Shihui Fu* and Guang Gong

Polaris: Transparent Succinct Zero-Knowledge Arguments for R1CS with Efficient Verifier

Abstract: We present a new zero-knowledge succinct argument of knowledge (zkSNARK) scheme for Rank-1 Constraint Satisfaction (RICS), a widely deployed NP-complete language that generalizes arithmetic circuit satisfiability. By instantiating with different commitment schemes, we obtain several zkSNARKs where the verifier's costs and the proof size range from $O(\log^2 N)$ to $O(\sqrt{N})$ depending on the underlying polynomial commitment schemes when applied to an N-gate arithmetic circuit. All these schemes do not require a trusted setup. It is plausibly post-quantum secure when instantiated with a secure collision-resistant hash function. We report on experiments for evaluating the performance of our proposed system. For instance, for verifying a SHA-256 preimage (less than 23k AND gates) in zero-knowledge with 128 bits security, the proof size is less than 150kB and the verification time is less than 11ms, both competitive to existing systems.

Keywords: zkSNARK, verifiable computation, zero-knowledge proof, polynomial commitment

DOI 10.2478/popets-2022-0027
Received 2021-05-31; revised 2021-09-15; accepted 2021-09-16.

1 Introduction

With a surge in applying blockchain technologies to various applications and cloud outsource computing, zero-knowledge proof systems gain a great interests from its theoretical interests to practical implementations. Zero-knowledge proof allows a powerful party (the prover) to convince another weak one (the verifier) that a statement is true without revealing any information beyond the fact that the statement is true. Since introduced by Goldwasser et al. in [36] and followed by the generic constructions proposed in the seminal work of Kilian [41] and Micali [46] based on probabilistically checkable proofs (PCPs), there are tremendous works that have been done to bring zero-knowledge proofs to practical systems, and it has become fundamental tools in the design of real-world systems with strong privacy properties, such as verifiable outsourced computations, anonymous credentials, privacy-preserving cryptocurrencies and smart contracts.

The fundamental challenge associated with maintaining transactions privacy in blockchain applications is that participants must be able to agree on the state of the network and reject any invalid transactions or unfair play while still respecting the external privacy constraints endowed by the workflows. Simply, participants must be able to agree that something is the same without seeing it. Many applications further require that a transaction consists of a single non-interactive message that can be verified by anyone; such messages are cheap to communicate and can be stored for later use (e.g., on a public ledger). Constructions that satisfy these properties are known as zero-knowledge succinct non-interactive arguments (zkSNARGs) [33], and refer to the proof constructions where one can prove possession of certain information, e.g., a secret key, without revealing that information, and without any interaction between the prover and verifier. Additionally, if the prover can convince the verifier not only that the secret exists, but that he in fact knows such a secret – again, without revealing any information about the secret, we call such a proof construction *zero-knowledge succinct non-interactive argument of knowledge* (zkSNARK).

Currently, there are a large amount of constructions that achieve different tradeoffs between proof size, proving time, and verification time under different trust models as well as cryptographic assumptions. Some constructions also achieve better efficiency by relying on a trusted preprocessing model in which a one-time expensive setup procedure is performed in order to generate a compact verification key for proof, which is later used to verify proof instances efficiently. Essentially, it needs a third party to run a probabilistic algorithm to generate parameters for the argument system, publish its output, and "forget" the secret randomness used to generate it. As stated in [16], when using cryptographic proofs in distributed systems, relying on a central party negates the benefits of distributed trust and, even though it is

***Corresponding Author: Shihui Fu:** University of Waterloo, E-mail: shihui.fu@uwaterloo.ca
Guang Gong: University of Waterloo, E-mail: ggong@uwaterloo.ca

invoked only once in a system's life, a party trusted by all users typically does not exist!

A proof system is called *transparent* if it does not involve any trusted setup. Recent efforts have thus focused on designing proof systems with transparent setup. We continue along this line of research and focus on obtaining transparent proof systems with better concrete efficiency characteristics: succinctness (the proof size is polylogarithmic in the original computation length N), prover-scalability (proof generation time scales linearly or quasi-linearly in N), and verifier-scalability (verification time is polylogarithmic in N).

1.1 Our Results

In this paper we propose Polaris, a zkSNARK with quasi-linear prover time and both polylogarithmic proof size and verification time in the size of the arithmetic circuit representing the statement. Inspired by the encoding of an R1CS instance as a univariate polynomial in quadratic arithmetic program (QAP) [32] and a degree-3 multivariate polynomial in Spartan [48], we instantiate the encoding of the R1CS instance as a univariate polynomial by leveraging bivariate interpolations. Polaris is a combination of the univariate analogue of encoding and Aurora's univariate sumcheck protocol [16]. Polaris also has attractive features: it uses a transparent setup and can only use lightweight cryptography.

The efficiency features of Polaris stem from an efficient arithmetic layered circuit that the verifier can delegate the query computations to the prover, and validate the result using the GKR protocol [35]. Our concrete contributions are:

- **A query computation circuit.** Recall that the univariate sumcheck in Aurora [16] invokes a low degree test (LDT) protocol on Reed-Solomon (RS) code as a subroutine to test the sum of a univariate polynomial on a subset. To ensure security, at the end of the LDT, the verifier needs oracle access to some points of the polynomials. In both Ligero [7] and Aurora [16], the verifier constructs a circuit by itself and evaluates it locally, which takes linear time. Instead of evaluating the query computations locally, we design a (layered) arithmetic circuit to delegate the query computations to the prover, and validate the results using the GKR protocol. In this way, we can eliminate the linear overhead to evaluate these points locally, making the verification time of the overall protocol polylogarithmic.
- **An encoding of R1CS instances as low degree univariate polynomials.** In QAP [32], the authors encode an R1CS instance as a degree-N univariate polynomial. However, this offline encoding step alone typically costs $O(N^2 \log N)$ because it involves the FFT to encode the computation as $O(N)$ degree-N univariate polynomials. In Spartan [48], the resulting multivariate polynomial in the offline preprocessing phase is actually quadratic in the instance size, since adding the low degree extensions can increase the degrees for each variable. Instead, we interpret the encoding of the R1CS instance as a univariate polynomial by leveraging bivariate interpolations. The size of resulting univariate polynomial is equal to the size of the instance and the encoding cost is $O(N \log N)$. This avoids any asymptotic overhead when we employ a polynomial commitment scheme.
- **Transparent zkSNARK for R1CS.** We construct a new zkSNARK for R1CS with polylogarithmic for both proof size and verification time. By instantiating with different polynomial commitment schemes, we obtain several zkSNARKs where the verifier's costs and the proof size range from polylogarithmic to sublinear depending on the underlying commitment scheme. These schemes do not require a trusted setup. Our scheme thus provides an alternative choice for the design of transparent zkSNARKs. Table 1 compares the asymptotic costs of our zkSNARK (instantiated with the polynomial commitment schemes of FRI [8] or Virgo-VPD [56]) with some known schemes.
- **Evaluation.** We implement and evaluate Polaris, based on our new construction instantiated with the polynomial commitment scheme of FRI [8]. The implementation is based on the open-source library `libiop` [2] and `Virgo` [6]. Our experimental evaluation demonstrates that Polaris offers a much lower verification time than Ligero and Aurora for instances with large sizes as we reduce the complexity from linear to polylogarithmic.

1.2 Related Work

More recent improvements for QAP/QSP based zkSNARK have been proposed in the directions for removing the need for a trusted setup to generate the argument system parameters and construct transparent zkSNARK schemes. The scheme of [54], Hyrax, is transparent, and the proof size and verification time are sublinear. Virgo's [56] model of computation is the same as Hyrax's, so it achieves sublinear verification costs only for low-depth, uniform circuits as well. Fractal [29] achieves sublinear verification costs, but it is a preprocessing zkSNARK for R1CS via establishing a connection between holography and preprocessing in the ran-

Table 1. Asymptotic comparison of some recent zkSNARKs when applied to an N-gate arithmetic circuit. For Virgo, we assume a depth-D layered uniform circuit.

scheme	setup	prover time	proof length	verifier time	computational model
Ligero [7]	**public**	$O(N \log N)$	$O(\sqrt{N})$	$O(N)$	**arithmetic circuit**
Ligero++ [20]	**public**	$O(N \log N)$	$O(\log^2 N)$	$O(N)$	**arithmetic circuit**
Aurora [16]	**public**	$O(N \log N)$	$O(\log^2 N)$	$O(N)$	**R1CS**
Fractal [29], Spartan$_{\mathrm{RO}}$ [48]	**public**	$O(N \log N)$	$O(\log^2 N)$	$O(\log^2 N)$	**R1CS**
Virgo [56]	**public**	$O(N \log N)$	$O(D \log N)$	$O(D \log N)$	**uniform circuit**
Stark [9]	**public**	$O(N \log N)$	$O(\log^2 N)$	$O(\log^2 N)$	**uniform circuit**
Spartan$_{\mathrm{DL}}$ [48]	**public**	$O(N)$	$O(\sqrt{N})$	$O(\sqrt{N})$	**R1CS**
Marlin [28]	**private**	$O(N \log N)$	$O(\log N)$	$O(\log N)$	**R1CS**
Polaris$_{\mathrm{RO}}$	**public**	$O(N \log N)$	$O(\log^2 N)$	$O(\log^2 N)$	**R1CS**

dom oracle model, but it does not offer short proofs. Marlin [28] also achieves sublinear verification costs in the holographic setting, but it needs a trusted setup to generate a linear-size structured reference string (SRS). Spartan [48] (the DLOG variant) offers the best asymptotic cost for the prover for statements expressed in R1CS through the sumcheck protocol to build low degree multilinear polynomials (LDPs) where it employs a polynomial commitment scheme as a black box to commit some polynomial evaluations. But the proofs are $O(\sqrt{N})$ group elements and the verifier must perform $O(\sqrt{N})$ exponentiations. The variant Spartan$_{\mathrm{RO}}$ in the Spartan family offers a tradeoff between prover costs and verifier costs. Bulletproof [22] is also along this line, but it is to iteratively check inner products committed by DLOG computations, which suffers the heavy cost for both communication between provers and verifiers, since it leverages some benefits from MPC.

Based on "(MPC)-in-the-head", Ames et al. [7] proposed a zero-knowledge interactive PCPs (IPCPs) called Ligero. It only uses symmetric key operations and the prover is fast in practice and the verification time is linear in the size of the circuit. Later, it is categorized as interactive oracle proofs (IOPs, a multi-round variant of IPCPs), and in the same model Ben-Sasson et al. built Stark [9], a transparent zero-knowledge proof in the RAM model of computation. Their verification time is also linear to the description of the RAM program, and succinct (logarithmic) in the time required for program execution. Recently, Ben-Sasson et al. proposed Aurora [16], in the IOP model with a logarithmic factor reduction of the proof size and prover time compared with Stark. Ligero++ [20] is an optimized variant of Ligero, which further reduces the proof sizes from $O(\sqrt{N})$ to polylogarithmic, but it still suffers a linear verification cost. All these proof systems have the verifier oracle access to an encoding of the input using a Reed-Solomon (RS) code, and achieve non-interactivity through Fiat-Shamir transform [31]. Comparing to our scheme, we expect the verification in our scheme is more efficient, as our scheme has a better asymptotic complexity.

Differentiation from some existing schemes. Next, we compare our new scheme with some existing schemes, such as Aurora [16], Stark [9] and Marlin [28].

- The main difference between Stark [9] and Polaris is that the computational models they support are different. Stark supports only *uniform* computations specified by a succinctly represented program and a time bound. This requires circuits with a sequence of identical sub-circuits, otherwise it does not achieve sublinear verification costs. Any circuit can be converted to this form [14], but the verification time will be linear. In contrast, Polaris can support *non-uniform* computations specified by an explicit circuit (or R1CS), and has a polylogarithmic verification time. Both Stark and Polaris have argument size $O(\log^2 N)$, but additional costs in Stark (e.g., due to switching networks [9, 16]) result in Stark proof sizes much larger than Polaris' proofs.
- Aurora [16] is a *non*-holographic protocol for R1CS with $O(N)$ verification costs: the verifier constructs an oracle query circuit by itself and evaluates it locally, which takes linear time. This is the best possible because just reading the circuit description takes at least linear time. Polaris inherits a variant of Aurora's univariate sumcheck, but the verifier performs it *exponentially faster*, in time $O(\log^2 N)$, by delegating the oracle query computations to the prover, and validating the results using the GKR protocol. Another difference is that Aurora encapsulates the entry-wise product check (univariate rowcheck in [16]) as a low degree test, while Polaris uses a direct one. Thus, in Polaris, we can make a black-box use of any efficient extractable polynomial commitment scheme to check the entry-wise product and obtain a family of zkSNARKs where each variant employs a different polynomial commitment scheme.

– Marlin [28] and Polaris follow different design paradigms: Marlin requires a *trusted setup* to generate the linear-size SRS even though it is universal and updatable, while our scheme does not need any trusted setup. Both Marlin and Polaris obtain sublinear verifiers in the holographic setting. The main difference is that we use a GKR protocol to reduce the verifier's cost, compared with the approach in Marlin where two additional univariate sumcheck protocols are used. Naturally, without such additional univariate sumcheck protocols, the verifier in Polaris does not need any oracle access to the low-degree polynomial extensions (of the row, column and non-zero entries) that encode the non-zero entries of the circuit matrices, so Polaris does not need any offline preprocessing phase to compute them (nine univariate polynomials in Marlin) which each incurs $O(N \log N)$ time. The only holographic part is within the GKR protocol, whose verifier runs in sublinear time when given (multivariate low degree) encodings of the circuit's wiring predicates and the circuit's input (constant-size in Polaris). We design an $O(N)$-size circuit to engage the doubly-efficient GKR protocol to achieve holography. This immediately leads to a linear-time prover (time-optimal) in this subroutine, compared to the approach in Marlin where two additional univariate sumcheck protocols are used and each incurs an $O(N \log N)$-time prover. Another difference is that we except our building blocks only use symmetric cryptography, which makes the final argument plausibly post-quantum secure. In contrast, the authors in Marlin instantiated an extension of extractable polynomial commitment whose security relies on the Strong Diffie–Hellman Assumption in bilinear groups, which is not post-quantum secure.

2 Preliminaries

For $n \in \mathbb{N}$, let $[n] = \{1, 2, \ldots, n\}$. We denote by $A_{i,j}$ the entry lying in the i-th row and the j-th column of matrix A. We use $\mathbb{F}$ to denote a finite field and λ the security parameter. We denote by $\mathsf{negl}(n)$ a negligible function defined over the integers, meaning that for every polynomial $p(\cdot)$ and all sufficiently large n's, $\mathsf{negl}(n) < \frac{1}{p(n)}$. We use "PPT algorithms" to refer to probabilistic polynomial time algorithms.

2.1 Problem Instances in R1CS

Let $\mathcal{R}$ be a binary ordered NP relation and $\mathcal{L} \subset \{0,1\}^*$ the language corresponding to $\mathcal{R}$. For each instance $\mathbb{x} \in \mathcal{L}$, let $\mathcal{R}_{\mathbb{x}} \subset \{0,1\}^*$ denote the corresponding set of witnesses for the fact that $\mathbb{x} \in \mathcal{L}$, i.e., $\mathcal{R}_{\mathbb{x}} = \{w : (\mathbb{x}, w) \in \mathcal{R}\}$. Let $\mathcal{R}_{\mathcal{L}}$ denote the corresponding language of valid instance-witness pairs, i.e., $\mathcal{R}_{\mathcal{L}} = \{(\mathbb{x}, w) : \mathbb{x} \in \mathcal{L} \text{ and } w \in \mathcal{R}_{\mathbb{x}}\}$.

As an NP-complete language, we focus on the rank-1 constraint satisfiability (R1CS). The main reason is that R1CS is a popular target for compiler toolchains that accept applications expressed in high-level languages, and it generalizes circuits by allowing "native" field arithmetic and having no fan-in/fan-out restrictions, but it is simple enough that one can design efficient argument systems for it [16, 48].

Definition 1. *(R1CS instance). An R1CS instance $\mathbb{x}$ is a tuple $(\mathbb{F}, A, B, C, v, m, n)$, where A, B, C are $m \times m$ matrices over $\mathbb{F}$, v is the vector with entries consisting of instance's public input and output, $m \geq 1 + |v|$ and there are at most n non-zero entries in each matrix.*

Below, we use the notation (x, y, z) to denote the concatenation of the three vectors x, y, z in a natural way. Without loss of generality, we assume that $m = O(n)$.

Definition 2. *(R1CS relation $\mathcal{R}_{\text{R1CS}}$). An R1CS instance $\mathbb{x} = (\mathbb{F}, A, B, C, v, m, n)$ is said to be satisfiable if there exists a witness $w \in \mathbb{F}^{m-|v|-1}$ such that $(Az) \circ (Bz) = Cz$ where $z := (1, v, w)$ and "$\circ$" denotes the entry-wise product.*

Clearly, R1CS relation generalizes the problem of arithmetic circuit satisfiaility. For example, the matrices A, B, C represent the circuit's gates, or more precisely, the vectors Az, Bz and Cz encode the left input, right input and output vectors of the gates in the circuit respectively. The witness w consists of the circuit's private input and wire values.

