [49] address the above issue with a new formalism for encoding computations called *quadratic arithmetic programs (QAPs)*. By building on the work of

[1] Although we use the word "proof", we mean proofs that are computationally sound [32].

Ishai et al. [60], Groth [56], and Lipmaa [69], GGPR construct a zkSNARK for the arithmetic circuit satisfiability problem in which the prover's running time is quasi-linear in the size of the circuit (which is nearly optimal), the size of a proof is a constant number of group elements, and the verifier incurs a constant-time computation to verify the proof (in addition to the cost to process the public inputs and outputs of the circuit). Unfortunately, GGPR's zkSNARK requires a *trusted* setup that produces a large $O(\mathcal{C})$-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's machinery, but they retain a trusted setup [18, 22, 28, 57, 58], or require interaction [81].

The above state of affairs has motivated another class of works (called "transparent" zkSNARKs or zkSNARKs without trusted setup) that aim to address the limitation listed above for GGPR—while also providing performance similar to (or even better than) GGPR [49].[2] They prove security in the random oracle model, which is acceptable in practice. There are roughly four works in this class. First, Hyrax [91] extends a line of work [40, 84, 86–89] that refines the doubly-efficient interactive proofs (IPs) of Goldwasser et al. [53]. Second, Ligero [3] builds an interactive PCP [64] using the "MPC in the head" paradigm [61] and then transforms it to a zkSNARK. Third, zkSTARKs [14] and Aurora [19] build on interactive oracle proofs (IOPs) [20, 78], a generalization of PCPs and IPs. Fourth, Bulletproofs [34] builds on the work of Bootle et al. [30] and leverages a special-purpose argument protocol for inner product operations.

Unfortunately, these works suffer from the following problems: (1) Hyrax [90] requires arithmetic circuits to be layered, and the verifier's costs and the proof size scale linearly in the depth (i.e., the number of layers) of the circuit, so Hyrax is restricted to low-depth circuits (note that converting an arbitrary circuit into a layered form can increase its size quadratically [53]);[3] (2) Hyrax [90] achieves sub-linear verification costs only when its layered circuits have sufficient data-parallelism (i.e., a threshold number of identical sub-circuits); (3) zkSTARKs [14] assume circuits with a sequence of repeated sub-circuits (i.e., uniform circuits); although any circuit can be converted to this form [16, 18], such a transformation incurs 10–1000$\times$ blowup in circuit size (which translates to a similar factor increase in the prover's costs) when compared to equivalent but non-uniform circuits [90]; (4) The verifier in Aurora [19], Ligero [3], and Bulletproofs [34] incurs costs that are linear in the size of an arithmetic circuit, so they are not truly succinct; and (5) zkSTARKs [14] and Aurora [19] rely on a non-standard conjecture for soundness [14, Appendix B].

The goal of this work is to address the aforementioned problems associated with existing transparent SNARKs. Specifically, we desire a succinct zero-knowledge argument protocol for an arithmetic circuit satisfiability instance $(\mathcal{C}, x, y)$ with the following properties.

1. The argument's computational model should be general: the argument should apply to arbitrary circuits without assuming structure in layering, data-parallelism, or other uniformity.
2. The argument should not require a trusted setup nor a structured common reference string.
3. The prover should only incur $O(|\mathcal{C}|)$ cryptographic operations with small constants.
4. Proofs should be succinct (i.e., the size of a proof should be sub-linear in the size of $\mathcal{C}$).
5. The cost to verify a proof should be sub-linear in the size of $\mathcal{C}$ and linear in $|x| + |y|$.
6. Security should hold under standard cryptographic hardness assumptions (e.g., the hardness of computing discrete logarithms); non-interactivity can rely on random oracles [12] (such non-falsifiable assumptions are inherent [51]).

**Contributions.** This paper presents a new zkSNARK without trusted setup, which we call *Spartan*, for proving the satisfiability of arithmetic circuits over a large finite field (e.g., the set $\{0, 1, \ldots p-1\}$ for a large prime $p$). Spartan is the first transparent zkSNARK to achieve sub-linear verification costs for arbitrary circuits.[4] At its core, Spartan contains a new public coin interactive argument protocol with succinct communication costs and sub-linear verification costs where security holds under the hardness of computing discrete logarithms. We make our interactive argument a zero-knowledge succinct interactive argument (under standard cryptographic hardness assumptions) by using prior techniques [13, 42] and their use in the context of doubly-efficient interactive proofs [91, 95]. Finally, we make the succinct zero-knowledge argument non-interactive in the random oracle model using the Fiat-Shamir heuristic [46]. For an arithmetic circuit satisfiability instance $(\mathcal{C}, x, y)$ where $x$ and $y$ are public inputs and outputs of an arithmetic circuit $\mathcal{C}$, Spartan's prover incurs $O(|\mathcal{C}|)$ cryptographic operations and $O(|\mathcal{C}| \log |\mathcal{C}|)$ non-cryptographic finite field operations; the

[2] The work of Kilian [66] and Micali [73] do not suffer from the trusted setup issue, but as mentioned earlier, they are infeasible to be used in practice.
[3] For a depth-$d$ circuit, converting to a layered form increases the circuit size by a factor of $O(d)$.
[4] To our knowledge, even the classical PCP-based transparent zkSNARKs [66, 73] do not achieve sub-linear verification costs unless one uses uniform circuits, which as described above incur 10–1000$\times$ blowup in the prover's costs compared to using equivalent non-uniform circuits.

proof produced is of length $O(|\mathcal{C}|^{1/c})$ and can be verified in $O(|\mathcal{C}|^{1-1/c})$ time (for $c \geq 2$). Using a prior idea [91, §6.1], the proof length can be reduced to $O(\log |\mathcal{C}|)$ at the cost of making the verifier's costs $O(|\mathcal{C}|)$ (we refer to this variant as Spartan-log). Using a different prior idea [93, §3.1], the prover time can be made $O(|\mathcal{C}|)$, which is time-optimal (we refer to this variant as Spartan-opt). In more detail, Spartan makes the following contributions.

**(1) A new interactive argument with sub-linear verification costs (a succinct sum-check protocol).** We design a new public coin succinct interactive argument under standard cryptographic hardness assumptions where the verifier incurs sub-linear costs for arbitrary circuits. Beyond argument protocols, our interactive argument can be viewed as a succinct variant of the sum-check protocol [70] (a seminal interactive proof protocol used in several contexts [8, 9, 53]) when the low-degree polynomial over which the protocol is run can be decomposed into several multilinear polynomials. The latter maybe of independent interest.

Our core technical insight is that the sum-check protocol [70] when applied to a suitably-constructed low-degree polynomial yields a powerful—but *highly inefficient*—probabilistic proof protocol, but the inefficiency can be tamed, for certain choices of low-degree polynomials (e.g., a low-degree polynomial that can be decomposed into several multilinear polynomials), by employing a polynomial commitment scheme [65]. Specifically, we construct our argument through a careful integration of four techniques: (1) the sum-check protocol [70], (2) a cryptographic commitment scheme for multilinear polynomials [91, §6.1], which Hyrax employs for a related use; (3) an encoding of a circuit satisfiability instance as a low-degree polynomial composed of multilinear polynomials, which Blumberg et al. [29, §4.2] use in a multi-prover interactive proof protocol; and (4) *computation commitments*, a new primitive for creating a commitment to a mathematical description of a circuit, which is critical for achieving sub-linear verification costs.

***Computation commitments to achieve sub-linear verification costs.*** Achieving sub-linear verification costs appears to be fundamentally unrealizable because the verifier must process a circuit for which the proof is produced before it can verify a purported proof. To address this, our observation is that when verifying a proof, the verifier must evaluate a low-degree polynomial that encodes the circuit structure at a random point in its domain (§5), which incurs $O(|\mathcal{C}|)$ costs to the verifier. Our primitive, computation commitments, enables verifiably delegating the necessary polynomial evaluations to the prover by employing a polynomial commitment scheme for multilinear polynomials. Specifically, in Spartan, the verifier reads the entire circuit for which the proof is produced and retains a short cryptographic commitment to a set of multilinear polynomials that encode the structure of the circuit. Later, when producing a proof, the prover evaluates the necessary polynomials and proves that the polynomial evaluations are consistent with the commitment retained by the verifier. While the verifier incurs $O(|\mathcal{C}|)$ cost to compute a computation commitment, the cost is amortized over *all* future proofs produced for that circuit.[5] This amortization is similar to that of GGPR: the verifier can verify a proof individually—without batching—while enjoying sub-linear costs. However, unlike GGPR's trusted setup, creating a computation commitment does not involve any secret trapdoors. Section 6 provides details.

**(2) Shed light on connections among different strands of theory.** Our interactive argument sheds light on interconnections among different lines of work on probabilistic proofs—from the perspective of zkSNARKs—including doubly-efficient IPs, MIPs, and short PCPs. Section 5.2 provides details in context, but here is a brief summary: (1) Spartan is a light-weight mechanism to transform the short PCPs of Babai et al. [8] to a succinct interactive argument—*without* employing Kilian's approach of using Merkle trees; (2) Spartan is also a simple mechanism to compile the two-prover interactive proof protocol of Blumberg et al. [29] (and of Babai et al. [9]) to a single prover argument and then into a zkSNARK—without relying on general and expensive techniques [27]; and (3) Spartan is a way to eliminate the requirement of layered circuits in doubly-efficient IPs [40, 53, 84, 87–89] (and arguments built atop them [91, 94, 95]) by applying cryptography. Finally, in all the three cases, Spartan achieves sub-linear costs to the verifier for *any* arithmetic circuit over which the protocol is run (after a one-time linear-time preprocessing of the circuit by the verifier). Thus, Spartan unifies three different lines of prior theory and achieves theoretical improvements.

**(3) Simplicity.** Compared to other transparent zkSNARKs, especially those based on short PCPs and IOPs, Spartan is perhaps simpler. While this claim is of non-technical nature and cannot be rigorously proved, Spartan relies only on well-studied building blocks. The core ingredients in Spartan are the sum-check protocol [70], a classical protocol, and a cryptographic commitment scheme for multilinear polynomials [90] that relies only on simple homomorphic commitments [77] and a sigma protocol for inner product operations. Thus, we believe that this work makes it easy for practitioners to refine, optimize, and incorporate Spartan into their applications, thereby taking several steps toward making it possible to deploy transparent zkSNARKs at large scale.

**Roadmap and paper organization.** Section 2 describes related work; Section 3 introduces concepts and terminology for describing Spartan; Section 4 provides a top-down overview of Spartan and its sub-protocols; Sections 5 and 6

[5] If the circuit has regular structure (e.g., data-parallelism), the cost of creating such a commitment is sub-linear in $|\mathcal{C}|$.

| | setup | prover | proof length | verifier | assumption | computational model |
|---|---|---|---|---|---|---|
| GGPR-based [21, 49, 57] | private | $O(n \log n)$ | $O(1)$ | $O(1)$ | KOE | arbitrary circuits |
| Ligero [3] | public | $O(n \log n)$ | $O(\sqrt{n})$ | $O(n)$ | CRHF | arbitrary circuits |
| Hyrax [89] | public | $O(n + m \cdot g)$ | $O(m + \lvert w \rvert^{1/c})$ | $O(m + \lvert w \rvert^{1-1/c})$ | DLOG | layered circuits |
| Hyrax-log [89] | public | $O(n + m \cdot g)$ | $O(m + \log \lvert w \rvert)$ | $O(m + \lvert w \rvert)$ | DLOG | layered circuits |
| Bulletproofs [34] | public | $O(n)$ | $O(\log n)$ | $O(n)$ | DLOG | arbitrary circuits |
| zkSTARKs [14] | public | $O(n \log^2 n)$ | $O(\log^2 n)$ | $O(\log^2(n))$ | CRHF | uniform circuits |
| Aurora [19] | public | $O(n \log n)$ | $O(\log^2 n)$ | $O(n)$ | CRHF | arbitrary circuits |
| Spartan | public | $O(n \log n)$ | $O(n^{1/c})$ | $O(n^{1-1/c})$ | DLOG | arbitrary circuits |
| Spartan-log | public | $O(n \log n)$ | $O(\log n)$ | $O(n)$ | DLOG | arbitrary circuits |
| Spartan-opt | public | $O(n)$ | $O(n^{1/c})$ | $O(n^{1-1/c})$ | DLOG | arbitrary circuits |
| Spartan-opt-log | public | $O(n)$ | $O(\log n)$ | $O(n)$ | DLOG | arbitrary circuits |

Figure 1: A comparison of zkSNARK schemes geared toward practice. The depicted costs are for an arithmetic circuit satisfiability instance $(\mathcal{C}, x, y)$. $n$ denotes the number of gates in an arithmetic circuit (i.e., $n = |\mathcal{C}|$). For Hyrax [91], we assume a layered arithmetic circuit of depth $d$, width $g$, and $k$ copies (i.e., $n = d \cdot g \cdot k$); $w$ denotes a witness to $\mathcal{C}$; and $m = d \cdot \log g$. For Hyrax and Spartan, $c$ is a constant $\geq 2$. The verifier incurs $O(|x| + |y|)$ additional cost in all schemes. Furthermore, all zkSNARKs without trusted setup listed above achieve non-interactivity in the random oracle model using the Fiat-Shamir heuristic [46]. Spartan-opt-log applies refinements from Spartan-log to Spartan-opt to reduce proof sizes at the cost of increasing the verifier's running time. Finally, it is worth noting that Hyrax's prover performs $O(|w|)$ cryptographic operations whereas Spartan (and others) perform $O(n)$ cryptographic operations, so for circuits where $|w| < n$, Hyrax's prover performs fewer cryptographic operations.

describe technical details of Spartan and offer security proofs; Section 7 describes extensions.

## 2 Related work

This section compares Spartan with prior approaches to zkSNARKs (and related constructs). For ease of exposition, we introduce terminology and notation. Recall that for any problem instance $X$ (e.g., a given arithmetic circuit), if $X$ is in an NP language $L$ (e.g., the set of satisfiable arithmetic circuits), there exists a witness $w$ such that a deterministic verifier can check if $X \in L$ in time $O(poly(|X|))$. We use $n$ to denote the running time of such a deterministic verifier. Figure 1 depicts asymptotic costs for the prover and the verifier, proof length, and assumptions made by different zkSNARK schemes. For transparent zkSNARKs in the table, the assumptions we state are in addition to the random oracle assumption made by them to achieve non-interactivity.

