Ursa Minor: The Implementation Framework for Polaris

Mohammadtaghi Badakhshan(✉), Guiwen Luo, Tanmayi Jandhyala, and Guang Gong

University of Waterloo, Waterloo, Canada
{mbadakhshan,guiwen.luo,tjandhya,ggong}@uwaterloo.ca

Abstract. This paper conducts an analysis of algorithms within Polaris, a plausibly post-quantum zero-knowledge succinct non-interactive argument of knowledge (zkSNARK) protocol, by decomposing it into its construction components for detailed investigation. Recognizing the need for fast implementation in real-world applications, we introduce the Ursa Minor, an implementation framework tailored to evaluate Polaris's efficiency. Our contribution in this framework are twofold: Firstly, we proposed a concrete GKR arithmetic circuit to be integrated in Polaris. Secondly, we optimized the efficiency of FRI protocol employed in Polaris, by eliminating the field inversion operations.

Keywords: Zero-Knowledge Proof · Post-Quantum · Privacy-Preserving Application

1 Introduction

Zero-Knowledge Succinct Non-Interactive Arguments of Knowledge (zkSNARKs) have gained significant interest in academia for their ability to enable efficient and privacy-preserving verifiable computations. With the potential threat of quantum computers posing to the security of classical zkSNARKs, researchers have been exploring post-quantum secure zkSNARKs to ensure their resilience against quantum adversaries.

Polaris [9] is such a novel zkSNARK protocol that achieves plausibly post-quantum security, leveraging several construction components that are widely used in academia. Polaris is built upon an Rank-1 Constraint System (R1CS) like arithmetician over binary field, and integrates with sumcheck, GKR and FRI procotols [3,11,16]. The sumcheck protocol enables efficient zero-knowledge proofs for polynomial evaluations, and the GKR protocol extends this capability to handle general arithmetic circuits. Additionally, the FRI protocol is used to do succinct proximity proof to low-degree polynomials. Because the security of those protocols does not rely on the hardness assumption of classical math

Part of the research of Mohammadtaghi Badakhshan on this project is supported by the Ripple Graduate Fellows award from May 1st, 2023 to April 30th, 2024.

S. Petkova-Nikova and D. Panario (Eds.): WAIFI 2024, LNCS 15176, pp. 258–273, 2025.
https://doi.org/10.1007/978-3-031-81824-0_16

problems, Polaris is plausibly quantum secure. The landscape of post-quantum secure zkSNARKs continues to evolve, with ongoing research efforts focused on enhancing transparency and efficiency. Several notable constructions include Ligero [2], STARK [4], Aurora [5], Fractal [7], Spartan-RO [15].

Classic zkSNARKs, such as Groth16 [12] and Halo2 [1] built upon elliptic curve pairings, would usually show the efficiency advantages against those plausibly post-quantum secure protocols built upon GKR and FRI. However, efficient implementation of these protocols is crucial for enabling practical deployment in real-world applications. To address concerns regarding efficiency, this paper proposes the fast implementation of Polaris by emphasizing on the optimization of GKR and FRI protocols.

Contributions. Our main contributions are summarized as follows:

- In Sect. 4, we present an instantiation of the FRI protocol. This instantiation eliminates the field inversion operations in both the Commit phase and Query phase, expecting to show better efficiency.
- In Sect. 5, we present an instantiation of the GKR circuit tailored for the Polaris implementation. By designing the circuit as a satisfiability circuit, we ensure the verifiable computation of values essential for the Polaris protocol while minimizing the number of gates. This would reduce the communication overhead, and the verifier and the prover complexities.

2 Preliminaries

2.1 Algebraic Foundations

Here we first introduce the algebraic foundations of the paper. They are summarized from the existing literature.

Finite Fields. In this paper, $\mathbb{F}$ is used to denote a general finite field and $\mathbb{F}_2$ denotes the binary field $\{0, 1\}$, while $\mathbb{F}_{2^{64}}$ and $\mathbb{F}_{2^{192}}$ specifically refer to the following 2 finite fields,

$$\mathbb{F}_{2^{64}} := \mathbb{F}_2[X]/(X^{64} + X^4 + X^3 + X + 1), \tag{1}$$

the field over which we do the arithmetization for circuits. And

$$\mathbb{F}_{2^{192}} := \mathbb{F}_{2^{64}}[Y]/(Y^3 + Y + 1), \tag{2}$$

from which the FRI protocol's random challenges are picked up. Those two fields are adopted in Preon [6], a SNARK-based post-quantum secure signature scheme.

Interpolations. Given n points $\{(x_i, y_i) \in \mathbb{F}^2 \mid 0 \leq i < n\}$, we define the unique polynomial

$$P(X) := \sum_{j=0}^{n-1} a_j X^j \tag{3}$$

of degree less than n, such that

$$P(x_i) = y_i,\ 0 \leq i < n.$$

Vanishing Polynomial. For a set $H \subset \mathbb{F}$, we define $Z_H(X)$ as the following polynomial

$$\mathbb{Z}_H(X) := \prod_{x \in H} (X - x) \tag{4}$$

that vanishes on H.

2.2 FRI

FRI protocol is a low-degree test for polynomials. It is used to determine whether a polynomial f is low-degree with respect to the size of the evaluation domain, without actually knowing f itself [3,13].

FRI protocol begins with a polynomial $f_0(X)$ and its evaluation domain L_0, which is an affine subspace in $\mathbb{F}$. The polynomial $f_0(X)$ and domain L_0 undergo a stepwise reduction process using a random folding procedure, resulting in a sequence of polynomials

$$f_0(X), f_1(X), \cdots, f_{\mathsf{r}}(X) \in \mathbb{F}[X], \tag{5}$$

and a sequence of domains

$$L_0 \supseteq L_1 \supseteq \cdots \supseteq L_{\mathsf{r}}. \tag{6}$$

Suppose d_k is the upper bound of polynomial's degree, i.e., $\deg f_k(X) < d_k$. We assume the degrees decrease with the same ratio as the domains, which means the quotients

$$\frac{d_k}{d_{k+1}} = \frac{|L_k|}{|L_{k+1}|} \tag{7}$$

are the same, called *reduction factors*, and in this papar we always assume the reduction factors to be 2.