2.2 Zero-Knowledge Interactive Arguments

An interactive proof allows a prover $\mathcal{P}$ to convince a PPT verifier $\mathcal{V}$ the validity of some statements through interaction over some number of rounds. We are interested in the interactive proof systems where both the prover and the verifier are efficient with respect to the size of the instance.

An interactive argument system for an NP relationship $\mathcal{R}$ is an interactive proof between a computationally-bounded prover $\mathcal{P}$ and a PPT verifier $\mathcal{V}$. Therefore, in an argument the computational power of the prover is also restricted to be a PPT one. Naturally, we require the argument to satisfy a proof of knowledge property additionally. Intuitively,

this means that in order to produce a convincing proof of a statement, the prover must convince the verifier that it "knows" a witness w for the instance $\mathbb{x}$ such that $(\mathbb{x}, w) \in \mathcal{R}$. We formalize interactive arguments of knowledge in the following.

Definition 3. *(Public-coin succinct interactive argument of knowledge). Let $(\mathcal{P}, \mathcal{V})$ denote a pair of PPT interactive algorithms and* pp *public parameters given as input the security parameter λ. Then a protocol between $\mathcal{P}$ and $\mathcal{V}$ is called a public-coin succinct interactive argument of knowledge for a language $\mathcal{L}$ if:*

- ***Completeness.*** *For every instance-witness pair $(\mathbb{x}, w) \in \mathcal{R}_{\mathcal{L}}$, and every $r \in \{0,1\}^*$,*

$$\Pr[\langle \mathcal{P}(\mathsf{pp}, w), \mathcal{V}(\mathsf{pp}, r)\rangle(\mathbb{x}) = 1] = 1.$$

- ***Soundness.*** *For every instance $\mathbb{x} \notin \mathcal{L}$, every PPT prover $\mathcal{P}^*$, and every $w, r \in \{0,1\}^*$,*

$$\Pr[\langle \mathcal{P}^*(\mathsf{pp}, w), \mathcal{V}(\mathsf{pp}, r)\rangle(\mathbb{x}) = 1] \leq \mathsf{negl}(\lambda).$$

- ***Knowledge soundness.*** *For any PPT adversary $\mathcal{A}$, there exists a PPT extractor $\mathcal{E}$ such that for every instance $\mathbb{x}$ and every $w, r \in \{0,1\}^*$, if $\Pr[\langle \mathcal{A}(\mathsf{pp}, w), \mathcal{V}(\mathsf{pp}, r)\rangle(\mathbb{x}) = 1] \geq \mathsf{negl}(\lambda)$, then*

$$\Pr[(\mathbb{x}, w') \in \mathcal{R}_{\mathcal{L}} | w' \leftarrow \mathcal{E}^{\mathcal{A}}(\mathsf{pp}, \mathbb{x})] \geq \mathsf{negl}(\lambda).$$

- ***Succinctness.*** *The running time of $\mathcal{V}$ and the total communication (proof size) between $\mathcal{P}$ and $\mathcal{V}$ are sublinear in the size of the instance $\mathbb{x}$.*
- ***Public coin.*** *$\mathcal{V}$'s challenge in each round is independent of $\mathcal{P}$'s messages in previous rounds.*

Definition 4. *(Witness-extended emulation [38]). An interactive argument system $(\mathsf{Setup}, \mathcal{P}, \mathcal{V})$ for $\mathcal{L}$ has witness-extended emulation if for all deterministic polynomial time $\mathcal{P}^*$ there exists an expected polynomial time emulator $\mathcal{E}$ such that for all non-uniform polynomial time adversaries $\mathcal{A}$ and all $z_{\mathcal{V}} \in \{0,1\}^*$, the following probabilities differ by at most $\mathsf{negl}(\lambda)$:*

$$\Pr\left[\mathcal{A}(t) = 1 : \begin{array}{l}\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda) \\ (\mathbb{x}, z_{\mathcal{P}}) \leftarrow \mathcal{A}(\mathsf{pp}) \\ t \leftarrow \mathsf{tr}\langle \mathcal{P}^*(\mathsf{pp}, z_{\mathcal{P}}), \mathcal{V}(\mathsf{pp}, z_{\mathcal{V}})\rangle(\mathbb{x})\end{array}\right]$$

and

$$\Pr\left[\begin{array}{c}\mathcal{A}(t)=1 \text{ and} \\ t \text{ accepting} \Rightarrow (\mathbb{x},w)\in\mathcal{R}_{\mathcal{L}}\end{array} : \begin{array}{l}\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda) \\ (\mathbb{x}, z_{\mathcal{P}}) \leftarrow \mathcal{A}(\mathsf{pp}) \\ (t,w) \leftarrow \mathcal{E}^{\mathcal{P}^*(\mathsf{pp}, z_{\mathcal{P}})}(\mathbb{x})\end{array}\right],$$

where tr *denotes the random variable that corresponds to the transcript of the interaction between $\mathcal{P}^*$ and $\mathcal{V}$. Here, the oracle called by $\mathcal{E}$ permits rewinding the prover to a specific point and resuming with fresh randomness for the verifier from this point onwards.*

We use Setup to denote the generation phase of the public parameters pp. We adapt the following definitions from [34, 48] for our zero-knowledge notation.

Definition 5. *An interactive argument (Setup, $\mathcal{P}$, $\mathcal{V}$) for $\mathcal{L}$ is computational zero-knowledge if for every PPT interactive machine $\mathcal{V}^*$, there exists a PPT algorithm $\mathcal{S}$ called the simulator, running in time polynomial in the length of its first input such that for every instance $\mathbb{x} \in \mathcal{L}$, $w \in \mathcal{R}_{\mathbb{x}}$, and $z \in \{0,1\}^*$, the following holds when the distinguishing gap is considered as a function of $|\mathbb{x}|$:*

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

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

2.3 Commitment Schemes

A commitment scheme is a cryptographic primitive that allows one to commit to a chosen value while keeping it hidden to others, with the ability to reveal the committed value later. In this subsection, we introduce the following two commitment schemes, polynomial commitment scheme and Merkle tree, which we employ in our protocol.

A polynomial commitment scheme is a commitment scheme in which a prover commits to a polynomial f over $\mathbb{F}$ of degree at most d with a message that is much smaller than sending all the coefficients of f. The prover can later produce an interactive or non-interactive argument that $f(\theta) = \eta$ for arbitrary $\theta, \eta \in \mathbb{F}$.

We follow the notations from Bünz [23] where they generalize the definition of Kate et al. [40], to allow interactive evaluation proofs, and these definitions are also used in Spartan [48]. In any list of arguments or returned tuple $(a, b, c; d, e)$, those variables listed before the semicolon are public and the ones after it are secret. When there is no secret information, the semicolon is omitted.

A polynomial commitment scheme is a tuple of four protocols $\mathsf{PC} = (\mathsf{Setup}, \mathsf{Commit}, \mathsf{Open}, \mathsf{Eval})$:

- $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, d)$: takes as input the degree d of a polynomial; generates public parameters pp.
- $(\mathcal{C}, \mathcal{S}) \leftarrow \mathsf{Commit}(\mathsf{pp}; f)$: takes as input a secret polynomial f with degree at most d over $\mathbb{F}$; outputs a

public commitment $\mathcal{C}$ and (optional) a secret opening hint $\mathcal{S}$.
- $b \in \{0,1\} \leftarrow \mathsf{Open}(\mathsf{pp}, \mathcal{C}, f, \mathcal{S})$: verifies the opening of commitment $\mathcal{C}$ to the polynomial f provided with opening hint $\mathcal{S}$; output b.
- $b \in \{0,1\} \leftarrow \mathsf{Eval}(\mathsf{pp}, \mathcal{C}, \theta, \eta, d; f, \mathcal{S})$ is an interactive public-coin protocol between a PPT prover $\mathcal{P}$ and verifier $\mathcal{V}$. Both $\mathcal{P}$ and $\mathcal{V}$ hold as input a commitment $\mathcal{C}$, points $\theta, \eta \in \mathbb{F}$ and a degree d. The prover $\mathcal{P}$ additionally knows the polynomial $f(X) \in \mathbb{F}[X]$ with degree at most d and its opening hint $\mathcal{S}$. In the protocol, the prover attempts to convince the verifier that $f(\theta) = \eta$. At the end of the protocol, $\mathcal{V}$ outputs b.

A polynomial commitment scheme is correct if an honest committer can successfully convince the verifier of any evaluation, and is evaluation binding if no efficient adversary can convince the verifier that the committed polynomial f evaluates to different values $\eta_0 \neq \eta_1 \in \mathbb{F}$ at the same point $\theta \in \mathbb{F}$. Also, we require the polynomial commitment scheme to satisfy the knowledge soundness property. We formally consider the following definition [48].

Definition 6. *A tuple of four protocols* $\mathsf{PC} = (\mathsf{Setup}, \mathsf{Commit}, \mathsf{Open}, \mathsf{Eval})$ *is a polynomial commitment scheme for polynomials over* $\mathbb{F}$ *if the following conditions hold.*

- ***Completeness.*** *For every polynomial* $f(X) \in \mathbb{F}[X]$*, and every point* $\theta \in \mathbb{F}$*,*

$$\Pr\left[b = 1 : \begin{array}{c} d \leftarrow \deg(f(X)) \\ \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, d) \\ (\mathcal{C}, \mathcal{S}) \leftarrow \mathsf{Commit}(\mathsf{pp}; f) \\ \eta \leftarrow f(\theta) \\ b \leftarrow \mathsf{Eval}(\mathsf{pp}, \mathcal{C}, \theta, \eta, d; f, \mathcal{S}) \end{array}\right] = 1.$$

- ***Binding.*** *For any PPT adversary* $\mathcal{A}$*, and every degree parameter* $d \geq 1$*,*

$$\Pr\left[\begin{array}{c} b_0 = b_1 \neq 0 \\ \text{and } f_0 \neq f_1 \end{array} : \begin{array}{c} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, d) \\ (\mathcal{C}, f_0, f_1, \mathcal{S}_0, \mathcal{S}_1) \leftarrow \mathcal{A}(\mathsf{pp}) \\ b_0 \leftarrow \mathsf{Open}(\mathsf{pp}, \mathcal{C}, f_0, \mathcal{S}_0) \\ b_1 \leftarrow \mathsf{Open}(\mathsf{pp}, \mathcal{C}, f_1, \mathcal{S}_1) \end{array}\right] \leq \mathsf{negl}(\lambda).$$

- ***Knowledge soundness.*** *Eval is a public-coin interactive argument of knowledge with witness-extended emulation (Definition 4) for the following NP relation given* $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, d)$ *on the degree parameter* $d \geq 1$*:*

$$\mathcal{R}_{\mathsf{Eval}}(\mathsf{pp}) = \left\{ ((\mathcal{C}, \theta, \eta), (f, \mathcal{S})) : \begin{array}{c} f \in \mathbb{F}[X] \text{ and } \deg(f) \leq d \\ \text{and } f(\theta) = \eta \\ \text{and } \mathsf{Open}(\mathsf{pp}, \mathcal{C}, f, \mathcal{S}) = 1 \end{array} \right\}$$

Note that in the foregoing definition, we did not give explicitly the formal definition of evaluation binding. This is because if the Setup, Commit, and Open parts of a polynomial scheme PC form a binding commitment scheme (the second condition in Definition 6), then witness-extended emulation implies evaluation binding [23].

A Merkle hash tree [45] is a data structure that allows to commit ℓ messages (or a vector) by a single hash value such that revealing any message requires only $O(\log \ell)$ proof size and verification time. A Merkle tree is a tuple of four subprotocols $\mathsf{MT} = (\mathsf{Setup}, \mathsf{Commit}, \mathsf{Open}, \mathsf{Verify})$.
- $\mathcal{H} \leftarrow \mathsf{Setup}(1^\lambda)$: picks a hash function from the collision-resistant hash function family for Merkle trees.
- $\mathsf{root}_c \leftarrow \mathsf{Commit}(c)$: takes as input a vector $c = (c_1, \ldots, c_\ell)$; assigns all entries to the leaves of the tree and computes internal nodes via applying the underlying hash function $\mathcal{H}$ on the values assigned to the children; output the root of the tree.
- $(c_i, \pi_i) \leftarrow \mathsf{Open}(c, i)$: takes as input an index $i \leq \ell$; output the entry c_i together with a proof π_i that consists of all the values assigned to nodes on the path from the root to c_i.
- $b \in \{0,1\} \leftarrow \mathsf{Verify}(\mathsf{root}_c, i, c_i, \pi_i)$: verifies the opening of c_i by recomputing the entire path bottom-up and comparing final outcome to the commitment root_c; output b.

The binding property of a Merkle hash tree follows from the collision-resistant property of the hash function used to construct the Merkle tree.

2.4 Univariate Sumcheck

Given a subset L of $\mathbb{F}$ and an integer $k < |L|$, the Reed-Solomon (RS) code, denoted by $\mathrm{RS}[L, k]$, is all evaluations over L of univarite polynomials of degree less than k. We use the notation $f|_L$ to denote the vector of the evaluations $(f(a))_{a \in L}$. If $\deg(f) < k$, then $f|_L \in \mathrm{RS}[L, k]$. Likewise, any vector of size $|L|$ can be viewed as some univariate polynomial of degree less than $|L|$ evaluated on L. Thus we use the notations of univariate polynomials over $\mathbb{F}$ and their evaluations on L interchangeable.

We now describe an interactive proof protocol, called the univariate sumcheck, which was recently proposed by Ben-Sasson et al. in [16]. The univariate sumcheck is an RS-encoded IOPP for testing whether a low-degree univariate polynomial f sums to a given value on a subset $H \subseteq \mathbb{F}$. This protocol is a univariate analogue of the seminal multivariate sumcheck protocol [44, 49]. The underlying idea of the univariate sumcheck protocol relies on the following lemma.

Lemma 1 ([16, 24]). *Let H be an affine subspace of $\mathbb{F}$, and let $g(X)$ be a univariate polynomial over $\mathbb{F}$ of degree strictly less than $|H|$. Then*

$$\sum_{a\in H} g(a) = g_{|H|-1}\cdot c_1 \ ,$$

where $g_{|H|-1}$ is the coefficient of $X^{|H|-1}$ in g, and c_1 is the coefficient of X in $\mathbb{Z}_H(X) = \prod_{a\in H}(X-a)$.

Here $c_1 \neq 0$, since $\mathbb{Z}_H(X)$ has only simple zeros. Lemma 1 suggests the following approach. To test the result of $\sum_{a\in H} f(a)$ for $f(X) \in \mathbb{F}[X]$ with degree less than d ($d \geq |H|$), by the Euclidean algorithm, we can decompose f into two polynomials g and h such that $f(X) \equiv g(X) + \mathbb{Z}_H(X)\cdot h(X)$ with $\deg(g) < |H|$ and $\deg(h) < d - |H|$. This decomposition is unique for every f. In particular, f and g agree on H, thus so do their sums on H. Therefore, if the claimed sum $\mu = \sum_{a\in H} f(a)$ sent by the prover is correct, then $c_1\cdot f(X) - \mu\cdot X^{|H|-1} - c_1\cdot\mathbb{Z}_H(X)\cdot h(X)$ must be a polynomial of degree less than $|H|-1$. To test this, the prover and the verifier engage in a low degree test (LDT) protocol on the RS code. The LDT protocol [16] is denoted as $\langle \mathcal{P}_{\mathsf{LDT}}(\vec{\rho}, p), \mathcal{V}_{\mathsf{LDT}}(\vec{m}, \deg(p))\rangle(L)$, where $\vec{\rho}$ is a sequence of polynomials, p is their rational constraint and $\vec{m}$ is the degrees of the sequence of polynomials.

The protocol of the univariate sumcheck in [16] works as follows. To prove $\mu = \sum_{a\in H} f(a)$ for $f(X) \in \mathbb{F}[X]$ with degree less than d, the prover $\mathcal{P}$ and the verifier $\mathcal{V}$ pick an affine subspace L of $\mathbb{F}$, where $|L| > d$ and $|L| = O(|H|)$. Then the prover $\mathcal{P}$ decomposes $f(X) \equiv g(X) + \mathbb{Z}_H(X)\cdot h(X)$ with $\deg(g) < |H|$ and $\deg(h) < d - |H|$, and computes the vectors $f|_L$ and $h|_L$. $\mathcal{P}$ then commits these two vectors using the Merkle tree and sends the roots to $\mathcal{V}$. The verifier $\mathcal{V}$ outputs a rational constraint $N(X, Y_1, Y_2) = c_1\cdot Y_1 - \mu\cdot X^{|H|-1} - c_1\cdot\mathbb{Z}_H(X)\cdot Y_2$. Then the prover $\mathcal{P}$ and the verifier $\mathcal{V}$ engage an LDT to test that the degree of $((f(X), h(X)), N(X, f(X), h(X)))$ is less than $((d, d-|H|), |H|-1)$. At the end of the LDT, $\mathcal{V}$ needs oracle access to κ (we can set $\kappa = 4$ in a binary field, see [8] for more details) points of $f|_L$ and $h|_L$. $\mathcal{P}$ sends these points with their Merkle tree proofs, and $\mathcal{V}$ validates their correctness to complete the low degree test. A similar variant of the univariate sumcheck protocol where H and L are multiplication cosets of prime fields or extension fields, is adopted in [56]. The univariate sumcheck on multiplication cosets is also stated in [16]. We denote the univariate sumcheck protocol as $\langle \mathcal{P}_{\mathsf{SC}}(f), \mathcal{V}_{\mathsf{SC}}(r)\rangle(\mathbb{F}, L, H, d, \mu)$.

