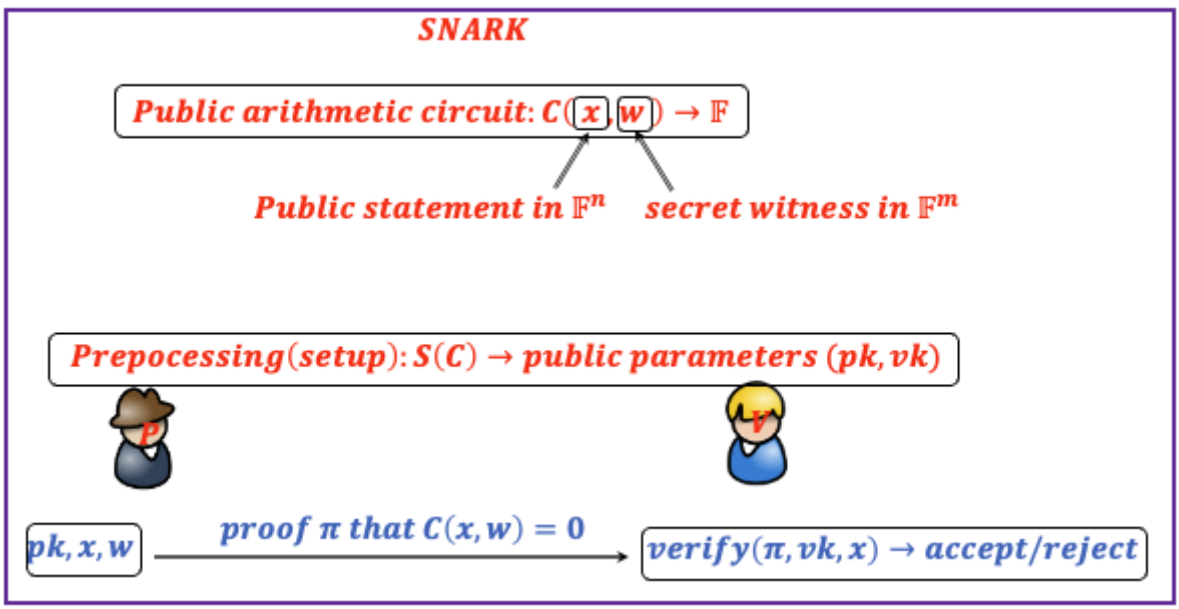
# Part II The technical architecture of the Aleo blockchain (Marlin and Varuna)

## Marlin and Varuna

### 1. Overview of Zero-Knowledge Proof Technology

Zero-knowledge proofs are widely applied in blockchain applications and have become crucial tools for privacy and scalability. Among these, zk-SNARKs (succinct non-interactive arguments of knowledge) are favored for their short proof size and efficient verification. In practice, the efficiency of verification is significantly determined by the proof's length/computational complexity, which is associated with the application's circuit size. A typical SNARK protocol includes the following processes:




In general, the length of a SNARK proof has a sublinear relationship with the witness (the prover's private input). Verifiers need to review the instance and circuit information during proof verification, making the verification complexity related to the instance length and the size of the circuit. Achieving succinct proofs and efficient verification ultimately boils down to the preprocessing setup, specifically the setup process illustrated above.

- **Proof Conciseness:** To achieve shorter proofs, where the proof length is logarithmically related to the circuit gate size $len(\pi) = O_\lambda(\log(|C|))$

- **Verification Efficiency:** If verifiers do not have time to review circuit-related information, the setup algorithm generates a verification key (vk), usually a short summary of the circuit.

In the specific schemes of zk-SNARKs widely used today, such as Groth16, Sonic, and Plonk, as well as optimized implementation protocols like Halo2 and Marlin, all these specific zero-knowledge proof schemes have a preprocessing setup process. During this process, some public parameters (common reference string, CRS) are secretly generated. These parameters will be involved in the subsequent proof generation and verification processes. The term "secret" here refers to the issue of "toxic waste" introduced during the generation of these public parameters. Anyone who can access this toxic waste could potentially forge proofs, making the management of this toxic waste especially critical. After the CRS is generated, it needs to be immediately destroyed to prevent misuse. Nowadays, MPC algorithm can avoid these problems.