**Short PCPs.** Babai et al. [8] and Feige et al. [45] introduce *probabilistically checkable proofs (PCPs)* [4, 5, 44], a type of proof that proves the membership of a problem instance in an NP language while requiring a randomized verifier to inspect only a small (e.g., a constant [59]) number of bits in the proof. The length of a proof in early constructions is $O(poly(n))$ (and are called "short PCPs" in the PCP literature [24, 25]), so on their own, PCPs do not offer succinctness. Kilian [66] constructs the first succinct interactive argument for NP by employing short PCPs in conjunction with a cryptographic hash function. In Kilian's approach, the prover cryptographically commits to a short PCP using a Merkle tree [72] and then decommits the proof at locations that a verifier wishes to inspect; Micali [73] made a similar approach non-interactive in the random oracle model. Kilian's approach achieves $O(polylog(n))$ communication (which translates to the size of a non-interactive proof in Micali's construction). While many works on PCPs focus on proving inapproximability results [45], there is long line of work to reduce the length of a PCP, and to improve the asymptotic costs of proof generation and verification [17, 23, 25]. Despite these advances, the PCP machinery remains too expensive to be used in practice, so Kilian's and Micali's approach to build succinct arguments via short PCPs—even when using state-of-the-art PCPs—remains concretely inefficient.

Spartan achieves similar asymptotics for the prover as instantiating Kilian's approach with a state-of-the-art PCP, but with significantly smaller constants. To achieve this, Spartan employs stronger cryptography (in the form of a polynomial commitment scheme) than a simple hash function (§5), but it still only makes standard cryptographic hardness assumptions. However, the verifier's work (and the size of the proof) are asymptotically worse under Spartan. Note that the asymptotic costs of the verifier in Spartan are inherited from the specific polynomial commitment scheme we use, so any future improvements in polynomial commitment schemes translate to improvements to Spartan. Finally, as we discuss in Section 5, Spartan can be seen as a replacement for Kilian's approach to transform the short PCP scheme of Babai et al. [8]—where the PCP length is asymptotically longer than in the state-of-the-art constructions [17, 23], but has the required algebraic structure for applying stronger cryptography—to a succinct interactive argument.

**Linear PCPs and QAPs.** Ishai, Kushilevitz, and Ostrovsky (IKO) [60] 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 [4, 5]. Such linear PCPs are far simpler than state-of-the-art 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 cryptographic commitment protocol in which a prover can commit to an exponentially-long PCP without ever materializing the entire PCP.[6] Because of the specific linear PCP construction that they use, which is based on Hadamard codes [4, 5], the prover's work is $O(n^2)$. Setty et al. [82] 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 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.

Gennaro, Gentry, Parno, and Raykova (GGPR) [49] introduce *quadratic arithmetic programs (QAPs)*, a remarkable encoding of computations amenable to probabilistic checking. Building on IKO[60], Groth [56], and Lipmaa [69], GGPR design a zkSNARK in which the prover's work is $O(n \log n)$. Bitansky et al. [28] and Setty et al. [81] observe that QAPs can be viewed as linear PCPs. Zaatar [81] leverages this observation to build an interactive argument with $O(n \log n)$ asymptotics for the prover by composing a QAP-based linear PCP with a refined variant of IKO's cryptographic machinery [82, 83]. 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, it is not publicly verifiable, and it achieves succinctness only when proofs for a batch of statements are verified at once. Note that Zaatar and IKO do not support zero-knowledge, but it is not fundamental [60, §3.1].

Pinocchio [76] optimizes and implements GGPR [49] in entirety. It avoids the issues listed above for Zaatar and IKO—at the cost of making q-type, knowledge of exponent (KOE) 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 [51]). Among other things, BCGTV [18], BCTV [22], and Groth [57] refine and optimize zkSNARKs in Pinocchio and the work of Bitansky et al. [28]. However, all these works require a trusted setup, which must be performed by an incorruptible party and the trapdoor used in the setup process must be kept secret for ensuring soundness. The setup incurs $O(n)$ cryptographic operations and produces an $O(n)$-sized structured common reference string (CRS). To cope with trusted setup, recent works [58, 71] propose schemes to update the structured CRS after it is generated. However, at least one updating entity must be trusted.

**Interactive proofs.** While PCP-based arguments make cryptographic hardness assumptions, interactive proofs (IPs) are unconditionally secure. The early works on interactive proofs [7, 54] focus on studying the power of IPs in the context of intractable languages. However, in 2008, Goldwasser, Kalai, and Rothblum (GKR) [53] propose an elegant: interactive proof protocol 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 width of the circuit. Cormode, Mitzenmacher, and Thaler (CMT) [40] refine the GKR protocol to bring down 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 [84, 86–89] further refine the approach of GKR and CMT to reduce constants and to improve asymptotics when circuits have further structure such as data-parallelism. This line of work culminated in interactive arguments [94, 96] and zkSNARKs [91, 95]. Section 5.2 discusses, in context, how these works relate to Spartan.

**Interactive oracle proofs.** Ben-Sasson et al. [20] and Reingold et al. [78] introduce a new proof model that combines aspects of PCPs and IPs under two different names: interactive oracle proofs (IOPs) and probabilistically checkable interactive proofs (PCIPs). In their model model, the prover and the verifier interact over a sequence of rounds (as in IPs), but the verifier examines only a small number of bits in the prover's messages (as in PCPs), so this model also generalizes interactive PCPs [64]. They also extend Kilian's and Micali's techniques [66, 73] to transform IOPs to zkSNARKs in the random oracle model. There are two zkSNARKs in this line of work: zkSTARKs [14] and Aurora [19]. These works only require symmetric key operations (as in the classical PCP-based arguments [66, 73]), but they suffer from limitations discussed in Section 1, which includes relying on a non-standard conjecture related to Reed-Solomon codes for proving soundness. Furthermore, it is not entirely clear how these works can achieve sub-linear verification costs—without employing uniform circuits (which are far more verbose than equivalent non-uniform circuits [90]).

[6] Spartan is analogous to IKO in this aspect: The prover in IKO and Spartan do not materialize a PCP. The difference between these two works is that Spartan works with the polynomially-sized short PCPs of Babai et al. [8] rather than the exponentially-sized linear PCPs of Arora et al. [4, 5].

**Multi-party computation (MPC).** Ishai et al. [61] propose the "MPC in the head" paradigm to build zero-knowledge arguments from secure multi-party computation protocols. While the earlier instantiations of this paradigm, ZKBoo [52] and ZKB++ [37], produce proofs of size $O(n)$, Ligero [3] makes the proof size $O(\sqrt{n})$ by employing a sophisticated multi-party computation protocol and coding techniques to build an interactive PCP [64]. Like Aurora [19] and zkSTARKs [14], Ligero only relies on symmetric key primitives such as hash functions. However, as discussed in Section 1, the verifier's running time in these works is $O(n)$.

**Knowledge of discrete logarithms.** Bootle et al. [30] building on its predecessors [11, 55] provide a zero-knowledge argument for the satisfiability of arithmetic circuits under the hardness of computing discrete logarithms. The core building block in their argument is a special-purpose argument for proving the correctness of an inner product operation over two committed vectors. The proof size is $O(\log n)$, which Bulletproofs [34] improves by $\approx 3\times$. Unfortunately, as mentioned in Section 1, the verifier in these works incurs $O(n)$ costs.

# 3 Preliminaries

This section introduces notation, terminology, and concepts that we build on. We borrow some of our notation, terminology, and phrasing from prior work [29, 40, 53, 87, 89, 94].

## 3.1 Arithmetic circuits, satisfiability, and assignments

**Arithmetic circuits.** An arithmetic circuit $\mathcal{C}$ over a finite field $\mathbb{F}$ (e.g., $\mathbb{F}_p$ for a large prime $p$) is a directed network of input gates, output gates, and internal gates with wires connecting those gates. We assume each gate has two in-neighbor wires and computes an addition (or multiplication) over $\mathbb{F}$ (input gates have no in-neighbors). We consider two types of input gates in an arithmetic circuit: (1) public input gates, and (2) non-deterministic input gates (we explain differences between them below). Thus, we can think of $\mathcal{C}$ as a function that maps a public input $x \in \mathbb{F}^m$ and a non-deterministic input $w \in \mathbb{F}^\ell$ to a public output $y \in \mathbb{F}^n$. Let $|\mathcal{C}|$ denote the size of $\mathcal{C}$, that is, the number of gates in $\mathcal{C}$.

**Definition 3.1** (Arithmetic circuit satisfiability)**.** Given an arithmetic circuit, $\mathcal{C} : \mathbb{F}^m \times \mathbb{F}^q \to \mathbb{F}^n$, $x \in \mathbb{F}^m$, and $y \in \mathbb{F}^n$, *the arithmetic circuit satisfiability* is the decision problem of checking whether there exists a non-deterministic input $w \in \mathbb{F}^q$ such that $\mathcal{C}(x,w) = y$. This is captured by the following relation: $R_{(\mathcal{C},x,y)} = \{w \in \mathbb{F}^q : \mathcal{C}(x,w) = y\}$. We refer to the tuple $(\mathcal{C},x,y)$ as an *arithmetic circuit satisfiability instance*.

Note that in the above definition if $x = \perp$ and $y = 1$, then we get a traditional arithmetic circuit satisfiability problem. We consider our generic setup because in many applications of zkSNARKs (e.g., proof-based verifiable computation; see [92] for a survey), it is natural to have non-trivial $x$ and $y$, which correspond to public input and output of a given delegated computation respectively.

**Definition 3.2** (Satisfying assignment)**.** A *satisfying assignment* to an arithmetic circuit satisfiability instance $(\mathcal{C},x,y)$ is a vector of values one for each gate in $\mathcal{C}$ with the constraint that values assigned to public input gates are consistent with $x$, the values assigned to output gates are consistent with $y$, and the value assigned to any non-input gate $g$ is consistent with an evaluation of $g$ with values assigned to $g$'s in-neighbor gates.

**Lemma 3.1.** *If there exists a satisfying assignment to $(\mathcal{C},x,y)$ then there exists a witness $w$ such that $\mathcal{C}(x,w) = y$.*

*Proof.* Given a satisfying assignment $z$ to $(\mathcal{C},x,y)$, output values assigned to non-deterministic input gates of $\mathcal{C}$ in $z$ as witness $w$. Since $z$ is a satisfying assignment to $(\mathcal{C},x,y)$, $\mathcal{C}(x,w) = y$. □

## 3.2 Polynomials and low-degree extensions

**Polynomials.** Recall that a polynomial $G$ over a finite field $\mathbb{F}$ 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 $G$ is the maximum degree of any monomial in $G$. Finally, the degree of a polynomial $G$ in a particular variable $x_i$ is the maximum exponent that $x_i$ takes in any of the monomials in $G$.

**Definition 3.3.** A *multivariate* polynomial is a polynomial with more than one variable (otherwise it is called a *univariate* polynomial). A multivariate polynomial is called a *multilinear* polynomial if the degree of the polynomial in each variable is at most one.

**Definition 3.4.** A multivariate polynomial over a finite field $\mathbb{F}$ is called low-degree polynomial if the degree of the polynomial in each variable is small compared to $|\mathbb{F}|$, for example $\log |\mathbb{F}|$.

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

$$\begin{aligned}\widetilde{Z}(x_1,\ldots,x_m) &= \sum_{e\in\{0,1\}^m} Z(e)\cdot\prod_{i=1}^{m}(x_i\cdot e_i + (1-x_i)\cdot(1-e_i)) \\ &= \sum_{e\in\{0,1\}^m} Z(e)\cdot\chi_e \\ &= \langle (Z(0),\ldots,Z(2^m-1)),(\chi_0,\ldots,\chi_{2^m-1})\rangle\end{aligned}$$

For any $r = (r_1,\ldots,r_m) \in \mathbb{F}^m$, $\widetilde{Z}(r)$ can be computed, using the above expression, in $O(2^m)$ field operations [84, 87].

### 3.3 The sum-check protocol

We now describe a key technical tool that we employ in Spartan, called the sum-check protocol [70].

Suppose there is an $m$-variate low-degree polynomial, $G : \mathbb{F}^m \to \mathbb{F}$ where the degree of each variable in $G$ is at most $\ell$. Suppose that a verifier $\mathcal{V}_{SC}$ is interested in checking the following claim by an untrusted prover $\mathcal{P}_{SC}$:

$$H = \sum_{x_1\in\{0,1\}}\sum_{x_2\in\{0,1\}}\cdots\sum_{x_m\in\{0,1\}} G(x_1,x_2,\ldots,x_m)$$

Of course, given a description of the polynomial $G(\cdot)$, $\mathcal{V}_{SC}$ can deterministically evaluate the sum over the Boolean hypercube and verify whether the sum is $H$. But, this computation takes time exponential in $m$.

Lund et al. [70] describe a seminal 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 sum-check protocol, $\mathcal{V}_{SC}$ interacts with $\mathcal{P}_{SC}$ over a sequence of $\ell$ rounds. At the end of this interaction, $\mathcal{V}_{SC}$ outputs `accept` or `reject` indicating whether it believes in $\mathcal{P}_{SC}$'s claim. The principal cost to $\mathcal{V}_{SC}$ is to evaluate the polynomial $G(\cdot)$ at a random point in its domain $(r_1,r_2,\ldots,r_m) \in_R \mathbb{F}^m$. Figure 2 depicts the sum-check protocol from $\mathcal{V}_{SC}$'s perspective. It offers the following guarantees (note that there are no assumptions about the computational power of the prover).