Let $f^{(k)}$ ($0 \leq k \leq \mathsf{r}$) be a Reed Solomn codeword defined as follows,

$$\begin{aligned} f^{(k)} : L_k &\to \mathbb{F}, \\ x &\mapsto f_k(x). \end{aligned} \tag{8}$$

The notation caveat should be noted here that $f_k(X)$ represents a polynomial, while $f^{(k)}$ is an array of points representing a Reed Solomn codeword. The interpolant of $f^{(k)}$ would construct the polynomial $f_k(x)$. The rate of Reed Solomn codeword is defined as

$$\rho = \frac{d_k}{|L_k|}. \tag{9}$$

FRI is an interactive protocol containing a Commit phase and a Query phase, running for r rounds.

Commit Phase. During the k-th round of the Commit phase ($0 \leq k \leq \mathsf{r}-1$), the prover commits to $f^{(k)}$ and the verifier has oracle access to $f^{(k)}$. In this paper, Merkle tree commitment is employed.

During the last round $k = \mathsf{r}$, prover sends the $f^{(\mathsf{r})}$ to the verifier. By this point, the degree of $f_{\mathsf{r}}(X)$, which is the interpolant of $f^{(\mathsf{r})}$, should be no more than $\rho \cdot |L_{\mathsf{r}}| - 1$.

Query Phase. The verifier will validate that the prover adheres to the prescribed procedures.

Specifically, the verifier will randomly select $s^{(0)}$ from L_0, and iteratively computes a sequence of points $s^{(0)}, s^{(1)}, \cdots, s^{(\mathsf{r})}$. The verifier will query the value $f_{k+1}(s^{(k+1)})$, and two other points from $f^{(k)}$, and check the round consistency among those three points ($0 \leq k \leq \mathsf{r}-1$). The verifier will repeat this check for ℓ times, and accept only when all checks pass.

If verifier accepts, it means that the degree of the original polynomial $f_0(X)$ should be no more than $\rho \cdot |L_0| - 1$.

2.3 GKR

GKR [11] is a public coin interactive proof protocol for any language computable by a log-space uniform layered (fan-in 2) arithmetic circuit $\mathcal{C}$, in which a prover $\mathcal{P}$ can run the computation and interactively prove the correctness of the result (output gate(s)) to a verifier $\mathcal{V}$. Consider a circuit with depth denoted by d and size represented by S, where the size is defined as the total number of gates. For any layer ℓ within the circuit, S_ℓ indicates the number of gates at that layer. Specifically, $\ell = 0$ corresponds to the output layer, while $\ell = d$ represents the input layer. Additionally, ν is the size of the input to the circuit $\mathcal{C}$. The communication cost is $O(S_0 + d\log(S))$, the cost of $\mathcal{V}$ is $O(\nu + d\log(S))$, and the runtime of the $\mathcal{P}$ is bounded by $O(S^3)$. In the following, we briefly describe the GKR protocol following Thaler's presentation of the protocol in [17].

Circuit Encoding. At each layer ℓ, the gates are numerically labeled in binary from 0 to $S_\ell - 1$, assuming S_ℓ is a power of two (expressed as $S_\ell = 2^{k_\ell}$). The functions $\mathsf{in1}_\ell$ and $\mathsf{in2}_\ell$, each defined as $\mathsf{in1}_\ell, \mathsf{in2}_\ell : \{0,1\}^{k_\ell} \to \{0,1\}^{k_{\ell+1}}$, map a binary gate label at layer ℓ to its input gates at layer $\ell+1$. This mapping explicitly encodes the wiring-how outputs from gates at layer $\ell+1$ serve as inputs to a gate at layer ℓ. Accordingly, the functions add_ℓ and mult_ℓ, representing addition and multiplication gates at layer ℓ, are defined as

$$\mathsf{add}_\ell, \mathsf{mult}_\ell : \{0,1\}^{k_\ell} \times \{0,1\}^{k_{\ell+1}} \times \{0,1\}^{k_{\ell+1}} \to \{0,1\}.$$

For a gate labeled a at layer ℓ, these functions take as input the labels of three gates (a, b, c), and return 1 if and only if $(b, c) = (\mathsf{in1}_\ell(a), \mathsf{in2}_\ell(a))$. $\widetilde{\mathsf{add}}_\ell$ and $\widetilde{\mathsf{mult}}_\ell$ denote the Multilinear Extension (MLE) of add_ℓ and mult_ℓ. Additionally, the function $W_\ell : \{0,1\}^{k_\ell} \to \mathbb{F}$, maps gate at layer ℓ to the outputted value of the

gate. Accordingly, $\widetilde{W}_\ell$ denote the MLE of W_ℓ. The following equation describes how $\widetilde{W}_\ell$ can be derived form $\widetilde{W}_{\ell+1}$, $\widetilde{\mathsf{add}}_\ell$, and $\widetilde{\mathsf{mult}}_\ell$:

$$\widetilde{W}_\ell(z) = \sum_{b,c\in\{0,1\}^{k_{\ell+1}}} \Big(\widetilde{\mathsf{add}}_\ell(z,b,c)\left(\widetilde{W}_{\ell+1}(b)+\widetilde{W}_{\ell+1}(c)\right) + \widetilde{\mathsf{mult}}_\ell(z,b,c)\left(\widetilde{W}_{\ell+1}(b)\cdot\widetilde{W}_{\ell+1}(c)\right)\Big).$$

Multivariate Sum-Check. The GKR protocol consists of an iteration for each layer. In the iteration corresponding to layer $\ell < d$ of the circuit, $\mathcal{P}$ claims a specific value for $\widetilde{W}_\ell(r_\ell)$, with $r_\ell \in \mathbb{F}^{k_\ell}$ being a randomly selected point. Note that r_ℓ may have non-Boolean entries. In order to check the claim, $\mathcal{P}$ and $\mathcal{V}$ cooperate in a multivariate sum-check protocol [14] to the polynomial $f^{(\ell)}_{r_\ell}$ defined as

$$f^{(\ell)}_{r_\ell}(b,c) = \widetilde{\mathsf{add}}_\ell(r_\ell,b,c)\left(\widetilde{W}_{\ell+1}(b)+\widetilde{W}_{\ell+1}(c)\right) + \widetilde{\mathsf{mult}}_\ell(r_\ell,b,c)\left(\widetilde{W}_{\ell+1}(b)\cdot\widetilde{W}_{\ell+1}(c)\right).$$

Given that $\mathcal{V}$ does not know the polynomial $f^{(\ell)}_{r_\ell}$, to evaluate $f^{(\ell)}_{r_\ell}(b^*,c^*)$ in the final round of the sum-check protocol at a randomly chosen point $(b^*,c^*) \in \mathbb{F}^{k_{\ell+1}} \times \mathbb{F}^{k_{\ell+1}}$, $\mathcal{V}$ asks $\mathcal{P}$ to provide $z_1 = \widetilde{W}_{\ell+1}(b^*)$ and $z_2 = \widetilde{W}_{\ell+1}(c^*)$, which are then verified in the subsequent iteration $(i+1)$ through the *round consistency check* process. However, $\mathcal{V}$ can independently evaluate $\widetilde{\mathsf{add}}_\ell(r_\ell,b^*,c^*)$ and $\widetilde{\mathsf{mult}}_\ell(r_\ell,b^*,c^*)$, according to the circuit's structure.