3 An Encoding of R1CS Instances as Univariate Polynomials

This section describes an encoding of R1CS instance as a low degree univariate polynomial. This encoding is inspired by the encoding of an R1CS instance as a univariate polynomial in Aurora [16] and a degree-3 multivariate polynomial recently proposed by Setty [48]. The encoding presented in this section relies on the bivariate Lagrange interpolation.

For a given R1CS instance $\mathbb{x} = (\mathbb{F}, A, B, C, v, m, n)$, let H be an affine subspace of $\mathbb{F}$ such that $|H| = m$ (padding m to the nearest power of Char($\mathbb{F}$) if necessary). We can interpret matrices A, B, $C \in \mathbb{F}^{H\times H}$ as bivariate functions: $H \times H \to \mathbb{F}$. Specifically, any entry in them can be accessed with a 2-entry identifier $(x, y) \in H \times H$. Furthermore, given a purported witness $w \in \mathbb{F}^{|H|-1-|v|}$ to $\mathbb{x}$, we also interpret $z = (1, v, w) \in \mathbb{F}^H$ as a univariate function $Z\colon H \to \mathbb{F}$, so any element of z can be assessed with an entry in H.

We now define the following function $F_w(\cdot)$ that is used to encode the vector z which includes the private witness w,

$$F_w(X) = \left(\sum_{y\in H} A(X,y)\cdot Z(y)\right)\cdot\left(\sum_{y\in H} B(X,y)\cdot Z(y)\right) - \left(\sum_{y\in H} C(X,y)\cdot Z(y)\right).$$

The following lemma easily follows from the definition of R1CS relation (Definition 2).

Lemma 2. *A pair $(\mathbb{x}, w)$ is a valid instance-witness pair, i.e., $(\mathbb{x}, w) \in \mathcal{R}_{\text{R1CS}}$ if and only if $F_w(x) = 0$ for any $x \in H$.*

As our protocol relies on the polynomial arithmetic of $F_w(\cdot)$, we need its polynomial extension. Let

$$\bar{A}(X) = \sum_{y\in H} A(X,y)\cdot Z(y),$$
$$\bar{B}(X) = \sum_{y\in H} B(X,y)\cdot Z(y),$$
$$\bar{C}(X) = \sum_{y\in H} C(X,y)\cdot Z(y).$$

Thus, to find the coefficients of polynomial extension $F_w(X)$, it suffices to find the coefficients of $\bar{A}(X)$, $\bar{B}(X)$ and $\bar{C}(X)$, which each has degree at most $|H|-1$. In turn, $F_w(X)$ can be computed from the coefficients of $\bar{A}(X)$, $\bar{B}(X)$ and $\bar{C}(X)$ in time $O(|H|\log|H|)$ via

fast polynomial multiplication and subtraction. To find the coefficients of the latter three polynomials, we can compute the vectors Az, Bz, Cz and then interpolate them over H. However, this step alone typically costs at least quadratic time because it involves computing the matrix-vector product, say Az, which is quadratic in $|H|$ and not within our budget of a prover with running cost $O(n \log n)$.

To work around this more efficiently, we define

$$\Delta_H(X,Y) := \frac{\mathbb{Z}_H(X) - \mathbb{Z}_H(Y)}{X - Y},$$

which is a polynomial of individual degree $|H| - 1$ because $X - Y$ divides $\mathbb{Z}_H(X) - \mathbb{Z}_H(Y)$. When H is an affine subspace, there exist coefficients $c_0, \ldots, c_k \in \mathbb{F}$, where $k := \dim(H)$, such that $\mathbb{Z}_H(X) = X^{p^k} + \sum_{i=1}^{k} c_i X^{p^{i-1}} + c_0$ ($c_0 = 0$ if H is linear). Then $\Delta_H(X,Y) = (X-Y)^{p^k-1} + \sum_{i=1}^{k} c_i (X-Y)^{p^{i-1}-1}$ for the case of $X \neq Y$. If instead $X = Y$ then we use a different approach: observing that the polynomial $\Delta_H(X,X)$ is the formal derivative of $\mathbb{Z}_H(X)$, we compute $\Delta_H(x,x) = \frac{d\mathbb{Z}_H}{dX}\big|_{X=x}$, which equals to the (non-zero) coefficient c_1. The bivariate polynomial $\Delta_H(X,Y)$ vanishes on the square $H \times H$ except for on the diagonal, where it takes the constant value c_1. See [43, Chapter 3.4] and [14, Remark C.8] for how to find the coefficients $c_0, \ldots, c_k$ in $O(\log^2 |H|)$ field operations. This trick is similar to the one used in [15, 28].

To find the coefficients of polynomial $\bar{M}(X)$ for $M \in \{A, B, C\}$, we can evaluate each of them over H and then interpolate. Let $\mathsf{row}, \mathsf{col} : [n] \to H$ and $\mathsf{val} : [n] \to \mathbb{F}$ respectively represent the row index, column index, and value of the non-zero entries of the matrix $M \in \{A, B, C\}$ in some canonical order. Thus, for every $x \in H$,

$$\begin{aligned}
&\bar{M}(x) \\
&= \sum_{y \in H} \left(\sum_{i \in [n]} \Delta_H(x, \mathsf{row}(i)) \Delta_H(y, \mathsf{col}(i)) \frac{\mathsf{val}(i)}{c_1^2} \right) \cdot Z(y) \\
&= \sum_{i \in [n]} \Delta_H(x, \mathsf{row}(i)) \sum_{y \in H} \Delta_H(y, \mathsf{col}(i)) \frac{\mathsf{val}(i)}{c_1^2} \cdot Z(y) \\
&= \sum_{i \in [n]} \Delta_H(x, \mathsf{row}(i)) \cdot \mathsf{val}(i)/c_1 \cdot Z(\mathsf{col}(i)) \\
&= \sum_{i \in [n] \text{ s.t. } \mathsf{row}(i) = x} \mathsf{val}(i) \cdot Z(\mathsf{col}(i)).
\end{aligned}$$

The leads to the following strategy to find the values of $\bar{M}(X)$ on H. For R1CS instance $\mathbb{x}$, when receiving the matrix $M \in \mathbb{F}^{H \times H}$ in a sparse form, the prover initializes for each $x \in H$ a variable for $\bar{M}(x)$ that is initially set to 0. Then for each $i \in [n]$, compute the term $\mathsf{val}(i) \cdot Z(\mathsf{col}(i))$ and add it to the variable for $\bar{M}(\mathsf{row}(i))$. The foregoing strategy can be evaluated in time $O(n)$. Therefore, the coefficients of $\bar{M}(X)$ can be found in $O(n + |H| \log |H|)$. A similar observation was made by Chiesa et al. [28] in a different context.

Overall the entire encoding can be evaluated in time $O(|H| \log |H|)$, which is inevitable due to the fast multiplication of two polynomials with degree $|H| - 1$. We remark that, unlike the offline phase in Marlin [28] where the algebraic holographic proof needs an indexer to output the nine univariate polynomial extensions of $\{\mathsf{row}_M, \mathsf{col}_M, \mathsf{val}_M\}_{M \in \{A,B,C\}}$, our encoding does not need any such offline preprocessing phase since the verifier does not need oracle access to these polynomials.

Next, we need to test that the univariate polynomial $F_w(X)$ is equal to zero everywhere on the affine subspace H of $\mathbb{F}$, it is equivalent via the factor theorem to test whether there exists a polynomial $G(X)$ with degree $\deg(F_w) - |H| \leq |H| - 2$ such that $F_w(X) = \mathbb{Z}_H(X) \cdot G(X)$. Thus, the prover only needs to compute and send $G(X)$ to the verifier, who can probabilistically check the identity at a random point r_x of $\mathbb{F} \setminus H$. This introduces a soundness error, which we quantify below.

Lemma 3. *If there exists an $x \in H$ such that $F_w(x) \neq 0$, then*

$$\Pr_{r_x \in \mathbb{F} \setminus H} [F_w(r_x) = \mathbb{Z}_H(r_x) \cdot G(r_x)] \leq \frac{2|H| - 2}{|\mathbb{F}| - |H|}.$$

The following theorem summarizes our results in this section, it is a univariate analogue of the multivariate variant described in Spartan [48].

Theorem 1. *For any R1CS instance $\mathbb{x} = (\mathbb{F}, A, B, C, v, |H|, n)$, there exist two univariate polynomials F_w and G with degree $\deg(F_w) \leq 2|H| - 2$ and $\deg(G) \leq |H| - 2$ such that $F_w(r_x) = \mathbb{Z}_H(r_x) \cdot G(r_x)$ for a random $r_x \in \mathbb{F} \setminus H$ if and only if thers exists a witness $w \in \mathbb{F}^{|H|-1-|v|}$ such that $(\mathbb{x}, w) \in \mathcal{R}_{\text{R1CS}}$ (except for a soundness error that is negligible in λ) under the assumption that $|\mathbb{F}|$ is exponential in λ and $|H| = O(\lambda)$.*

4 A Construction of zkSNARK for R1CS

In this section, we present our new construction, a zkSNARK without a trusted setup. We first construct an interactive argument that is correct and sound, then extend it to be zero-knowledge as well as non-interactive

(using the Fiat-Shamir transform [31]). The main idea is a combination of the univariate encoding and the univariate sumcheck in Aurora [16] described in Section 2.4.

According to Theorem 1, to verify whether an R1CS instance $\mathbb{x} = (\mathbb{F}, A, B, C, v, |H|, n)$ is satisfiable, the verifier $\mathcal{V}$ can check if $F_w(r_x) = \mathbb{Z}_H(r_x) \cdot G(r_x)$ for a random $r_x \in \mathbb{F} \setminus H$. A naive way is just to send the coefficients of $G(\cdot)$ to $\mathcal{V}$, and check the equality at a random point. In order to check the equality, the verifier $\mathcal{V}$ first needs to evaluate the polynomials $G(X)$ and $\mathbb{Z}_H(X)$ at $r_x \in \mathbb{F} \setminus H$. The latter can be evaluated in time $O(\log^2 |H|)$ locally. To evaluate $G(r_x)$ using Horner's method, $O(|H|)$ field operations and $O(|H|)$ coefficients are required for the verifier. This will incur linear communication and verification cost.

So our first observation is that to evaluate $G(r_x)$ without incurring $O(|H|)$ communication from $\mathcal{P}$ to $\mathcal{V}$ and $O(|H|)$ verification cost, we can employ a polynomial commitment scheme for univariate polynomials. This allows the verifier $\mathcal{V}$ to delegate the computations of polynomial evaluations to a prover, and validates the results in time that is sublinear in the size of the polynomial.

In slightly more details, the prover $\mathcal{P}$ sends a commitment to $G(\cdot)$ to the verifier $\mathcal{V}$ at the beginning of the protocol using a polynomial commitment scheme for univariate polynomials. Then the verifier $\mathcal{V}$ selects a random point $r_x \in \mathbb{F} \setminus H$ and sends it to $\mathcal{P}$. The prover $\mathcal{P}$ evaluates G at r_x and sends the result to $\mathcal{V}$. Finally, the prover and the verifier engage in a subprotocol PC.Eval to test whether the evaluation is correct.

To check $F_w(r_x) = \mathbb{Z}_H(r_x) \cdot G(r_x)$, $\mathcal{V}$ also needs to evaluate $F_w(r_x)$. Recall the definition of F_w,

$$F_w(r_x) = \left(\sum_{y \in H} A(r_x, y) \cdot Z(y)\right) \cdot \left(\sum_{y \in H} B(r_x, y) \cdot Z(y)\right) - \left(\sum_{y \in H} C(r_x, y) \cdot Z(y)\right).$$

Thus, the verifier $\mathcal{V}$ needs to evaluate $A(r_x, y)$, $B(r_x, y)$, $C(r_x, y)$ and $Z(y)$ for all $y \in H$. While the evaluations of $Z(y)$ for all $y \in H$ is equal to $z = (1, v, w)$, the communication from $\mathcal{P}$ to $\mathcal{V}$ is still at least linear in the size of witness. We address this issue by applying a univariate sumcheck protocol.

The method is a combination of two protocols: the univariate sumcheck protocol and a randomized mini protocol. Similarly as the multivariate variant in Spartan [48], the structure of the individual terms in $F_w(r_x)$ are also in a form suitable for the application of the univariate sumcheck protocol. Specifically, we note that $F_w(r_x) = \bar{A}(r_x) \cdot \bar{B}(r_x) - \bar{C}(r_x)$. This immediately suggests the following solution: The prover makes three separate claims to $\mathcal{V}$, say that $\bar{A}(r_x) = v_A$, $\bar{B}(r_x) = v_B$, and $\bar{C}(r_x) = v_C$. Then the verifier $\mathcal{V}$ can verify

$$v_A \cdot v_B - v_C = G(r_x) \cdot \mathbb{Z}_H(r_x).$$

Of course, the verifier must still verify three new claims from the prover $\mathcal{P}$:

$$\bar{A}(r_x) = v_A, \bar{B}(r_x) = v_B, \text{ and } \bar{C}(r_x) = v_C.$$

To do so, $\mathcal{P}$ and $\mathcal{V}$ could run three independent instances of the univariate sumcheck protocol to verify these three claims. But this will result in a 3-fold increase in complexity. However, we can do this much more efficiently. We adopt a standard technique used in [16, 27, 48, 54] to combine these three claims into a single claim. The idea is to have the verifier choose r_A, r_B, $r_C \in \mathbb{F}$ uniformly at random and send them to the prover, and then to test whether

$$c = r_A \cdot \bar{A}(r_x) + r_B \cdot \bar{B}(r_x) + r_C \cdot \bar{C}(r_x).$$

We can rewrite c as follows.

$$\begin{aligned} c &= r_A \cdot \sum_{y \in H} A(r_x, y) \cdot Z(y) + r_B \cdot \sum_{y \in H} B(r_x, y) \cdot Z(y) \\ &\quad + r_C \cdot \sum_{y \in H} C(r_x, y) \cdot Z(y) \\ &= \sum_{y \in H} \big(r_A \cdot A(r_x, y) + r_B \cdot B(r_x, y) + r_C \cdot C(r_x, y)\big) \cdot Z(y). \end{aligned}$$

Denote

$$Q_{r_x}(Y) := \big(r_A \cdot A(r_x, Y) + r_B \cdot B(r_x, Y) + r_C \cdot C(r_x, Y)\big) \cdot Z(Y).$$

We state formally the subprotocol as follows.

- $\mathcal{V}$ chooses r_A, r_B, $r_C \in \mathbb{F}$ uniformly at random, sends them to $\mathcal{P}$, and computes $c = r_A \cdot v_A + r_B \cdot v_B + r_C \cdot v_C$.
- $\mathcal{P}$ and $\mathcal{V}$ invoke the univariate sumcheck protocol to verify

$$c = \sum_{y \in H} Q_{r_x}(y).$$

The soundness follows directly from the Schwartz-Zippel lemma.

Lemma 4 ([48]). *Assuming that $\bar{A}(r_x) \neq v_A$, or $\bar{B}(r_x) \neq v_B$, or $\bar{C}(r_x) \neq v_C$, and let $c = r_A \cdot v_A + r_B \cdot v_B + r_C \cdot v_C$. Then*

$$\Pr_{r_A, r_B, r_C \in \mathbb{F}} \left[r_A \cdot \bar{A}(r_x) + r_B \cdot \bar{B}(r_x) + r_C \cdot \bar{C}(r_x) = c\right] \leq \frac{1}{|\mathbb{F}|}.$$

The efficiency of this combined verification corresponds to a single invocation of the univariate sumcheck. The prover and verifier, in addition to the cost of running the univariate sumcheck protocol, pay only a constant cost.

Now we are ready to present the full protocol provided that there exists a polynomial commitment scheme for univariate polynomials, $\mathsf{PC} = (\mathsf{Setup}, \mathsf{Commit}, \mathsf{Open}, \mathsf{Eval})$. The full protocol is given in Protocol 1.

