Ring Proof Specification

22-08-2024-draft-6

Abstract

This document describes a cryptographic scheme based on SNARKs (Succinct Non-Interactive Arguments of Knowledge) that enables a prover to demonstrate knowledge of a secret scalar t and a secret index k within a group of public keys, where each public key is a point on an elliptic curve. The scheme ensures that, when combined with a public elliptic curve point H, the relation $R = PK_k + t\text{ů}H$ is satisfied. It leverages elliptic curve operations, a polynomial commitment scheme, and the Fiat-Shamir heuristic to achieve non-interactivity and zero-knowledge properties.

1. Notation

1.1. Basics

Basic Sets

- $\mathbb{N}_k = \{0, \dots, k-1\}$
- $\mathbb{B} = \mathbb{N}_2$

Vectors Operations

- $\overline{x} = (x_0, \dots, x_{n-1}),\ \overline{x}_i = x_i,\ 0 \le i < n$
- $\overline{a} \| \overline{b} = (a_0, \dots, a_{n-1}, b_0, \dots, b_{m-1})$
- $x^{\|n} = (x, \dots, x) \in X^n$

Kronecker Delta

- $\delta_{ij} = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases}$

Lagrange Basis Polynomials

- $L_i = L_{\mathbb{D},i} \in \mathbb{F}[X]^{<N}, \quad i \in \mathbb{N}_N, \quad L_i(\omega^j) = \delta_{ij}$

1.2. Curves and Fields

- $\langle \omega \rangle = \mathbb{D} \subseteq \mathbb{F}^*, \quad |\mathbb{D}| = N \in \mathbb{N}$
    - Cyclic subgroup generated by ω in the multiplicative group $\mathbb{F}^*$.

Elliptic Curves

- $J = J/\mathbb{F}$ – Elliptic curve J defined over the field $\mathbb{F}$.
- $\tilde{\mathbb{J}} = J(\mathbb{F})$ – Group of $\mathbb{F}$-rational points on J.
- $\mathbb{J} \subset \tilde{\mathbb{J}}$ – Prime order subgroup of $\tilde{\mathbb{J}}$.

Scalar Field

- $\mathbb{F}_{\mathbb{J}}$ – Field associated with the elliptic curve $\mathbb{J}$, with $|\mathbb{F}_{\mathbb{J}}| = |\mathbb{J}|$.
- $N_J = \lceil \log_2 |\mathbb{F}_{\mathbb{J}}| \rceil$ – Number of bits to represent an element of $\mathbb{F}_{\mathbb{J}}$.
- $N_K = N - N_J - 4$ – Maximum size of the ring handled with a domain of size N.

1.3. Support Functions

Unzip

- unzip : $\mathbb{J}^k \to (\mathbb{F}^k, \mathbb{F}^k)$; $\quad \overline{p} \to (\overline{p}_x, \overline{p}_y)$
  - Given a vector $\overline{p}$ of k elliptic curve points, unzip separates $\overline{p}$ into two vectors: $\overline{p}_x$ and $\overline{p}_y$, containing the x and y coordinates of each point, respectively.

Polynomial Interpolation

- Interpolate : $\mathbb{F}^k \to \mathbb{F}[x]^{<k}$; $\quad \overline{x} \to f$
  - Construct a polynomial by interpolating the vector values over $\mathbb{D}$

Polynomial Commitment Scheme

- PCS.Commit : $\mathbb{F}[x] \to \mathbb{G}$; $\quad f \to C_f$
  - Commits to a polynomial f over $\mathbb{F}$, with commitment in group $\mathbb{G}$.
- PCS.Open : $(\mathbb{G}, \mathbb{F}) \to (\mathbb{F}, \Pi)$; $\quad (C_f, x) \to (y, \pi)$
  - Evaluates the committed polynomial f at point x, returning evaluation y and proof π. The proof domain Π depends on the PCS.
- PCS.Verify : $(\mathbb{G}, \mathbb{F}, \mathbb{F}, \Pi) \to \mathbb{B}$; $\quad (C_f, x, y, \pi) \to (0|1)$
  - Verifies whether $y = f(x)$ given the commitment C_f and proof π.

Fiat-Shamir Transform

- FS : $\mathbb{S} \to \mathbb{F}$; $\mathbf{s} \to x$
  - Maps a serializable object $\mathbf{s} \in \mathbb{S}$ to $\mathbb{F}$, typically via some cryptographically secure hash function.