Currently, there are three main types of preprocessing setup:

- `trusted setup per circuit` : A representative scheme is Groth16. In this type of setup, each circuit has a corresponding toxic waste random number, which is crucial and must be permanently discarded after one use. Otherwise, if a prover gains access to this toxic waste, they could forge proofs. Therefore, any change in the circuit requires the regeneration of both the proving and verification parameters (pp and vp). Generating the Common Reference String (CRS) consumes a significant amount of computational and human resources, making this approach more suited for static, unchanging application scenarios. For instance, Zcash implements this scheme to facilitate privacy-preserving transactions based on the UTXO model.
  - $S(circuit; r)$
- `universal (updatable) setup` : Sonic and Plonk are typical examples of this setup, with Plonk being an optimized scheme of Sonic. In such setup, the random number r/toxic waste is only involved in the generation of global parameters and is independent of the circuit; this algorithm runs only once. The algorithms for generating proving and verification parameters (pp and vp) are deterministic, depending on the global parameters (gp) and the circuit. For different circuits, it is only necessary to run this deterministic algorithm each time. Therefore, a universal SRS (Structured Reference String) can support a broader and more diverse range of application scenarios. Although a universal setup is more expensive and generates larger proofs compared to a circuit-specific setup, its ability to accommodate more complex and diverse application scenarios makes it more popular among users.
  - $S = (S_{init}, S_{index}) : S_{init}(\lambda, r) \rightarrow gp, S_{index}(gp, Circuit) \rightarrow (pp, vp)$

- `transparent setup` : This setup does not contain any secret randomness, meaning there is no trusted setup process. The proving and verification parameters (pp and vp) it generates are only related to the circuit. A representative protocol is Bulletproofs, which are used for range proofs.

| zk-protocol | size of proof | verifier time | setup | Post-quantum? |
|---|---|---|---|---|
| Groth16 | $\approx 200Bytes$<br>$O_\lambda(1)$ | $\approx 1.5ms$<br>$O_\lambda(1)$ | trusted per circuit | no |
| Plonk/Marlin | $\approx 400Bytes$<br>$O_\lambda(1)$ | $\approx 3ms$<br>$O_\lambda(1)$ | Universal trusted setup | no |
| Bulletproofs | $\approx 1.5KB$<br>$O_\lambda(\log(\|C\|))$ | $\approx 3sec$<br>$O_\lambda(\|C\|)$ | transparent | no |
| STARK | $\approx 100KB$<br>$O_\lambda(\log^2(\|C\|))$ | $\approx 10ms$<br>$O_\lambda(\log^2(\|C\|))$ | transparent | yes |

Aleo uses the Varuna scheme, which is an improved version of the Marlin scheme. The main difference between Varuna and Marlin is that Marlin's arithmetic language only allows R1CS, while Varuna can support both R1CS and custom gates and lookups(custom constraints lookup arguments), such as Plonkish. This means that in the arithmetic representation of circuits, Marlin is limited to three matrices, whereas Varuna can utilize a greater number of matrices, offering better extensibility. Beyond addition and multiplication circuits, Varuna can support various types of arithmetic circuits with corresponding distinct construction methods, referred to as generalized R1CS.

| construction | argument size over BN-256 (bytes) | argument size over BLS12-381 (bytes) |
|---|---|---|
| Sonic [Mal+19] | 1152 | 1472 |
| MARLIN [this work] | 704 | 880 |
| Groth16 [Gro16] | 128 | 192 |

| zkSNARK construction | | sizes | | | time complexity | | | |
|---|---|---|---|---|---|---|---|---|
| | | \|ipk\| | \|ivk\| | $\|\pi\|$ | generator | indexer | prover | verifier |
| Sonic [Mal+19] | $\mathbb{G}_1$ | $8m$ | — | 20 | 8 f-MSM($M$) | 4 v-MSM($3m$) | 273 v-MSM($m$) | 7 pairings |
| | $\mathbb{G}_2$ | $8m$ | 3 | — | 8 f-MSM($M$) | — | — | |
| | $\mathbb{F}_q$ | — | — | 16 | — | $O(m \log m)$ | $O(m \log m)$ | $O(\|\mathbb{x}\| + \log m)$ |
| MARLIN [this work] | $\mathbb{G}_1$ | $4m$ | 2 | 13 | 1 f-MSM($3M$) | 12 v-MSM($m$) | 22 v-MSM($m$) | 2 pairings |
| | $\mathbb{G}_2$ | — | 2 | — | — | — | — | |
| | $\mathbb{F}_q$ | — | — | 8 | — | $O(m \log m)$ | $O(m \log m)$ | $O(\|\mathbb{x}\| + \log m)$ |
| Groth16 [Gro16] | $\mathbb{G}_1$ | $4n$ | $O(\|\mathbb{x}\|)$ | 2 | 4 f-MSM($n$) | | 4 v-MSM($n$) | 1 v-MSM($\|\mathbb{x}\|$) |
| | $\mathbb{G}_2$ | $n$ | $O(1)$ | 1 | 1 f-MSM($n$) | N/A | 1 v-MSM($n$) | 3 pairings |
| | $\mathbb{F}_q$ | — | — | — | $O(m + n \log n)$ | | $O(m + n \log n)$ | — |

$n$: number of multiplication gates in the circuit
$m$: total number of (addition or multiplication) gates in the circuit
$M$: maximum supported circuit size (maximum number of addition and multiplication gates)

