Zircuit

Polynomial Commitment Schemes for Zero-Knowledge Proof Systems

A Hands-on Workshop

Prerequisites

- **Workshop Repository** : https://github.com/zircuit-labs/devcon

- **SageMath:** https://doc.sagemath.org/html/en/installation/index.html

Introduction

- Polynomial commitments are crucial cryptographic tools used in many zero-knowledge proof systems.
- They allow a prover to commit to a polynomial while enabling efficient evaluation proofs for arbitrary points.
- They provide succinctness to many of the widely adopted SNARK/STARK zero-knowledge proof systems.

Given a potentially large-degree polynomial $f(x)$, how can we ***efficiently*** check that $f(z) = c$ for a given z ?

Introduction

Today, we will explore three distinct classes of polynomial commitment schemes:

- **Pairing-Based Commitments:** Based on bilinear pairings over elliptic curves.
- **Discrete Logarithm-Based Commitments:** Relies on the hardness of the discrete logarithm problem.
- **Hash Function-Based Commitments:** Based on cryptographic hash functions and error-correcting codes.

Introduction

More specifically, we will delve into the following protocols:

- **Pairing-Based:** The Kate, Zaverucha, Goldberg's polynomial commitment scheme **(KZG)**
- **Discrete Logarithm-Based:** The "Inner Product Argument" **(IPA)** by Bootle *et al.*
- **Hash Function-Based:** The "Fast Reed-Solomon IOP of Proximity" **(FRI)** by Ben-Sasson's *et al.*

Introduction

- In this workshop, we will compare these three commitment schemes by examining their mathematical constructions, properties, and trade-offs.
- We will engage in hands-on proof-of-concept implementation of some parts of their core logic.
- We will try to keep the presentation as informal as possible; however, a certain familiarity with finite fields, elliptic curves, and hash functions is recommended.

Introduction

This workshop ***will not*** focus on:

- The required mathematical background;
- Protocol's security properties and proofs;
- Non-interactivity (Fiat-Shamir heuristic);
- Complexity analysis and possible optimizations.

Notation

- All scalars and polynomials' coefficients will be defined over a finite field $\mathbb{F}_r$ (the scalars' field).
- We will denote the polynomial we want to commit to as $f(x) \in \mathbb{F}_r[x]$.
- The degree of $f(x)$ is less than n.
- We denote a commitment to $f(x)$ with C_f.
- Evaluation proofs are computed for the statement $f(z) = c$ where $z, c \in \mathbb{F}_r$.

Notation

Unless expressly stated, we will denote:

- Scalars with small roman letters (e.g., s, a, b, bf)
- Elliptic curve points and vectors with capital roman letters (e.g., P, G, A, B)
- Random values with small Greek letters (e.g., α, λ, ρ)
- The i-th coefficient of a polynomial $f(x)$ with f_i .

Polynomial Commitment Schemes

Polynomial commitment schemes allow a prover to commit to a certain polynomial $f(x)$ and later prove evaluations of the polynomial at arbitrary points efficiently.

They consist of four main primitives:

- **Setup**: Generates all protocols' parameters;
- **Commit**: Computes a commitment for $f(x)$:

$$f(x) \to C_f$$

- **Prove**: Computes an evaluation proof for $f(x)$ in z:

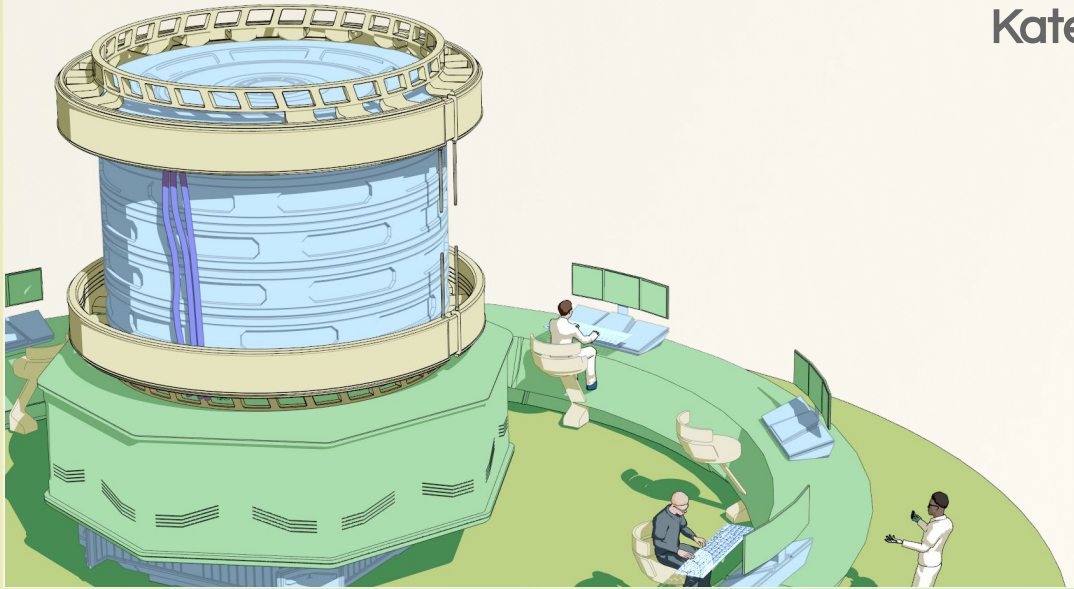
$$f(x), z \to \pi_{f,z}$$

- **Verify**: Checks an evaluation proof:

$$C_f, z, c, \pi_{f,z} \to \text{If } f(z) = c, \text{ Accept. Otherwise, Reject.}$$

The KZG Polynomial Commitment Scheme

Kate, Zaverucha, Goldberg (2010)



KZG Polynomial Commitment

- The KZG Polynomial Commitment scheme is a pairing-based scheme that uses bilinear pairings on elliptic curves.

- The scheme requires a ***trusted setup*** , a set of parameters generated using a trapdoor that must remain secret.

- However, evaluation proofs are of constant size, and verification takes constant time with respect to the degree of the committed polynomials.

KZG Polynomial Commitment - Pairing

- We define KZG over an elliptic curve E defined over a finite field $\mathbb{F}_p$.
- The curve possesses a bilinear pairing e, a map satisfying:

$$e(a \cdot G_1, b \cdot G_2) = e(G_1, G_2)^{ab} = v$$

  for certain given generators G_1, G_2. Computing either a or b from v is considered hard.

- Note that a, b do not necessarily belong to $\mathbb{F}_p$ but usually lie in a different scalar field $\mathbb{F}_r$.

KZG Polynomial Commitment - Setup

- A trusted setup generates a Structured Reference String (SRS) as:

$$SRS = \{G_1, s \cdot G_1, s^2 \cdot G_1, \ldots, s^{n-1} \cdot G_1, G_2, s \cdot G_2\}$$

- $s \in \mathbb{F}_r$ should remain secret; otherwise, the entire security of the scheme is at risk.

- MPC protocols for SRS generation can mitigate the risk of leaking s.

KZG Polynomial Commitment - Commit

- The commitment to a polynomial

$$f(x) = \sum_{i=0}^{n-1} f_i x^i \quad \text{with } f_i \in \mathbb{F}_r$$

is computed as:

$$C_f = \sum_{i=0}^{n-1} f_i \cdot (s^i \cdot G_1)$$

where the $\{s^i \cdot G_1\}_i$ are taken from the SRS.

KZG Polynomial Commitment - Prove

- To prove that $f(z) = c$ for a certain $z \in \mathbb{F}_r$, the Prover computes:

$$g(x) = \frac{f(x) - c}{x - z}$$

- $g(x)$ is a polynomial in $\mathbb{F}_r[x]$ if and only if $f(z) = c$.

- The Prover commits to $g(x)$ as

$$C_g = \sum_{i=0}^{n-2} g_i \cdot (s^i \cdot G_1)$$

  and sends C_g to the Verifier.