Round Consistency Check. To verify z_1 and z_2, $\mathcal{V}$ aims to *reduce* these verifications into a single task: validating the $\mathcal{P}$'s claim of $\widetilde{W}_{\ell+1}(r_{\ell+1})$. This claim represents the summation outcome at the next layer denoted as

$$\widetilde{W}_{\ell+1}(r_{\ell+1}) = \sum_{b,c\in\{0,1\}^{k_{\ell+2}}} f^{(\ell+1)}_{r_{\ell+1}}(b,c).$$

To do so, let define the unique line $\lambda : \mathbb{F} \to \mathbb{F}^{k_{\ell+1}}$, such that $\lambda(0) = b^*$ and $\lambda(1) = c^*$. $\mathcal{P}$ then sends the polynomial $q = \widetilde{W}_{\ell+1}\big|_\lambda$ to $\mathcal{V}$, representing the restriction of $\widetilde{W}_{\ell+1}$ to the line λ. Upon receiving the polynomial, $\mathcal{V}$ first verifies that $q(0) = z_1$ and $q(1) = z_2$. Subsequently, $\mathcal{V}$ selects a random $r^* \in \mathbb{F}$ and sets $r_{\ell+1} = \lambda(r^*)$, then checks if $q(r^*) = \widetilde{W}_{\ell+1}(r_{\ell+1})$, which is the claim made by $\mathcal{P}$ for the next round.

Final Round Check. In the final round, $\mathcal{V}$ independently checks $\widetilde{W}_d(r_d)$.

3 Polaris

Polaris [9] is a zkSNARK protocol without a trusted setup that has quasi-linear time complexity for the prover and polylogarithmic proof size. Its verification

time is relative to the size of the arithmetic circuit representing the statement to be proven. It achieves this efficiency by encoding the R1CS instance as a univariate polynomial in a quadratic arithmetic program (QAP) [10]. The main source of Polaris' efficiency is an arithmetic layered circuit design that allows the verifier to delegate query computations to the prover and verify the results using the GKR protocol [11]. Polaris combines univariate polynomial encoding described in Sect. 3.2 with univariate sumcheck protocol in Aurora [5]. By accomplishing this, the protocol constructs an interactive proof that is complete and sound, and then extends it to incorporate zero-knowledge and non-interativeness using Fiat-Shamir protocol [8]. Figure 1 below shows the building blocks of the Polaris protocol.

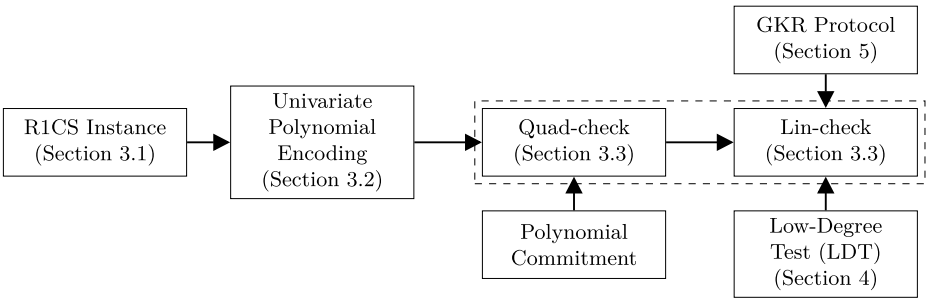

Fig. 1. Structure of the Polaris protocol with key sub-protocols.

3.1 R1CS Instance

The R1CS instance can be represented as a tuple $\mathbb{x} = (\mathbb{F}, A, B, C, \mathbf{v}, m, n)$, where $\mathbb{F}$ is the finite field, A, B are input matrices and C is the output matrix of degree $m \times m$ from the R1CS construction. $\mathbf{v}$ represents the vector containing the instance's public parameter. There are at most n non-zero entries in each matrix.

An R1CS relation $\mathcal{R}_{\text{R1CS}}$ is said to be satisfiable if there exists a witness $\mathbf{w} \in \mathbb{F}^{m-|v|-1}$ consisting of the circuit's private input and wire values such that

$$(A\mathbf{z}) \circ (B\mathbf{z}) = C\mathbf{z},$$

where $\mathbf{z} := (1,\mathbf{w},\mathbf{v})$ and "$\circ$" denotes the Hadamard product of the two vectors $A\mathbf{z}$ and $B\mathbf{z}$. This relation $\mathcal{R}_{\text{R1CS}}$ represents the input and output vectors of the gates in the arithmetic circuit.

3.2 Univariate Polynomial Encoding

Let H be an s-dimensional affine space of $\mathbb{F}$ such that $|H| = m$. The vector $\mathbf{z} = (1, \mathbf{v}, \mathbf{w}) \in \mathbb{F}^H$ is interpreted as a univariate function $Z : H \to \mathbb{F}$. This allows the accessing of any element of vector $\mathbf{z}$ using an index in H. The function $F_{\mathbf{w}}(\cdot)$ encodes $\mathbf{z}$, which contains the private witness $\mathbf{w}$, into a polynomial form for use in the protocol. It is defined as

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

A witness-instance pair $(\mathbb{x}, \mathbf{w})$ is deemed valid, i.e., $(\mathbb{x}, \mathbf{w}) \in \mathcal{R}_{\text{R1CS}}$ if and only if $F_{\mathbf{w}}(x) = 0$ for any $x \in H$. Polaris utilises the polynomial extension of $F_{\mathbf{w}}(\cdot)$, for its arithmetisation in the protocol. We assign