2. Parameters

2.1. Scheme Specific

- $\square \in \mathbb{J}$ – Padding element, a point on $\mathbb{J}$ with unknown discrete logarithm.
- $H \in \mathbb{J}$ – Pedersen blinding base point.
- $\overline{H} = (H, 2H, 4H, \ldots, 2^{N_J - 1}H) \in \mathbb{J}^{N_J}$ – Vector of scaled multiples of H.
- $S \in \tilde{\mathbb{J}} \setminus \mathbb{J}$ – Point in $\tilde{\mathbb{J}}$ used as seed for accumulation, ensuring the result is never the identity.

2.2. Public Data

- $\overline{PK} \in \mathbb{J}^{N_K}$ – Vector of public keys in the ring, padded with $\square$ to length N_K if needed.

2.3. Witness Data

- $t \in \mathbb{F}_{\mathbb{J}}$ – Prover's one-time secret.
- $k \in \mathbb{N}_{N_K}$ – Prover's index within the ring, identifying which public key in $\overline{PK}$ belongs to the prover.

2.4. Preprocessing

2.4.1. Public Input Preprocessing Concatenate ring points with scaled multiples of H:

$$\overline{P} = \overline{PK} \| \overline{H} = (P_0, \ldots, P_{N-5}) \in \mathbb{J}^{N-4}$$

$$\overline{p}_x = (P_{x,0}, \ldots, P_{x,N-5}, 0, 0, 0, 0) \in \mathbb{F}^N$$

$$\overline{p}_y = (P_{y,0}, \ldots, P_{y,N-5}, 0, 0, 0, 0) \in \mathbb{F}^N$$

Ring items selector:

$$\overline{s} = 1^{\| N_K} \;\|\; 0^{\| N - N_K} \in \mathbb{F}^N$$

2.4.1 Interpolation The resulting vectors are interpolated over $\mathbb{D}$:

$$p_x = \text{Interpolate}(\overline{p}_x)$$

$$p_y = \text{Interpolate}(\overline{p}_y)$$

$$s = \text{Interpolate}(\overline{s})$$

2.4.2. Commit to the constructed vectors

$$C_{p_x} = \text{PCS.Commit}(p_x)$$
$$C_{p_y} = \text{PCS.Commit}(p_y)$$
$$C_s = \text{PCS.Commit}(s)$$

2.5. Relation to Prove

Knowledge of k and t such that $R = PK_k + tH$.

$$\mathfrak{R}_H = \{(R, \overline{PK}; k, t) \mid R = PK_k + tH \; ; \; R \in \mathbb{J}, \; \overline{PK} \in \mathbb{J}^{N_K}, \; k \in \mathbb{N}_{N_K}, \; t \in \mathbb{F}_{\mathbb{J}}\}$$

---

3. Prover

3.1. Witness Polynomials

3.1.1. Bits Vector

- $\overline{k} \in \mathbb{B}^{N_K}$ – Binary vector representing the index k in the ring. $\overline{k}$ has N_K elements where $k_i = \delta_{ik}$
- $\overline{t} \in \mathbb{B}^{N_J}$ – Binary representation of the secret scalar t, with t_i representing the i-th bit of t in little-endian order, i.e., $t = \sum t_i 2^i$, for $i \in \mathbb{N}_{N_J}$

The bits vector $\overline{b}$ is constructed by concatenating $\overline{k}$ and $\overline{t}$, followed by a single 0.

$$\overline{b} = \overline{k} \| \overline{t} \| (0)$$

3.1.2. Conditional Sum Accumulator Vectors

$$ACC_0 = S, \quad ACC_i = ACC_{i-1} + b_{i-1}P_{i-1}, \quad i = 1, \ldots, N-4$$

- The accumulator is initialized with the seed point S.
- The accumulator is updated at each index i based on the previous value and the product of b_{i-1} and P_{i-1}.