- **Completeness.** If $H = \sum_{x\in\{0,1\}^m} G(x)$, then a correct $\mathcal{P}_{SC}$ can always make $\mathcal{V}_{SC}$ output `accept`.
- **Soundness.** If $H \neq \sum_{x\in\{0,1\}^m} G(x)$, then for any prover $\mathcal{P}^*_{SC}$, $\Pr\{\mathcal{V}_{SC}$ outputs `accept`$\} \leq \epsilon_{sc}$ where $\epsilon_{sc} = \ell\cdot m/|\mathbb{F}|$.

## 4 Overview of Spartan

This section provides an overview of Spartan, a zkSNARK [26] without trusted setup. The most novel aspect of Spartan is a new interactive argument for NP with succinct verification and communication costs. Abstractly speaking, the new interactive argument is a succinct variant of the sum-check protocol (which maybe of independent interest). In the classical sum-check protocol [70], the verifier must evaluate the low-degree polynomial over which the sum-check is run at a random point in its domain (line 17 in Figure 2), whereas our variant makes the verifier's computational cost to be sub-linear in the cost of evaluating the low-degree polynomial—as long as the the low-degree polynomial is composed of several multilinear polynomials. This is achieved by verifiably delegating the task of evaluating multilinear polynomials to the prover. Besides reducing the verifier's computational costs, our protocol makes communication costs from the prover to the verifier to be sub-linear in the communication required for the verifier to locally evaluate the low-degree polynomial.

**Achieving sub-linear verification and communication costs.** We observe that when one applies the sum-check protocol to a suitably-constructed encoding of an arithmetic circuit satisfiability instance $(\mathcal{C},x,y)$, the sum-check protocol offers an extremely simple probabilistic proof protocol—for proving the satisfiability of arithmetic circuits—that is unconditionally secure. But, this protocol is highly inefficient as it requires $O(|\mathcal{C}|)$ communication from the prover to the verifier as well $O(|\mathcal{C}|)$ computation at the verifier. We further observe that this inefficiency can be addressed by employing simple cryptography, specifically a cryptographic commitment scheme for polynomials, which in our choice

```
function SumCheck(G, m, H, ℓ)
  (r_1, r_2, ..., r_m) ←_R 𝔽^m // sample m field elements randomly
  e ← H
  for i = 1, 2, ..., m do
    // An honest P_SC returns G_i(·) as {G_i(0), G_i(1), ... G_i(ℓ)} since G_i(·) is a degree-ℓ univariate polynomial
    // when i = 1, G_1(X_1) = Σ_{(x_2,...,x_m)∈{0,1}^{m-1}} G(X_1, x2, ..., x_m)
    // when i > 1, G_i(X_i) = Σ_{(x_{i+1},...,x_m)∈{0,1}^{m-i}} G(r_1, ..., r_{i-1}, X_i, x_{i+1}, ..., x_m)
    G_i(·) ← ReceiveFromProver()

    if G_i(0) + G_i(1) ≠ e then
      return reject

    SendToProver(r_i)
    e ← G_i(r_i) // evaluate G_i(r_i) using its point-value form received from the prover

  // The following check requires computing G(·) at (r_1, r_2, ..., r_m) ∈ 𝔽^m
  if G(r_1, r_2, ..., r_m) ≠ e then
    return reject
  else
    return accept
```

Figure 2: Description of the sum-check protocol from the perspective of $\mathcal{V}_{SC}$. $\mathcal{V}_{SC}$ checks if the $m$-variate polynomial $G(\cdot)$ sums to $H$ over the Boolean hypercube $\{0,1\}^m$ with the assistance of an untrusted prover $\mathcal{P}_{SC}$. The degree of $G(\cdot)$ in each variable is at most $\ell$.

of the low-degree polynomial requires support for a restricted class of polynomials called multilinear polynomials (§3.2). The end result is a simple—yet highly efficient—interactive argument with succinct communication costs (§5 describes this protocol). However, the verifier still incurs $O(|\mathcal{C}|)$ computation (for an arbitrary circuit with no structure).

We address the latter issue by again employing a commitment scheme for multilinear polynomials. Specifically, we observe that during verification, the verifier only needs to evaluate a multilinear extension of several functions (i.e., several multilinear polynomials) that encode the circuit structure. Instead of the verifier evaluating those polynomials locally (which incurs $O(n)$ costs), the verifier computes a succinct cryptographic commitment to each of those polynomials—which we refer to as a *computation commitment*—and retains the commitment (we emphasize that there is no secret trapdoor, so it is a public computation). This incurs $O(|\mathcal{C}|)$ costs, but this cost is amortized over *all* future verification of proofs for the same circuit (note that those proofs can be verified individually, so there is no need to batch verify proofs to obtain amortization benefits). Thus, during verification of a proof for a circuit $\mathcal{C}$, the verifier asks the prover to evaluate the necessary multilinear polynomials at a point required for its verification in the interactive argument. For the verifier, the cost of verifying such an evaluation incurs computation and communication that are sub-linear in $|\mathcal{C}|$. Section 6 provides more details.

**Achieving zero-knowledge and non-interactivity.** Our interactive argument contains a single invocation of the sum-check protocol, so we can make it zero-knowledge by essentially using the zero-knowledge variant of the sum-check protocol developed in the context of doubly-efficient interactive proofs [91, 95]. Instead of running the sum-check protocol as depicted in Figure 2, the the prover in the zero-knowledge variant of the sum-check protocol commits to all its messages and proves—using a simple sigma protocol—that the verifier's checks pass underneath the commitments.[7] Zhang et al. [95, Section 4] in particular describe a zero-knowledge variant of the sum-check protocol in isolation, which we can use in the place of the traditional sum-check protocol in our interactive argument.[8] The only difference is in the specific polynomial on which the sum-check protocol is run, so we will not discuss this further. Finally, since our protocol is public coin, we can make it non-interactive in the random oracle model using the Fiat-Shamir heuristic [46].

[7] Recent work of Xie et al. [93] proposes, in the context of doubly-efficient interactive proofs, an alternate approach to achieve zero-knowledge in the sum-check protocol, by building on the work for Chiesa et al. [38]. This approach appears more light-weight compared to using cryptographic commitments and sigma protocols, especially for the verifier.

[8] Wahby et al. [91] describe several optimizations to reduce communication and computation in the zero-knowledge sum-check protocol, which apply well to our context.

## 5 A new public coin, succinct interactive argument for NP

This section describes a new public coin, succinct interactive argument for the arithmetic circuit satisfiability problem under standard cryptographic hardness assumptions. Our description, terminology, and notation is influenced by prior work that we build on [29, 40, 53, 87, 91].

**Interactive arguments.** An interactive argument is an interactive protocol between two entities: a prover $\mathcal{P}$ and a verifier $\mathcal{V}$. Both entities start with an NP statement (e.g., an arithmetic circuit satisfiability instance) and $\mathcal{P}$'s goal is to convince the validity of the statement (e.g., that the given arithmetic circuit satisfiability instance is indeed satisfiable). The prover is given an additional input: a purported witness $w$ to the NP statement that is being proven. $\mathcal{V}$ and $\mathcal{P}$ exchange a sequence of messages over a bounded number of rounds and at the end of the interaction $\mathcal{V}$ outputs its decision (`accept` or `reject`) indicating whether $\mathcal{V}$ believes that the circuit satisfiability instance is satisfiable. We now formalize properties necessary for an interactive protocol to be called an interactive argument.

**Definition 5.1.** An interactive protocol between a pair of probabilistic polynomial time algorithms $\langle \mathcal{P}, \mathcal{V} \rangle$ is called an interactive argument for the arithmetic circuit satisfiability problem if the following conditions hold.

- **Completeness:** For any satisfiable arithmetic circuit satisfiability instance $X = (\mathcal{C}, x, y)$, there exists a witness $w$ such that for all $r \in \{0,1\}^*$, $\Pr\{\langle \mathcal{P}(w), \mathcal{V}(r) \rangle(X) = \texttt{accept}\} \geq 1 - \epsilon_c$, where $\epsilon_c$ is the completeness error.
- **Soundness:** For any non-satisfiable arithmetic circuit satisfiability instance $X = (\mathcal{C}, x, y)$, any probabilistic polynomial time prover $\mathcal{P}^*$, and for all $w, r \in \{0,1\}^*$, $\Pr\{\langle \mathcal{P}^*(w), \mathcal{V}(r) \rangle(X) = \texttt{accept}\} \leq \epsilon_s$, where $\epsilon_s$ is the soundness error.

Such an interactive argument is called *succinct* if the total communication between $\mathcal{P}$ and $\mathcal{V}$ is sub-linear in $|\mathcal{C}|$ and is called *public coin* if $\mathcal{V}$'s messages are chosen uniformly at random (i.e., independent of $\mathcal{P}$'s messages). The rest of this section describes a public coin, succinct interactive argument underneath Spartan. Toward the end of this section, we prove that our protocol satisfies the above two properties.

### 5.1 Spartan's starting point: Using the sum-check protocol as a foundation for succinct arguments

Suppose $\mathcal{P}$ holds a satisfying assignment $z$ to an arithmetic circuit satisfiability instance $(\mathcal{C}, x, y)$, the goal of $\mathcal{V}$ is to check if $\mathcal{P}$ indeed holds a satisfying assignment. Of course, $\mathcal{P}$ could send its purported satisfying assignment $z$ to $\mathcal{V}$ and then $\mathcal{V}$ can deterministically verify whether $z$ is a satisfying assignment to $(\mathcal{C}, x, y)$. But, this incurs communication costs proportional to $|\mathcal{C}|$; $\mathcal{V}$ also incurs $O(|\mathcal{C}|)$ computational costs to verify the purported witness. Thus, it is not succinct.

As discussed in Section 3.3, the sum-check protocol [70] offers an efficient mechanism—with succinct communication costs—where a verifier $\mathcal{V}$ can probabilistically check a claim made by an untrusted prover $\mathcal{P}$ about the sum of polynomial evaluations over a Boolean hypercube. However, to build a succinct interactive argument using the sum-check protocol, we must solve the following sub-problems:

1. We must identify a low-degree multivariate polynomial $G(\cdot)$ such that it fits the structure of the sum-check protocol. Specifically, for a given an arithmetic circuit satisfiability instance $(\mathcal{C}, x, y)$, we must identify an $m$-variate low-degree polynomial that sums to a specific value (e.g., $0 \in \mathbb{F}$) over the $m$-dimensional Boolean hypercube $\{0,1\}^m$ *if and only if* $(\mathcal{C}, x, y)$ is satisfiable (i.e., there exists a satisfying assignment $z$ to $(\mathcal{C}, x, y)$).
2. We must make the total communication costs of the protocol to be sub-linear in $|\mathcal{C}|$. This is non-trivial because in the sum-check protocol $\mathcal{V}$ must evaluate a multivariate low-degree polynomial $G(\cdot)$ at a random point in its domain, and $G(\cdot)$ depends on the entire satisfying assignment to $(\mathcal{C}, x, y)$. Thus, a straightforward evaluation of $G(\cdot)$ at an arbitrary point in its domain would require $O(|\mathcal{C}|)$ communication (to receive a satisfying assignment) from $\mathcal{P}$ to $\mathcal{V}$.
3. We must make $\mathcal{V}$'s costs to participate in the protocol to be sub-linear in $|\mathcal{C}|$. This seems non-trivial as $\mathcal{V}$ must at least read the arithmetic circuit satisfiability instance $(\mathcal{C}, x, y)$ for which the proof is produced in entirety and $\mathcal{C}$ has no structure (e.g., data-parallelism or uniformity).

The next subsection describes prior solutions to the three sub-problems. We then describe Spartan's solution to the first two sub-problems. The next section describes how Spartan solves the last sub-problem (§6).

### 5.2 Prior solutions to address the above sub-problems (or works closely related to Spartan)

Prior literature on probabilistic proofs, starting with the work of Babai et al. [8, 9], offers a low-degree polynomial $G(\cdot)$ 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 [9] (or the correct execution of a program under a pointer machine [8] or other NP-complete problems). These capture a general model of computation, but those representations are many orders of magnitude more verbose than arithmetic circuits over a large finite field (verbosity translates to constants in the prover's and verifier's costs in the argument protocol). Fortunately, Blumberg, Thaler, Vu, and Walfish [29] offer an elegant low-degree polynomial as part of a multi-prover interactive proof (MIP) protocol in the context of proof-based verifiable computation [8, 48, 53, 92]. Specifically, their low-degree polynomial enables verifying the satisfiability of an arithmetic circuit via a combination of the sum-check protocol and other machinery discussed below. 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) provers that are not allowed to communicate (or collude) with one other. For example, in the protocol of Blumberg et al. [29] (which follows the high-level structure of the two-prover protocol of Babai et al. [9]), $\mathcal{V}$ interacts with the first prover to run the sum-check protocol. In the last step, $\mathcal{V}$ must evaluate $G(\cdot)$ at a random point (which as described above would require $O(|\mathcal{C}|)$ computation by $\mathcal{V}$ and $O(|\mathcal{C}|)$ communication from $\mathcal{P}$ to $\mathcal{V}$). Instead, $\mathcal{V}$ interacts with a second prover—via another protocol called *low-degree tests* [6, 74]—to learn the evaluation of $G(\cdot)$ at a random point. Despite a sophisticated analysis of soundness error, their protocol achieves only 23 bits of security when using a finite field of size 300 bits. Although MIPs require two (or more) non-colluding provers, there exists an elegant compiler to transform MIPs (such as the construction of Blumberg et al. [29]) to SNARKs [27]. However, the compiler of Bitansky and Chiesa [27] relies on fully-homomorphic encryption (FHE) [50], which despite massive improvements, remains orders of magnitude more expensive than simple public-key operations.

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. [29] and Babai et al. [9] (and other similar two-prover interactive proofs) into a succinct interactive argument (and then into a zkSNARK without trusted setup). For non-interactivity, Spartan assumes a random oracle model whereas the compiler of Bitansky and Chiesa [27] 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 [85, §3] sketches a mechanism to compile the two-prover protocol of Blumberg et al. [29] into a single prover argument using a polynomial commitment scheme. However, the sketch does not offer a security proof nor prescribe a concrete polynomial commitment scheme. More fundamentally, it does not solve the third sub-problem to achieve sub-linear verification costs for the verifier.