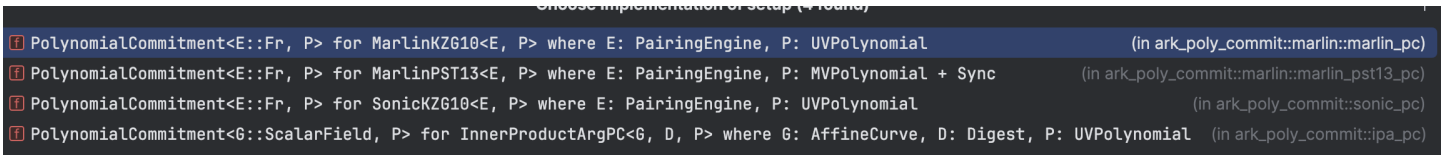
We will give a detailed introduction to the Marlin scheme.

## 2. Overview of the Marlin Protocol

The Marlin protocol encompasses three main processes: updatable setup, Prove, and Verify.

**(1) Updatable Setup mainly includes two algorithms:**

- Setup Algorithm: A deterministic algorithm used for generating SRS, based on different polynomial commitment schemes and security assumptions. This algorithm, which needs to run only once and is independent of the circuit, can be found in Marlin's GitHub repository. For detailed information, refer to marlin-poly-commit.




- Index Algorithm: Based on the SRS generated above and different circuits, it generates a provingKey and a verifierKey. This deterministic algorithm can be run by anyone.

**(2) Prove: The process where the prover generates a proof using the provingKey, instance, and witness.**

https://github.com/arkworks-rs/marlin/blob/master/src/ahp/prover.rs

**(3) Verify: The process where the verifier uses the verifierKey and instance to verify the proof.**

https://github.com/arkworks-rs/marlin/blob/master/src/ahp/verifier.rs

Constructing an effective SNARK for a general circuit typically requires two main components:

- **Polynomial Commitment Scheme**
- **IOP (Interactive Oracle Proof):** Essentially a multi-round interactive protocol where the verifier sends a challenge in each round, and the prover responds with an oracle. Through this oracle, the verifier can make queries/inquiries.

The Marlin protocol discusses various polynomial commitment schemes under different security models, including:

- Opening multiple polynomials with the same degree bound at a single point
- Opening multiple polynomials with multiple degree bounds at a single point
- Opening multiple polynomials with multiple degree bounds at multiple points

Let's dive into a few specific schemes.

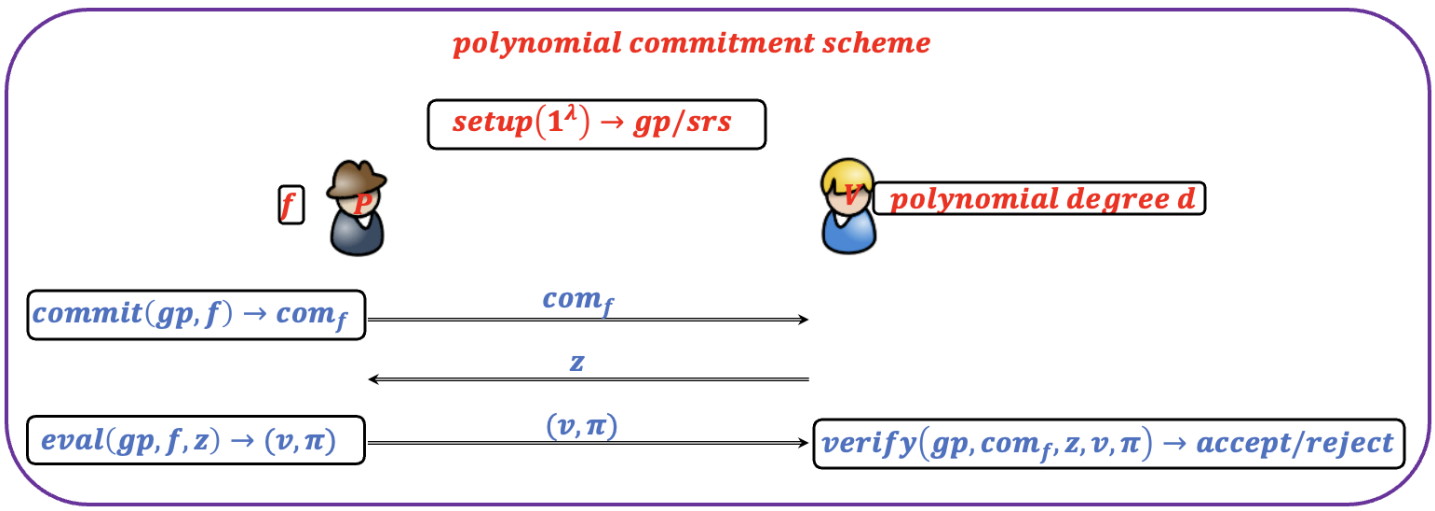
# 3. Polynomial Commitment Schemes

## 3.1. Definition of Polynomial Commitments

A polynomial commitment allows a prover to generate a commitment to a polynomial and later open this commitment at a specific point, producing a proof that the opened value matches the committed polynomial. The most straightforward method would be to send the polynomial to the verifier for direct validity verification, but this approach scales linearly with the polynomial's degree( $d$ ), which is impractical for polynomials of billion-degree ( $d$ ) magnitudes in real applications. Instead, the goal is to construct proofs that are both succinct and efficiently verifiable.