The resulting accumulator points are finally separated into x and y coordinates:

$$(\overline{acc}_x, \overline{acc}_y) = \text{unzip}(\overline{ACC})$$

3.1.3. Inner Product Accumulator Vector

$$acc_{ip_0} = 0, \quad acc_{ip_i} = acc_{ip_{i-1}} + b_{i-1}s_{i-1}, \quad i = 1, \ldots, N-4$$

- The accumulator is initialized with 0.
- The accumulator is updated at each index i based on the previous value and the product of b_{i-1} and s_{i-1}

3.1.4. Interpolation and Commitments

The resulting vectors are interpolated over $\mathbb{D}$ with random values $\{r_i\}$ appended as padding for the final entries. This padding helps obscure the resulting polynomial, even when committing to identical witness values.

$$b = \text{Interpolate}(\overline{b} \| (r_1, r_2, r_3))$$

$$\begin{aligned} acc_x &= \text{Interpolate}(\overline{acc}_x \| (r_4, r_5, r_6)) \\ acc_y &= \text{Interpolate}(\overline{acc}_y \| (r_7, r_8, r_9)) \\ acc_{ip} &= \text{Interpolate}(\overline{acc}_{ip} \| (r_{10}, r_{11}, r_{12})) \end{aligned}$$

Commit to the witness derived polynomials:

$$C_b = \text{PCS.Commit}(b)$$

$$\begin{aligned} C_{acc_{ip}} &= \text{PCS.Commit}(acc_{ip}) \\ C_{acc_x} &= \text{PCS.Commit}(acc_x) \\ C_{acc_y} &= \text{PCS.Commit}(acc_y) \end{aligned}$$

3.2. Constraints

Constraints are polynomials constructed to evaluate to zero when satisfied; a non-zero evaluation indicates a violation.

Note. When evaluating a polynomial f at $x = \omega^k \in \mathbb{D}$ for some $k \in \mathbb{N}$, $f(\omega x)$ gives the value of the polynomial at the next position in the evaluation domain ($\omega x = \omega^{k+1}$).

3.2.1. Inner Product

$$c_1(x) = \big(acc_{ip}(\omega x) - acc_{ip}(x) - b(x)s(x)\big)(x - \omega^{N-4})$$

This constraint ensures the inner product accumulator $acc_{ip}(x)$ is correctly updated, satisfying $acc_{ip}(\omega x) = acc_{ip}(x) + b(x)s(x)$.

The factor $(x - \omega^{N-4})$ ensures the constraint holds at all points including $x = \omega^{N-4}$, where $c_1(x)$ automatically vanishes.

3.2.2. Conditional Addition

$$\begin{aligned} c_2(x) = \Bigg(& b(x)\Big(\big(acc_x(x) - p_x(x)\big)^2\big(acc_x(x) + p_x(x) + acc_x(\omega x)\big) \\ & \quad - \big(p_y(x) - acc_y(x)\big)^2\Big) \\ & + \big(1 - b(x)\big)\big(acc_x(\omega x) - acc_x(x)\big)\Bigg) \times (x - \omega^{N-4}) \end{aligned}$$

$$
\begin{aligned}
c_3(x) = \Bigg(& b(x)\Big(\big(acc_x(x) - p_x(x)\big)\big(acc_y(\omega x) + acc_y(x)\big) \\
& - \big(p_y(x) - acc_y(x)\big)\big(acc_x(\omega x) - acc_x(x)\big)\Big) \\
& + \big(1 - b(x)\big)\big(acc_x(\omega x) - acc_x(x)\big)\Bigg) \times (x - \omega^{N-4})
\end{aligned}
$$

These constraints enforce correct elliptic curve addition for the x and y components, respectively, controlled by the Boolean variable $b(x)$:

- **When** $b(x) = 1$: $c_2(x)$ and $c_3(x)$ enforce the correct elliptic curve addition for the x and y components, respectively.
- **When** $b(x) = 0$: both constraints ensure the accumulator remains unchanged.

The factor $(x - \omega^{N-4})$ nullifies the constraint at $x = \omega^{N-4}$, where it does not apply.

3.2.3. Booleanity

$$c_3(x) = b(x)\big(1 - b(x)\big)$$

Ensures that the polynomial $b(x)$ acts as a Boolean variable, taking only values 0 or 1.

- **If** $b(x) = 0$ or $b(x) = 1$, then $c_3(x) = 0$.
- **If** $b(x)$ takes any value other than 0 or 1, $c_3(x)$ will be non-zero, violating the constraint.