**Short PCPs.** The short PCP construction of Babai et al. [8] does not require two provers. Instead, the prover $\mathcal{P}$ writes down a string—a probabilistically checkable proof (PCP)—where a verifier $\mathcal{V}$ only needs to inspect at a small number of locations in the string. In the construction of Babai et al. [8], the string 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 string allows $\mathcal{V}$ to conduct the sum-check protocol by accessing only a small subset of the bits in the string), and (2) a low-degree extension (LDE) of a satisfying assignment $Z$ 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 2, constructing a succinct argument via such short PCPs—using Kilian's approach [66]—remains highly impractical.

Note that Spartan can be viewed as a more direct transformation of ideas in the short PCP construction of Babai et al. [8] into a succinct interactive argument: (1) The prover in Spartan does not write down a low-degree extension of $Z$ (but instead cryptographically commits to a low-degree extension of $Z$ using $Z$ 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. [8] avoid multilinear extensions (MLE) of a satisfying assignment 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. [60], and the compiler of Bitansky and Chiesa [27].

**Doubly-efficient interactive proofs.** Doubly-efficient interactive proofs [40, 53, 84, 86–89] solve all the three sub-problems—by restricting themselves to deterministic circuits in a layered form where inputs are at layer 0 and outputs are at layer $d$ ($d > 1$). They apply a sequence of sum-check protocols, one for each layer, where they transform a claim about values computed at layer $i$ to a claim about values computed at layer $i-1$ in the arithmetic 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. However, 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 $\mathcal{V}$ (in other words, these works cannot support of all of NP while achieving succinct communication costs). Besides generality, the lack of support for non-determinism requires using Boolean circuits instead of arithmetic circuits in practice (Boolean circuits are orders of magnitude more verbose than an equivalent arithmetic circuits for many high-level programs of interest). This is because highly efficient transformations from high-level programs to arithmetic circuits make extensive use of non-determinism for integer and bitwise operations [76, 83], storage [33, 79], and RAM [16, 21, 90].

Zhang et al. [94] extend doubly-efficient IPs to the complexity class NP by employing a polynomial commitment scheme [65, 75] and then transform their interactive argument to a zkSNARK [95] using prior transformations [13, 42]. However, their polynomial commitment scheme requires a trusted setup. In a different work, Wahby et al. [91] transform the Giraffe IP [89] (a doubly-efficient interactive proof in the GKR [53] line of work) into a zkSNARK without requiring a trusted setup. They rely on an optimized variant of prior transformations [13, 42] in conjunction with a new polynomial commitment scheme; the polynomial commitment scheme builds on ideas from Groth [55] and is tailored to a specific type of polynomials called multilinear polynomials (§3.2). However, the zkSNARKs of Wahby et al. [91] and Zhang et al. [95] retain the requirement of layered circuits in the original GKR protocol. Note that converting an arbitrary arithmetic circuit into a layered form increases its size quadratically in the worst case [53].

To achieve sub-linear verification costs, Zhang et al. [94] and Wahby et al. [91] focus on data-parallel computations where a circuit is composed of many identical sub-circuits. Unfortunately, this can pose severe restrictions in practice. To mitigate perils of such a requirement, Wahby et al. [91] design an irregular circuit layer, called a redistribution layer (RDL), that allows sharing witness values across different sub-circuits. Naturally, the verifier incurs linear costs for RDL. Concretely, in two out of their three benchmarks, the verifier's dominant cost is computation related to RDL.

To avoid the requirement of layered circuits in the GKR protocol [53], Kalai sketches "squashed GKR" [62]: 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 original layered circuit. To avoid the verifier materializing the witness, the proposal employs a commitment scheme for low-degree polynomials and low-degree tests. However, the proposal produces only designated verifier proofs and it relies on heavy-weight machinery (e.g., FHE) for the commitment scheme. A more recent version of the proposal [63] avoids prescribing a specific commitment scheme. Nevertheless, neither of the two proposals addresses the third sub-problem to achieve sub-linear verification costs for the verifier.

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.[9] Specifically, we observe that in the sum-check protocol (applied to a suitable low-degree polynomial $G(\cdot)$), $\mathcal{V}$ can delegate the required evaluation of $G(\cdot)$ at a random point in its domain to the prover $\mathcal{P}$ via a polynomial commitment scheme. Furthermore, if we use the low-degree polynomial from Blumberg et al. [29] as $G(\cdot)$, then, as we explain below, $\mathcal{V}$ only needs evaluations of multilinear polynomials at random points in their domain. Indeed, Spartan employs the commitment scheme for multilinear polynomials from Wahby et al. [91, §6.1], which provides the necessary building block without requiring a trusted setup.

### 5.3 A solution to sub-problem #1: encoding a circuit satisfiability instance as a low-degree polynomial

This section describes details of a low-degree polynomial $G(\cdot)$ from Blumberg et al. [29]. We borrow their exposition, but introduce small changes to their notation, phrasing, and terminology.

**Encoding arithmetic circuit satisfiability instances as functions.** Given an arithmetic circuit satisfiability instance $(\mathcal{C}, x, y)$, the following four function capture its structure. To introduce these functions, label gates in $\mathcal{C}$ with $\lceil \log |\mathcal{C}| \rceil$-bit identifiers, so that one can reference each gate in $\mathcal{C}$ uniquely with its identifier. Let $s = \lceil \log |\mathcal{C}| \rceil$.

The four functions are as follows: (1) add : $\{0,1\}^{3s} \to \{0,1\}$ denotes a function that takes as input three gate identifiers in $\mathcal{C}$ and outputs a Boolean; (2) mul : $\{0,1\}^{3s} \to \{0,1\}$ is defined similarly to add; (3) io : $\{0,1\}^{3s} \to \{0,1\}$ captures the public input/output gates of $\mathcal{C}$; and (4) $I_{x,y} : \{0,1\}^s \to \mathbb{F}$ captures $\mathcal{C}$'s public input $x$ and output $y$. Specifically:

[9] In theory, the work of Canetti et al [35] proposes 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. [8]) and builds a Merkle tree. Later, when verifying proof, the verifier (knowing only 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, while polynomial in the size of the circuit, is too expensive to be implemented in practice. In contrast, Spartan offers a more straightforward approach where the public computation is linear in the size of the circuit.

$$\text{add}(a,b,c) = \begin{cases} 1 & \text{if gate } a \text{ adds the outputs of gates } b \text{ and } c \\ 0 & \text{otherwise} \end{cases}$$

$$\text{mul}(a,b,c) = \begin{cases} 1 & \text{if gate } a \text{ multiplies the outputs of gates } b \text{ and } c \\ 0 & \text{otherwise} \end{cases}$$

$$\text{io}(a,b,c) = \begin{cases} 1 & \text{if } a \text{ is a public input gate, and } b = c = 0 \\ 1 & \text{if } a \text{ is an output gate, and } b \text{ and } c \text{ are in-neighbors of } a \\ 0 & \text{otherwise} \end{cases}$$

$$I_{x,y}(a) = \begin{cases} x_a & \text{if } a \text{ is a public input gate } (x_a \text{ refers to an element of } x \text{ assigned to } a) \\ y_a & \text{if } a \text{ is an output gate} \\ 0 & \text{otherwise} \end{cases}$$

**A function that encodes a purported satisfying assignment.** This section describes a function $F_{x,y}(\cdot)$ that can be used to encode a purported satisfying assignment $Z$ to $(\mathcal{C}, x, y)$ such that $F_{x,y}(\cdot)$ exhibits a desirable behavior (as detailed below) *if and only if* $Z$ is a satisfying assignment to $(\mathcal{C}, x, y)$.

Given a satisfying assignment $Z$ to $(\mathcal{C}, x, y)$, it is natural to think of $Z$ as a function that maps the identifier of a gate $g$ to the value assigned to $g$ in $Z$. That is, $Z : \{0,1\}^s \to \mathbb{F}$. Now, consider the following function:

$$F_{x,y}(a,b,c) = \text{io}(a,b,c) \cdot (I_{x,y}(a) - Z(a)) + \text{add}(a,b,c) \cdot (Z(a) - (Z(b) + Z(c))) + \text{mul}(a,b,c) \cdot (Z(a) - Z(b) \cdot Z(c))$$

**Lemma 5.1** (Blumberg et al. [29])**.** *$F_{x,y}(a,b,c) = 0$ for all $(a,b,c) \in \{0,1\}^{3s}$ if and only if $Z$ is a satisfying assignment to $(\mathcal{C}, x, y)$.*

*Proof.* If $Z$ is a satisfying assignment to $(\mathcal{C}, x, y)$ then it is easy to see that $F_{x,y}(a,b,c) = 0$ for all $(a,b,c) \in \{0,1\}^{3s}$. For the other direction (i.e., $Z$ is not a satisfying assignment to $(\mathcal{C}, x, y)$ yet $F_{x,y} = 0$ for all $(a,b,c) \in \{0,1\}^{3s}$), then there are five cases (and they all yield a contradiction).

1. If $a \in \{0,1\}^s$ is the identifier of a public input gate and if $Z(a) \neq x_a$, then $F_{x,y}(a,0,0) = I_{x,y}(a) - Z(a) = x_a - Z(a) \neq 0$.
2. If $a \in \{0,1\}^s$ is the identifier of a non-output addition gate with in-neighbors $b$ and $c$, and if $Z(a) \neq (Z(b) + Z(c))$, then $F_{x,y}(a,b,c) = Z(a) - (Z(b) + Z(c)) \neq 0$.
3. If $a \in \{0,1\}^s$ is the identifier of a non-output multiplication gate with in-neighbors $b$ and $c$, and if $Z(a) \neq Z(b) \cdot Z(c)$, then $F_{x,y}(a,b,c) = Z(a) - Z(b) \cdot Z(c) \neq 0$.
4. If $a \in \{0,1\}^s$ is the identifier of an output addition gate with in-neighbors $b$ and $c$, and if $y_a \neq (Z(b) + Z(c))$, then $F_{x,y}(a,b,c) = I_{x,y}(a) - Z(a) + (Z(a) - (Z(b) + Z(c))) = y_a - (Z(b) + Z(c)) \neq 0$.
5. If $a \in \{0,1\}^s$ is the identifier of an output multiplication gate with in-neighbors $b$ and $c$, and if $y_a \neq Z(b) \cdot Z(c)$, then $F_{x,y}(a,b,c) = I_{x,y}(a) - Z(a) + (Z(a) - Z(b) \cdot Z(c)) = y_a - Z(b) \cdot Z(c) \neq 0$.

□

Unfortunately $F_{x,y}(\cdot)$ is a function, *not* a polynomial, so it cannot be directly used in the sum-check protocol. But, it provides the necessary building block as described next.

**A polynomial that encodes a purported satisfying assignment.** Consider the following polynomial extension $\widetilde{F}_{x,y} : \mathbb{F}^{3s} \to \mathbb{F}$ of $F_{x,y}$, where $\widetilde{f}$ refers to a polynomial extension of a function $f$.

$$\begin{aligned} \widetilde{F}_{x,y}(u_1,u_2,u_3) = \widetilde{\text{io}}(u_1,u_2,u_3) \cdot (\widetilde{I}_{x,y}(u_1) - \widetilde{Z}(u_1)) + \\ \widetilde{\text{add}}(u_1,u_2,u_3) \cdot (\widetilde{Z}(u_1) - (\widetilde{Z}(u_2) + \widetilde{Z}(u_3))) + \\ \widetilde{\text{mul}}(u_1,u_2,u_3) \cdot (\widetilde{Z}(u_2) - \widetilde{Z}(u_2) \cdot \widetilde{Z}(u_3)) \end{aligned}$$

Since $\widetilde{F}_{x,y}(\cdot)$ is a low-degree polynomial over $\mathbb{F}$ in $3s$ variables, a verifier $\mathcal{V}$ could check if $\sum_{a,b,c\in\{0,1\}^{3s}} \widetilde{F}_{x,y}(a,b,c) = 0$ using the sum-check protocol with a prover $\mathcal{P}$. But, this is insufficient: the above sum being zero over the $3s$-dimensional Boolean hypercube does not imply that $F_{x,y}(a,b,c)$ is zero for all $(a,b,c) \in \{0,1\}^{3s}$. This is because the $2^{3s}$ terms in the sum might cancel each other making the final sum zero—even when some of the individual terms are not zero. The refinement below addresses this issue.

**A polynomial that sums to zero if it encodes a satisfying assignment.** Consider another polynomial $Q_{x,y}$ whose coefficients are evaluations of $\widetilde{F}_{x,y}$ over the Boolean hypercube $\{0,1\}^{3s}$. More specifically:

$$Q_{x,y}(t) = \sum_{u\in\{0,1\}^{3s}} \widetilde{F}_{x,y}(u) \cdot t^{\sum_{i=0}^{3s-1} u_i \cdot 2^i}, \text{ where } u_i \text{ is the } i\text{th bit of } u$$

Note that the bit-string $u \in \{0,1\}^{3s}$ is interpreted as concatenation of three $s$-bit elements and passed as a single element to $\widetilde{F}_{x,y}(\cdot)$. Observe that $Q_{x,y}(t)$ is a *zero-polynomial* (i.e., it evaluates to zero for all points in its domain) *if and only if* $\widetilde{F}_{x,y}$ evaluates to zero at all points in the $3s$-dimensional Boolean hypercube (and hence *if and only if* $\widetilde{F}_{x,y}$ encodes a satisfying assignment to $(\mathcal{C},x,y)$). To check if $Q_{x,y}(t)$ is a zero-polynomial, it suffices to check if $Q_{x,y}(\tau) = 0$ where $\tau \in_R \mathbb{F}$. This is because, any non-zero univariate polynomial of degree $d$ has at most $d$ roots.