Protocol 1. Given an R1CS instance $\mathbb{x} = (\mathbb{F}, A, B, C, v, m, n)$, we fix an affine subspace $H \subseteq \mathbb{F}$ such that $|H| = m$ (padding m to the nearest power of $\mathrm{Char}(\mathbb{F})$ if necessary), and a sufficiently large affine subspace $L \subseteq \mathbb{F}$ such that $|L| = O(|H|) > 2|H|$ and $L \cap H = \varnothing$.

- $\mathsf{pp} \leftarrow \mathsf{PC.Setup}(1^\lambda, m-2)$.
- $b \leftarrow \langle \mathcal{P}(\mathsf{pp}, w), \mathcal{V}(\mathsf{pp}, r)\rangle(\mathbb{x})$:
  1. $\mathcal{P}$ computes $F_w(X)$, $G(X) = F_w(X)/\mathbb{Z}_H(X)$, and $(\mathcal{C}_G, \mathcal{S}_G) \leftarrow \mathsf{PC.Commit}(\mathsf{pp}; G)$, then sends $\mathcal{C}_G$ to $\mathcal{V}$.
  2. $\mathcal{V}$ samples $r_x \in \mathbb{F} \setminus H$ and sends r_x to $\mathcal{P}$.
  3. $\mathcal{P}$ computes $\eta = G(r_x)$ and sends η to $\mathcal{V}$.
  4. $b_e \leftarrow \langle \mathcal{P}_{\mathsf{PC.Eval}}(G, \mathcal{S}_G), \mathcal{V}_{\mathsf{PC.Eval}}(r)\rangle(\mathsf{pp}, \mathcal{C}_G, r_x, \eta, m-2)$.
  5. $\mathcal{V}$ aborts and output $b = 0$ if $b_e = 0$.
  6. $\mathcal{P}$ computes $v_A = \bar{A}(r_x)$, $v_B = \bar{B}(r_x)$, $v_C = \bar{C}(r_x)$, and send (v_A, v_B, v_C) to $\mathcal{V}$.
  7. $\mathcal{V}$ computes $\gamma = \mathbb{Z}_H(r_x)$, and aborts with output $b = 0$ if $v_A \cdot v_B - v_C \neq \eta \cdot \gamma$.
  8. $\mathcal{V}$ samples r_A, r_B, $r_C \in \mathbb{F}$, and sends (r_A, r_B, r_C) to $\mathcal{P}$.
  9. $\mathcal{V}$ computes $c = r_A \cdot v_A + r_B \cdot v_B + r_C \cdot v_C$.
  10. $\mathcal{P}$ and $\mathcal{V}$ invoke a univariate sumcheck protocol $b_s \leftarrow \langle \mathcal{P}_{\mathsf{SC}}(Q_{r_x}), \mathcal{V}_{\mathsf{SC}}(r)\rangle(\mathbb{F}, L, H, 2m-1, c)$.
  11. $\mathcal{V}$ aborts and output $b = 0$ if $b_s = 0$.
  12. $\mathcal{V}$ output $b = 1$.

4.1 Efficiency Analysis

Both parties run the univariate sumcheck as a subroutine. In addition, the prover $\mathcal{P}$ has the following costs:

1. $O(m \log m)$ cost to compute the encoding $F_w(X)$ as stated in Section 3;
2. the cost of $\mathsf{PC.Commit}$ and $\mathsf{PC.Eval}$ for a univariate polynomial $G(\cdot)$ with degree $m-2$;
3. $O(m \log m)$ costs to compute the polynomial $G(\cdot)$, which can be achieved via running a divide-and-conquer algorithm to obtain the $\log m$ coefficients of $\mathbb{Z}_H(\cdot)$ in time $O(\log^2 m)$, and performing the standard polynomial division of $F_w(X)$ by $\mathbb{Z}_H(X)$ in time $O((2m-2)\log m)$;
4. $O(m)$ costs to evaluate $G(r_x)$, $\bar{A}(r_x)$, $\bar{B}(r_x)$, and $\bar{C}(r_x)$.

The verifier $\mathcal{V}$ has

1. the cost of $\mathsf{PC.Eval}$ for a univariate polynomial $G(\cdot)$ with degree $m-2$;
2. $O(\log^2 m)$ cost to evaluate $\mathbb{Z}_H(r_x)$, which can be achieved via running a divide-and-conquer algorithm to obtain the $O(\log m)$ coefficients of $\mathbb{Z}_H(X)$ in time $O(\log^2 m)$, and evaluating $\mathbb{Z}_H$ at a single point in time $O(\log m)$.

The amount of communication mainly consists of the size of the commitment to $G(\cdot)$, the communication in $\mathsf{PC.Eval}$ for $G(r_x)$ and the communication in the univariate sumcheck protocol.

As stated above, the particular choice of polynomial commitment schemes impacts the costs of our construction as well as the assumptions, so we list some prior polynomial commitment schemes (see Appendix B for more details) without a trusted setup as well as their comparisons in Table 2.

4.2 Succinct Verification Cost

In this subsection, we reduce the cost of the verifier $\mathcal{V}$ in the univariate sumcheck protocol. As described in Section 2.4, at the end of the univariate sumcheck protocol, due to the low degree test, $\mathcal{V}$ needs oracle access to the evaluations of $Q_{r_x}(\cdot)$ at κ points $r_y \in L$:

$$Q_{r_x}(r_y) = \big(r_A \cdot A(r_x, r_y) + r_B \cdot B(r_x, r_y) + r_C \cdot C(r_x, r_y)\big) \cdot Z(r_y).$$

As the only term in $Q_{r_x}(r_y)$ that depends on the prover's witness is $Z(r_y)$, the prover $\mathcal{P}$ can commit to $Z|_L$ at the beginning of the protocol, and opens to points the verifier queries with their Merkle tree proofs. All the other terms in the above expression are determined by the problem instance $\mathbb{x}$, and can be computed locally by $\mathcal{V}$, which, however, takes at least linear time.

Recall the closed-form expression for the evaluations of $M(\cdot,\cdot)$ at $(r_x, r_y) \in (\mathbb{F} \setminus H) \times L$ where $M \in \{A, B, C\}$. We may rewrite it as

$$M(r_x, r_y) = \sum_{i \in [n]} \Delta_H(r_x, \mathsf{row}(i)) \cdot \Delta_H(r_y, \mathsf{col}(i)) \cdot \frac{\mathsf{val}(i)}{c_1^2}.$$

In Spartan [48], Setty introduced a new cryptographic compiler, called SPARK, to transform an existing extractable polynomial commitment scheme for dense multilinear polynomials to one that can efficiently handle sparse multilinear polynomials. By composing SPARK with computation commitments, the verification cost is reduced to be sublinear in n. Note that here $M(r_x, r_y)$ is in a similar form as that in Spartan [48]. So the SPARK compiler can be directly applied to our case, but it requires a modification for our setting where the multilinear extensions $\widetilde{\mathrm{eq}}(i, r_x)$ and $\widetilde{\mathrm{eq}}(j, r_y)$ in SPARK need to be replaced with $\Delta_H(r_x, \mathsf{row}(i))$ and $\Delta_H(r_y, \mathsf{col}(i))$.

Table 2. A comparison of transparent polynomial commitment schemes

scheme	$\mathcal{P}_{\mathsf{Eval}}$	$\lvert\mathcal{C}\rvert$	communication	$\mathcal{V}_{\mathsf{Eval}}$	assumption
DARK [23]	$O(d \log d)$ $\mathbb{G}_U$	$O(1)\lvert\mathbb{G}_U\rvert$	$O(\log d)$ $\mathbb{G}_U$	$O(\log d)$ $\mathbb{G}_U$	r-strong RSA/adaptive root
FRI [8], Virgo-VPD [56]	$O(d \log d)$ $\mathbb{H}$	$O(1)\lvert\mathbb{H}\rvert$	$O(\log^2 d)$ $\mathbb{H}$	$O(\log^2 d)$ $\mathbb{H}$	CRHF
Hyrax-PC [53]	$O(d)$ $\mathbb{G}$	$O(\sqrt{d})\lvert\mathbb{G}\rvert$	$O(\log d)$ $\mathbb{G}$	$O(\sqrt{d})\mathbb{G}$	DLOG
IPP-PC [25]	$O(d)$ $\mathbb{G}_1$	$O(1)\lvert\mathbb{G}_T\rvert$	$O(\log d)$ $\mathbb{G}_T$	$O(\sqrt{d})$ $\mathbb{G}_2$	SXDH
Dory [42]	$O(d)$ $\mathbb{G}_1$	$O(1)\lvert\mathbb{G}_T\rvert$	$O(\log d)$ $\mathbb{G}_T$	$O(\log d)$ $\mathbb{G}_2$	SXDH

$\mathbb{G}_U$ is a group of unknown order, $\mathbb{G}$ is a group, $\mathbb{H}$ is either the size of a hash output, or the time it takes to compute a hash, $\mathbb{G}_1$, $\mathbb{G}_2$, $\mathbb{G}_T$ denote the two source groups and the target group of a pairing P. The communication column refers to the amount of communication required in the interactive argument for $\mathsf{PC.Eval}$.

Next we propose an alternative approach to reduce the cost of the verifier to polylogarithmic for our construction. Specifically, we note that $\mathbb{Z}_H(\mathsf{row}(i)) = \mathbb{Z}_H(\mathsf{col}(i)) = 0$ because $\mathsf{row}(i), \mathsf{col}(i) \in H$ for every $i \in [n]$. Therefore,

$$\begin{aligned} & M(r_x, r_y) \\ &= \sum_{i\in[n]} \frac{\mathbb{Z}_H(r_x) - \mathbb{Z}_H(\mathsf{row}(i))}{r_x - \mathsf{row}(i)} \cdot \frac{\mathbb{Z}_H(r_y) - \mathbb{Z}_H(\mathsf{col}(i))}{r_y - \mathsf{col}(i)} \cdot \frac{\mathsf{val}(i)}{c_1^2} \\ &= \sum_{i\in[n]} \frac{\mathbb{Z}_H(r_x)}{r_x - \mathsf{row}(i)} \cdot \frac{\mathbb{Z}_H(r_y)}{r_y - \mathsf{col}(i)} \cdot \frac{\mathsf{val}(i)}{c_1^2} \\ &= \frac{\mathbb{Z}_H(r_x) \cdot \mathbb{Z}_H(r_y)}{c_1^2} \cdot \sum_{i\in[n]} \frac{\mathsf{val}(i)}{(r_x - \mathsf{row}(i))(r_y - \mathsf{col}(i))}. \end{aligned}$$

The first part $\mathbb{Z}_H(r_x) \cdot \mathbb{Z}_H(r_y)/c_1^2$ can be evaluated in time $O(\log m)$ locally by the verifier. Note that the value of $\mathbb{Z}_H(r_x)$ has been computed in previous steps, so the verifier only needs to evaluate $\mathbb{Z}_H(r_y)$ here.

Instead of evaluating the second part $\sum_{i\in[n]} \mathsf{val}(i)/((r_x - \mathsf{row}(i))(r_y - \mathsf{col}(i)))$ locally, which takes linear time, the verifier can delegate this computation to the prover, and validate the result using the GKR protocol, as described in Appendix A. In this way, we can eliminate the overhead to evaluate these points locally, making the verification time of the overall protocol polylogarithmic. To avoid any asymptotic overhead for the prover and verifier, we design an arithmetic circuit for the computation mentioned above. The details of the circuit are presented in Figure 1.

Note that the above circuit is layered: There are three identical copies of a sub-circuit, where each sub-circuit firstly computes $2n$ subtractions (with size $O(n)$ and depth $O(1)$), n multiplications (with size $O(n)$ and depth $O(1)$) and n divisions (with size $O(n)$ and depth $O(1)$); the outputs of Step 3 are then fed into a binary tree of addition gates with size $O(n)$ and depth $O(\log n)$ to compute the final sums. Furthermore, there is no sharing of elements across the three data-parallel units, so it truly data-parallel. Therefore, the whole circuit is of size $O(n)$ and depth $O(\log n)$, with 2 inputs and 3 outputs. By Lemma 5 (see Appendix A for details), the prover time is $O(n)$, the proof size and the verification time are $(\log^2 n)$ for the query delegation computation. Note that the values $\mathsf{row}_M(i)$, $\mathsf{col}_M(i)$, $\mathsf{val}_M(i)$, $i \in [n]$ are directly hard-coded into the circuit. This needs a (public) per-circuit preprocessing step for $\mathcal{V}$, leading to a sublinear verifier in the holographic setting.

Input: $(r_x, r_y) \in (\mathbb{F} \setminus H) \times L$

Output: $\left(\sum_{i\in[n]} \frac{\mathsf{val}_M(i)}{(r_x - \mathsf{row}_M(i))(r_y - \mathsf{col}_M(i))}\right)_{M\in\{A,B,C\}}$

1. For $i = 1, 2, \ldots, n$:
   Compute $r_x - \mathsf{row}_M(i)$ and $r_y - \mathsf{col}_M(i)$ for each $M \in \{A, B, C\}$.
2. For $i = 1, 2, \ldots, n$:
   Compute $(r_x - \mathsf{row}_M(i)) \cdot (r_y - \mathsf{col}_M(i))$ for each $M \in \{A, B, C\}$.
3. For $i = 1, 2, \ldots, n$:
   Compute $\frac{\mathsf{val}_M(i)}{(r_x - \mathsf{row}_M(i)) \cdot (r_y - \mathsf{col}_M(i))}$ for each $M \in \{A, B, C\}$.
4. Compute and output the three sums

$$\left(\sum_{i\in[n]} \frac{\mathsf{val}_M(i)}{(r_x - \mathsf{row}_M(i))(r_y - \mathsf{col}_M(i))}\right)_{M\in\{A,B,C\}}$$

Fig. 1. Arithmetic circuit $\mathfrak{C}$ for computing the evaluations of $\left(\sum_{i\in[n]} \frac{\mathsf{val}_M(i)}{(r_x - \mathsf{row}_M(i))(r_y - \mathsf{col}_M(i))}\right)_{M\in\{A,B,C\}}$

Theorem 2. *Given a polynomial commitment scheme for univariate polynomials, Protocol 2 is a public-coin succinct interactive argument for the R1CS relation where security holds under the assumptions needed for the polynomial commitment scheme and assuming $|\mathbb{F}|$ is exponential in λ and exponentially bigger than the size parameters m, n of R1CS instance.*

The proof is given in Appendix C.

4.3 Zero-Knowledge

The interactive protocol from the prior subsections is not zero-knowledge, so we present the modifications on the protocol to achieve zero-knowledge. Intuitively, there are three different building blocks that may reveal

Protocol 2. Given an R1CS instance $\mathbb{x} = (\mathbb{F}, A, B, C, v, m, n)$, we fix an affine subspace $H \subseteq \mathbb{F}$ such that $|H| = m$ (padding m to the nearest power of Char($\mathbb{F}$) if necessary), and a sufficiently large affine subspace $L \subseteq \mathbb{F}$ such that $|L| = O(|H|) > 2|H|$ and $L \cap H = \varnothing$.

– $(\mathsf{pp}, \mathcal{H}) \leftarrow (\mathsf{PC.Setup}(1^\lambda, m-2), \mathsf{MT.Setup}(1^\lambda))$: generates public parameters pp and picks a hash function from the collision-resistant hash function family for Merkle trees.