3.2.4. Conditional Addition Boundary Given the seed point $S = (s_x, s_y)$ and the expected result delta from the seed point $R = (r_x, r_y)$, the constraints are:

$$
\begin{aligned}
c_5(x) &= \big(acc_x(x) - s_x\big)L_0(x) + \big(acc_x(x) - r_x - s_x\big)L_{N-4}(x) \\
c_6(x) &= \big(acc_y(x) - s_y\big)L_0(x) + \big(acc_y(x) - r_y - s_y\big)L_{N-4}(x)
\end{aligned}
$$

These constraints ensure the accumulator components take specific values at the conditional addition boundaries:

- **At** $x = \omega^0$: $L_0(x) = 1$ and $L_{N-4}(x) = 0$, enforcing $acc_x(\omega^0) = s_x$ and $acc_y(\omega^0) = s_y$.
- **At** $x = \omega^{N-4}$: $L_0(x) = 0$ and $L_{N-4}(x) = 1$, enforcing $acc_x(\omega^{N-4}) = r_x + s_x$ and $acc_y(\omega^{N-4}) = r_y + s_y$.

3.2.5. Inner Product Boundary

$$c_7(x) = acc_{ip}(x)L_0(x) + \big(acc_{ip}(x) - 1\big)L_{N-4}(x)$$

This constraint ensure the accumulator components take specific values at the conditional addition boundaries:

- **At** $x = \omega^0$: $L_0(x) = 1$ and $L_{N-4}(x) = 0$, enforcing $acc_{ip}(\omega^0) = 0$.
- **At** $x = \omega^{N-4}$: $L_0(x) = 0$ and $L_{N-4}(x) = 1$, enforcing $acc_{ip}(\omega^{N-4}) = 1$.

3.3. Constraints Aggregation

3.3.1. Aggregation Polynomial The protocol aggregates all constraints into a single polynomial for efficiency.

Using the Fiat-Shamir heuristic, sample the aggregation coefficients:

$$\{\alpha_i\}_{i=1}^{7} \leftarrow \mathrm{FS}(C_b, C_{acc_{ip}}, C_{acc_x}, C_{acc_y})$$

Construct the aggregated polynomial:

$$c(x) = \left(\sum_{i=1}^{7} \alpha_i c_i(x)\right) \cdot \prod_{k=1}^{3} \left(x - \omega^{N-k}\right)$$

The factor $\prod_{k=1}^{3} \left(x - \omega^{N-k}\right)$ ensures that $c(x)$ vanishes at the last three points of the domain, thereby enforcing the constraints across the entire evaluation domain, including the last three points where random evaluation values were used during the witness polynomials interpolation phase.

3.3.2. Quotient Polynomial The quotient polynomial is computed as:

$$q(x) = \frac{c(x)}{x^N - 1}$$

Dividing by $X^N - 1$ ensures that the aggregated constraints encoded in $c(x)$ are enforced consistently across the entire evaluation domain $\mathbb{D}$ while reducing the degree of the polynomial.

3.3.3. Quotient Polynomial Commitment and Challenge The prover commits to the quotient polynomial q:

$$C_q = \mathrm{PCS.Commit}(q)$$

The prover receives the evaluation point ζ in response:

$$\zeta \leftarrow \mathrm{FS}(C_q)$$

3.3.4. Relevant Polynomials Evaluation Evaluate the relevant polynomials at the sampled evaluation point ζ:

$$p_{x,\zeta} = p_x(\zeta)$$

$$p_{y,\zeta} = p_y(\zeta)$$
$$s_\zeta = s(\zeta)$$
$$b_\zeta = b(\zeta)$$
$$acc_{ip,\zeta} = acc_{ip}(\zeta)$$
$$acc_{x,\zeta} = acc_x(\zeta)$$
$$acc_{y,\zeta} = acc_y(\zeta)$$

3.3.5. Linearization Polynomial The linearization polynomials are constructed to enable the verifier to evaluate certain parts of the constraint polynomials at $\zeta\omega$ while independently reconstructing the evaluation of q at ζ using the "relevant polynomial evaluations" provided by the prover as part of the proof.

In particular we require these contributions just for the accumulators contraints.