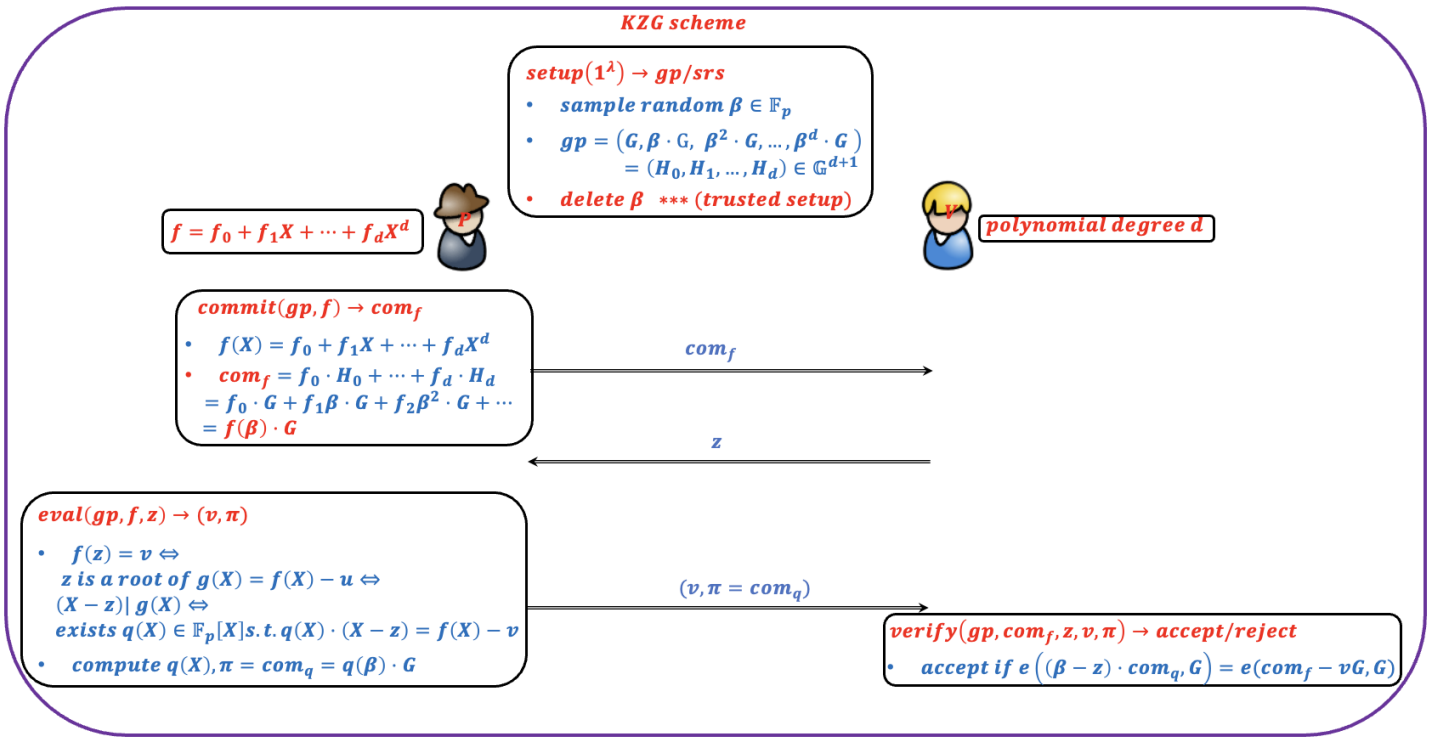
The polynomial commitment process involves the following algorithms:

- $KeyGen/setup(1^\lambda) \rightarrow gp$ : Generates a public key, public parameters, or a common reference string. This is represented directly by $gp$ (global parameters).
- $commit(gp, f) \rightarrow com_f$ : Given the $gp$ (global parameters) and a polynomial, the prover can compute a commitment to the polynomial.
- $eval(gp, f, z) \rightarrow (v, \pi)$ : The prover computes the value $v$ of the polynomial at a point $z$ , along with the proof $\pi$ .
- $verify(gp, com_f, z, v, \pi) \rightarrow accept\ or\ reject$ : The verifier checks the correctness of the value $v$ computed by the polynomial at point $z$ .