**Rewriting $Q_{x,y}(\tau)$ in a form suitable for the sum-check protocol.** The core observation is that $\tau^{\sum_{i=0}^{3s-1} u_i \cdot 2^i}$ can be rewritten as a multilinear polynomial in variables $\{u_0,\ldots,u_{3s-1}\}$. Define a polynomial $S : \{0,1\}^{3s} \times \{0,1\}^{\lceil \log 3s \rceil} \to \mathbb{F}$ where $u_i$ refers to the $i$th bit of $u$.

$$S_\tau(u,i) = \begin{cases} \tau^{2^i} & \text{if } u_i = 1 \\ 1 & \text{otherwise} \end{cases}$$

$$S_\tau(u,i) = 1 + (\tau^{2^i} - 1) \cdot u_i$$

Notice that $\tau^{\sum_{i=0}^{3s-1} u_i \cdot 2^i} = \prod_{i=0}^{3s-1} S_\tau(u,i)$. Thus:

$$\begin{aligned} Q_{x,y}(\tau) &= \sum_{u\in\{0,1\}^{3s}} \widetilde{F}_{x,y}(u) \cdot \prod_{i=0}^{3s-1} S_\tau(u,i) \\ &= \sum_{u\in\{0,1\}^{3s}} G_{x,y,\tau}(u) \end{aligned}$$

$G_{x,y,\tau}(\cdot)$ is a low-degree multivariate polynomial in variables $\{u_0,u_1,\ldots,u_{3s-1}\}$. If multilinear extensions of add, mul, io, and $Z$ are used in $\widetilde{F}_{x,y}$, then $G_{x,y,\tau}(\cdot)$ is a low-degree polynomial with degree at most three in each variable (i.e., $m = 3s = 3\lceil \log |C| \rceil, \ell = 3$ in the terminology of the sum-check protocol in Section 3.3). Section 5.5 describes how $G_{x,y,\tau}(\cdot)$ gets used in the Spartan interactive argument.

### 5.4 A solution to sub-problem #2: making the communication cost of evaluating $G_{x,y,\tau}(\cdot)$ sub-linear in $|\mathcal{C}|$

The previous subsection established that for $\mathcal{V}$ to verify if an arithmetic circuit satisfiability instance $(\mathcal{C},x,y)$ is satisfiable, it can check if $\sum_{u\in\{0,1\}^{3s}} G_{x,y,\tau}(u) = 0$, where $G_{x,y,\tau}(\cdot)$ is a low-degree polynomial in $3s$ variables with a degree of at most 3 in each variable. If the verifier $\mathcal{V}$ uses the sum-check protocol to verify the above sum with the prover $\mathcal{P}$, then we need a mechanism for $\mathcal{V}$ to evaluate $G_{x,y,\tau}(\cdot)$ at a random point in its domain $(z_1,z_2,z_3) \in \mathbb{F}^{3s}$ (Figure 2, line 17)—without incurring $O(|\mathcal{C}|)$ communication from $\mathcal{P}$ to $\mathcal{V}$. Our solution to this problem is for $\mathcal{V}$ to employ a polynomial commitment scheme. We now discuss details.

**Details.** Recall that

$$G_{x,y,\tau}(u_1,u_2,u_3) = \widetilde{F}_{x,y}(u_1,u_2,u_3) \cdot \prod_{i=0}^{3s-1} S_\tau((u_1,u_2,u_3),i)$$

Thus, to evaluate $G_{x,y,\tau}(\cdot)$ at an arbitrary point in its domain $(z_1,z_2,z_3) \in \mathbb{F}^{3s}$, $\mathcal{V}$ must be able to evaluate $\widetilde{F}_{x,y}(\cdot)$ as the other terms in the expression can be evaluated locally without any information from $\mathcal{P}$. Now, recall that

$$\begin{aligned} \widetilde{F}_{x,y}(u_1,u_2,u_3) = {} & \widetilde{\text{io}}(u_1,u_2,u_3) \cdot (\widetilde{I}_{x,y}(u_1) - \widetilde{Z}(u_1)) + \\ & \widetilde{\text{add}}(u_1,u_2,u_3) \cdot (\widetilde{Z}(u_1) - (\widetilde{Z}(u_2) + \widetilde{Z}(u_3))) + \\ & \widetilde{\text{mul}}(u_1,u_2,u_3) \cdot (\widetilde{Z}(u_2) - \widetilde{Z}(u_2) \cdot \widetilde{Z}(u_3)) \end{aligned}$$

Given the above closed-form expression, to evaluate $\widetilde{F}_{x,y}(\cdot)$ at $(z_1, z_2, z_3) \in \mathbb{F}^{3s}$, $\mathcal{V}$ must evaluate $\widetilde{Z}(\cdot)$ at three different points: $z_1$, $z_2$, and $z_3$ where each $z_i \in \mathbb{F}^s$. This is because all other terms in the above expression for $\widetilde{F}_{x,y}(\cdot)$ can be computed locally by $\mathcal{V}$ using $(\mathcal{C}, x, y)$ without access to a purported satisfying assignment (Section 6 discusses how to reduce the cost of those evaluations to be sub-linear in $|\mathcal{C}|$). To evaluate $\widetilde{Z}(\cdot)$ at $z_1, z_2$, and $z_3$, $\mathcal{V}$ can of course receive a purported satisfying assignment from $\mathcal{P}$ *before* the sum-check protocol begins and then evaluate $\widetilde{Z}(\cdot)$ at three points in $O(|\mathcal{C}|)$ time, but this incurs $O(|\mathcal{C}|)$ communication, which we wish to avoid. We now discuss how a polynomial commitment scheme enables achieving succinct communication.

**Achieving succinct communication.** The core idea is simple: make $\mathcal{P}$ cryptographically commit to $\widetilde{Z}(\cdot)$ *before* the sum-check protocol begins using a *polynomial commitment scheme* [65, 75]. Informally speaking, a polynomial commitment scheme enables the following. A *sender* (e.g., $\mathcal{P}$) can send a commitment to a polynomial to a *receiver* (e.g., $\mathcal{V}$), and then later decommit evaluations of the committed polynomial at any point in its domain. Such schemes are *evaluation binding*: a probabilistic polynomial time sender cannot commit to one polynomial but decommit evaluations of a different polynomial at locations requested by the receiver (except for a small probability). Thus, in our context, $\mathcal{V}$ can request a commitment to $\widetilde{Z}(\cdot)$ before the sum-check protocol begins and then decommit evaluations of $\widetilde{Z}(\cdot)$ at the required three points $z_1, z_2$, and $z_3$ in the domain of $\widetilde{Z}(\cdot)$.

There exist many polynomial commitment schemes in the literature [65, 75, 94], but most of them require a trusted setup that produces a structured common reference string with a secret trapdoor (analogous to GGPR-style protocols), which we wish to avoid. Fortunately, Wahby et al. [91, §6.1] describe a polynomial commitment scheme tailored for multilinear polynomials in a related context. Their scheme, which is public coin and interactive, does not require a trusted setup nor makes non-standard assumptions. The non-interactive version of their scheme relies on the Fiat-Shamir heuristic [46], so it requires non-standard assumptions and is proven secure in the random oracle model. Nevertheless, both the interactive and non-interactive variants are consistent with our desired list of qualities (§1). Finally, although their commitment scheme applies only to multilinear polynomials, it is acceptable in our context. This is because, for efficiency, as in prior work [29, 40], we always use multilinear extensions (§3.2) in the construction of $G_{x,y,\tau}(\cdot)$, so $\widetilde{Z}(\cdot)$ is a multilinear polynomial in our context.

More formally, we consider a polynomial commitment scheme for multilinear polynomials with a tuple of four algorithms $\Pi_{pc} = (\mathsf{PolyGen}, \mathsf{PolyCommit}, \mathsf{PolyEval}, \mathsf{PolyEvalVerify})$. Without loss of generality, a multilinear polynomial over a finite field $\mathbb{F}$ in $m$ variables can be represented with a unique vector $Z \in \mathbb{F}^n$ where $n = 2^m$: $Z$ contains evaluations of the given multilinear polynomial at all the $2^m$ binary inputs in a canonical order. Our description of the four algorithms is tailored to this view of multilinear polynomials since the vector $Z$ is explicitly realized by $\mathcal{P}$ in our particular use case. Note that in the interactive variant of the polynomial commitment scheme, PolyEval and PolyEvalVerify are interactive protocols rather than local algorithms.

- $pp \leftarrow \mathsf{PolyGen}(1^\lambda, n)$: takes as input security parameter $\lambda$ and the size of the vector $Z$; produces a public parameter $pp$ (e.g., the description of a group $\mathbb{G}$ in which the discrete logarithm problem is hard and sequence of randomly chosen generators for $\mathbb{G}$). For simplicity, $pp$ includes $n$ as well.
- $\hat{Z} \leftarrow \mathsf{PolyCommit}(pp, Z)$: takes as input public parameters generated by the above algorithm and a vector $Z \in \mathbb{F}^n$ that uniquely represents the multilinear polynomial to be committed; produces a commitment $\hat{Z}$.
- $(v, \pi) \leftarrow \mathsf{PolyEval}(pp, Z, z)$: takes as input public parameters produced by PolyGen, a vector $Z \in \mathbb{F}^n$ representing a unique multilinear polynomial, and a point $z \in \mathbb{F}^m$ in the domain of the polynomial; outputs $v = \widetilde{Z}(z)$ where $v \in \mathbb{F}$, and a proof of correct evaluation $\pi$.
- $b \leftarrow \mathsf{PolyEvalVerify}(pp, \hat{Z}, z, v, \pi)$: takes as input public parameters generated by PolyGen, a commitment to a multilinear polynomial produced by PolyCommit, $z \in \mathbb{F}^m$, $v \in \mathbb{F}$, and a proof $\pi$; verifies if $v$ is the correct evaluation of the polynomial underneath the commitment $\hat{Z}$ using $pp$, $\pi$, and $\hat{Z}$; outputs $b = 1$ denoting `accept` or $b = 0$ indicating `reject`.

**Definition 5.2.** A tuple of four algorithms $\Pi_{pc} = (\mathsf{PolyGen}, \mathsf{PolyCommit}, \mathsf{PolyEval}, \mathsf{PolyEvalVerify})$ is a polynomial commitment scheme for multilinear polynomials over a finite field $\mathbb{F}$ if the following conditions hold.

- **Completeness.** For any multilinear polynomial represented by a vector $Z \in \mathbb{F}^n$ and security parameter $\lambda$,

$$\Pr\left\{ \begin{array}{c} pp \leftarrow \mathsf{PolyGen}(1^\lambda, n, c); \hat{Z} \leftarrow \mathsf{PolyCommit}(pp, Z); (v, \pi) \leftarrow \mathsf{PolyEval}(pp, Z, z): \\ \mathsf{PolyEvalVerify}(pp, \hat{Z}, z, v, \pi) = 1 \wedge v = \widetilde{Z}(z) \end{array} \right\} \geq 1 - \mathrm{negl}(\lambda)$$

- **Soundness.** For any probabilistic polynomial time adversary $\mathcal{A}$, size parameter $n \geq 1$, and security parameter $\lambda$,

$$\Pr\left\{ \begin{array}{c} pp \leftarrow \mathsf{PolyGen}(1^\lambda, n, c);\ (Z, z, v, \pi) = \mathcal{A}(1^\lambda, pp);\ \hat{Z} \leftarrow \mathsf{PolyCommit}(pp, Z): \\ \mathsf{PolyEvalVerify}(pp, \hat{Z}, z, v, \pi) = 1 \wedge v \neq \widetilde{Z}(z) \end{array} \right\} \leq \mathrm{negl}(\lambda)$$

**Lemma 5.2.** *There exists a polynomial commitment scheme for multilinear polynomials with perfect completeness and soundness where for a multilinear polynomial represented by a vector $Z \in \mathbb{F}^n$, PolyCommit produces a commitment of size $O(n^{1/c})$, PolyEval produces a proof of size $O(n^{1/c})$, and PolyEvalVerify runs in time $O(n^{1-1/c})$.*

*Proof.* The desired result is implied by more general version of the lemma [91, Lemma 5]. □

Additionally, Wahby et al. [91] prove that their polynomial commitment scheme satisfies a stronger notion of soundness called generalized special soundness. Specifically, they prove that once the prover in their scheme commits to a multilinear polynomial by sending a commitment $\hat{Z}$, there exists an extractor algorithm that, given oracle access to the prover, can extract the underlying vector $Z$ (and hence the entire polynomial $\widetilde{Z}(\cdot)$ and thus the purported satisfying assignment in our context). We note that our interactive argument, which employs their commitment scheme for $\widetilde{Z}(\cdot)$, satisfies a stronger notion of knowledge soundness (this can be proved by adapting their proof that their interactive argument satisfies knowledge soundness).

### 5.5 The Spartan succinct interactive argument

We now provide an end-to-end description of the Spartan interactive argument. The verifier $\mathcal{V}$ and the prover $\mathcal{P}$ start with an arithmetic circuit satisfiability instance $(\mathcal{C}, x, y)$ and $\mathcal{P}$'s goal is to convince $\mathcal{V}$ that it there is a satisfying assignment $Z$ to $(\mathcal{C}, x, y)$, which from Lemma 3.1 implies that there exists a witness $w$ such that $\mathcal{C}(x, w) = y$. The following inline figure depicts the entire interactive argument protocol.

**Inputs to both $\mathcal{V}$ and $\mathcal{P}$:** a circuit satisfiability instance $(\mathcal{C}, x, y)$ and public parameters $pp$ for the polynomial commitment scheme produced by $\mathsf{PolyGen}(1^\lambda, n, c)$ for security parameter $\lambda$, $n = |\mathcal{C}|$, and a constant $c \geq 2$.