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

To find the coefficients of the three polynomials, we can compute the vector products $A\mathbf{z}$, $B\mathbf{z}$ and $C\mathbf{z}$, and then interpolate those vector results over H. Given that A, B and C are sparse matrices, the prover employs *sparse encoding* approach for efficiently finding the coefficient of each $\bar{A}$, $\bar{B}$, and $\bar{C}$. To do so, we define three functions for each matrix M, where $M \in \{A, B, C\}$. These functions, $\mathsf{row}, \mathsf{col} : [n] \to H$, and $\mathsf{val} : [n] \to \mathbb{F}$, respectively map from the set of indices $[n]$ to the row and column indices, and to the value of the non-zero entries within the matrix M. Here $[n] := \{1, 2, \cdots, n\}$. Thus, for every $x \in H$,

$$\bar{M}(x) = \sum_{i \in [n] \text{ s.t. } \mathsf{row}(i) = x} \mathsf{val}(i) \cdot Z(\mathsf{col}(i)).$$

We can represent any sequence of length N over $\mathbb{F}$ as a univariate polynomial by using Lagrange interpolation. For a sequence $\{f_i\}$, applying Lagrange interpolation would result in a polynomial f, where $f(\rho(i)) = f_i$. Here, $\rho(i) = i_0\alpha_0 + \cdots + i_{s-1}\alpha_{s-1}$ with $(i_0, \cdots, i_{s-1})$ as the binary representation of i. The points used in the Lagrange interpolation are $(\rho(i), f_i), 0 \leq i < 2^s - 1$. In other words, we have

$$f(x) = \sum_{i=0}^{2^s - 1} f_i \sigma_i(x), \tag{10}$$

where $\{\sigma_i \,|\, 0 \leq i < 2^s\}$ is the Lagrange basis, given by

$$\sigma_i(x) = \frac{\prod_{j \neq i}(x - \rho(j))}{\prod_{j \neq i}(\rho(i) - \rho(j))}. \tag{11}$$

Since $f(x)$ is a polynomial with coefficients in $\mathbb{F}$, we can naturally extend f from a function mapping $N_{2^s} \to \mathbb{F}$ to a function over $\mathbb{F}$.

Given the definition of the vanishing polynomial in Equation (4), we can further represent $\mathbb{Z}_H(x)$, an affine linearized polynomial of $\mathbb{F}$ with dimension s, as below,

$$\mathbb{Z}_H(x) = x^{2^k} + \sum_{i=1}^{s} c_i x^{2^{i-1}} + c_0, c_i \in \mathbb{F}.$$

If H is linear, then $c_0 = 0$. Note that $c_1 \neq 0$, since $\mathbb{Z}_H(x)$ has no repeated roots. This also means that $\mathbb{Z}_H$ has degree $|H|$. To perform univariate encoding, Polaris makes use of the following bivariate polynomial:

$$\Delta_H(x, y) := \frac{\mathbb{Z}_H(x) - \mathbb{Z}_H(y)}{x - y}, \tag{12}$$

which is a polynomial of degree $|H| - 1$ because $\mathbb{Z}_H(x) - \mathbb{Z}_H(y)$ is divisible by $x - y$. With the above notation, the Lagrange basis in 11 is given as

$$\sigma_i(x) = \frac{\Delta_H(x, \rho(i))}{c_1} = \frac{1}{c_1} \frac{Z_H(x)}{x - \rho(i)}, 0 \leq i < 2^s,$$

and (10) now becomes

$$f(x) = \frac{1}{c_1} \sum_{i \in N_{2^s}} f_i \Delta_H(x, \rho(i)) = \frac{1}{c_1} \sum_{i \in N_{2^s}} f_i \frac{Z_H(x)}{x - \rho(i)}. \tag{13}$$

Equation (13) represents a univariate polynomial leveraged from bivariate interpolation.

3.3 Quadratic and Linear Checks

In the Polaris protocol, the prover $\mathcal{P}$ wants to convince the verifier $\mathcal{V}$ that $(\mathbb{x}, \mathbf{w}) \in \mathcal{R}_{\text{R1CS}}$ which is true if and only if the univariate $\mathbb{F}_{\mathbf{w}}(x) = 0$ at all points within the affine subspace H of $\mathbb{F}$. According to the factor theorem (Theorem 1 in [9]), this verification is the same as determining if there's a polynomial $G(X)$ where $\deg(F_{\mathbf{w}}) - |H| \leq |H| - 2$, such that

$$F_{\mathbf{w}}(X) = \mathbb{Z}_H(X) \cdot G(X). \tag{14}$$

This is verified through two distinct checks, namely, the Quad-Check and the Lin-Check, as presented in the following sections.

Quad-Check. In the Quad-Check protocol, $\mathcal{V}$ conducts a probabilistic check of Equation (14) at a randomly selected point. To realize this, $\mathcal{P}$ commits to the polynomial $G(X)$ using FRI univariate polynomial commitment and sends the commitment to $\mathcal{V}$. Section 4 presents the instantiation of the FRI commitment scheme in our protocol. Then, $\mathcal{V}$ sends the randomly selected point $r_x \in F \setminus H$

to $\mathcal{P}$. Then, the prover evaluates $\eta = G(r_x)$ and sends the result to $\mathcal{V}$. Finally, $\mathcal{V}$ query the commitment to verify the evaluation.

To verify $F_w(r_x) = \mathbb{Z}_H(r_x) \cdot G(r_x)$, the verifier also needs to compute $F_w(r_x)$. Recall that

$$F_w(r_x) = \bar{A}(r_x) \cdot \bar{B}(r_x) - \bar{C}(r_x).$$

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 check if

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

This equation is represents the quadratic-check of the R1CS instance, or *quad-check*, owing to it checking the R1CS verifiability in $\rho(i)$, where i is within the range of the matrix degree m. But the verifier must now additionally verify three new claims from the prover $\mathcal{P}$:

$$\bar{A}(r_x) = v_A, \bar{B}(r_x) = v_B, \text{ and } \bar{C}(r_x) = v_C. \tag{15}$$

To do so, $\mathcal{P}$ and $\mathcal{V}$ engage in the Lin-check protocol.

Lin-Check. In the Lin-Check protocol, the three claims presented in Equation (15) are combined into one to be evaluated in a single claim. Accordingly, the verifier chooses r_A, r_B, $r_C \in \mathbb{F}$ uniformly at random and sends them to the prover, which can test if

$$c = r_A \cdot \bar{A}(r_x) + r_B \cdot \bar{B}(r_x) + r_C \cdot \bar{C}(r_x). \tag{16}$$

We can rewrite c as follows.