## 3.2. KZG Polynomial Commitment Scheme

This section details the specific algorithms of the KZG polynomial commitment scheme.




- `setup`: $\mathbb{F}_p$ is a finite field of prime order $p$, and $\mathbb{G}$ is a cyclic group with a generator $G$. In this algorithm, $\beta$ represents the toxic waste that must be permanently destroyed. In practice, Multi-Party Computation (MPC) is introduced to generate this value. In actual schemes, particularly in universal setup zero-knowledge proof schemes, this algorithm functions as $S_{init}$, participating only in the generation of $gp$ and being independent of the circuit. The size of $gp$ is linearly related to $d$, where $d$ is typically on the order of billions, implying that $gp$ is very large.
- `commit`: Making a commitment to a polynomial involves calculating the value of the polynomial at $\beta$, even though the value of $\beta$ is not known. However, this calculation can be performed using $gp$.
- `eval`: The computational effort required for the prover to calculate the quotient polynomial $q(X)$ is substantial, as the degree $d$ of this polynomial is typically very large, often on the order of billions. After calculating the quotient polynomial, a commitment to this polynomial must be made. This means that the computational effort for proof is very large. How to optimize this in practice will be discussed later.
- `verify`: The verification equation $(\beta - z) \cdot com_q = com_f - v \cdot G$ involves the toxic waste $\beta$, but the verifier cannot know the information about $\beta$. Therefore, it is necessary to introduce bilinear mapping/pairing for the final verification step.

## 3.3. MarlinKZG10

In the marlin-poly-commit repository, several polynomial commitment schemes are implemented, including:

- Inner-product-argument Polynomial Commitment (PC)
- Marlin variant of the Kate-Zaverucha-Goldberg (KZG) Polynomial Commitment: marlinKZG10
- Sonic/AuroraLight variant of the KZG Polynomial Commitment
- Marlin variant of the Papamanthou-Shi-Tamassia multivariate Polynomial Commitment: Marlin-PST13

This article will not detail every scheme but will focus on some of the representative ones.

Marlin requires polynomial commitment schemes to satisfy:

- Extractability: A stronger notion than binding.
- Efficiency:
  - For the prover: The time to create a polynomial commitment should be short, implying the need to reduce the polynomial's degree *d* (low-degree); the verification of proofs should be efficient, meaning the size of the generated proof/quotient polynomial commitment should be proportional to the polynomial's degree, not to the highest degree *D*.
  - For the verifier: the commitment size, proof size, time to verify an opening must be independent of the claimed degrees for the polynomials
- hiding：commitments and proofs of evaluation reveal no information about the committed polynomial beyond the publicly stated degree bound and the evaluation itself.

To meet the functional and security requirements specified in the Marlin project, Marlin employs an extended Kate-Zaverucha-Goldberg (KZG) polynomial commitment scheme, achieving extractability within the algebraic group model (AGM).

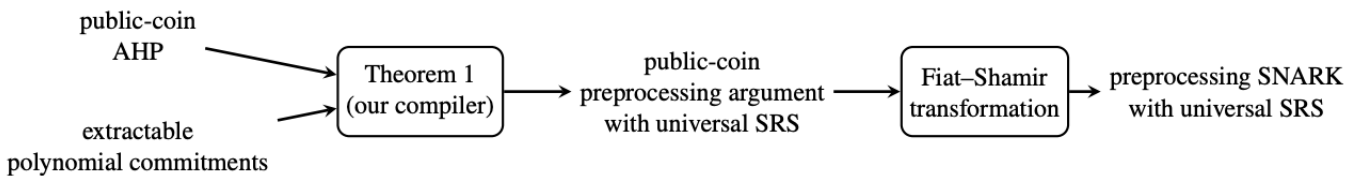



**Figure 3:** Diagram of our methodology to construct preprocessing SNARGs with universal SRS.

**(1) Setup**

In the code, g = G, gamma_g = $\gamma G$, where $\gamma$ is introduced to provide the hiding feature for commitments. If the hiding feature is not considered, then the components marked in blue in the scheme are not necessary.

**Setup.** On input a security parameter $\lambda$ (in unary), and a maximum degree bound $D \in \mathbb{N}$, $\mathsf{PC_s}$.Setup samples public parameters $(\mathsf{ck}, \mathsf{rk})$ as follows. Sample a bilinear group $\langle \mathsf{group} \rangle \leftarrow \mathsf{SampleGrp}(1^\lambda)$, and parse $\langle \mathsf{group} \rangle$ as a tuple $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, q, G, H, e)$. Sample random elements $\beta, \gamma \in \mathbb{F}_q$. Then compute the vector

$$\mathbf{\Sigma} := \begin{pmatrix} G & \beta G & \beta^2 G & \dots & \beta^D G \\ \gamma G & \gamma\beta G & \gamma\beta^2 G & \dots & \gamma\beta^D G \end{pmatrix} \in \mathbb{G}_1^{2D+2} \ .$$

Set $\mathsf{ck} := (\langle \mathsf{group} \rangle, \mathbf{\Sigma})$ and $\mathsf{rk} := (D, \langle \mathsf{group} \rangle, \gamma G, \beta H)$, and then output the public parameters $(\mathsf{ck}, \mathsf{rk})$. These public parameters will support polynomials over the field $\mathbb{F}_q$ of degree at most $D$.