– $b \leftarrow \langle \mathcal{P}(\mathsf{pp}, w), \mathcal{V}(\mathsf{pp}, r)\rangle(\mathbb{x})$:

1. $\mathcal{P}$ finds the unique univariate polynomial $Z : \mathbb{F} \to \mathbb{F}$ such that $Z|_H = (1, v, w)$, evaluates $Z|_L$ and runs $\mathsf{root}_Z \leftarrow \mathsf{MT.Commit}(Z|_L)$. Then $\mathcal{P}$ sends root_Z to $\mathcal{V}$.
2. $\mathcal{P}$ computes $F_w(X)$, $G_1(X) = F_w(X)/\mathbb{Z}_H(X)$, and $(\mathcal{C}_{G_1}, \mathcal{S}_{G_1}) \leftarrow \mathsf{PC.Commit}(\mathsf{pp}; G_1)$, then sends $\mathcal{C}_{G_1}$ to $\mathcal{V}$.
3. $\mathcal{V}$ samples $r_x \in \mathbb{F} \setminus H$ and sends r_x to $\mathcal{P}$.
4. $\mathcal{P}$ computes $\eta = G_1(r_x)$ and sends η to $\mathcal{V}$.
5. $b_e \leftarrow \langle \mathcal{P}_{\mathsf{PC.Eval}}(G_1, \mathcal{S}_{G_1}), \mathcal{V}_{\mathsf{PC.Eval}}(r)\rangle(\mathsf{pp}, \mathcal{C}_{G_1}, r_x, \eta, m-2)$.
6. $\mathcal{V}$ aborts and outputs $b = 0$ if $b_e = 0$.
7. $\mathcal{P}$ computes $v_A = \bar{A}(r_x)$, $v_B = \bar{B}(r_x)$, $v_C = \bar{C}(r_x)$, and sends (v_A, v_B, v_C) to $\mathcal{V}$.
8. $\mathcal{V}$ computes $\gamma = \mathbb{Z}_H(r_x)$, and aborts with output $b = 0$ if $v_A \cdot v_B - v_C \neq \eta \cdot \gamma$.
9. $\mathcal{V}$ samples r_A, r_B, $r_C \in \mathbb{F}$, and sends (r_A, r_B, r_C) to $\mathcal{P}$.
10. $\mathcal{V}$ computes $c = r_A \cdot v_A + r_B \cdot v_B + r_C \cdot v_C$.
11. $\mathcal{P}$ computes $Q_{r_x}(\cdot)$ and decomposes uniquely $Q_{r_x}(Y) = G_2(Y) + \mathbb{Z}_H(Y) \cdot h(Y)$ with $\deg G_2 < m$ and $\deg h < m-1$. $\mathcal{P}$ evaluates $h|_L$ and runs $\mathsf{root}_h \leftarrow \mathsf{MT.Commit}(h|_L)$. $\mathcal{P}$ then sends root_h to $\mathcal{V}$.
12. Let $p(Y) = c_1 \cdot Q_{r_x}(Y) - c \cdot Y^{m-1} - c_1 \cdot \mathbb{Z}_H(Y) \cdot h(Y)$. $\mathcal{P}$ and $\mathcal{V}$ then invoke a low degree test (LDT): $\langle \mathcal{P}_{\mathsf{LDT}}(((Q_{r_x}, h), p)), \mathcal{V}_{\mathsf{LDT}}(((2m-1, m-1), m-1))\rangle(L)$. If the LDT fails, $\mathcal{V}$ aborts and outputs $b = 0$. Otherwise, at the end of the LDT, $\mathcal{V}$ needs oracle access to κ points of $Q_{r_x}(\cdot)$, $h(\cdot)$ and $p(\cdot)$ at query indices $\mathcal{Q} \subseteq L$.
13. For each index $i \in \mathcal{Q}$, let a_i be the corresponding point in L.
    (a) $\mathcal{P}$ opens $(Z(a_i), \pi_i^Z) \leftarrow \mathsf{MT.Open}(i, Z|_L)$ and $(h(a_i), \pi_i^h) \leftarrow \mathsf{MT.Open}(i, h|_L)$ and sends them to $\mathcal{V}$.
    (b) $\mathcal{V}$ executes $\mathsf{MT.Verify}(\mathsf{root}_Z, i, Z(a_i), \pi_i^Z)$ and $\mathsf{MT.Verify}(\mathsf{root}_h, i, h(a_i), \pi_i^h)$ for point $a_i \in L$ opened by $\mathcal{P}$. If any verification fails, $\mathcal{V}$ aborts and outputs $b = 0$.
    (c) $\mathcal{P}$ and $\mathcal{V}$ then invoke the GKR protocol: $\langle \mathcal{P}_{\mathsf{GKR}}, \mathcal{V}_{\mathsf{GKR}}\rangle(\mathfrak{C}, (r_x, a_i))$, where the circuit $\mathfrak{C}$ computes the evaluation and outputs $\left(\sum_{i\in[n]} \mathsf{val}_M(i)/((r_x - \mathsf{row}_M(i))(a_i - \mathsf{col}_M(i)))\right)_{M\in\{A,B,C\}}$ (denoted by $(\Gamma_{i,A}, \Gamma_{i,B}, \Gamma_{i,C})$, see Figure 1). If the check in GKR fails, $\mathcal{V}$ aborts and outputs $b = 0$.
    (d) $\mathcal{V}$ computes $Q_{r_x}(a_i) = 1/c_1^2 \cdot \gamma \cdot \mathbb{Z}_H(a_i) \cdot \left(r_A \cdot \Gamma_{i,A} + r_B \cdot \Gamma_{i,B} + r_C \cdot \Gamma_{i,C}\right) \cdot Z(a_i)$, together with $h(a_i)$, $\mathcal{V}$ checks the constraint $p(a_i) = N(a_i, Q_{r_x}(a_i), h(a_i))$ at index $i \in \mathcal{Q}$. If the check fails, $\mathcal{V}$ aborts and outputs $b = 0$.
14. $\mathcal{V}$ outputs $b = 1$.

information about the NP witness. First, the prover $\mathcal{P}$ sends evaluations of $G_1(r_x)$, $\bar{A}(r_x)$, $\bar{B}(r_x)$ and $\bar{C}(r_x)$ to the verifier and these polynomials are directly defined by the circuit and the witness w. Second, $\mathcal{P}$ and $\mathcal{V}$ invoke the low degree test on $((Q_{r_x}(Y), h(Y)), p(Y))$ and the proof of LDT reveals information about these polynomials, which are directly related to the witness input w. Third, at the end of the univariate sumcheck protocol, due to the low degree test, the prover $\mathcal{P}$ opens some evaluations of $Z(\cdot)$, which is defined by the witness w. Therefore, all these types of information give the verifier partial information about the NP witness.

Fortunately, making the protocol zero-knowledge only requires introducing small modifications to the constructions and analysis. To mitigate the first leakage, we take the technique proposed by Chiesa et al. [27]. To ensure zero-knowledge, the prover $\mathcal{P}$ masks these polynomials $\bar{A}(\cdot)$, $\bar{B}(\cdot)$ and $\bar{C}(\cdot)$ by random polynomials so that the evaluations do not leak any information. For the correctness and soundness purposes, these random polynomials are committed using the Merkle trees and opened at random points chosen by the verifier. To mitigate leakages of the last two types, we take the standard approaches proposed in [7, 11, 12, 16]. Specifically, for the second leakage, the prover samples a random polynomial of the same degree and then runs the univariate sumcheck protocol on their random linear combination. For the last leakage, we increase the degree of the RS code $Z|_L$ by κ, such that opening any κ points of the codeword does not leak any information about the message.

Below we describe the modifications required for each of these leakages.

1. Modification to the polynomial $Z(\cdot)$. To eliminate the leakage of queries on $Z(\cdot)$, the prover samples a random polynomial $R_Z(\cdot)$ of degree κ, and defines

$$\widetilde{Z}(Y) := Z(Y) + \mathbb{Z}_H(Y) \cdot R_Z(Y).$$

Note here that $\widetilde{Z}(y) = Z(y)$ for all $y \in H$, yet any κ evaluations of $\widetilde{Z}(\cdot)$ outside H do not leak any information about $Z(\cdot)$, as it is masked by a random polynomial $R_Z(\cdot)$, thus the values in witness w.

2. Modifications to the evaluations of $\bar{A}(\cdot)$, $\bar{B}(\cdot)$ and $\bar{C}(\cdot)$. The prover selects three random polynomials $R_A(\cdot)$, $R_B(\cdot)$ and $R_C(\cdot)$ with degree $|H|-1$ and defines

$$\widetilde{A}(X) := \sum_{y\in H} A(X,y)\cdot\widetilde{Z}(y) + \mathbb{Z}_H(X)\cdot\sum_{y\in H} R_A(y),$$

$$\widetilde{B}(X) := \sum_{y\in H} B(X,y)\cdot\widetilde{Z}(y) + \mathbb{Z}_H(X)\cdot\sum_{y\in H} R_B(y),$$

$$\widetilde{C}(X) := \sum_{y\in H} C(X,y)\cdot\widetilde{Z}(y) + \mathbb{Z}_H(X)\cdot\sum_{y\in H} R_C(y).$$

As $R_A(\cdot)$, $R_B(\cdot)$ and $R_C(\cdot)$ are randomly selected by $\mathcal{P}$ of degree equal to $|H|-1$, the sums $\sum_{y\in H} R_A(y)$, $\sum_{y\in H} R_B(y)$ and $\sum_{y\in H} R_C(y)$ are also random by Lemma 1 (because the leading coefficients are random). Note here that, for example, $\widetilde{A}(x) = \bar{A}(x)$ for all $x \in H$, revealing evaluation of $\widetilde{A}(\cdot)$ outside H does not leak any information about $\bar{A}(\cdot)$, thus the values in witness.

3. Modification to the polynomial $Q_{r_x}(\cdot)$. When replaced the three evaluations of $\bar{A}(\cdot)$, $\bar{B}(\cdot)$, $\bar{C}(\cdot)$ and polynomial $Z(\cdot)$ with their corresponding zero-knowledge versions, the polynomial $Q_{r_x}(\cdot)$ in the univariate sumcheck now becomes

$$\widetilde{Q}_{r_x}(Y) := \Big(r_A\cdot A(r_x,Y) + r_B\cdot B(r_x,Y) + r_C\cdot C(r_x,Y)\Big)\cdot\widetilde{Z}(Y) + \mathbb{Z}_H(r_x)\cdot\Big(r_A\cdot R_A(Y) + r_B\cdot R_B(Y) + r_C\cdot R_C(Y)\Big).$$

To make it zero-knowledge, the prover first samples a random polynomial $S_Q(\cdot)$ of the same degree (i.e., $2|H|+\kappa-1$) as $\widetilde{M}_{r_x}(\cdot)$, sends $s_1 = \sum_{y\in H} S_Q(y)$ to $\mathcal{V}$. The verifier then replies with a random challenge element $\alpha_1 \in \mathbb{F}$. Finally, the prover and verifier run the univariate sumcheck protocol on their linear combination:

$$\alpha_1\cdot\widetilde{c} + s_1 = \sum_{y\in H}\Big(\alpha_1\cdot\widetilde{Q}_{r_x}(y) + S_Q(y)\Big),$$

where $\widetilde{c} = r_A\cdot\widetilde{A}(r_x) + r_B\cdot\widetilde{B}(r_x) + r_C\cdot\widetilde{C}(r_x)$. This ensures that both $\widetilde{c}$ and s_1 can be correctly computed because of the random linear combination and the linearity of the univariate sumcheck, while revealing no information about $\widetilde{Q}_{r_x}(\cdot)$ during the protocol, as it is masked by an (almost) uniformly random polynomial (RS codeword) [16, 56].

Recall that $G_1(X) = F_w(X)/\mathbb{Z}_H(X)$ in Protocol 2. Thus, when we replace $\bar{A}(\cdot)$, $\bar{B}(\cdot)$ and $\bar{C}(\cdot)$ with their corresponding zero-knowledge versions, the closed-form of $G_1(\cdot)$ can be computed as:

$$F_w(X)/\mathbb{Z}_H(X) + \bar{A}(X)\cdot\sum_{y\in H} R_B(y) + \bar{B}(X)\cdot\sum_{y\in H} R_A(y) + \mathbb{Z}_H(X)\cdot\sum_{y\in H} R_A(y)\cdot\sum_{y\in H} R_B(y) - \sum_{y\in H} R_C(y).$$

Even though the sums $\sum_{y\in H} R_A(y)$, $\sum_{y\in H} R_B(y)$ and $\sum_{y\in H} R_C(y)$ are random, we cannot efficiently simulate the PC.Eval prover with this polynomial $G_1(\cdot)$. To solve this problem, we can apply a zero-knowledge Eval protocol. The idea is a simple blinding of the polynomial as the modification in the second type above. To do this, instead, the prover commits to a random polynomial $S_G(X)$ of the same degree (i.e., $|H|$), and sends $s_2 = S_G(r_x)$ to $\mathcal{V}$. The verifier then replies with a random challenge $\alpha_2 \in \mathbb{F}$. The prover and verifier can then use the standard Eval protocol to evaluate $\alpha_2\cdot G_1(X) + S_G(X)$ at r_x. A similar observation was made by Chiesa et al. in [27] and Bünz et al. in DARK [23].

To obtain the final zero-knowledge version, we replace the three aforementioned ingredients of Protocol 2 with their corresponding zero-knowledge versions. In addition, for the correctness and soundness purposes, these random polynomials $R_A(\cdot)$, $R_B(\cdot)$, $R_C(\cdot)$ and $S_Q(\cdot)$ also need to be committed using the Merkle trees at the beginning of the protocol and opened at κ points chosen by the verifier.

We have the same advantage in our construction as stated in [56], namely the GKR protocol used in Protocol 2 remains unchanged in the final zero-knowledge version. This is because evaluations of $r_A\cdot A(\cdot,\cdot) + r_B\cdot B(\cdot,\cdot) + r_C\cdot C(\cdot,\cdot)$ is independent of the witness input. Therefore, we can apply the plain version of the GKR protocol without zero-knowledge, avoiding any expensive cryptographic primitives. We present the full zero-knowledge protocol, combining all the previous discusses, in Protocol 3.

Theorem 3. *Given a polynomial commitment scheme for univariate polynomials, Protocol 3 is a public-coin zero-knowledge succinct interactive argument for the R1CS relation where security holds under the assumptions needed for the polynomial commitment scheme and assuming $|\mathbb{F}|$ is exponential in λ and exponentially bigger than the size parameters m, n of R1CS instance.*

The proof is given in Appendix D.

Efficiency. The running time of MT.Commit is $O(n\log n)$. The circuit described in Figure 1 is a regular circuit with size $O(n)$, depth $O(\log n)$ and input and

Protocol 3. (Zero-knowledge argument for R1CS) Given an R1CS instance $\mathbb{x} = (\mathbb{F}, A, B, C, v, m, n)$, we fix an affine subspace $H \subseteq \mathbb{F}$ such that $|H| = m$ (padding m to the nearest power of Char($\mathbb{F}$) if necessary), and a sufficiently large affine subspace $L \subseteq \mathbb{F}$ such that $|L| = O(|H|) > 2|H|$ and $L \cap H = \varnothing$.