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

where $y \in H$. We denote

$$Q_{r_x}(Y) \;:=\; \big(r_A \cdot A(r_x, Y) + r_B \cdot B(r_x, Y) + r_C \cdot C(r_x, Y)\big) \cdot Z(Y). \tag{17}$$

Consequently, to verify $c = \sum_{y \in H} Q_{r_x}(y)$, $\mathcal{P}$ and $\mathcal{V}$ engage in the univariate sumcheck protocol realized by the FRI low degree test (LDT). Section 4 presents our instantiation of the FRI LDT protocol. At the end of the FRI protocol, $\mathcal{V}$ needs oracle access to the evaluations of $Q_{r_x}(\cdot)$ at κ points $r_y \in L$, where L is an affine subspace $L \subseteq \mathbb{F}$ such that $|L| > 2|H|$ and $L \cap H = \varnothing$.

We denote the evaluations of $A(r_x, r_y)$, $B(r_x, r_y)$, and $C(r_x, r_y)$ in $Q_{r_x}(r_y)$ as $M(r_x, r_y)$ where $M \in \{A, B, C\}$ using $\mathsf{row}(i)$, $\mathsf{col}(i)$, and $\mathsf{val}(i)$ functions as the following equation:

$$M(r_x, r_y) = \frac{\mathbb{Z}_H(r_x) \cdot \mathbb{Z}_H(r_y)}{c_1^2} \cdot \sum_{i \in [n]} \frac{\mathsf{val}(i)}{(r_x - \mathsf{row}(i))\,(r_y - \mathsf{col}(i))}, \tag{18}$$

where we denote the second part as $C_M(r_x, r_y)$, such that

$$C_M(r_x, r_y) := \sum_{i \in [n]} \frac{\mathsf{val}(i)}{(r_x - \mathsf{row}(i))\,(r_y - \mathsf{col}(i))}. \tag{19}$$

Given that $\mathsf{row}(i), \mathsf{col}(i) \in H$, $r_x \in \mathbb{F} \backslash H$ and $r_y \in L$, such that $L \cap H = \varnothing$, the denominators in $C_M(r_x, r_y)$ are non-zero. To reduce the computational burden on the verifier, the task of evaluating $C_M(r_x, r_y)$ at κ points $r_y \in L$ is not performed locally. Instead, this computation is outsourced to the prover. The verifier then uses the GKR protocol to validate the results provided by the prover. In Sect. 5, we explain how we utilized the GKR protocol and designed the corresponding circuit for $C_M(r_x, r_y)$.

3.4 Adding Zero-Knowledge

The interactive protocol reveals information about the witness $\mathbf{w}$ when $\mathcal{P}$ sends evaluations of $G(r_x), \bar{A}(r_x), \bar{B}(r_x), \bar{C}(r_x)$ and $Z(\cdot)$, and invokes the low degree test on related polynomials of $((Q_{r_x}(Y))$. To prevent these "leakages" and achieve zero-knowledge, three main modifications are employed:

1. **Eliminating leakage of queries on $Z(\cdot)$.** The prover chooses a random polynomial $R_Z(\cdot)$ of degree κ and computes a $\widetilde{Z}(Y) := Z(Y) + \mathbb{Z}_H(Y) \cdot R_Z(Y)$. Even though $\widetilde{Z}(y) = Z(y)$ for $y \in H$, $\widetilde{Z}(\cdot)$ polynomial evaluations outside H preserve zero knowledge as $R_Z(\cdot)$ masks the information of polynomial $Z(\cdot)$.
2. **Modifications to the evaluations of $\bar{A}(\cdot), \bar{B}(\cdot), \bar{C}(\cdot)$.** The prover $\mathcal{P}$ samples some random polynomials $R_A(\cdot)$, $R_B(\cdot)$, $R_C(\cdot)$ of degree $|H| - 1$ and provides that

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

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

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

   As the $R(\cdot)$ are random, revealing the evaluations of $\widetilde{A}(\cdot), \widetilde{B}(\cdot), \widetilde{C}(\cdot)$ outside H does not leak information about the values in the witness w.
3. **Modifications to the polynomial $Q(r_x)(\cdot)$.** To uphold the zero-knowledge property of the $Q_{r_x}(\cdot)$ polynomial from Equation (17) in the univariate sumcheck phase, the prover $\mathcal{P}$ picks a random polynomial $S_Q(\cdot)$ of degree $2|H| + \kappa - 1$, and sends an $s_1 = \sum_{y \in H} S_Q(y)$ to $\mathcal{V}$. To this, $\mathcal{V}$ responds with a random challenge $\alpha_1 \in \mathbb{F}$. $\mathcal{P}$ and $\mathcal{V}$ then run a sumcheck on the following linearised representation:

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

where $\widetilde{c} = r_A \cdot \widetilde{A}(r_x) + r_B \cdot \widetilde{B}(r_x) + r_C \cdot \widetilde{C}(r_x)$, referenced from Equation (16). This ensures $\widetilde{c}$ and s_1 can be correctly computed due to the random linear combination of the sumcheck, while revealing no information about $\widetilde{Q}r_x(\cdot)$ as it is masked by the random polynomial $S_Q(\cdot)$ [5] [19] .

To obtain the full zero-knowledge protocol, we replace relevant components with their zero-knowledge versions, with the additional need for the prover $\mathcal{P}$ to commit to the random polynomials using Merkle tree commitments at the beginning, to be later opened at κ points by $\mathcal{V}$. A key advantage is that the GKR protocol remains unchanged, avoiding expensive cryptographic computations.

4 FRI Instantiation

In the FRI protocol, let r be the number of rounds, let $\{\beta_0 = 1, \beta_1, \beta_2, \cdots, \beta_{191}\}$ be one of the $\mathbb{F}_2$-basis of $\mathbb{F}_{2^{192}}$ defined in Sect. 2.1.