Accumulator inner product (c_1) contribution:

$$l_1(x) = (\zeta - \omega^{N-4})acc_{ip}(x)$$

Conditional addition accumulators ($c_{2,3}$) contributions:

$$l_2(x) = (\zeta - \omega^{N-4})\big(b_\zeta(acc_{x,\zeta} - p_{x,\zeta})^2 acc_x(x) + (1 - b_\zeta)acc_y(x)\big)$$

$$l_3(x) = (\zeta - \omega^{N-4})\Big(\big(b_\zeta(acc_{y,\zeta} - p_{y,\zeta}) + 1 - b_\zeta\big)acc_x(x) + b_\zeta(acc_{x,\zeta} - p_{x,\zeta})acc_y(x)\Big)$$

Linearized constraints are aggregated using $\{\alpha_i\}$ coefficients and evaluated at $\zeta\omega$:

$$l(x) = \sum_{i=1}^{3} \alpha_i l_i(x)$$
$$l_{\zeta\omega} = l(\zeta\omega)$$

3.3.6. Sample Aggregation Coefficients Sample the aggregation coefficients $\{\nu_i\}$ using the Fiat-Shamir heuristic and compute the aggregate polynomial agg:

$$\{\nu_i\}_{i=1}^{8} \leftarrow \mathrm{FS}(p_{x,\zeta}, p_{y,\zeta}, s_\zeta, b_\zeta, acc_{ip,\zeta}, acc_{x,\zeta}, acc_{y,\zeta}, l_{\zeta\omega})$$

Construct the aggregate polynomial:

$$agg(x) = \nu_1 p_x(x) + \nu_2 p_y(x) + \nu_3 s(x) + \nu_4 b(x) + \nu_5 acc_{ip}(x) + \nu_6 acc_x(x) + \nu_7 acc_y(x) + \nu_8 q(x)$$

3.3.7. Proof Construction Open the aggregate polynomial agg at ζ and the linearization polynomial l at $\zeta\omega$:

$$\Pi_\zeta = \text{PCS.Open}(agg, \zeta)$$

$$\Pi_{\zeta\omega} = \text{PCS.Open}(l, \zeta\omega)$$

Construct the proof as follows:

$$\Pi = (C_b, C_{acc_{ip}}, C_{acc_x}, C_{acc_y}, p_{x,\zeta}, p_{y,\zeta}, s_\zeta, b_\zeta, acc_{ip,\zeta}, acc_{x,\zeta}, acc_{y,\zeta}, C_q, l_{\zeta\omega}, \Pi_\zeta, \Pi_{\zeta\omega})$$

4. Verifier

4.1. Inputs

Commitments to the ring public keys and the selector, prepared during the pre-processing phase:

$$(C_{p_x}, C_{p_y}, C_s)$$

The claimed accumulation result, allegedly $PK_k + tH$ for some k and t known to the prover. This is the primary element to be assessed:

$$R = (r_x, r_y)$$

Proof which contains all the necessary commitments, evaluations, and openings needed for the verifier to perform the validation checks:

$$\Pi = (C_b, C_{acc_{ip}}, C_{acc_x}, C_{acc_y}, p_{x,\zeta}, p_{y,\zeta}, s_\zeta, b_\zeta, ip_\zeta, ac_{x,\zeta}, ac_{y,\zeta}, C_q, l_{\zeta\omega}, \Pi_\zeta, \Pi_{\zeta\omega})$$

4.2. Verification

4.2.1. Fiat-Shamir Challenges Recovery of aggregation coefficients and evaluation point:

$$\{\alpha_i\}_{i=1}^{7} \leftarrow \text{FS}(C_b, C_{ip}, C_{acc_x}, C_{acc_y})$$

$$\zeta \leftarrow \text{FS}(C_q)$$

$$\{\nu_i\}_{i=1}^{8} \leftarrow \text{FS}(p_{x,\zeta}, p_{y,\zeta}, s_\zeta, b_\zeta, acc_{ip,\zeta}, acc_{x,\zeta}, acc_{y,\zeta}, l_{\zeta\omega})$$

4.2.2. Contributions to the Constraints Evaluated at ζ The following expressions represent the contributions to the constraint polynomials evaluated at the point ζ:

$$\tilde{c}_{1,\zeta} = -(acc_{ip,\zeta} + b_\zeta s_\zeta)(\zeta - \omega^{N-4})$$

$$\tilde{c}_{2,\zeta} = \left\{b_\zeta \left[(acc_{x,\zeta} - p_{x,\zeta})^2(acc_{x,\zeta} + p_{x,\zeta}) - (p_{y,\zeta} - acc_{y,\zeta})^2\right] - (1 - b_\zeta)acc_{y,\zeta}\right\}(\zeta - \omega^{N-4})$$