- $(\mathsf{pp}, \mathcal{H}) \leftarrow (\mathsf{PC.Setup}(1^\lambda, 2m + \kappa - 1), \mathsf{MT.Setup}(1^\lambda))$: generates public parameters pp and picks a hash function from the collision-resistant hash function family for Merkle tree.
- $b \leftarrow \langle \mathcal{P}(\mathsf{pp}, w), \mathcal{V}(\mathsf{pp}, r)\rangle(\mathbb{x})$:
  1. $\mathcal{P}$ finds the unique univariate polynomial $Z : \mathbb{F} \to \mathbb{F}$ such that $Z|_H = (1, v, w)$. $\mathcal{P}$ samples a polynomial $R_Z(Y)$ of degree κ with random coefficients and sets $\widetilde{Z}(Y) = Z(Y) + \mathbb{Z}_H(Y) \cdot R_Z(Y)$. $\mathcal{P}$ evaluates $\widetilde{Z}|_L$ and runs $\mathsf{root}_{\widetilde{Z}} \leftarrow \mathsf{MT.Commit}(\widetilde{Z}|_L)$. Then $\mathcal{P}$ sends $\mathsf{root}_{\widetilde{Z}}$ to $\mathcal{V}$.
  2. $\mathcal{P}$ samples three polynomials $R_A(Y)$, $R_B(Y)$ and $R_C(Y)$ of degree $m-1$ with random coefficients. $\mathcal{P}$ evaluates $R_A|_L$, $R_B|_L$, $R_C|_L$, and runs $\mathsf{root}_{R_A} \leftarrow \mathsf{MT.Commit}(R_A|_L)$, $\mathsf{root}_{R_B} \leftarrow \mathsf{MT.Commit}(R_B|_L)$, $\mathsf{root}_{R_C} \leftarrow \mathsf{MT.Commit}(R_C|_L)$. Then $\mathcal{P}$ sends root_{R_A}, root_{R_B} and root_{R_C} to $\mathcal{V}$.
  3. $\mathcal{P}$ computes $\widetilde{F}_w(X) = \widetilde{A}(X) \cdot \widetilde{B}(X) - \widetilde{C}(X)$, and $G_1(X) = \widetilde{F}_w(X)/\mathbb{Z}_H(X)$. Then $\mathcal{P}$ runs $(\mathcal{C}_{G_1}, \mathcal{S}_{G_1}) \leftarrow \mathsf{PC.Commit}(\mathsf{pp}; G_1)$, and sends $\mathcal{C}_{G_1}$ to $\mathcal{V}$.
  4. $\mathcal{V}$ samples $r_x \in \mathbb{F} \setminus H$ and sends r_x to $\mathcal{P}$.
  5. $\mathcal{P}$ computes $\eta = G_1(r_x)$ and sends η to $\mathcal{V}$.
  6. $b_e \leftarrow \langle \mathcal{P}_{\mathsf{PC.Eval}}(G_1, \mathcal{S}_{G_1}), \mathcal{V}_{\mathsf{PC.Eval}}(r)\rangle(\mathsf{pp}, \mathcal{C}_{G_1}, r_x, \eta, m)$.
  7. $\mathcal{V}$ aborts and outputs $b = 0$ if $b_e = 0$.
  8. $\mathcal{P}$ computes $v_A = \widetilde{A}(r_x)$, $v_B = \widetilde{B}(r_x)$, $v_C = \widetilde{C}(r_x)$, and sends (v_A, v_B, v_C) to $\mathcal{V}$.
  9. $\mathcal{V}$ computes $\gamma = \mathbb{Z}_H(r_x)$, and aborts with output $b = 0$ if $v_A \cdot v_B - v_C \neq \eta \cdot \gamma$.
  10. $\mathcal{V}$ samples r_A, r_B, $r_C \in \mathbb{F}$, and sends (r_A, r_B, r_C) to $\mathcal{P}$.
  11. $\mathcal{V}$ computes $\widetilde{c} = r_A \cdot v_A + r_B \cdot v_B + r_C \cdot v_C$.
  12. $\mathcal{P}$ samples a polynomial $S_Q(Y)$ of degree $2m + \kappa - 1$ with random coefficients and computes $s_1 = \sum_{y \in H} S_Q(y)$. $\mathcal{P}$ evaluates $S_Q|_L$ and runs $\mathsf{root}_{S_Q} \leftarrow \mathsf{MT.Commit}(S_Q|_L)$. Then $\mathcal{P}$ sends s_1 and root_{S_Q} to $\mathcal{V}$.
  13. $\mathcal{V}$ samples $\alpha \in \mathbb{F}$ randomly, and sends α to $\mathcal{P}$.
  14. $\mathcal{P}$ computes $\alpha \cdot \widetilde{Q}_{r_x}(Y) + S_Q(Y)$ and decomposes uniquely as $G_2(Y) + \mathbb{Z}_H(Y) \cdot h(Y)$ with $\deg G_2 < m$ and $\deg h < m + \kappa$. $\mathcal{P}$ evaluates $h|_L$ and runs $\mathsf{root}_h \leftarrow \mathsf{MT.Commit}(h|_L)$. $\mathcal{P}$ then sends root_h to $\mathcal{V}$.
  15. Let $p(Y) = c_1 \cdot (\alpha \cdot \widetilde{Q}_{r_x}(Y) + S_Q(Y)) - (\alpha \cdot \widetilde{c} + s_1) \cdot Y^{m-1} - c_1 \cdot \mathbb{Z}_H(Y) \cdot h(Y)$. $\mathcal{P}$ and $\mathcal{V}$ then invoke a low degree test (LDT): $\langle \mathcal{P}_{\mathsf{LDT}}((\widetilde{Q}_{r_x}, S_Q, h), p), \mathcal{V}_{\mathsf{LDT}}((2m + \kappa, 2m + \kappa, m + \kappa), m - 1)\rangle(L)$. If the LDT fails, $\mathcal{V}$ aborts and outputs $b = 0$. Otherwise, at the end of the LDT, $\mathcal{V}$ needs oracle access to κ points of $\widetilde{Q}_{r_x}(\cdot)$, $S_Q(\cdot)$ $h(\cdot)$ and $p(\cdot)$ at query indices $\mathcal{Q} \subseteq L$.
  16. For each index $i \in \mathcal{Q}$, let a_i be the corresponding point in L.
      (a) $\mathcal{P}$ opens
      $$(\widetilde{Z}(a_i), \pi_i^{\widetilde{Z}}) \leftarrow \mathsf{MT.Open}(i, \widetilde{Z}|_L), \quad (R_A(a_i), \pi_i^{R_A}) \leftarrow \mathsf{MT.Open}(i, R_A|_L)$$
      $$(R_B(a_i), \pi_i^{R_B}) \leftarrow \mathsf{MT.Open}(i, R_B|_L), \quad (R_C(a_i), \pi_i^{R_C}) \leftarrow \mathsf{MT.Open}(i, R_C|_L)$$
      $$(S_Q(a_i), \pi_i^{S_Q}) \leftarrow \mathsf{MT.Open}(i, S_Q|_L), \quad (h(a_i), \pi_i^{h}) \leftarrow \mathsf{MT.Open}(i, h|_L),$$
      and sends them to $\mathcal{V}$.
      (b) $\mathcal{V}$ executes
      $$\mathsf{MT.Verify}(\mathsf{root}_{\widetilde{Z}}, i, \widetilde{Z}(a_i), \pi_i^{\widetilde{Z}}),\ \mathsf{MT.Verify}(\mathsf{root}_{R_A}, i, R_A(a_i), \pi_i^{R_A}),\ \mathsf{MT.Verify}(\mathsf{root}_{R_B}, i, R_B(a_i), \pi_i^{R_B}),$$
      $$\mathsf{MT.Verify}(\mathsf{root}_{R_C}, i, R_C(a_i), \pi_i^{R_C}),\ \mathsf{MT.Verify}(\mathsf{root}_{S_Q}, i, S_Q(a_i), \pi_i^{S_Q}),\ \mathsf{MT.Verify}(\mathsf{root}_h, i, h(a_i), \pi_i^{h}),$$
      for point $a_i \in L$ opened by $\mathcal{P}$. If any verification fails, $\mathcal{V}$ aborts and outputs $b = 0$.
      (c) $\mathcal{P}$ and $\mathcal{V}$ then invoke the GKR protocol: $\langle \mathcal{P}_{\mathsf{GKR}}, \mathcal{V}_{\mathsf{GKR}}\rangle(\mathfrak{C}, (r_x, a_i))$, where the circuit $\mathfrak{C}$ computes the evaluation and outputs $\left(\sum_{i \in [n]} \mathsf{val}_M(i)/((r_x - \mathsf{row}_M(i))(a_i - \mathsf{col}_M(i)))\right)_{M \in \{A,B,C\}}$ (denoted by $(\Gamma_{i,A}, \Gamma_{i,B}, \Gamma_{i,C})$, see Figure 1). If the check in GKR fails, $\mathcal{V}$ aborts and outputs $b = 0$.
      (d) $\mathcal{V}$ computes $\widetilde{Q}_{r_x}(a_i) = 1/c_1^2 \cdot \gamma \cdot \mathbb{Z}_H(a_i) \cdot \left(r_A \cdot \Gamma_{i,A} + r_B \cdot \Gamma_{i,B} + r_C \cdot \Gamma_{i,C}\right) \cdot \widetilde{Z}(a_i) + \gamma \cdot \left(r_A \cdot R_A(a_i) + r_B \cdot R_B(a_i) + r_C \cdot R_C(a_i)\right)$, together with $S_Q(a_i)$, $h(a_i)$, $\mathcal{V}$ checks the constraint $p(a_i) = N(a_i, \alpha \cdot \widetilde{Q}_{r_x}(a_i) + S_Q(a_i), h(a_i))$ at index $i \in \mathcal{Q}$. If the check fails, $\mathcal{V}$ aborts and outputs $b = 0$.
  17. $\mathcal{V}$ outputs $b = 1$.

output sizes $O(1)$. Other than the cost of invoking PC, the prover time is $O(n \log n)$, and both the total communication and the verification time are $O(\log^2 n)$. We list the costs of our construction instantiated with different polynomial commitment schemes in Table 3. In all other four candidate constructions for polynomial commitment schemes except Hyrax-PC [53] and IPP-PC [25], the communication costs between the prover and verifier and the verification times are polylogarithmic. Furthermore, the constructions instantiated with FRI

[8] and Virgo-VPD [56] are plausibly post-quantum resistant because the underlying polynomial commitment schemes rely only on symmetric cryptography, whereas others are not due to their reliance on groups of either unknown order or a pairing. Note that all these candidate constructions do not require a trusted setup.

Knowledge soundness. Our zero-knowledge protocol is also a proof of knowledge in the random oracle model. Informally speaking, given the root and sufficiently many authentication paths, there exists a PPT extractor that reconstructs the leaves with high probability. Additionally, in our protocol the leaves are RS encoding of the witness vector, which can be efficiently decoded by the extractor. The proof is similar to Virgo [56] and that in [17, 51], and we omit the details here.

Finally, the interactive argument constructed in prior subsections is public coin, so it can be made non-interactive in the random oracle model using the Fiat-Shamir transform [31].

5 Evaluation

In this section, we present some experimental results of our new zero-knowledge argument system. The system is implemented in C++ based on the open-source implementation of Aurora [16]. We use the open-source implementations of the univariate sumcheck protocol and (heuristic) FRI in `libiop` [2]. To implement the GKR protocol for the query computations, we modify the implementation in `Virgo` [6]. The compiler to generate the statements of zero-knowledge proofs as R1CS instances is taken from `libsnark` [3]. For efficient computation of the linear coefficient of the affine subspace polynomial $\mathbb{Z}_H$, we consider our implementation over the binary field $\mathbb{F}_{2^{256}}$ at the standard security level of 128 bits. All the experiments were run on a machine with an Intel Core i7-9800X processor and 128GB of RAM.

Virgo [56] is specialized to layered circuits over $\mathbb{F}_{p^2}$ where $p = 2^{61} - 1$, so it is not sufficient to provide 128 bits security. The implementation of Virgo does not support natively a larger Mersenne prime $2^{127} - 1$. Modifying and switching to such a field incurs a moderate slow down, which is at least 17× slower than Spartan [48], so we do not include it as a baseline here.

For Ligero, Aurora and Fractal, we use their open-source implementations from `libiop` with the same binary field $\mathbb{F}_{2^{256}}$. The implementations of Aurora and Fractal support two sets of parameters: proven and non-proven (also known as heuristic). We use the default one, i.e., heuristic, which relies on non-standard conjectures related to RS codes. Note that very recent work makes progress toward proving some of these heuristic [10].

For Marlin, we use its open-source implementation from [4], which uses `BLS12-381` for efficient curve arithmetic. Actually, it is a little unfair to compare Polaris with Marlin, as they are under different trust models. In particular, Marlin achieves a constant-size argument (880 bytes over `BLS12-381` reported in [28]) by relying on a trusted setup to generate a linear-size SRS. In contrast, our construction achieves polylogarithmic argument size but without any trusted setup.

For $\text{Spartan}_{\text{DL}}$, we use its open-source implementation (SpartanSNARK) from [5], which uses `curve25519-dalek` [1] for efficient curve arithmetic. Note that one of the variants in Spartan's family is the $\text{Spartan}_{\text{RO}}$, which has a better asymptotics in verifier's costs than $\text{Spartan}_{\text{DL}}$. However, the open-source implementation does not provide such an alternative. In addition, for $\text{Spartan}_{\text{RO}}$, the verifier will incur some hidden computation costs in order to realize efficiently computation commitments for query computation, and it adds additional memory cost on storing metadata at the verifier side. For example, the verifier reported in $\text{Spartan}_{\text{DL}}$ [48] requires about 15.1s to preprocess an R1CS instance at 2^{20} constraints. In contrast, $\text{Polaris}_{\text{RO}}$ does not require any such preprocessing. Although we cannot give a concrete advantage in performance than $\text{Spartan}_{\text{RO}}$, we expect the verifier in our scheme is faster.

Prover running time. Table 4 shows the prover's running times under Polaris and its baselines. Polaris is a little faster than Fractal and a little slower than Aurora. As shown in the table, Polaris is also a little faster than Marlin, despite sharing the same asymptotics. Recall that Polaris employs an $O(\log N)$-depth, $O(N)$-size circuit (see Figure 1) for the oracle access computations, which yields a much faster prover (linear in the GKR protocol) than Marlin's two additional univariate sumcheck protocols ($O(N \log N)$). Spartan beats all other systems since it offers a linear-time (time-optimal) prover among all. Ligero is also competitive to other systems because its proof system is actually an IPCP and has a constant round complexity.

Argument size. Table 5 shows argument sizes of Polaris and other four schemes. Polaris offers a smaller argument size than Fractal, despite sharing the same asymptotics. Note that it seems that Spartan offers much shorter arguments, which is competitive with Polaris concretely, although the latter has a better asymptotics. One reason is that for relatively small number of constraints, $\sqrt{N}$ is smaller than $\log^2 N$. Some additional experiments show that Polaris (237KBs) will overtake

Table 3. Costs of our construction instantiated with different polynomial commitment schemes

	PC scheme in Polaris	prover time	communication	verifier time	assumption
$\text{Polaris}_{\text{CL}}$	DARK [23]	$O(n \log n)$	$O(\log^2 n)$	$O(\log^2 n)$	r-strong RSA/adaptive root
$\text{Polaris}_{\text{RO}}$	FRI [8], Virgo-VPD [56]	$O(n \log n)$	$O(\log^2 n)$	$O(\log^2 n)$	CRHF
$\text{Polaris}_{\text{DL}}$	Hyrax-PC [53]	$O(n \log n)$	$O(\sqrt{n})$	$O(\sqrt{n})$	DLOG
$\text{Polaris}_{\text{DH}}$	Dory-PC [42]	$O(n \log n)$	$O(\log^2 n)$	$O(\log^2 n)$	SXDH

Table 4. Prover's performance (in seconds) for various R1CS instance sizes.

	Ligero	Aurora	Fractal	Marlin	Spartan	$\text{Polaris}_{\text{RO}}$
2^{10}	0.10	0.24	0.30	0.71	0.04	0.27
2^{11}	0.16	0.45	0.59	1.22	0.08	0.50
2^{12}	0.29	0.89	1.19	2.18	0.13	0.99
2^{13}	0.58	1.84	2.43	3.92	0.22	2.15
2^{14}	1.11	3.89	5.00	7.26	0.53	4.64
2^{15}	1.38	7.95	10.22	13.35	0.85	9.71
2^{16}	2.65	16.68	21.15	25.38	1.75	20.02
2^{17}	5.45	34.34	43.81	47.86	3.22	40.33
2^{18}	11.12	70.54	89.76	92.38	6.01	81.10
2^{19}	22.84	145.53	185.80	181.15	12.02	163.32
2^{20}	45.00	304.77	378.73	365.11	21.74	329.71

Spartan (356KBs) when the number of constraints is 2^{23} and beyond.

Table 5. Argument sizes (in KBs) for various R1CS instance sizes.

	Ligero	Aurora	Fractal	Spartan	$\text{Polaris}_{\text{RO}}$
2^{10}	554	74	130	32	97
2^{11}	629	84	143	37	108
2^{12}	1066	94	155	42	119
2^{13}	1178	103	162	48	127
2^{14}	2075	115	175	54	137
2^{15}	3195	126	193	63	151
2^{16}	5907	135	202	72	160
2^{17}	5671	146	218	85	172
2^{18}	10562	160	233	98	181
2^{19}	10703	171	239	120	190
2^{20}	20025	186	255	142	204

Verifier running time. Table 6 shows the verifier's running times under Polaris and its baselines. As we can see, Polaris offers a verifier that is faster than Ligero, Aurara and Spartan, both concretely and asymptotically. Moreover, when verifying a SHA-256 preimage (less than 23k AND gates [26]), our verification time (10.4ms) is about 1.7× faster than Spartan (18.2ms), 25× faster than Aurora (262ms), 78× faster than Ligero (814.3ms). Marlin offers the fastest verifier as the number of constraints increases since it incurs $O(\log N)$ verification costs. This should come as no surprise: the main feature of the trusted-setup approach is that proofs can be very short (a few hundred bytes) and very cheap to verify (a few milliseconds), such as [37]. Our scheme offers a tradeoff compared to Marlin: a slightly faster prover, at the cost of slightly larger verification time. With a smaller size circuit, we expect Polaris can offer a more efficient prover and verifier than Marlin.

Table 6. Verifier's performance (in milliseconds) for various R1CS instance sizes.

	Ligero	Aurora	Fractal	Marlin	Spartan	$\text{Polaris}_{\text{RO}}$
2^{10}	43.6	29.7	7.8	10.7	9.7	8.1
2^{11}	80.4	22.3	8.3	10.8	10.2	8.5
2^{12}	149.1	37.7	8.4	10.6	11.1	8.8
2^{13}	289.8	70.3	9.0	10.8	13.4	9.5
2^{14}	560.2	136.4	9.3	10.5	17.3	9.7
2^{15}	814.3	262.0	10.0	10.8	18.2	10.4
2^{16}	1570.9	511.4	10.1	10.5	29.4	10.9
2^{17}	3076.5	1025.1	10.5	9.7	35.9	11.5
2^{18}	6144.8	2050.3	10.9	10.0	47.8	12.0
2^{19}	12829	4097.5	11.3	8.3	54.7	12.7
2^{20}	25609	8288.4	11.6	7.6	83.1	13.3

6 Concluding Remarks

Polaris, a new zkSNARK scheme has improved verifier's performance compared with Ligero/Ligero++, Aurora for R1CS circuits, without a trusted setup, and with plausible post-quantum security, since the underlying cryptographic schemes involved are only symmetric cryptography, i.e., collision-resistant hash functions for Merkle tree commitments.

For blockchain privacy, a zkSNARK scheme deployed in the real-world systems is the QAP/QSP-based zkSNARK [32] in 2013. An earlier implemented version of the zkSNARK in Zcash [13] used the method from [18]. However, a vulnerability has been found [47] in 2019. Currently Zcash advised not to use that implementation, and it is currently updated to the version [39] using the method in [37]. Contrastively, the approaches listed in Table 1 (i.e., Ligero/Ligero++, Aurora, Fractal, Virgo, Spartan and our Polaris) do not need any trusted setup and perform heavy pairing cryptographic operations, and possess plausible post-quantum security, which can eliminate vulnerabilities in implementations of those heavy pairing operations as well as memory attacks on single point failure for accessing CRSs.