**$\mathcal{P}$'s input:** a satisfying assignment $Z$ to $(\mathcal{C}, x, y)$

**$\mathcal{V}$'s output:** `accept` or `reject` indicating its decision.

**Interactive protocol between $\mathcal{P}$ and $\mathcal{V}$.**

- $\mathcal{P} \rightarrow \mathcal{V}$: $\mathcal{P}$ sends to $\mathcal{V}$ a commitment $\hat{Z}$ to the unique multilinear polynomial $\widetilde{Z}(\cdot)$ using the PolyCommit algorithm discussed in Section 5.4. $\mathcal{V}$ outputs `reject` if the received commitment is not well-formed.
- $\mathcal{V} \rightarrow \mathcal{P}$: $\mathcal{V}$ samples a random $\tau \in_R \mathbb{F}$ and sends $\tau$ to $\mathcal{P}$.
- $\mathcal{V} \leftrightarrow \mathcal{P}$: $\mathcal{V}$ and $\mathcal{P}$ execute the sum-check protocol using the polynomial $G_{x,y,\tau}(\cdot)$ from Section 5.3 with $H = 0, m = 3s$, and $\ell = 3$. On line 17 of the sum-check protocol (Figure 2), $\mathcal{V}$ must evaluate $G_{x,y,\tau}(\cdot)$ at $r = (r_1, \ldots, r_{3s}) = (z_1, z_2, z_3) \in \mathbb{F}^{3s}$ (where each $z_i \in \mathbb{F}^s$) and check if $G_{x,y,\tau}(r) = e$, where $e \in \mathbb{F}$. This is done as follows.
  - $\mathcal{V} \rightarrow \mathcal{P}$: $\mathcal{V}$ sends to $\mathcal{P}$ three points $z_1, z_2$, and $z_3$ where each $z_i \in \mathbb{F}^m$ and $m = \log n$.
  - $\mathcal{P} \rightarrow \mathcal{V}$: $\mathcal{P}$ runs $(v_i, \pi_i) \leftarrow \mathsf{PolyEval}(pp, Z, z_i)$ for all $i \in \{1, 2, 3\}$ and sends to $\mathcal{V}$ three evaluations along with three proofs $(v_i, \pi)$ for all $i \in \{1, 2, 3\}$.
  - $\mathcal{V}$ runs $b_i \leftarrow \mathsf{PolyEvalVerify}(pp, \hat{Z}, z_i, v_i, \pi_i)$ for all $i \in \{1, 2, 3\}$ outputting `reject` if any $b_i = 0$. (Section 5.6 reduces the number of decommits from three to one.)
  - $\mathcal{V}$ locally evaluates:
    $v_{x,y} \leftarrow \widetilde{I}_{x,y}(z_1)$
    $v_{io} \leftarrow \widetilde{io}(z_1, z_2, z_3)$
    $v_{add} \leftarrow \widetilde{add}(z_1, z_2, z_3)$
    $v_{mul} \leftarrow \widetilde{mul}(z_1, z_2, z_3)$
    (The latter three evaluations take time $O(|\mathcal{C}|)$; Section 6 makes this sub-linear in $|\mathcal{C}|$.)

- $\mathcal{V}$ locally checks and returns `accept` if the following condition holds (otherwise $\mathcal{V}$ returns `reject`):
  $e = (v_{io} \cdot (v_{x,y} - v_1) + v_{add} \cdot (v_1 - (v_2 + v_3) + v_{mul} \cdot (v_1 - v_2 \cdot v_3)) \cdot \prod_{i=0}^{3s-1} S_\tau(u,i)$

The following theorem establishes that Spartan is an interactive argument.

**Theorem 5.1.** *The above protocol is an interactive argument (Definition 5.1) under the discrete logarithm assumption with $\epsilon_c = 0$ and $\epsilon_s = |\mathcal{C}|^3/|\mathbb{F}| + \ell \cdot m/|\mathbb{F}| + negl(\lambda)$, where $\ell = 3$ and $m = 3\log|\mathcal{C}|$, $negl(\cdot)$ is a negligible function, and $\lambda$ is the security parameter.*

*Proof.* Perfect completeness follows from the perfect completeness of the sum-check protocol and of the commitment scheme for multilinear polynomials.

We now prove soundness and show that $\epsilon_s$ is as claimed. Suppose a probabilistic polynomial time prover $\mathcal{P}^*$ commits to a multilinear polynomial $\widetilde{Z}(\cdot)$ before the sum-check protocol begins. There are two cases. The first is when the following event $E$ happens: $v_1 = \widetilde{Z}(z_1) \wedge v_2 = \widetilde{Z}(z_2) \wedge v_3 = \widetilde{Z}(z_3)$ (i.e., $v_i = \widetilde{Z}(z_i)$ for any $z_i$ in the domain of $\widetilde{Z}(\cdot)$). The second case is the event $\neg E$, which denotes the complement of $E$.

By standard laws of probability, we have the following. For any non-satisfiable arithmetic circuit satisfiability instance $X = (\mathcal{C}, x, y)$, any probabilistic polynomial time prover $\mathcal{P}^*$, and for all $w, r \in \{0,1\}^*$,

$$\begin{aligned}\Pr\{\langle \mathcal{P}^*(w), \mathcal{V}(r)\rangle(X) = \texttt{accept}\} &= \Pr\{(\langle \mathcal{P}^*(w), \mathcal{V}(r)\rangle(X) = \texttt{accept}) \wedge E\} + \Pr\{(\langle \mathcal{P}^*(w), \mathcal{V}(r)\rangle(X) = \texttt{accept}) \wedge \neg E\} \\ &\leq \Pr\{(\langle \mathcal{P}^*(w), \mathcal{V}(r)\rangle(X) = \texttt{accept})|E\} + \Pr\{\neg E\} \\ &\leq (|\mathcal{C}|^3/|\mathbb{F}| + 9\log|\mathcal{C}|/|\mathbb{F}|) + \text{negl}(\lambda), \text{ (lemmas 5.3 and 5.2)}\end{aligned}$$

We now prove a supporting lemma.

**Lemma 5.3.** *For any non-satisfiable arithmetic circuit satisfiability instance $X = (\mathcal{C}, x, y)$, any probabilistic polynomial time prover $\mathcal{P}^*$, and for all $w, r \in \{0,1\}^*$, $\Pr\{\langle \mathcal{P}^*(w), \mathcal{V}(r)\rangle(X) = \texttt{accept}|E\} \leq |\mathcal{C}|^3/|\mathbb{F}| + \ell \cdot m/|\mathbb{F}|$, where $\ell = 3$ and $m = 3\log|\mathcal{C}|$.*

*Proof.* The case where $E$ happens is analogous to the one where $\mathcal{V}$ has an oracle access to a multilinear polynomial $\widetilde{Z}(\cdot)$ and hence to the polynomial $G_{x,y,\tau}(\cdot)$. We analyze this case as follows.

If $(\mathcal{C}, x, y)$ is not satisfiable, then $Q_{x,y}(t)$ is not a zero-polynomial. By the Schwartz-Zippel lemma, $Q_{x,y}(t) = 0$ for at most $d/|\mathbb{F}|$ values of $t$ in the domain of $Q_{x,y}(\cdot)$, where $d$ is the degree of $Q_{x,y}(\cdot)$. In our context, $d = 2^{3s} = 2^{3\log s} = |\mathcal{C}|^3$, so $Q_{x,y}(\cdot)$ is zero for at most $|\mathcal{C}|/|\mathbb{F}|$ points in the domain of $Q_{x,y}(\cdot)$.

There are two cases to consider. First, if $\mathcal{V}$ chooses a $\tau \in \mathbb{F}$ where $Q_{x,y}(\tau) = 0$, then the Spartan interactive argument protocol may make the verifier to incorrectly output `accept`. Second, If $\mathcal{V}$ chooses a $\tau \in \mathbb{F}$ where $Q_{x,y}(\tau) \neq 0$, then a malicious prover begins with a false claim in the sum-check protocol. By the soundness of the sum-check protocol, the malicious prover succeeds with probability at most $\ell \cdot m/|\mathbb{F}|$, where $m = 3s = 3\log|\mathcal{C}|$ and $\ell = 3$ and the probability is over $\mathcal{V}$'s randomness. Given these two cases, we can use a standard union bound to establish the desired result. □

□

### 5.6 Reducing the number of polynomial decommits

In the above interactive argument protocol, $\mathcal{V}$ must decommit $\widetilde{Z}(\cdot)$ at three points in its domain. We now discuss a prior interactive protocol [29] to reduce this to one. Note that a similar technique appears in doubly-efficient interactive proofs to reduce two evaluations to one [40, 53, 87, 89, 94]. The reduction is as follows.

Suppose $\gamma(t) : \mathbb{F} \to \mathbb{F}^s$ denotes the degree-2 curve that passes through the points at which $\mathcal{V}$ wishes to evaluate $\widetilde{Z}(\cdot)$ i.e., $\gamma(0) = z_1, \gamma(1) = z_2, \gamma(2) = z_3$. Through standard interpolation, $\gamma$ is defined by the following closed form:

$$\gamma(t) = 2^{-1}(t-2)(t-1) \cdot z_1 - (t-2)t \cdot z_2 + 2^{-1}t(t-1) \cdot z_3$$

1. $\mathcal{P}$ sends to $\mathcal{V}$ three evaluations of $\widetilde{Z}(\cdot)$ at $z_1, z_2$, and $z_3$—*without* any proofs. Of course, $\mathcal{P}$ can respond with *any* values (denote them as $v_1, v_2$, and $v_3$ that are purported to be evaluations of $\widetilde{Z}(\cdot)$ at $z_1, z_2$, and $z_3$ respectively).

2. $\mathcal{V}$ asks $\mathcal{P}$ to send a degree-$2s$ univariate polynomial $K^*$ where a correct prover would send $\widetilde{Z}(\cdot)$ restricted to $\gamma(t)$ i.e., $K^* = \widetilde{Z} \circ \gamma$ ($K^*$ can be represented with $2s+1$ elements in $\mathbb{F}$).
3. $\mathcal{V}$ checks that $K^*$ agrees with its definition i.e., $K^*(0)=v_1, K^*(1)=v_2, K^*(2)=v_3$. If all these checks pass, $\mathcal{V}$ samples $\rho \in_R \mathbb{F}$ and sets $z_4 = \gamma(\rho)$ and $v_4 = K^*(\rho)$.

The above protocol provides the following guarantee to $\mathcal{V}$, where the probability is over $\mathcal{V}$'s random choices. Except for a probability $2s/|\mathbb{F}|$, $\mathcal{V}$ can assume that $\widetilde{Z}(z_i) = v_i$ for all $1 \leq i \leq 3$ as long as it can verify that $\widetilde{Z}(z_4) = v_4$. Thus, instead of verifying three evaluations of $\widetilde{Z}(\cdot)$ via the polynomial commitment scheme, $\mathcal{V}$ verifies if $\widetilde{Z}(z_4) = v_4$. Figure 3 depicts the Spartan interactive argument protocol that employs this optimization.

# 6 Computation commitments: A technique to achieve sub-linear verification time

This section describes a scheme to create a short cryptographic commitment to the description of an arithmetic circuit. We refer to such commitments as *computation commitments*. The primary application of this scheme is to make the verifier's costs in Spartan's interactive argument (§5) to be sub-linear in $|\mathcal{C}|$, thereby addressing the third sub-problem; so, we describe computation commitments in that context. We note that this scheme can be used to achieve sub-linear verification costs in Hyrax [91] and the protocol of Zhang et al. [94, 95]—even when circuits do not have data-parallelism or other structure. However, it may less profitable in the context of those systems, on concrete terms, because a separate commitment must be created for each layer of an arithmetic circuit.

## 6.1 Motivation and scheme

Recall from Section 5.3 that an arithmetic circuit can be encoded as four functions: add, mul, io, and $I_{x,y}$. We now examine the cost of evaluating these functions as part of verifying a proof.

Of these four functions, only the last one, $I_{x,y}$, depends on the concrete values of the public inputs and outputs of an arithmetic circuit satisfiability instance $(\mathcal{C}, x, y)$. Since $\widetilde{I}_{x,y}(\cdot)$ is a multilinear extension of $I_{x,y}(\cdot)$, the cost of evaluating $\widetilde{I}_{x,y}(\cdot)$ at any point in its domain is linear in $|x| + |y|$ [87]. Thus, the verifier can afford to evaluate $\widetilde{I}_{x,y}(\cdot)$ at a random point in its domain when verifying a proof for $(\mathcal{C}, x, y)$.

The other three functions depend only on the structure of the arithmetic circuit $\mathcal{C}$ (i.e., they do not depend on $x$, $y$, or a purported satisfying assignment to $(\mathcal{C}, x, y)$). However, evaluating $\widetilde{\text{add}}(\cdot)$, $\widetilde{\text{mul}}(\cdot)$, and $\widetilde{\text{io}}(\cdot)$ at a point $(z_1, z_2, z_3)$ in their domain (lines 26–28 in Figure 3) incurs $O(|\mathcal{C}|)$ cost. The $O(|\mathcal{C}|)$ cost is unavoidable if the circuit $\mathcal{C}$ has no structure such as data-parallelism or other homogeneity. We now discuss a new primitive, called *computation commitments*, that a verifier can use to verifiably outsource this work to the prover by incurring a one-time cost of $O(|\mathcal{C}|)$. We emphasize that the one-time computation by the verifier is a public computation, so there are no secret trapdoors.

Computation commitments is a set of four algorithms: $\Pi_{cc} = (\mathsf{ComputationGen}, \mathsf{ComputationCommit}, \mathsf{ComputationDecommit}, \mathsf{ComputationDecommitVerify})$.

1. $pp \leftarrow \mathsf{ComputationGen}(1^\lambda, n)$: On input security parameter $\lambda$ and an upper bound on the circuit size $n$, it produces public parameters $pp$.
2. $\hat{\mathcal{C}} \leftarrow \mathsf{ComputationCommit}(pp, \mathcal{C})$: Produces a short commitment to $\mathcal{C}$ using public parameters $pp$.
3. $(v, \pi) \leftarrow \mathsf{ComputationDecommit}(pp, \mathcal{C}, z)$: Produces an evaluation $v$ along with a proof $\pi$ when given with public parameters $pp$, a circuit $\mathcal{C}$, and a location $z$ in the domain of $\hat{C}$.
4. $b \leftarrow \mathsf{ComputationDecommitVerify}(pp, \hat{\mathcal{C}}, z, v, \pi)$: Outputs `true` if $v$ is the correct evaluation of object underneath commitment $\hat{\mathcal{C}}$ at location $z$ using the supplied proof $\pi$ and public parameters $pp$.

We now discuss a concrete scheme for computation commitments. We assume a polynomial commitment scheme for multilinear polynomials $\Pi_{pc}$ (the scheme from the prior section suffices for our purposes).