### (2) Commit

- The process of committing to a univariate polynomial is as follows:
  Normally, the commitment to a polynomial $p(X) = a_0 + a_1 X + \ldots + a_t X^t$ would be $commit(p(X)) = p(\beta) \cdot G$. However, this method does not offer the hiding property since it reveals the value of the polynomial at $\beta$. To ensure hiding, a random blinding polynomial, $\overline{p(X)}$, is added to the calculation. The final commitment is $commit(p(X)) = p(\beta) \cdot G + \overline{p(\beta)} \cdot \gamma G$.
- For multiple univariate polynomials, represented as $[p_i]_{i=1}^n$, to achieve hiding, it's not necessary to add a random blinding polynomial $\overline{p_i(X)}$ to every univariate polynomial. It depends on the randomly generated vector $\vec{w} = [w_i]_{i=1}^n$.

**Commit.** On input $\mathsf{ck}$, univariate polynomials $\boldsymbol{p} := [p_i]_{i=1}^n$ over $\mathbb{F}_q$, and randomness $\boldsymbol{\omega} := [\omega_i]_{i=1}^n$, $\mathsf{PC_s}$.Commit outputs commitments $\boldsymbol{c} := [c_i]_{i=1}^n$ that are computed as follows. If for any $p_i \in \boldsymbol{p}$, $\deg(p_i) > D$, abort. Else, for each $i \in [n]$, if $\omega_i$ is not $\perp$, then obtain random univariate polynomial $\bar{p}_i$ of degree $\deg(p_i)$ from $\omega_i$, otherwise $\bar{p}_i$ is set to be a zero polynomial. For each $i \in [n]$, output $c_i := p_i(\beta)G + \gamma\bar{p}_i(\beta)G$. Note that because $p_i$ and $\bar{p}_i$ have degree at most $D$, the above terms are linear combinations of terms in $\mathsf{ck}$.

### (3) Eval -> $(\pi)$

The open algorithm in the code, specifically the 'compute_witness_polynomial && open_with_witness_polynomial', essentially generates a commitment/proof for the quotient polynomial/witness polynomial.

- For a univariate polynomial:
  - The witness polynomial $w(X) = \dfrac{p(X) - p(z)}{X - z}$, and its proof is $w(\beta) \cdot G$.
  - To maintain the hiding feature, $\overline{w(X)} = \dfrac{\overline{p(X)} - \overline{p(z)}}{X - z}$, and its proof is $\overline{w(\beta)} \cdot \gamma G$. Since the prover introduces a new polynomial, the value of this new polynomial at point $z$ cannot be obtained through SRS, thus the generated proof includes an extra term $\overline{v} = \overline{p}(z)$.

- For multiple univariate polynomials, a linear combination is used with a value $\xi$ randomly chosen by the verifier.

**Open.** On input ck, univariate polynomials $\boldsymbol{p} := [p_i]_{i=1}^{n}$ over $\mathbb{F}_q$, evaluation point $z \in \mathbb{F}_q$, opening challenge $\xi \in \mathbb{F}_q$, and randomness $\boldsymbol{\omega} := [\omega_i]_{i=1}^{n}$, which is the same randomness used for $\mathsf{PC_s}$.Commit, $\mathsf{PC_s}$.Open outputs an evaluation proof $\pi \in \mathbb{G}_1$ that is computed as follows. If for any $p_i \in \boldsymbol{p}$, $\deg(p_i) > D$, abort. For each $i \in [n]$, if $\omega_i$ is not $\perp$, then obtain random univariate polynomial $\bar{p}_i$ of degree $\deg(p_i)$ from $\omega_i$, otherwise $\bar{p}_i$ is set to be a zero polynomial. Then compute the linear combination of polynomials $p(X) := \sum_{i=1}^{n} \xi^i p_i(X)$ and $\bar{p}(X) := \sum_{i=1}^{n} \xi^i \bar{p}_i(X)$. Compute witness polynomials $w(X) := \frac{p(X)-p(z)}{X-z}$ and $\bar{w}(X) := \frac{\bar{p}(X)-\bar{p}(z)}{X-z}$. Set $\mathsf{w} := w(\beta)G + \gamma\bar{w}(\beta)G \in \mathbb{G}_1$ and $\bar{v} := \bar{p}(z) \in \mathbb{F}_q$. The evaluation proof is $\pi := (\mathsf{w}, \bar{v})$.