$$\tilde{c}_{3,\zeta} = \{b_\zeta\left[(acc_{x,\zeta} - p_{x,\zeta})acc_{y,\zeta} + (p_{y,\zeta} - acc_{y,\zeta})acc_{x,\zeta}\right] - (1 - b_\zeta)acc_{x,\zeta}\}(\zeta - \omega^{N-4})$$

$$
\begin{aligned}
c_4 &= b_\zeta(1 - b_\zeta) \\
c_5 &= (acc_{x,\zeta} - s_x)L_0(\zeta) + (acc_{x,\zeta} - r_x - s_x)L_{N-4}(\zeta) \\
c_6 &= (acc_{y,\zeta} - s_y)L_0(\zeta) + (acc_{y,\zeta} - r_y - s_y)L_{N-4}(\zeta) \\
c_7 &= acc_{ip,\zeta}L_0(\zeta) + (acc_{ip,\zeta} - 1)L_{N-4}(\zeta)
\end{aligned}
$$

Note: The tilde ($\tilde{}$) above the first three polynomials indicates that these are only partial contributions, representing the components evaluated at ζ. The components evaluated at $\zeta\omega$ are added later by the linearization aggregated polynomial found within the proof ($l_{\zeta\omega}$).

4.2.3. Evaluation of the Quotient Polynomial at ζ Aggregate the contributions along with the linearization polynomial evaluated at $\zeta\omega$ to compute the evaluation of the quotient polynomial at ζ:

$$
q_\zeta = \frac{\left(\sum_{i=1}^{7} \alpha_i c_i + l_{\zeta\omega}\right) \prod_{k=1}^{3} (\zeta - \omega^{N-k})}{\zeta^N - 1}
$$

Compute the aggregate commitment C_{agg} using the aggregation coefficients ν_i:

$$
C_{agg} = \nu_1 C_{p_x} + \nu_2 C_{p_y} + \nu_3 C_s + \nu_4 C_b + \nu_5 C_{acc_{ip}} + \nu_6 C_{acc_x} + \nu_7 C_{acc_y} + \nu_8 C_q
$$

Compute the aggregate evaluation agg_ζ using the same coefficients:

$$
agg_\zeta = \nu_1 p_{x,\zeta} + \nu_2 p_{y,\zeta} + \nu_3 s_\zeta + \nu_4 b_\zeta + \nu_5 acc_{ip,\zeta} + \nu_6 acc_{x,\zeta} + \nu_7 acc_{y,\zeta} + \nu_8 q_\zeta
$$

Verify the aggregate polynomial opening at ζ using Π_ζ:

$$
\text{PCS.Verify}(C_{agg}, \zeta, agg_\zeta, \Pi_\zeta)
$$

4.2.4. Evaluation of the Linearization Polynomial at $\zeta\omega$ Compute the individual linearization polynomial commitments:

$$
C_{l_1} = (\zeta - \omega^{N-4})C_{acc_{ip}}
$$

$$
\begin{aligned}
C_{l_2} &= (\zeta - \omega^{N-4}) \left(b_\zeta(acc_{x,\zeta} - p_{x,\zeta})^2 C_{acc_x} + (1 - b_\zeta)C_{acc_y}\right) \\
C_{l_3} &= (\zeta - \omega^{N-4}) \left((b_\zeta(acc_{y,\zeta} - p_{y,\zeta}) + 1 - b_\zeta\right) C_{acc_x} + b_\zeta(acc_{x,\zeta} - p_{x,\zeta})C_{acc_y}\big)
\end{aligned}
$$

Aggregate the linearization polynomial commitments using $\{\alpha_i\}$ coefficients.

$$
C_l = \sum_{i=1}^{3} \alpha_i C_{l_i}
$$

Verify the aggregate linearization polynomial opening at $\zeta\omega$ using $\Pi_{\zeta\omega}$:

$$
\text{PCS.Verify}(C_l, \zeta\omega, l_{\zeta\omega}, \Pi_{\zeta\omega})
$$

5. Acknowledgements

This specification is primarily derived from Sergey Vasilyev's original writeup and reference implementation, as cited in the references.

6. References

- These notes on hackmd: `https://hackmd.io/@davxy/r1SVPqQc0`.
- Sergey Vasilyev original writeup: `https://hackmd.io/ulW5nFFpTwClHsD0kusJAA`
- W3F reference implementation: `https://github.com/w3f/ring-proof`