1. $pp \leftarrow \mathsf{ComputationGen}(1^\lambda, n)$: Invoke $\Pi_{pc}.\mathsf{PolyGen}(1^\lambda, n, c)$ and output public parameters $pp$ produced by the call.
2. $\hat{\mathcal{C}} \leftarrow \mathsf{ComputationCommit}(pp, \mathcal{C})$: Encode the circuit $\mathcal{C}$ into three functions $\text{add}(\cdot), \text{mul}(\cdot), \text{io}(\cdot)$, which in turn determine three multilinear polynomials $\widetilde{\text{add}}(\cdot), \widetilde{\text{mul}}(\cdot), \widetilde{\text{io}}(\cdot)$; call $\Pi_{pc}.\mathsf{PolyCommit}$ thrice once for each multilinear polynomial and output a vector of polynomial commitments. $\hat{\mathcal{C}} = (\hat{\text{add}}, \hat{\text{mul}}, \hat{\text{io}})$.
3. $(v, \pi) \leftarrow \mathsf{ComputationDecommit}(pp, \mathcal{C}, z)$: Invoke $\Pi_{pc}.\mathsf{PolyEval}$ thrice once for each multilinear polynomial in $\hat{\mathcal{C}} = (\hat{\text{add}}, \hat{\text{mul}}, \hat{\text{io}})$ at location $z$; output evaluations of the three multilinear polynomials along with proofs for each i.e., $v = (v_{add}, v_{mul}, v_{io})$ and $\pi = (\pi_{add}, \pi_{mul}, \pi_{io})$.

4. $b \leftarrow \mathsf{ComputationDecommitVerify}(pp, \hat{\mathcal{C}}, z, v, \pi)$: Invoke $\Pi_{pc}.\mathsf{PolyEvalVerify}$ thrice once for each of $\hat{C} = (\hat{\text{add}}, \hat{\text{mul}}, \hat{\text{io}})$ at location $z$ and the corresponding evaluation and proof; output `reject` if any invocation outputs `reject`.

It is straightforward to show that the above commitment scheme provides a standard notion of binding by essentially proving a reduction to the binding property of the underlying polynomial commitment scheme.

### 6.2 Using computation commitments in the Spartan interactive argument

We now discuss how the above commitment scheme gets used in the Spartan interactive argument.

**Creating computation commitments.** $\mathcal{V}$ calls $\mathsf{ComputationGen}(1^\lambda, n)$ to create public parameters $pp$. Then, when a verifier $\mathcal{V}$ reads the description of an arithmetic circuit $\mathcal{C}$, it calls $\mathsf{ComputationCommit}(pp, \mathcal{C})$, which outputs a commitment $\hat{\mathcal{C}}$. With the aforementioned choice of the polynomial commitment scheme, the total size of the computation commitment is $O(|\mathcal{C}|^{1/c})$ for any $c \geq 2$, which is of size that is sub-linear in $O(|\mathcal{C}|)$. And, the verifier incurs $O(|\mathcal{C}|)$ time to create such a commitment. However, as we discuss below, this one-time cost is amortized over *all* future verification of proofs associated with $\mathcal{C}$. Note that in practice, these commitments can be created at the time the verifier compiles a high-level program to a circuit satisfiability instance.

**Using computation commitments.** When participating in the Spartan interactive argument protocol, the verifier starts with a computation commitment to $\mathcal{C}$, say $\hat{\mathcal{C}}$. It must evaluate the polynomials underneath the commitment at location $z = (z_1, z_2, z_3)$ in the Spartan interactive argument. To assist the verifier, the prover invokes $\mathsf{ComputationDecommit}(pp, \mathcal{C}, z)$ to produce an evaluation along with a proof. The verifier then invokes $\mathsf{ComputationDecommitVerify}$ and outputs `reject` if the innovation returns `reject`. If not, the verifier uses the evaluation in the rest of the protocol. The verifier incurs an additional $O(|\mathcal{C}|^{1-1/c})$ time to invoke $\mathsf{ComputationDecommitVerify}$, and the prover, which is in charge of providing the evaluation, incurs $O(|\mathcal{C}|)$ cost to execute $\mathsf{ComputationDecommit}$.

Figure 5 depicts the Spartan succinct interactive argument that uses computation commitments to achieve sub-linear verification costs. It is straightforward to show that the modified protocol provides perfect completeness and soundness as in the previous section.

## 7 Discussion

**Achieving post-quantum security.** A reasonable criticism for this work is that Ligero [3], zkSTARKs [14], and Aurora [19] are plausibly post-quantum secure (as they rely only on symmetric key primitives), but Spartan as described in this paper is not. However, as detailed in Section 5, Spartan is based on the sum-check protocol (which is unconditionally secure) and a commitment scheme for multilinear polynomials (which is secure under the hardness of computing discrete logarithms). Thus, the cryptographic hardness assumptions made by Spartan stem from the latter primitive. Recent work by Baum et al. [10] proposes a lattice-based zero-knowledge argument protocol that is plausibly post-quantum secure. Although their verifier does not achieve sub-linear costs, we can substitute the DLOG-based primitives in the commitment scheme (used in Spartan) with one based on their scheme. In principle, this makes it possible to obtain a post-quantum secure variant of Spartan. We leave it to future work to work out all details.

## Acknowledgments

Careful comments from Sebastian Angel, Melissa Chase, Esha Ghosh, Satya Lokam, and Jonathan Lee helped improve this paper. Special thanks to Justin Thaler for his thorough comments, pointers to related work, and ideas to improve this work further.

## References


[1] Decentralized identity foundation. `https://identity.foundation/`.

[2] Verifiable credentials. `https://www.w3.org/TR/verifiable-claims-data-model/`.

[3] S. Ames, C. Hazay, Y. Ishai, and M. Venkitasubramaniam. Ligero: Lightweight sublinear arguments without a trusted setup. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, 2017.

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

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

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

[7] L. Babai. Trading group theory for randomness. In *Proceedings of the ACM Symposium on Theory of Computing (STOC)*, pages 421–429, 1985.

[8] L. Babai, L. Fortnow, L. A. Levin, and M. Szegedy. Checking computations in polylogarithmic time. In *Proceedings of the ACM Symposium on Theory of Computing (STOC)*, 1991.

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

[10] C. Baum, J. Bootle, A. Cerulli, R. del Pino, J. Groth, and V. Lyubashevsky. Sub-linear lattice-based zero-knowledge arguments for arithmetic circuits. In *Proceedings of the International Cryptology Conference (CRYPTO)*, 2018.

[11] S. Bayer and J. Groth. Efficient zero-knowledge argument for correctness of a shuffle. In *Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, Apr. 2012.

[12] M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, pages 62–73, 1993.

[13] M. Ben-Or, O. Goldreich, S. Goldwasser, J. Håstad, J. Kilian, S. Micali, and P. Rogaway. Everything provable is provable in zero-knowledge. In *Proceedings of the International Cryptology Conference (CRYPTO)*, pages 37–56, 1988.

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

[15] E. Ben-Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza. Zerocash: Decentralized anonymous payments from Bitcoin. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, 2014.

[16] E. Ben-Sasson, A. Chiesa, D. Genkin, and E. Tromer. Fast reductions from RAMs to delegatable succinct constraint satisfaction problems: Extended abstract. In *Proceedings of the Innovations in Theoretical Computer Science (ITCS) Conference*, 2013.

[17] E. Ben-Sasson, A. Chiesa, D. Genkin, and E. Tromer. On the concrete efficiency of probabilistically-checkable proofs. In *Proceedings of the ACM Symposium on Theory of Computing (STOC)*, pages 585–594, 2013.

[18] E. Ben-Sasson, A. Chiesa, D. Genkin, E. Tromer, and M. Virza. SNARKs for C: Verifying program executions succinctly and in zero knowledge. In *Proceedings of the International Cryptology Conference (CRYPTO)*, Aug. 2013.

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

[20] E. Ben-Sasson, A. Chiesa, and N. Spooner. Interactive Oracle Proofs. In *Theory of Cryptography Conference (TCC)*, 2016.

[21] E. Ben-Sasson, A. Chiesa, E. Tromer, and M. Virza. Scalable zero knowledge via cycles of elliptic curves. In *Proceedings of the International Cryptology Conference (CRYPTO)*, 2014.

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

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

[24] E. Ben-Sasson and M. Sudan. Simple PCPs with poly-log rate and query complexity. In *Proceedings of the ACM Symposium on Theory of Computing (STOC)*, pages 266–275, 2005.

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

[26] N. Bitansky, R. Canetti, A. Chiesa, and E. Tromer. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In *Proceedings of the Innovations in Theoretical Computer Science (ITCS) Conference*, 2012.

[27] N. Bitansky and A. Chiesa. Succinct arguments from multi-prover interactive proofs and their efficiency benefits. In *Proceedings of the International Cryptology Conference (CRYPTO)*, 2012.

[28] N. Bitansky, A. Chiesa, Y. Ishai, O. Paneth, and R. Ostrovsky. Succinct non-interactive arguments via linear interactive proofs. In *Theory of Cryptography Conference (TCC)*, 2013.

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

[30] J. Bootle, A. Cerulli, P. Chaidos, J. Groth, and C. Petit. Efficient zero-knowledge arguments for arithmetic circuits in the discrete log setting. In *Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, 2016.

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

[32] G. Brassard, D. Chaum, and C. Crépeau. Minimum disclosure proofs of knowledge. *J. Comput. Syst. Sci.*, 37(2):156–189, Oct. 1988.
[33] B. Braun, A. J. Feldman, Z. Ren, S. Setty, A. J. Blumberg, and M. Walfish. Verifying computations with state. In *Proceedings of the ACM Symposium on Operating Systems Principles (SOSP)*, 2013.
[34] B. Bnz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, 2018.
[35] R. Canetti, B. Riva, and G. N. Rothblum. Two protocols for delegation of computation. In *Proceedings of the 6th International Conference on Information Theoretic Security (ICITS)*, pages 37–61, 2012.
[36] J. P. M. Chase. ZSL Proof of Concept. `https://github.com/jpmorganchase/quorum/wiki/ZSL`, 2017.
[37] M. Chase, D. Derler, S. Goldfeder, C. Orlandi, S. Ramacher, C. Rechberger, D. Slamanig, and G. Zaverucha. Post-quantum zero-knowledge and signatures from symmetric-key primitives. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, 2017.
[38] A. Chiesa, M. A. Forbes, and N. Spooner. A zero knowledge sumcheck and its applications. *CoRR*, abs/1704.02086, 2017.
[39] A. Chiesa, E. Tromer, and M. Virza. Cluster computing in zero knowledge. In *Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, 2015.
[40] G. Cormode, M. Mitzenmacher, and J. Thaler. Practical verified computation with streaming interactive proofs. In *Proceedings of the Innovations in Theoretical Computer Science (ITCS) Conference*, 2012.
[41] C. Costello, C. Fournet, J. Howell, M. Kohlweiss, B. Kreuter, M. Naehrig, B. Parno, and S. Zahur. Geppetto: Versatile verifiable computation. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, May 2015.
[42] R. Cramer and I. Damgård. Zero-knowledge proofs for finite field arithmetic, or: Can zero-knowledge be for free? In *Proceedings of the International Cryptology Conference (CRYPTO)*, pages 424–441, 1998.
[43] 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 *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, 2016.
[44] I. Dinur. The PCP theorem by gap amplification. *Journal of the ACM (JACM)*, 54(3), June 2007.
[45] U. Feige, S. Goldwasser, L. Lovász, S. Safra, and M. Szegedy. Interactive proofs and the hardness of approximating cliques. *Journal of the ACM (JACM)*, 43(2):268–292, Mar. 1996.
[46] A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Proceedings of the International Cryptology Conference (CRYPTO)*, pages 186–194, 1986.
[47] D. Fiore, C. Fournet, E. Ghosh, M. Kohlweiss, O. Ohrimenko, and B. Parno. Hash first, argue later: Adaptive verifiable computations on outsourced data. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, 2016.
[48] R. Gennaro, C. Gentry, and B. Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In *Proceedings of the International Cryptology Conference (CRYPTO)*, 2010.
[49] R. Gennaro, C. Gentry, B. Parno, and M. Raykova. Quadratic span programs and succinct NIZKs without PCPs. In *Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, 2013.
[50] C. Gentry. *A fully homomorphic encryption scheme*. PhD thesis, Stanford University, 2009.
[51] C. Gentry and D. Wichs. Separating succinct non-interactive arguments from all falsifiable assumptions. In *Proceedings of the ACM Symposium on Theory of Computing (STOC)*, pages 99–108, 2011.
[52] I. Giacomelli, J. Madsen, and C. Orlandi. ZKBoo: Faster zero-knowledge for Boolean circuits. In *Proceedings of the USENIX Security Symposium*, 2016.
[53] S. Goldwasser, Y. T. Kalai, and G. N. Rothblum. Delegating computation: Interactive proofs for muggles. In *Proceedings of the ACM Symposium on Theory of Computing (STOC)*, 2008.
[54] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof-systems. In *Proceedings of the ACM Symposium on Theory of Computing (STOC)*, 1985.
[55] J. Groth. Linear algebra with sub-linear zero-knowledge arguments. In *Proceedings of the International Cryptology Conference (CRYPTO)*, 2009.
[56] J. Groth. Short pairing-based non-interactive zero-knowledge arguments. In *Proceedings of the International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT)*, 2010.
[57] J. Groth. On the size of pairing-based non-interactive arguments. In *Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, 2016.
[58] J. Groth, M. Kohlweiss, M. Maller, S. Meiklejohn, and I. Miers. Updatable and universal common reference strings with applications to zk-SNARKs. In *Proceedings of the International Cryptology Conference (CRYPTO)*, 2018.
[59] J. Håstad. Some optimal inapproximability results. In *Proceedings of the ACM Symposium on Theory of Computing (STOC)*, pages 1–10, 1997.
[60] Y. Ishai, E. Kushilevitz, and R. Ostrovsky. Efficient arguments without short PCPs. In *IEEE Conference on Computational Complexity*, 2007.
[61] Y. Ishai, E. Kushilevitz, R. Ostrovsky, and A. Sahai. Zero-knowledge from secure multiparty computation. In *Proceedings of*

*the ACM Symposium on Theory of Computing (STOC)*, pages 21–30, 2007.