#### (4) Verify

- For a single variable polynomial, the verification algorithm is consistent with KZG and is not further derived here.
- For multiple univariate polynomials, the verification method is a batch check, which essentially introduces $\xi$ for a linear combination.

**Check.** On input rk, commitments $\boldsymbol{c} := [c_i]_{i=1}^{n}$, evaluation point $z \in \mathbb{F}_q$, alleged evaluations $\boldsymbol{v} := [v_i]_{i=1}^{n}$, evaluation proof $\pi := (\mathsf{w}, \bar{v})$, and randomness $\xi \in \mathbb{F}_q$, $\mathsf{PC_s}$.Check proceeds as follows. Compute the linear combination $C := \sum_{i=1}^{n} \xi^i c_i$ Then compute the linear combination of evaluations $v := \sum_{i=1}^{n} \xi^i v_i$, and check the evaluation proof via the equality $e(C - vG - \gamma\bar{v}G, H) = e(\mathsf{w}, \beta H - zH)$.

### 3.4. Marlin-PST13

The KZG polynomial commitment scheme is not only applicable to univariate polynomials but also to multivariate polynomials and supports batch proofing. The PST13 scheme, utilizing KZG for multivariate $k - variate$ polynomial commitments, achieves batch proofs.

- Assume the verifier has commitment values $com_{f_1}, \ldots, com_{f_n}$ .
- The prover aims to prove $f_i(u_{i,j} = v_{i,j}), i \in [1, n], j \in [1, m]$ , and can generate a batch proof $\pi$ , which requires only a single group element.

## 4. Proof Properties of Polynomial Commitments

Combining polynomials commitment with Interactive Oracle Proofs (IOP), they are frequently used in protocols like zeroTest, sumcheck, and productCheck. The proof system in Marlin mainly utilizes the zeroTest and sumcheck protocols. Before introducing these protocols, it's necessary to discuss the concept of vanishing polynomials.