7 Acknowledgment

The work is supported by NSERC Strategic Project Grant.

References

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

[2] libiop: a C++ library for IOP-based zkSNARKs. https://github.com/scipr-lab/libiop.

[3] libsnark: a C++ library for zkSNARK proofs. https://github.com/scipr-lab/libsnark.

[4] Marlin. https://github.com/arkworks-rs/marlin.

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

[6] Virgo ZK reference implementation. https://github.com/sunblaze-ucb/virgo.

[7] Scott Ames, Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. Ligero: Lightweight sublinear arguments without a trusted setup. *CCS* 2017: 2087–2104.

[8] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Fast reed-solomon interactive oracle proofs of proximity. *ICALP* 2018: 14:1–14:17.

[9] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Scalable, transparent, and post-quantum secure computational integrity. *IACR Cryptology ePrint Archive*, 2018: 46.

[10] Eli Ben-Sasson, Dan Carmon, Yuval Ishai, Swastik Kopparty, and Shubhangi Saraf. Proximity gaps for Reed-Solomon codes. *FOCS* 2020: 900–909.

[11] Eli Ben-Sasson, Alessandro Chiesa, Michael A. Forbes, Ariel Gabizon, Michael Riabzev, and Nicholas Spooner. Zero knowledge protocols from succinct constraint detection. *TCC* 2017: 172–206.

[12] Eli Ben-Sasson, Alessandro Chiesa, Ariel Gabizon, and Madars Virza. Quasi-linear size zero knowledge from linear-algebraic PCPs. *TCC* 2016-A: 33–64.

[13] Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from bitcoin. *IEEE Symposium on Security and Privacy* 2014: 459–474.

[14] Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, and Eran Tromer. Fast reductions from RAMs to delegatable succinct constraint satisfaction problems. *ITCS* 2013: 401–414.

[15] Eli Ben-Sasson, Alessandro Chiesa, Lior Goldberg, Tom Gur, Michael Riabzev, and Nicholas Spooner. Linear-Size Constant-Query IOPs for Delegating Computation. *TCC* (2) 2019: 494–521.

[16] Eli Ben-Sasson, Alessandro Chiesa, Michael Riabzev, Nicholas Spooner, Madars Virza, and Nicholas P. Ward. Aurora: Transparent succinct arguments for R1CS. *EUROCRYPT* (1) 2019: 103–128.

[17] Eli Ben-Sasson, Alessandro Chiesa, and Nicholas Spooner. Interactive oracle proofs. *TCC* (2) 2016: 31–60.

[18] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. Succinct non-interactive zero knowledge for a von neumann architecture. *USENIX Security Symposium* 2014: 781–796.

[19] Eli Ben-Sasson, Lior Goldberg, Swastik Kopparty, and Shubhangi Saraf. DEEP-FRI: Sampling outside the box improves soundness. *ITCS* 2020: 5:1–5:32.

[20] Rishabh Bhadauria, Zhiyong Fang, Carmit Hazay, Muthuramakrishnan Venkitasubramaniam, Tiancheng Xie, and Yupeng Zhang. Ligero++: A new optimized sublinear IOP. *CCS* 2020: 2025–2038.

[21] Jonathan Bootle, Andrea Cerulli, Pyrros Chaidos, Jens Groth, and Christophe Petit. Efficient zero-knowledge arguments for arithmetic circuits in the discrete log setting. *EUROCRYPT* (2) 2016: 327–357.

[22] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Gregory Maxwell. Bulletproofs: Short proofs for confidential transactions and more. *IEEE Symposium on Security and Privacy* 2018: 315–334.

[23] Benedikt Bünz, Ben Fisch, and Alan Szepieniec. Transparent snarks from DARK compilers. *EUROCRYPT* (1) 2020: 677–706.

[24] Nigel P. Byott and Robin J. Chapman. Power sums over finite subspaces of a field. *Finite Fields and Their Applications*, 5(3): 254–265, 1999.

[25] Benedikt Bünz, Mary Maller, Pratyush Mishra, and Noah Vesely. Proofs for inner pairing products and applications. *IACR Cryptology ePrint Archive*, 2019: 1177.

[26] Matteo Campanelli, Rosario Gennaro, Steven Goldfeder, and Luca Nizzardo. Zero-knowledge contingent payments revisited: Attacks and payments for services. *CCS* 2017: 229–243.

[27] Alessandro Chiesa, Michael A. Forbes, and Nicholas Spooner. A zero knowledge sumcheck and its applications. *IACR Cryptology ePrint Archive*, 2017: 305.

[28] Alessandro Chiesa, Yuncong Hu, Mary Maller, Pratyush Mishra, Noah Vesely, and Nicholas P. Ward. Marlin: Preprocessing zkSNARKs with Universal and Updatable SRS. *EUROCRYPT* (1) 2020: 738–768.

[29] Alessandro Chiesa, Dev Ojha, and Nicholas Spooner. Fractal: Post-quantum and transparent recursive proofs from holography. *EUROCRYPT* (1) 2020: 769–793.

[30] Graham Cormode, Michael Mitzenmacher, and Justin Thaler. Practical verified computation with streaming interactive proofs. *ITCS* 2012: 90–112.

[31] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. *CRYPTO* 1986: 186–194.

[32] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct nizks without PCPs. *EUROCRYPT* 2013: 626–645.

[33] Craig Gentry and Daniel Wichs. Separating succinct non-interactive arguments from all falsifiable assumptions. *STOC* 2011: 99–108.

[34] Oded Goldreich. *The Foundations of Cryptography - Volume 1: Basic Techniques*. Cambridge University Press, 2001.

[35] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. Delegating computation: Interactive proofs for muggles. *Journal of the ACM*, 62(4): 27:1–27:64, 2015.

[36] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on Computing*, 18(1): 186–208, 1989.

[37] Jens Groth. On the size of pairing-based non-interactive arguments. *EUROCRYPT* (2) 2016: 305–326.

[38] Jens Groth and Yuval Ishai. Sub-linear zero-knowledge argument for correctness of a shuffle. *EUROCRYPT* 2008: 379–396.

[39] Daira Hopwood, Sean Bowe, Taylor Hornby, and Nathan Wilcox. *Zcash Protocol Specification*, March 2020.

[40] Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. Constant-size commitments to polynomials and their applications. *ASIACRYPT* 2010: 177–194.

[41] Joe Kilian. A note on efficient zero-knowledge proofs and arguments (extended abstract). *STOC* 1992: 723–732.

[42] Jonathan Lee. Dory: Efficient, transparent arguments for generalised inner products and polynomial commitments. *IACR Cryptology ePrint Archive*, 2020: 1274.

[43] Rudolf Lidl and Harald Niederreiter. *Finite Fields*. Second Edition. Cambridge University Press, 1997.

[44] Carsten Lund, Lance Fortnow, Howard J. Karloff, and Noam Nisan. Algebraic methods for interactive proof systems. *FOCS* (1) 1990: 2–10.

[45] Ralph C. Merkle. A certified digital signature. *CRYPTO* 1989: 218–238.

[46] Silvio Micali. Computationally sound proofs. *SIAM Journal on Computing*, 30(4): 1253–1298, 2000.

[47] NIST. Common vulnerabilities and exposures. https://nvd.nist.gov/vuln/detail/cve-2019-7167, March 2019.

[48] Srinath Setty. Spartan: Efficient and general-purpose zkSNARKs without trusted setup. *CRYPTO* (3) 2020: 704–737.

[49] Adi Shamir. IP = PSPACE. *Journal of the ACM*, 39(4): 869–877, 1992.

[50] Justin Thaler. Time-optimal interactive proofs for circuit evaluation. *CRYPTO* (2) 2013: 71–89.

[51] Paul Valiant. Incrementally verifiable computation or proofs of knowledge imply time/space efficiency. *TCC* 2008: 1–18.

[52] Alexander Vlasov and Konstantin Panarin. Transparent polynomial commitment scheme with polylogarithmic communication complexity. *IACR Cryptology ePrint Archive*, 2019: 1020.

[53] Riad S. Wahby, Ye Ji, Andrew J. Blumberg, Abhi Shelat, Justin Thaler, Michael Walfish, and Thomas Wies. Full accounting for verifiable outsourcing. *CCS* 2017: 2071–2086.

[54] Riad S. Wahby, Ioanna Tzialla, Abhi Shelat, Justin Thaler, and Michael Walfish. Doubly-efficient zkSNARKs without trusted setup. *IEEE Symposium on Security and Privacy* 2018: 926–943.

[55] Tiancheng Xie, Jiaheng Zhang, Yupeng Zhang, Charalampos Papamanthou, and Dawn Song. Libra: Succinct zero-knowledge proofs with optimal prover computation. *CRYPTO* (3) 2019: 733–764.

[56] Jiaheng Zhang, Tiancheng Xie, Yupeng Zhang, and Dawn Song. Transparent polynomial delegation and its applications to zero knowledge proof. *IEEE Symposium on Security and Privacy* 2020: 859–876.

A GKR Protocol

We first review a seminal protocol that has various applications in the literature of interactive proof, called the sumcheck protocol [44, 49]. In a sumcheck protocol, a (not necessarily efficient) prover takes as input a τ-variate polynomial $f\colon \mathbb{F}^\tau \to \mathbb{F}$ of degree $\leq d$ in each variable (think of d as significantly smaller than $|\mathbb{F}|$). His goal is to convince a verifier that

$$\sum_{x_1,\ldots,x_\tau \in \{0,1\}} f(x_1,\ldots,x_\tau) = \beta,$$

for some constant $\beta \in \mathbb{F}$. The verifier only has oracle access to f, and is given the constant $\beta \in \mathbb{F}$. Directly computing the sum requires exponential time in τ, as there are 2^τ combinations. Lund et al. [44] proposed a sumcheck protocol that is efficient in both its running time and its number of oracle queries and allows a verifier $\mathcal{V}$ to delegate the computation to a computationally unbounded prover $\mathcal{P}$, who can convince $\mathcal{V}$ that β is the correct sum. In the protocol, $\mathcal{V}$ interacts with $\mathcal{P}$ over a sequence of τ rounds. At the end of this interaction, $\mathcal{V}$ needs an oracle access to the evaluation of f at a random point $\theta \in \mathbb{F}^\tau$ and outputs accept or reject.

Using the sumcheck protocol as a building block, Goldwasser et al. [35] proposed an efficient general-purpose interactive proof protocol for layered arithmetic circuits, allowing the verifier to validate the circuit evaluation in logarithmic time with a logarithmic-size proof. We refer it to the GKR protocol and present it in more details below.

Let $\mathfrak{C}$ be a layered arithmetic circuit of fan-in 2 over $\mathbb{F}$, meaning that the circuit can be decomposed into layers, and wires only connect gates in adjacent layers (if $\mathfrak{C}$ is not layered it can easily be transformed into a layered circuit $\mathfrak{C}$ with a small blowup in size by a factor of $O(D)$ where D is the depth of the circuit $\mathfrak{C}$). We now number the layers from 0 to D with layer D referring to the input layer, and layer 0 referring to the output layer, thus each gate in the i-th layer takes inputs from two gates in the $(i+1)$-th layer. In the first message, $\mathcal{P}$ tells $\mathcal{V}$ the (claimed) output(s) of the circuit. The protocol then works its way in iterations towards the input layer, with one iteration devoted to each layer. In the first round, $\mathcal{V}$ and $\mathcal{P}$ run the sumcheck protocol to reduce the claim about the output to a claim about the values in layer 1. In the i-th round, both parties reduce a claim about layer $i-1$ to a claim about layer i through the sumcheck protocol. Finally, the protocol terminates with a claim about the input layer D, which can be

checked directly by $\mathcal{V}$, or is given as an oracle access. If the check passes, $\mathcal{V}$ accepts the claimed output.

More concretely, let S_i denote the number of gates at layer i of the circuit $\mathfrak{C}$. Assume S_i is a power of 2 and let $S_i = 2^{s_i}$. Number the gates at layer i from 0 to $S_i - 1$, and let $V_i : \{0,1\}^{s_i} \to \mathbb{F}$ denote the function that takes as input a binary gate label, and outputs the corresponding gate's value at layer i. With this definition, V_0 corresponds to the output of the circuit, and V_D corresponds to the input layer. Unfortunately, $V_i(\cdot)$ is not a polynomial, so it cannot be directly used in the sumcheck protocol. We consider its multilinear extension, the unique polynomial $\tilde{V}_i : \mathbb{F}^{s_i} \to \mathbb{F}$ such that $\tilde{V}_i(x_1, \ldots, x_{s_i}) = V_i(x_1, \ldots, x_{s_i})$ for all $x_1, \ldots, x_{s_i} \in \{0,1\}$.

With these definitions and the approach in [27], we can express the evaluation of $\tilde{V}_i$ as a summation of the evaluations of $\tilde{V}_{i+1}$:

$$\alpha_i \tilde{V}_i(u^{(i)}) + \beta_i \tilde{V}_i(v^{(i)}) = \sum_{x,y \in \{0,1\}^{s_{i+1}}} f_i(\tilde{V}_{i+1}(x), \tilde{V}_{i+1}(y)), \tag{1}$$

where $u^{(i)}, v^{(i)} \in \mathbb{F}^{s_i}$ are random vectors and $\alpha_i, \beta_i \in \mathbb{F}$ are random values. Here f_i depends on α_i, β_i, $u^{(i)}$, and $v^{(i)}$.

With Equation 1, the GKR protocol proceeds as follows. The prover $\mathcal{P}$ first sends the claimed output of the circuit to $\mathcal{V}$. From the claimed output, $\mathcal{V}$ defines a polynomial $\tilde{V}_0$ and computes $\tilde{V}_0(u^{(0)})$ and $\tilde{V}_0(v^{(0)})$ for random $u^{(0)}, v^{(0)} \in \mathbb{F}^{s_0}$. $\mathcal{V}$ selects $\alpha_0, \beta_0 \in \mathbb{F}$ randomly and computes $\alpha_0 \tilde{V}_0(u^{(0)}) + \beta_0 \tilde{V}_0(v^{(0)})$. $\mathcal{V}$ and $\mathcal{P}$ then execute the sumcheck protocol on Equation 1 for $i = 0$. As described before, at the end of the sumcheck, $\mathcal{V}$ needs an oracle access to the evaluation of $f_0(u^{(1)}, v^{(1)})$, where $u^{(1)}$, $v^{(1)}$ are randomly selected in $\mathbb{F}^{s_1}$. To compute $f_0(u^{(1)}, v^{(1)})$, $\mathcal{V}$ asks $\mathcal{P}$ to send $\tilde{V}_1(u^{(1)})$ and $\tilde{V}_1(v^{(1)})$. As f_0 only depends on $\alpha_0, \beta_0, u^{(0)}, v^{(0)}$ and the gates and wiring in layer 0, which are all known to $\mathcal{V}$ and can be computed by $\mathcal{V}$ directly. In this way, $\mathcal{V}$ and $\mathcal{P}$ reduce a claim about the output to two claims about values in layer 1. $\mathcal{V}$ and $\mathcal{P}$ then repeat the protocol recursively layer by layer to reduce a claim on layer i to one claim on layer $i+1$. Eventually, $\mathcal{V}$ receives two claimed evaluations $\tilde{V}_D(u^{(D)})$ and $\tilde{V}_D(v^{(D)})$. $\mathcal{V}$ then checks the correctness of these two claims directly by evaluating $\tilde{V}_D$, which is defined by the input of the circuit $\mathfrak{C}$. Let $\mathcal{P}_{\mathsf{GKR}}$ and $\mathcal{V}_{\mathsf{GKR}}$ be the algorithms for the GKR prover and verifier, we state the properties of the GKR protocol in the following lemma.

Lemma 5 ([30, 35, 50, 55, 56])**.** *Let $\mathfrak{C} : \mathbb{F}^n \to \mathbb{F}$ be a depth-D layered arithmetic circuit. $\langle \mathcal{P}_{\mathsf{GKR}}, \mathcal{V}_{\mathsf{GKR}} \rangle(\mathfrak{C}, x)$ is an interactive proof for the function computed by $\mathfrak{C}$ with soundness $O(D \log |\mathfrak{C}|/|\mathbb{F}|)$. The total communication is $O(D \log |\mathfrak{C}|)$ and the running time of the prover $\mathcal{P}$ is $O(|\mathfrak{C}|)$. When $\mathfrak{C}$ has regular wiring pattern, the running time of the verifier $\mathcal{V}$ is $O(n + D \log |\mathfrak{C}|)$.*

B Polynomial Commitment Schemes in Table 2

In particular, Supersonic [23] makes use of groups of unknown order to construct Diophantine ARguments of Knowledge (DARK) proofs for polynomial evaluations over the field. Virgo-VPD refers to the verifiable polynomial delegation scheme in [56]. As a core component Virgo relies on an inner product argument which can also be used as a univariate polynomial commitment. Hyrax-PC refers to the polynomial commitment scheme used in [53] and is based on the work of Bootle et al. [21]. This approach is also followed in Bünz et al. [25] for univariate and bivariate polynomials and in Dory [42] as well. The FRI protocol [8] is an efficient interactive oracle proof (IOP) for which a committed oracle is close to a Reed-Solomon codeword, meaning that the prover commits to a large sequence of field elements and the verifier queries only a few specific elements rather than reading the entire sequence. In order to be used as a polynomial commitment scheme, the protocol requires to query the polynomial values outside of the evaluation set. DEEP-FRI [19] shows that this is possible and a recent note by Vlasov and Panarin [52] makes the connection explicitly by building a polynomial commitment scheme from FRI.