Evaluation Domains. We assume that the verifier and prover have agreed upon the evaluation domains L_k $(0 \leq k \leq \mathsf{r})$, whose sizes are the powers of 2, and $m = \log_2(|L_0|)$. Those affine subspaces are adopted in Preon [6]. The evaluation domains are recursively defined as follows. First,

$$L_0 =< \beta_0, \beta_1, \cdots, \beta_{m-1} > +\beta_m. \tag{20}$$

For an integer i $(0 \leq i \leq 2^m - 1)$, its binary expression is

$$i = (i_{m-1} i_{m-2} \cdots i_1 i_0)_2 = i_0 + i_1 \cdot 2 + \cdots + i_{m-1} \cdot 2^{m-1}, i_j \in \{0, 1\}.$$

Then we can define the i-th element in L_0 as

$$L_0[i] = (i_0 \cdot \beta_0 + i_1 \cdot \beta_1 + \cdots + i_{m-1} \cdot \beta_{m-1}) + \beta_m,\ 0 \leq i < 2^m. \tag{21}$$

Let us define the polynomial

$$q_0(X) = X(X - \beta_0) \tag{22}$$

and

$$\beta_j^{(1)} = q_0(\beta_{j+1}),\ 0 \leq j \leq m-1,$$

then L_1 is defined as

$$L_1 = q_0(L_0) =< \beta_0^{(1)}, \beta_1^{(1)}, \cdots, \beta_{m-2}^{(1)} > +\beta_{m-1}^{(1)}, \tag{23}$$

and the i-th element in L_1 is

$$L_1[i] = (i_0 \cdot \beta_0^{(1)} + i_1 \cdot \beta_1^{(1)} + \cdots + i_{m-2} \cdot \beta_{m-2}^{(1)}) + \beta_{m-1}^{(1)},\ 0 \leq i < 2^{m-1}.$$

We can thus recursively define that, for $1 \leq k \leq m-1$,

$$\begin{aligned} q_k(X) &= X(X - \beta_0^{(k)}), \\ \beta_j^{(k+1)} &= q_k(\beta_{j+1}^{(k)}),\ \text{for } 0 \leq j \leq m-k-1, \\ L_{k+1} &= q_k(L_k) =< \beta_0^{(k+1)}, \beta_1^{(k+1)}, \cdots, \beta_{m-k-2}^{(k+1)} > +\beta_{m-k-1}^{(k+1)}, \end{aligned} \tag{24}$$

and for $0 \leq i < 2^{m-k-1}$, the i-th element of L_{k+1} is defined as

$$L_{k+1}[i] = (i_0 \cdot \beta_0^{(k+1)} + i_1 \cdot \beta_1^{(k+1)} + \cdots + i_{m-1} \cdot \beta_{m-k-2}^{(k+1)}) + \beta_{m-k-1}^{(k+1)}.$$

FRI Commit Phase.

Prover's input: $f^{(0)} : L_0 \to \mathbb{F}$, a purported Reed Solomn codeword corresponding to polynomial $f_0(X)$, with rate ρ.

Loop for $0 \leq k \leq \mathsf{r} - 1$:

1. Prover commits to all codewords.
   - $f^{(k)} : L_k \to \mathbb{F}$ is recursively defined in Step 3.
   - Prover computes a Merkle commitment to $f^{(k)}$ and sends out the Merkle root.
2. Verifier sends a uniformly random $\alpha^{(k)} \in \mathbb{F}$.
3. Prover defines the codeword $f^{(k+1)}$ with domain L_{k+1}, such that for each $0 \leq i < |L_{k+1}|$,
   - It is easy to check that

$$q_k(L_k[2i]) = q_k(L_k[2i+1]) = L_{k+1}[i].$$

   - The values in codeword $f^{(k+1)}$ is derived from the previous codeword,

$$f_{k+1}(L_{k+1}[i]) = \frac{f_k(L_k[2i]) - f_k(L_k[2i+1])}{L_k[2i] - L_k[2i+1]}(\alpha^{(k)} - L_k[2i]) + f_k(L_k[2i]).$$

     Here the denominator $L_k[2i] - L_k[2i+1] = \beta_0^{(k)}$, whose inverse can be precomputed.

For $k = \mathsf{r}$:

- $f^{(\mathsf{r})} : L_{\mathsf{r}} \to \mathbb{F}$ is defined in Step 2.
- prover sends out the last codeword $f^{(\mathsf{r})}$.

FRI Query Phase.

1. Verifier extracts all Merkle roots, all challenges $\alpha^{(0)}, \alpha^{(1)}, \cdots, \alpha^{(\mathsf{r}-1)}$, and the last codeword $f^{(\mathsf{r})}$. Verifier has oracle access to $f^{(0)}, f^{(1)}, \cdots, f^{(\mathsf{r}-1)}$.
2. Verifier computes the interpolant $f_{\mathsf{r}}(X)$ from points $f^{(\mathsf{r})}$, then check if the degree of $f_{\mathsf{r}}(X)$ is no more than $\rho \cdot |L^{(\mathsf{r})}| - 1$. If not, reject.
3. Verifier does the consistency check between two neighboring codewords. Repeat ℓ times:
   - Sample random index $s^{(0)} = i$ from $0 \leq i < |L_0|$, and for $0 \leq k \leq \mathsf{r} - 1$, compute $s^{(k+1)} = \lfloor s^{(k)}/2 \rfloor$.
   - If $s^{(\mathsf{r})}$ is repeated, resample $s^{(0)}$.

– **Round consistency check**:
Denote

$$x_1^{(k)} = L_k[2\, s^{(k+1)}],\ x_2^{(k)} = L_k[2\, s^{(k+1)} + 1],$$

the verifier first queries

$$f_{k+1}(L_{k+1}[s^{(k+1)}]), f_k(x_1^{(k)}), f_k(x_2^{(k)}),$$

and checks the Merkle commit paths for those three points, then checks that for every $k \in \{0, 1, 2, \cdots, \mathsf{r} - 1\}$,

$$f_{k+1}(L_{k+1}[s^{(k+1)}]) = \frac{f_k(x_1^{(k)}) - f_k(x_2^{(k)})}{x_1^{(k)} - x_2^{(k)}}(\alpha^{(k)} - x_1^{(k)}) + f_k(x_1^{(k)}). \quad (25)$$

If any one equation of the consistency check fails, reject.
Notice that $x_1^{(k)} - x_2^{(k)} = \beta_0^{(k)}$, whose inverse can be precomputed.

4. Accept if all checks pass. This implies that the degree of the original polynomial $f_0(X)$ is no more than $\rho \cdot |L_0| - 1$.