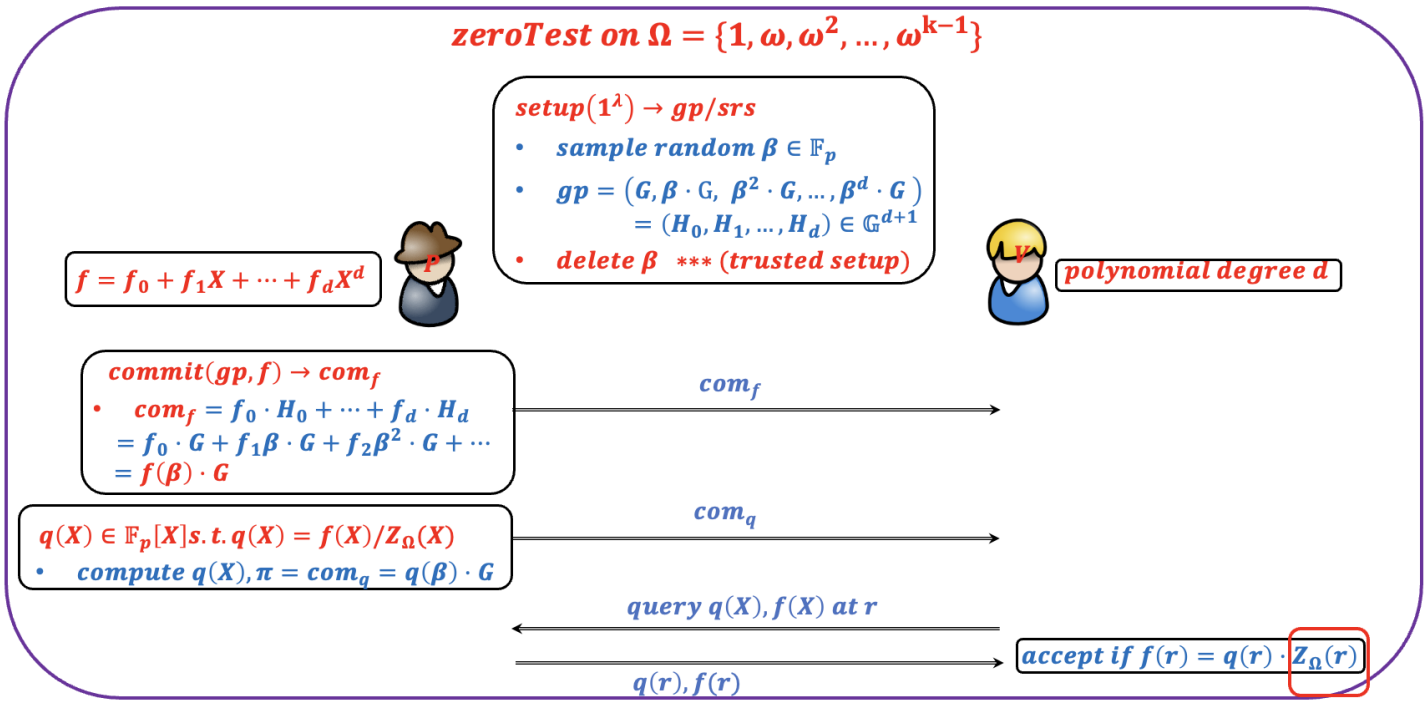
### 4.1. Vanishing Polynomial

In a prime field $\mathbb{F}_p$, let $\Omega$ be a subgroup of the field with order $k$. A vanish polynomial $Z_\Omega(X)$ takes the value 0 for all elements of $\Omega$. If the subgroup is a cyclic group generated by $\omega$, and if $\Omega = \{1, \omega, \ldots, \omega^{k-1}\}$, the vanishing polynomial $Z_\Omega(X) = (X-1)(X-\omega)(X-\omega^2)\ldots(X-\omega^{k-1}) = X^k - 1$. The definition of a vanishing polynomial plays a significant role here. Instead of calculating the product of $k$ polynomials, one only needs to compute $X^k - 1$. For $z \in \mathbb{F}_p$, computing $Z_\Omega(r)$ requires $\leq 2\log_2 k$ field operations.

- If $\Omega$ is a multiplicative subgroup, then $Z_\Omega(X) = x^k - 1$.
- If $\Omega$ is a $\xi - coset$ of some multiplicative subgroup $\Omega_0$, then $\Omega = \xi\Omega_0, Z_\Omega(X) = X^k - \xi^k$

## 4.2. ZeroTest on $\Omega$

The prover has a polynomial $f(X)$ and needs to prove that $f(X)$ equals 0 over the entire multiplicative subgroup $\Omega$, which essentially means proving that $f(X)$ can be divided by the vanishing polynomial of the multiplicative subgroup $Z_\Omega(X) = X^k - 1$




For this protocol:

- verifier time: $O(log(k))$ and two poly queries (but can be done in one)
- Prover time: dominated by the time to compute $q(X)$ and then commit to $q(X)$.

## 4.3. SumCheck Protocol

The prover needs to prove to the verifier that the value of a multivariate polynomial $g(b_1, \ldots, b_\ell)$ is $C_1$ . The specific method is as follows:

# Sum-Check Protocol [LFKN90]

- **Start**: P sends claimed answer $C_1$. The protocol must check that:

$$C_1 = \sum_{b_1 \in \{0,1\}} \sum_{b_2 \in \{0,1\}} \ldots \sum_{b_\ell \in \{0,1\}} g(b_1, \ldots, b_\ell).$$

- **Round 1**: P sends **univariate** polynomial $s_1(X_1)$ claimed to equal:

$$H_1(X_1) := \sum_{b_2 \in \{0,1\}} \ldots \sum_{b_\ell \in \{0,1\}} g(X_1, b_2, \ldots, b_\ell)$$

- V checks that $C_1 = s_1(0) + s_1(1)$.
- V picks $r_1$ at random from $\mathbb{F}$ and sends $r_1$ to P.
- **Round 2**: They recursively check that $s_1(r_1) = H_1(r_1)$.
  i.e., that $s_1(r_1) = \sum_{b_2 \in \{0,1\}} \ldots \sum_{b_\ell \in \{0,1\}} g(r_1, b_2, \ldots, b_\ell)$.

- **Round $\ell$ (Final round):** P sends univariate polynomial $s_\ell(X_\ell)$ claimed to equal

$$H_\ell := g(r_1, \ldots, r_{\ell-1}, X_\ell).$$

- V checks that $s_{\ell-1}(r_{\ell-1}) = s_\ell(0) + s_\ell(1)$.
- V picks $r_\ell$ at random, and needs to check that $s_\ell(r_\ell) = g(r_1, \ldots, r_\ell)$.
  - No need for more rounds. V can perform this check with one oracle query.

# Soundness

- If P does not send the prescribed messages, then V rejects with probability at least 1- $\frac{\ell \cdot d}{|\mathbb{F}|}$, where $d$ is the maximum degree of $g$ in any variable.
- E.g. $|\mathbb{F}| \approx 2^{128}, d = 3, \ell = 60$.
  - Then soundness error is at most $3 \cdot 60 / 2^{128} = 2^{-120}$.

## 5. References

- Preprocessing zkSNARKs with Universal and Updatable SRS
- Aleo Varuna Code
- Marlin Code
- PST13
- The Evolution of Polynomial Commitments
- A Survey of Elliptic Curves for Proof Systems
- Aurora: Transparent Succinct Arguments for R1CS

---

**About AlphaSwap**

AlphaSwap (previously AleoSwap) offers private, secure, and smooth trading experience on the Aleo blockchain.

**Official Website** | **Twitter** | **Discord** | **Telegram**