[62] 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.

[63] Y. T. Kalai. GKR based zero-knowledge proofs. ZKProof Standards workshop, `https://www.youtube.com/watch?v=x8pUxFptfb0`, Apr. 2019.

[64] Y. T. Kalai and R. Raz. Interactive PCP. In *Proceedings of the International Colloquium on Automata, Languages and Programming (ICALP)*, pages 536–547, 2008.

[65] A. Kate, G. M. Zaverucha, and I. Goldberg. Constant-size commitments to polynomials and their applications. In *Proceedings of the International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT)*, pages 177–194, 2010.

[66] J. Kilian. A note on efficient zero-knowledge proofs and arguments (extended abstract). In *Proceedings of the ACM Symposium on Theory of Computing (STOC)*, 1992.

[67] A. Kosba, A. Miller, E. Shi, Z. Wen, and C. Papamanthou. Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, 2016.

[68] A. Kosba, C. Papamanthou, and E. Shi. xJsnark: A framework for efficient verifiable computation. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, 2018.

[69] H. Lipmaa. Progression-free sets and sublinear pairing-based non-interactive zero-knowledge arguments. In *Theory of Cryptography Conference (TCC)*, 2012.

[70] C. Lund, L. Fortnow, H. Karloff, and N. Nisan. Algebraic methods for interactive proof systems. In *Proceedings of the IEEE Symposium on Foundations of Computer Science (FOCS)*, Oct. 1990.

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

[72] R. C. Merkle. A digital signature based on a conventional encryption function. In *Proceedings of the International Cryptology Conference (CRYPTO)*, 1988.

[73] S. Micali. CS proofs. In *Proceedings of the IEEE Symposium on Foundations of Computer Science (FOCS)*, 1994.

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

[75] C. Papamanthou, E. Shi, and R. Tamassia. Signatures of correct computation. In *Theory of Cryptography Conference (TCC)*, 2013.

[76] B. Parno, C. Gentry, J. Howell, and M. Raykova. Pinocchio: Nearly practical verifiable computation. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, May 2013.

[77] T. P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *Proceedings of the International Cryptology Conference (CRYPTO)*, 1991.

[78] O. Reingold, G. N. Rothblum, and R. D. Rothblum. Constant-round interactive proofs for delegating computation. In *Proceedings of the ACM Symposium on Theory of Computing (STOC)*, pages 49–62, 2016.

[79] S. Setty, S. Angel, T. Gupta, and J. Lee. Proving the correct execution of concurrent services in zero-knowledge. In *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, Oct. 2018.

[80] S. Setty, A. J. Blumberg, and M. Walfish. Toward practical and unconditional verification of remote computations. In *Proceedings of the USENIX Workshop on Hot Topics in Operating Systems (HotOS)*, May 2011.

[81] 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 *Proceedings of the ACM European Conference on Computer Systems (EuroSys)*, Apr. 2013.

[82] S. Setty, R. McPherson, A. J. Blumberg, and M. Walfish. Making argument systems for outsourced computation practical (sometimes). In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, Feb. 2012.

[83] 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 *Proceedings of the USENIX Security Symposium*, Aug. 2012.

[84] J. Thaler. Time-optimal interactive proofs for circuit evaluation. In *Proceedings of the International Cryptology Conference (CRYPTO)*, 2013.

[85] 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.

[86] J. Thaler, M. Roberts, M. Mitzenmacher, and H. Pfister. Verifiable computation with massively parallel interactive proofs. In *Proceedings of the USENIX Workshop on Hot Topics in Cloud Computing (HotCloud)*, 2012.

[87] V. Vu, S. Setty, A. J. Blumberg, and M. Walfish. A hybrid architecture for verifiable computation. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, 2013.

[88] R. S. Wahby, M. Howald, S. Garg, A. Shelat, and M. Walfish. Verifiable ASICs. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, 2016.

[89] R. S. Wahby, Y. Ji, A. J. Blumberg, A. Shelat, J. Thaler, M. Walfish, and T. Wies. Full accounting for verifiable outsourcing. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, 2017.

[90] R. S. Wahby, S. Setty, Z. Ren, A. J. Blumberg, and M. Walfish. Efficient RAM and control flow in verifiable outsourced computation. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, 2015.

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

*Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, 2018.

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

[93] T. Xie, J. Zhang, Y. Zhang, C. Papamanthou, and D. Song. Libra: Succinct zero-knowledge proofs with optimal prover computation. Cryptology ePrint Archive, Report 2019/317, 2019.

[94] Y. Zhang, D. Genkin, J. Katz, D. Papadopoulos, and C. Papamanthou. vSQL: Verifying arbitrary SQL queries over dynamic outsourced databases. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, 2017.

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

[96] Y. Zhang, D. Genkin, J. Katz, D. Papadopoulos, and C. Papamanthou. vRAM: Faster verifiable RAM with program-independent preprocessing. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, 2018.

**function** SpartanVerifier(Circuit $\mathcal{C}$, Public input $x$, Output $y$, Param $pp$)
  // receive a commitment to a multilinear polynomial $\widetilde{Z}(\cdot)$
  $\hat{Z} \leftarrow$ ReceiveFromProver()

  $\tau \xleftarrow{R} \mathbb{F}$
  SendToProver($\tau$)

  // apply the sum-check protocol to polynomial $G_{x,y,\tau}$
  $m \leftarrow 3s, H \leftarrow 0, \ell = 3$
  $(e, r_1, \ldots, r_{3s}) \leftarrow$ SumCheckWithoutFinalCheck$(G_{x,y,\tau}, m, H, \ell)$

  // receive evaluations of $\widetilde{Z}(\cdot)$ at $z_1 \leftarrow (r_1, \ldots r_s), z_2 \leftarrow (r_{s+1}, \ldots, r_{2s}), z_3 \leftarrow (r_{2s+1}, \ldots, r_{3s})$
  SendToProver$(z_1, z_2, z_3)$
  $(v_1, v_2, v_3) \leftarrow$ ReceiveFromProver()
  $(z_4, v_4) \leftarrow$ ReduceFromThreeToOne$(s, z_1, v_1, z_2, v_2, z_3, v_3)$

  // decommit the evaluation of $\widetilde{Z}(\cdot)$ at $z_4$
  SendToProver$(z_4)$
  $(v'_4, \pi_4) \leftarrow$ ReceiveFromProver()
  $b \leftarrow$ PolyEvalVerify$(pp, \hat{Z}, z_4, v_4, \pi_4)$ // verify if the returned decommitments are correct
  **if** $b \neq$ `true` **then**
    **return** `reject`

  // locally evaluate terms of $G_{x,y,\tau}(\cdot)$ that do not depend on a satisfying assignment
  $v_{x,y} \leftarrow \widetilde{I}_{x,y}(z_1)$
  $v_{io} \leftarrow \widetilde{\text{io}}(z_1, z_2, z_3)$
  $v_{add} \leftarrow \widetilde{\text{add}}(z_1, z_2, z_3)$
  $v_{mul} \leftarrow \widetilde{\text{mul}}(z_1, z_2, z_3)$

  // perform the final check in the sum-check protocol
  **if** $e \neq (v_{io} \cdot (v_{x,y} - v_1) + v_{add} \cdot (v_1 - (v_2 + v_3)) + v_{mul} \cdot (v_1 - v_2 \cdot v_3)) \cdot \prod_{i=0}^{3s-1} S_\tau(u, i)$ **then**
    **return** `reject`
  **else return** `accept`

**function** SumCheckWithoutFinalCheck($G$, $m$, $H$, $\ell$)
  $(r_1, r_2, \ldots, r_m) \leftarrow_R \mathbb{F}^m$ // sample $m$ field elements randomly
  $e \leftarrow H$
  **for** $i = 1, 2, \ldots, m$ **do**
    // An honest $\mathcal{P}_{SC}$ returns $G_i(\cdot)$ as $\{G_i(0), G_i(1), \ldots G_i(\ell)\}$ since $G_i(\cdot)$ is a degree-$\ell$ univariate polynomial
    // when $i = 1$, $G_1(X_1) = \sum_{(x_2, \ldots, x_m) \in \{0,1\}^{m-1}} G(X_1, x2, \ldots, x_m)$
    // when $i > 1$, $G_i(X_i) = \sum_{(x_{i+1}, \ldots, x_m) \in \{0,1\}^{m-i}} G(r_1, \ldots, r_{i-1}, X_i, x_{i+1}, \ldots, x_m)$
    $G_i(\cdot) \leftarrow$ ReceiveFromProver()

    **if** $G_i(0) + G_i(1) \neq e$ **then**
      **return** `reject`

    SendToProver$(r_i)$
    $e \leftarrow G_i(r_i)$ // evaluate $G_i(r_i)$ using its point-value form received from the prover
  **return** $e, (r_1, r_2, \ldots, r_m)$

Figure 3: The Spartan succinct interactive argument described from the perspective of the verifier. SumCheckWithoutFinalCheck differs from the protocol in Figure 2 in that it returns without performing the final check on $G(r_1, r_2, \ldots, r_m) = e$. PolyEvalVerify refers to a routine in the polynomial commitment scheme for multilinear polynomials (see text for details). ReduceFromThreeToOne is depicted in Figure 4. Our depiction is inspired by prior works [29, 87, 91].

1: **function** ReduceFromThreeToOne($s$, $z_1$, $v_1$, $z_2$, $v_2$, $z_3$, $v_3$)
2: // receive a degree-$2s$ polynomial $K(\cdot)$ in point-value representation
3: // a correct prover returns $K = \tilde{Z} \circ \ell$ for $\ell : \mathbb{F} \to \mathbb{F}^s$, where $\ell(t) = 2^{-1}(t-2)(t-1) \cdot z_1 - (t-2)t \cdot z_2 + 2^{-1}t(t-1) \cdot z_3$
4: $k_0, k_1, \ldots, k_{2s} \leftarrow$ ReceiveFromProver()
5: **if** $k_0 \neq v_1$ or $k_1 \neq v_2$ or $k_2 \neq v_3$ **then**
6: **return** reject
7: $r \xleftarrow{R} \mathbb{F}$
8: $z_4 \leftarrow \ell(r)$
9: $v_4 \leftarrow K^*(r)$ // $K^*$ is the polynomial interpolation of $k_0, \ldots, k_{2s}$
10: **return** $(z_4, v_4)$

Figure 4: An interactive protocol, depicted from the verifier's perspective, for reducing three evaluations of a polynomial into a single evaluation (see text for details).

1: **function** SpartanVerifier(Circuit $\mathcal{C}$, Public input $x$, Output $y$, PolyParam $pp_pc$, CompParam $pp_cc$, ComputationCommitment $\hat{\mathcal{C}}$)
2: // receive a commitment to a multilinear polynomial $\widetilde{Z}(\cdot)$
3: $\hat{Z} \leftarrow$ ReceiveFromProver()
4:
5: $\tau \xleftarrow{R} \mathbb{F}$
6: SendToProver($\tau$)
7:
8: // apply the sum-check protocol to polynomial $G_{x,y,\tau}$
9: $m \leftarrow 3s, H \leftarrow 0, \ell = 3$
10: $(e, r_1, \ldots, r_{3s}) \leftarrow$ SumCheckWithoutFinalCheck$(G_{x,y,\tau}, m, H, \ell)$
11:
12: // receive evaluations of $\widetilde{Z}(\cdot)$ at $z_1 \leftarrow (r_1, \ldots r_s), z_2 \leftarrow (r_{s+1}, \ldots, r_{2s}), z_3 \leftarrow (r_{2s+1}, \ldots, r_{3s})$
13: SendToProver$(z_1, z_2, z_3)$
14: $(v_1, v_2, v_3) \leftarrow$ ReceiveFromProver()
15: $(z_4, v_4) \leftarrow$ ReduceFromThreeToOne$(s, z_1, v_1, z_2, v_2, z_3, v_3)$
16:
17: // decommit the evaluation of $\widetilde{Z}(\cdot)$ at $z_4$
18: SendToProver$(z_4)$
19: $(v'_4, \pi_4) \leftarrow$ ReceiveFromProver()
20: $b \leftarrow$ PolyEvalVerify$(pp_{pc}, \hat{Z}, z_4, v_4, \pi_4)$ // verify if the returned decommitments are correct
21: **if** $b \neq$ `true` **then**
22: **return** `reject`
23:
24: $v_{x,y} \leftarrow \widetilde{I}_{x,y}(z_1)$ // locally evaluate $\widetilde{I}_{x,y}(\cdot)$
25:
26: // employ computation commitments to evaluate remaining terms of $G_{x,y,\tau}(\cdot)$
27: $z \leftarrow (z_1, z_2, z_3)$
28: $(v, \pi) \leftarrow$ ReceiveFromProver$(z)$
29: $b \leftarrow$ ComputationDecommitVerify$(pp_{pc}, \hat{Z}, z, v, \pi)$ // verify if the returned decommitment is correct
30: **if** $b \neq$ `true` **then**
31: **return** `reject`
32: $(v_{add}, v_{mul}, v_{io}) \leftarrow v$
33:
34: // perform the final check in the sum-check protocol
35: **if** $e \neq (v_{io} \cdot (v_{x,y} - v_1) + v_{add} \cdot (v_1 - (v_2 + v_3) + v_{mul} \cdot (v_1 - v_2 \cdot v_3)) \cdot \prod_{i=0}^{3s-1} S_\tau(u, i)$ **then**
36: **return** `reject`
37: **else return** `accept`

Figure 5: Spartan succinct interactive argument—with sub-linear verification costs via the use of computation commitments—from the verifier's perspective. This protocol differs from Figure 3 in one aspect: on line 28 of this protocol, the verifier takes the assistance of the prover to evaluate three multilinear polynomials $\widetilde{\text{add}}(\cdot), \widetilde{\text{mul}}(\cdot)$, and $\widetilde{io}(\cdot)$ at location $z = (z_1, z_2, z_3)$.