5 GKR Circuit

In this section we are going to present the circuit $\mathfrak{C}$ to provide a verifiable computation for $C_M(r_x, r_y)$. By considering that the arithmetic circuit can only include addition and multiplication operations, computing the multiplicative inverse is expensive. Therefore, we turn the straightline computation, in which we should compute the multiplicative inverse, into an *satisfiability* circuit instance [17]. Therefore, the circuit $\mathfrak{C}$ receives a set of inputs that includes $\overline{\mathsf{c}^{(d)}} = \{\mathsf{c}^{(d)}(i) \mid i \in [n]\}$, alongside r_x, r_y, $\overline{\mathsf{row}} = \{\mathsf{row}(i) \mid i \in [n]\}$, $\overline{\mathsf{col}} = \{\mathsf{col}(i) \mid i \in [n]\}$, and $\overline{\mathsf{val}} = \{\mathsf{val}(i) \mid i \in [n]\}$, where d denotes the depth of the circuit and $\mathsf{c}^{(\ell)}(i)$ is defined as follows:

$$\mathsf{c}^{(\ell)}(i) := \begin{cases} \frac{\mathsf{val}(i)}{(r_x - \mathsf{row}(i))(r_y - \mathsf{col}(i))}, & \ell = d,\ i \in [n] \% 1 \le i \le n \\ \mathsf{c}^{(\ell+1)}(2i-1) + \mathsf{c}^{(\ell+1)}(2i), & \ell \in [0, d), i \in [2^{\ell-d} n] \\ 0 & \text{otherwise.} \end{cases} \quad (26)$$

This definition specifies that $\mathfrak{C}$ encompasses all terms included in the summation outlined in Eq. (19), utilizing these terms as inputs (where $\ell = d$). To enable verifiable computation of this summation, $\mathfrak{C}$ is equipped with a *summation* component structured as a binary tree. Within this structure, for any level $\ell < d$, the function $\mathsf{c}^{(\ell)}(i)$ calculates the sum of two preceding terms from the immediately lower layer (i.e., layer $\ell + 1$), using addition gates. In Eq. (26), we simplify our notation by assuming a balanced binary tree structure for ease of definition. This assumption entails that the input layer, denoted by $\mathsf{c}^{(d)}(i)$, comprises a power of two elements. Consequently, the number of layers are $d = \log n$. The output layer, serving as the tree's root, is designated by $\mathsf{c}^{(0)}(1) = C_M(r_x, r_y)$. However,

this binary structure is not a strict requirement for the actual implementation. In practice, given that each addition gate necessitates two inputs, layers featuring an odd number of elements incorporate a zero as the supplemental input for the subsequent layer. This zero is consistently available at every layer of the circuit.

For $\mathfrak{C}$ to qualify as a satisfiability circuit, it must verify that each asserted term $\mathsf{c}^{(d)}(i)$ aligns with the parameters r_x, r_y, $\mathsf{row}(i)$, $\mathsf{col}(i)$, and $\mathsf{val}(i)$. Consequently, we need to design a *consistency* component embedded to the circuit. The number of layers $d = \log n$ is determined by the summation component presented above.

Layer $\ell = d-1$: (The immediate layer beyond the input layer) $\alpha(i) = r_x - \mathsf{row}(i)$, $\beta(i) = r_y - \mathsf{col}(i)$, where $\overline{\alpha} = \{\alpha(i) \mid i \in [n]\}$ and $\overline{\beta} = \{\beta(i) \mid i \in [n]\}$ are realized by addition gates.

Layer $\ell = d-2$: $\gamma(i) = \alpha(i)\beta(i)$ for $i \in [n]$, where $\overline{\gamma} = \{\gamma(i) \mid i \in [n]\}$ is realized by multiplication gates

Layer $\ell = d-3$: $\mathsf{val}'(i) = \gamma(i)\mathsf{c}^{(d)}(i)$, where $\mathsf{c}^{(d)}(i)$ is copied to this layer from the input layer by being added to zero in previous two layers. This zero is consistently available at every layer of the circuit. $\overline{\mathsf{val}'} = \{\mathsf{val}'(i) \mid i \in [n]\}$ is realized by multiplication gates.

Layer $\ell = d-4$: $\zeta(i) = \mathsf{val}'(i) + \mathsf{val}(i)$, where $\mathsf{val}(i)$ is also copied from the input layer to this layer by being added to zero. This layer performs an addition of $\mathsf{val}'(i)$ to $\mathsf{val}(i)$, such that, when $\mathsf{val}'(i)$ equals $\mathsf{val}(i)$, $\zeta(i) = 0$, since the addition is equivalent to the XOR operation in this field. This property ensures $\mathsf{val}'(i)$ and $\mathsf{val}(i)$ are equal. $\overline{\zeta} = \{\zeta(i) \mid i \in [n]\}$ is realized by addition gates.

Layers $\ell < d-4$: $\overline{\zeta}$ is copied to the upper layer till the output layer.

Figure 2 illustrates the GKR circuit used in Polaris. For clarity, only connections from the input layer are shown, as the other connections follow a similar pattern and have been omitted from this visual representation.

The summation component of the presented GKR circuit consists of $2n-1$ gates, under the assumption that it forms a balanced binary tree (i.e., n is a power of two). If this condition is not met, the circuit will have a number of gates that is close to this figure. The consistency component of the circuit has $(n+1)\log n + 7n + 2$ gates. Consequently the size of the circuit is $S = (n+1)\log n + 9n + 1$ ($2n$ of them are multiplication and the rest are addition gates). The prover only needs to send $\mathsf{c}^{(0)}(1)$ to the verifier as the output of the circuit because the verifier assumes that the other $n+1$ gate values should be zero; therefore, $S_0 = 1$ (S_0 is the size of the output layer). Note that the depth of the circuit is $d = \log(n)$. According to Sect. 2.3, the communication cost is $O(S_0 + d\log(S)) = O(\log(n) \cdot \log((n+1)\log n + 9n))$ which simplifies to $O(\log(n) \cdot \log(n \log n)) = O((\log(n))^2)$.