C Proof of Theorem 2

Proof. **Completeness.** If $c = \sum_{y \in H} Q_{r_x}(y)$, then by the definitions of G_2, h and Lemma 1, $c = \sum_{y \in H} G_2(y) = G_2^{(m-1)} \cdot c_1$, where $G_2^{(m-1)}$ is the coefficient of Y^{m-1} in $G_2(\cdot)$. Therefore, $p(Y) = c_1 \cdot Q_{r_x}(Y) - G_2^{(m-1)} \cdot c_1 \cdot Y^{m-1} - c_1 \cdot \mathbb{Z}_H(Y) \cdot h(Y) = c_1 \cdot \left(G_2(Y) - G_2^{(m-1)} \cdot Y^{m-1}\right)$, which is in $\mathrm{RS}[L, m-1]$. The rest follows from the completeness of the commitment schemes, the LDT and GKR protocols.

Soundness. Let ϵ_{PC}, ϵ_{MT}, ϵ_{LDT} and ϵ_{GKR} be the soundness errors of the polynomial commitment, Merkle tree, LDT and GKR protocols, respectively. We argue the soundness by combining the soundness errors in the following cases.

Case 1: $\nexists Z^* \in \mathrm{RS}[L, m]$ such that $\mathsf{com} = \mathsf{MT.Commit}(Z^*|L)$, i.e., com is not a valid commitment.

– If com is not a valid Merkle tree root, the verification passes with probability less than ϵ_{MT} in Step 13(b).
– If $\exists Z^{**} \notin \mathrm{RS}[L,m]$ such that $\mathsf{com} \leftarrow \mathsf{MT.Commit}(Z^{**}|L)$, and the points opened by $\mathcal{P}$ satisfy $\eta_i^* \neq Z^{**}(a_i)$ for some $i \in \mathcal{Q}$ in Step 13(a), the verification passes with probability less than ϵ_{MT}.
– If the value η^* returned by $\mathcal{P}$ in Step 4 satisfies $\eta^* \neq G_1^*(r_x)$, the verification passes with probability less than ϵ_{PC}.
– If the values v_A^*, v_B^* and v_C^* returned by $\mathcal{P}$ in Step 7 satisfy $v_A^* \cdot v_B^* - v_C \neq \eta^* \cdot \gamma$, or $v_A^* \neq \bar{A}^*(r_x)$, or $v_B^* \neq \bar{B}^*(r_x)$, or $v_C^* \neq \bar{C}^*(r_x)$, then by Lemmas 3 and 4, the verification passes with probability less than $(2m-2)/(|\mathbb{F}|-m)+1/|\mathbb{F}|$.
– If the output $(\Gamma_{i,A}^*, \Gamma_{i,B}^*, \Gamma_{i,C}^*)$ returned by $\mathcal{P}$ in Step 13(c) satisfies $\Gamma_{i,M}^* \neq \left(\sum_{i\in[n]} \mathsf{val}_M(i)/((r_x - \mathsf{row}_M(i))(a_i - \mathsf{col}_M(i)))\right)$ for some $M \in \{A,B,C\}$ and $i \in \mathcal{Q}$, the verification passes with probability less than ϵ_{GKR}.
– Otherwise, as $Q_{r_x} \notin \mathrm{RS}[L, 2m-1]$, by the LDT protocol, the verification in Step 12 passes with probability less than ϵ_{LDT}.

Case 2: $\exists Z^* \in \mathrm{RS}[L,m]$ such that $\mathsf{com} = \mathsf{MT.Commit}(Z^*|L)$. Let $Q_{r_x}^*(Y) = (r_A \cdot A(r_x,Y) + r_B \cdot B(r_x,Y) + r_C \cdot C(r_x,Y)) \cdot Z^*(Y)$. Suppose that $c^* \neq \sum_{y\in H} Q_{r_x}^*(y)$, then by Lemma 1, for all $h \in \mathrm{RS}[L,m-1]$, $p^* = c_1 \cdot Q_{r_x}^*(Y) - c^* \cdot Y^{m-1} - c_1 \cdot \mathbb{Z}_H(Y) \cdot h(Y) \notin \mathrm{RS}[L,m-1]$. Therefore,

– If the commitment in Step 11 is not a valid Merkle tree root, or the points opened by $\mathcal{P}$ in Step 13(a) are inconsistent with Z^* or h, the verification passes with probability less than ϵ_{MT}.
– If $Q_{r_x} \in \mathrm{RS}[L,2m-1]$, then either $h \notin \mathrm{RS}[L,m-1]$ or $p \notin \mathrm{RS}[L,m-1]$. Thus, by the LDT protocol, the verification passes with probability less than ϵ_{LDT}.
– Other cases are similar to Case 1, the verification passes with probability less than $\epsilon_{\mathsf{PC}} + \epsilon_{\mathsf{GKR}} + (2m-2)/(|\mathbb{F}|-m) + 1/|\mathbb{F}|$.

By the union bound, the soundness is no more than $O(\epsilon_{\mathsf{MT}} + \epsilon_{\mathsf{PC}} + \epsilon_{\mathsf{LDT}} + \epsilon_{\mathsf{GKR}} + (2m-2)/(|\mathbb{F}|-m) + 1/|\mathbb{F}|) \leq \mathsf{negl}(\lambda)$. □

D Proof of Theorem 3

Proof. **Completeness.** It follows from the completeness of Protocol 2 and the modifications to achieve zero-knowledge.

Soundness. It follows from the soundness of Protocol 2 and the zero-knowledge modifications. In particular, if $\exists \widetilde{Z}^{*'} \in \mathrm{RS}[L, m+\kappa+1]$, it can always be uniquely decomposed as $\widetilde{Z}^*(Y) = \widetilde{Z}^{*'}(Y) - \mathbb{Z}_H(Y) \cdot R_Z^*(Y)$ such that $\widetilde{Z}^*(y) = \widetilde{Z}^{*'}(y)$ for all $y \in H$. If $\widetilde{F}_w^*(X) \neq \widetilde{F}_w(X)$, using the fact that $\widetilde{F}_w^*$ and $\widetilde{F}_w$ are of degree at most $2m$, we have that the number of $r_x \in \mathbb{F} \setminus H$ for which $\widetilde{F}_w^*(r_x) = \widetilde{F}_w(r_x)$ is at most $2m$. This happens with probability at most $2m/(|\mathbb{F}|-m)$. In addition, suppose that $\widetilde{c} \neq \widetilde{c}^* = \sum_{y\in H} \widetilde{Q}_{r_x}^*(y)$ and let $s_1^* = \sum_{y\in H} S_Q^*(y)$, where $S_Q^*(y)$ is committed by $\mathcal{P}$ in Step 12. Then $\sum_{y\in H}\left(\alpha \cdot \widetilde{Q}_{r_x}^*(y) + S_Q^*(y)\right) = \alpha \cdot \widetilde{c}^* + s_1^* = \alpha \cdot \widetilde{c} + s_1$ if and only if $\alpha = \frac{s_1 - s_1^*}{\widetilde{c}^* - \widetilde{c}}$, which happens with probability $1/|\mathbb{F}|$. The probabilities of other cases are the same as the proof of Theorem 2, and we omit the details here.

Zero-knowledge. We describe a simulator $\mathcal{S}$ in Figure 2 that simulates $\mathcal{V}^*$'s view in the protocol, where $\mathcal{S}_{\mathsf{PC.Eval}}$ and $\mathcal{S}_{\mathsf{LDT}}$ are the simulators of the PC.Eval and LDT protocols, respectively. In particular, the simulator $\mathcal{S}_{\mathsf{LDT}}$ generates $p_{\mathrm{sim}} \in \mathrm{RS}[L, 1+\deg p]$ and can simulate the view of any sequence of random polynomials that are subject to the constraint and their evaluations at points indexed by $\mathcal{Q}$ are consistent with the oracle access of p_{sim}.

In the simulator $\mathcal{S}$, Steps 11 and 16(c) are the same as the real protocol, and no message is sent in Steps 4, 7, 9–11, 13, 16(b), 16(d) and 17. In Steps 1–3, 5, 12 and 16(a), both polynomials in each pair of $(\widetilde{Z}_{\mathrm{sim}}, \widetilde{Z})$, $(R_A^{\mathrm{sim}}, R_A)$, $(R_B^{\mathrm{sim}}, R_B)$, $(R_C^{\mathrm{sim}}, R_C)$, $(G_1^{\mathrm{sim}}, G_1)$ and $(S_Q^{\mathrm{sim}}, S_Q)$ are sampled randomly, thus their commitments and evaluations are indistinguishable. In Step 8, as η_{sim} is independent with $v_A^{\mathrm{sim}}, v_B^{\mathrm{sim}}$ and randomly distributed, so is v_C^{sim}. Thus, they are indistinguishable from (v_A, v_B, v_C) in the real protocol. The view of Steps 14 and 15 simulated by $\mathcal{S}_{\mathsf{LDT}}$ and the view of Step 6 simulated by $\mathcal{S}_{\mathsf{PC.Eval}}$ are indistinguishable from $(\widetilde{Q}_{r_x}, S_Q, h)$ and G_1 in the real protocol, respectively. □

$\mathcal{S}(\mathbb{x}, r)$:

1. Sample $\widetilde{Z}_{\text{sim}} \in \text{RS}[L, m+\kappa+1]$ uniformly at random and send $\mathsf{root}_{\widetilde{Z}_{\text{sim}}} \leftarrow \mathsf{MT.Commit}(\widetilde{Z}_{\text{sim}}|_L)$ to $\mathcal{V}^*$.
2. Sample R_A^{sim}, R_B^{sim}, $R_C^{\text{sim}} \in \text{RS}[L, m]$ uniformly at random and send $\mathsf{root}_{R_A^{\text{sim}}} \leftarrow \mathsf{MT.Commit}(R_A^{\text{sim}}|_L)$, $\mathsf{root}_{R_B^{\text{sim}}} \leftarrow \mathsf{MT.Commit}(R_B^{\text{sim}}|_L)$ and $\mathsf{root}_{R_C^{\text{sim}}} \leftarrow \mathsf{MT.Commit}(R_C^{\text{sim}}|_L)$ to $\mathcal{V}^*$.
3. Sample a degree m polynomial G_1^{sim} uniformly at random, compute $(\mathcal{C}_{G_1^{\text{sim}}}, \mathcal{S}_{G_1^{\text{sim}}}) \leftarrow \mathsf{PC.Commit}(\mathsf{pp}, G_1^{\text{sim}})$, and sends $\mathcal{C}_{G_1^{\text{sim}}}$ to $\mathcal{V}^*$.
4. Receive $r_x^{\text{sim}} \in \mathbb{F} \setminus H$ from $\mathcal{V}^*$.
5. Evaluate $\eta_{\text{sim}} = G_1^{\text{sim}}(r_x^{\text{sim}})$ and send η_{sim} to $\mathcal{V}^*$.
6. Call $\mathcal{S}_{\mathsf{PC.Eval}}$ to simulate the view of the evaluation protocol $\mathcal{S}_{\mathsf{PC.Eval}}(\mathsf{pp}, \mathcal{C}_{G_1^{\text{sim}}}, r_x^{\text{sim}}, \eta_{\text{sim}}, m, r)$.
7. Wait $\mathcal{V}^*$ for validation.
8. Sample two random elements $v_A^{\text{sim}}, v_B^{\text{sim}} \in \mathbb{F}$, compute $v_C^{\text{sim}} = v_A^{\text{sim}} \cdot v_B^{\text{sim}} - \eta_{\text{sim}} \cdot \mathbb{Z}_H(r_x^{\text{sim}})$, and send $(v_A^{\text{sim}}, v_B^{\text{sim}}, v_C^{\text{sim}})$ to $\mathcal{V}^*$.
9. Wait $\mathcal{V}^*$ for validation.
10. Receive $r_A^{\text{sim}}, r_B^{\text{sim}}, r_C^{\text{sim}} \in \mathbb{F}$ from $\mathcal{V}^*$.
11. Let $\widetilde{c}_{\text{sim}} = r_A^{\text{sim}} \cdot v_A^{\text{sim}} + r_B^{\text{sim}} \cdot v_B^{\text{sim}} + r_C^{\text{sim}} \cdot v_C^{\text{sim}}$.
12. Sample $S_Q^{\text{sim}} \in \text{RS}[L, 2m+\kappa]$ uniformly at random and send $s_1^{\text{sim}} = \sum_{y \in H} S_Q^{\text{sim}}(y)$ and $\mathsf{root}_{S_Q^{\text{sim}}} \leftarrow \mathsf{MT.Commit}(S_Q^{\text{sim}}|_L)$ to $\mathcal{V}^*$.
13. Receive $\alpha_{\text{sim}} \in \mathbb{F}$ from $\mathcal{V}^*$.
14. Given the random challenges $\mathcal{Q}_{\text{sim}}$ of $\mathcal{V}^*$, call $\mathcal{S}_{\mathsf{LDT}}$ to generate $p_{\text{sim}} \in \text{RS}[L, m-1]$. For each query point $a_i \in \mathcal{Q}_{\text{sim}}$, compute h_i^{sim} such that $p_{\text{sim}}(a_i) = c_1 \cdot (\alpha_{\text{sim}} \cdot \widetilde{Q}_{r_x}^{\text{sim}}(a_i) + S_Q^{\text{sim}}(a_i)) - (\alpha_{\text{sim}} \cdot c_{\text{sim}} + s_1^{\text{sim}}) \cdot a_i^{n-1} - c_1 \cdot \mathbb{Z}_H(a_i) \cdot h_i^{\text{sim}}$. Sample $h_{\text{sim}} \in \text{RS}[L, m+\kappa]$ uniformly at random such that $h_{\text{sim}}(a_i) = h_i^{\text{sim}}$, and send $\mathsf{root}_{h_{\text{sim}}} \leftarrow \mathsf{MT.Commit}(h_{\text{sim}}|_L)$ to $\mathcal{V}^*$.
15. Call $\mathcal{S}_{\mathsf{LDT}}$ to simulate the view of the LDT protocol $\mathcal{S}_{\mathsf{LDT}}(L, ((2m+\kappa, 2m+\kappa, m+\kappa), m-1))$.
16. For each index $i \in \mathcal{Q}_{\text{sim}}$, let a_i be the corresponding point in L.
    (a) Open
    $$(\widetilde{Z}_{\text{sim}}(a_i), \pi_i^{\widetilde{Z}_{\text{sim}}}) \leftarrow \mathsf{MT.Open}(i, \widetilde{Z}_{\text{sim}}|_L), \quad (R_A^{\text{sim}}(a_i), \pi_i^{R_A^{\text{sim}}}) \leftarrow \mathsf{MT.Open}(i, R_A^{\text{sim}}|_L)$$
    $$(R_B^{\text{sim}}(a_i), \pi_i^{R_B^{\text{sim}}}) \leftarrow \mathsf{MT.Open}(i, R_B^{\text{sim}}|_L), \quad (R_C^{\text{sim}}(a_i), \pi_i^{R_C^{\text{sim}}}) \leftarrow \mathsf{MT.Open}(i, R_C^{\text{sim}}|_L)$$
    $$(S_Q^{\text{sim}}(a_i), \pi_i^{S_Q^{\text{sim}}}) \leftarrow \mathsf{MT.Open}(i, S_Q^{\text{sim}}|_L), \quad (h_{\text{sim}}(a_i), \pi_i^{h_{\text{sim}}}) \leftarrow \mathsf{MT.Open}(i, h_{\text{sim}}|_L),$$
    and sends them to $\mathcal{V}^*$.
    (b) Wait $\mathcal{V}^*$ to validate the points.
    (c) Run the GKR protocol $\langle \mathcal{P}_{\mathsf{GKR}}, \mathcal{V}_{\mathsf{GKR}} \rangle(\mathfrak{C}, (r_x^{\text{sim}}, a_i))$ with $\mathcal{V}^*$, where the circuit $\mathfrak{C}$ computes the evaluation and outputs $\left(\sum_{i \in [n]} \mathsf{val}_M(i) / ((r_x^{\text{sim}} - \mathsf{row}_M(i))(a_i - \mathsf{col}_M(i)))\right)_{M \in \{A,B,C\}}$ (see Figure 1).
    (d) Wait $\mathcal{V}^*$ for validation.
17. Wait $\mathcal{V}^*$ for validation.

Fig. 2. The simulator $\mathcal{S}$ of the final zero-knowledge protocol