The input size to the circuit is $\nu = 4n + 3$. According to Sect. 2.3, the verifier's computation cost should be $O(\nu + d\log(S)) = O(n + \log(n) \cdot (n\log(n))) = O(n(\log(n))^2)$. However, in the implementation of the GKR protocol, this computation can be delegated to the prover via verifiable polynomial delega-

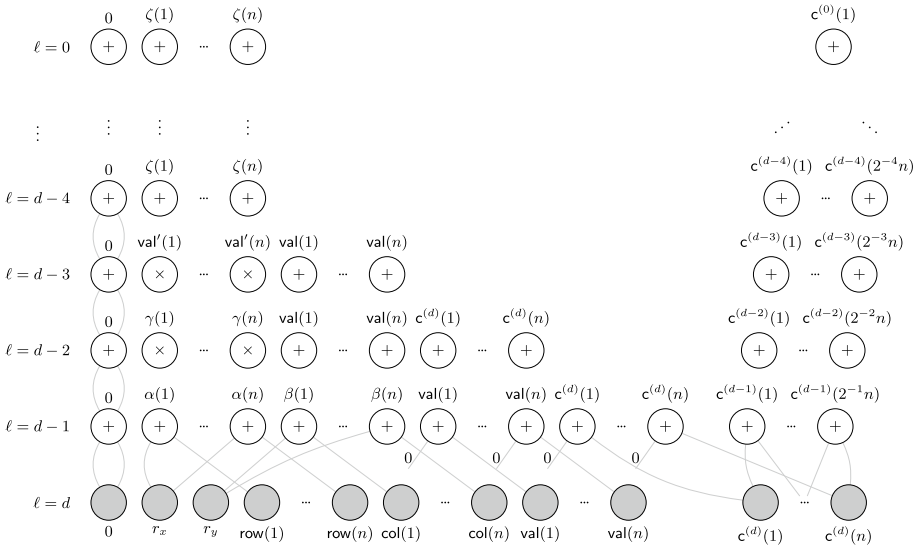

Fig. 2. The GKR satisfiability circuit for calculating $C_M(r_x, r_y)$ (Equation (19)). Sample connections are shown for clarity. $\forall i, \zeta(i) = 0$, and $\mathsf{c}^{(0)}(1) = C_M(r_x, r_y)$.

tion (VPD) schemes such as Virgo [18]. Finally, the prover's computation is $O(S^3) = O((n \log(n))^3)$.

6 Conclusion

In this paper, we introduced the Polaris protocol and its construction components. We presented an instantiation of the GKR circuit to minimize the number of gates, with the aim to reduce communication overhead, as well as the verifier and prover's complexities. We also explained an instantiation of the FRI protocol that eliminates the field inversion operations in both the Commit phase and Query phase, while would help to achieve better efficiency. The complete performance benchmark and the expected improvements will be demonstrated in the upcoming full implementation of Polaris.

References

1. The halo2 Book. https://zcash.github.io/halo2/. Accessed 19 Mar 2024
2. Ames, S., Hazay, C., Ishai, Y., Venkitasubramaniam, M.: Ligero: lightweight sublinear arguments without a trusted setup. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, pp. 2087–2104 (2017)
3. Ben-Sasson, E., Bentov, I., Horesh, Y., Riabzev, M.: Fast reed-solomon interactive oracle proofs of proximity. In: 45th International Colloquium on Automata, Languages, and Programming (ICALP 2018). Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik (2018)

4. Ben-Sasson, E., Bentov, I., Horesh, Y., Riabzev, M.: Scalable zero knowledge with no trusted setup. In: Boldyreva, A., Micciancio, D. (eds.) CRYPTO 2019. LNCS, vol. 11694, pp. 701–732. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-26954-8_23
5. Ben-Sasson, E., Chiesa, A., Riabzev, M., Spooner, N., Virza, M., Ward, N.P.: Aurora: transparent succinct arguments for R1CS. In: Ishai, Y., Rijmen, V. (eds.) EUROCRYPT 2019. LNCS, vol. 11476, pp. 103–128. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-17653-2_4
6. Chen, M.S., et al.: Preon: zk-SNARK based signature scheme (2023)
7. Chiesa, A., Ojha, D., Spooner, N.: FRACTAL: post-quantum and transparent recursive proofs from holography. In: Canteaut, A., Ishai, Y. (eds.) EUROCRYPT 2020. LNCS, vol. 12105, pp. 769–793. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-45721-1_27
8. Fiat, A., Shamir, A.: How to prove yourself: practical solutions to identification and signature problems. In: Conference on the Theory and Application of Cryptographic Techniques, pp. 186–194. Springer (1986)
9. Fu, S., Gong, G.: Polaris: transparent succinct zero-knowledge arguments for R1CS with efficient verifier. Proc. Priv. Enhancing Technol., 544–564 (2022). https://doi.org/10.2478/popets-2022-0027
10. Gennaro, R., Gentry, C., Parno, B., Raykova, M.: Quadratic span programs and Succinct NIZKs without PCPs. In: Johansson, T., Nguyen, P.Q. (eds.) EUROCRYPT 2013. LNCS, vol. 7881, pp. 626–645. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38348-9_37
11. Goldwasser, S., Kalai, Y.T., Rothblum, G.N.: Delegating computation: interactive proofs for muggles. In: Proceedings of the Fortieth Annual ACM Symposium on Theory of Computing, STOC 2008, pp. 113–122. Association for Computing Machinery, New York (2008). https://doi.org/10.1145/1374376.1374396
12. Groth, J.: On the size of pairing-based non-interactive arguments. In: Fischlin, M., Coron, J.-S. (eds.) EUROCRYPT 2016. LNCS, vol. 9666, pp. 305–326. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49896-5_11
13. Haböck, U.: A summary on the FRI low degree test. Cryptology ePrint Archive (2022)
14. Lund, C., Fortnow, L., Karloff, H., Nisan, N.: Algebraic methods for interactive proof systems. J. ACM **39**(4), 859–868 (1992)
15. Setty, S.: Spartan: efficient and general-purpose zkSNARKs without trusted setup. In: Annual International Cryptology Conference, pp. 704–737. Springer (2020)
16. Thaler, J.: A note on the GKR protocol (2015)
17. Thaler, J.: Proofs, arguments, and zero-knowledge. Now Found. Trends (2022). https://doi.org/10.1561/978163828125
18. Zhang, J., Xie, T.: Virgo: zero knowledge proofs system without trusted setup (2019). https://api.semanticscholar.org/CorpusID:221606763
19. Zhang, J., Xie, T., Zhang, Y., Song, D.: Transparent polynomial delegation and its applications to zero knowledge proof. In: 2020 IEEE Symposium on Security and Privacy (SP), pp. 859–876. IEEE (2020)