KZG Polynomial Commitment - Verify

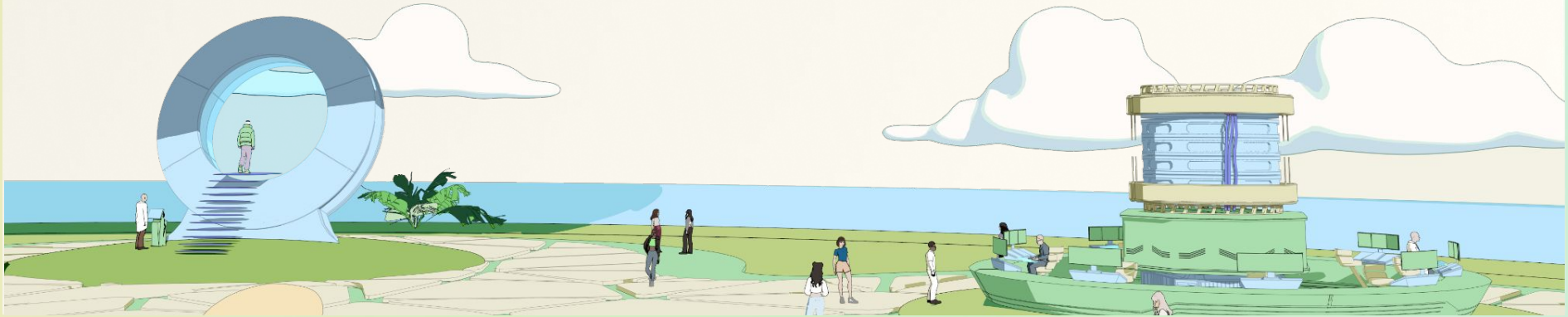
- The verifier checks if:

$$e(C_f, G_2) \stackrel{?}{=} e(C_g, s \cdot G_2 - z \cdot G_2) \cdot e(G_1, G_2)^c$$

- Why?

$$e\left(\sum_i f_i \cdot s^i G_1, G_2\right) = e\left(\sum_i \frac{f_i \cdot s^i - c}{s - z} \cdot G_1, sG_2 - zG_2\right) \cdot e(G_1, G_2)^c$$

Coding time!



KZG Polynomial Commitment - Summary

(Setup) $$SRS = G_1, s \cdot G_1, s^2 \cdot G_1, \ldots, s^{n-1} \cdot G_1, G_2, s \cdot G_2$$

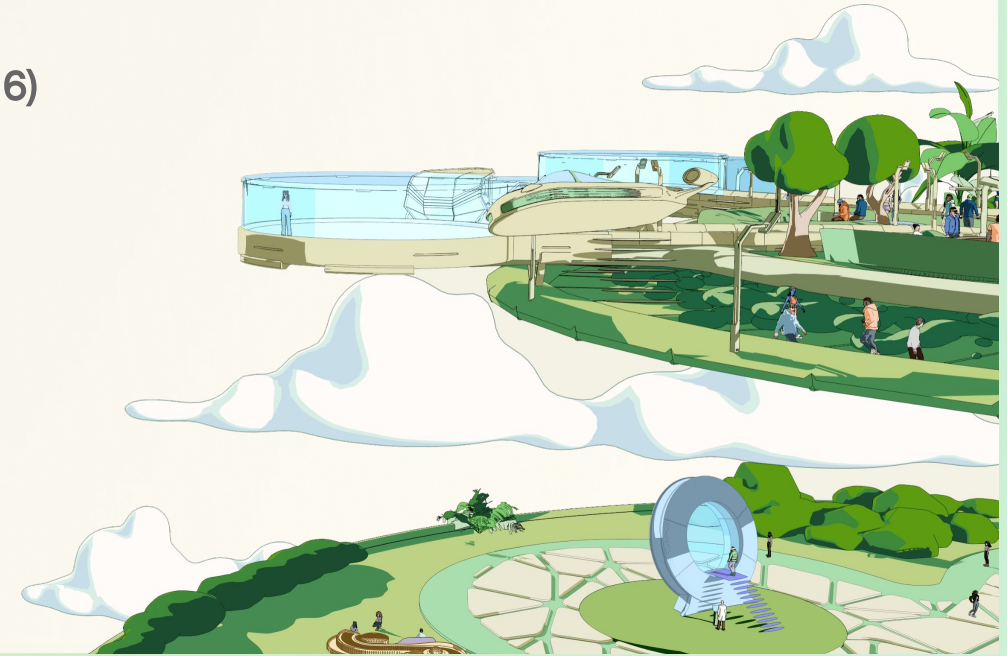
(Commit) $$f(x) = \sum_{i=0}^{n-1} f_i x^i \longrightarrow C_f = \sum_{i=0}^{n-1} f_i \cdot (s^i \cdot G_1)$$

(Prove) $$f(z) = c \longrightarrow g(x) = \frac{f(x) - c}{x - z} \in \mathbb{F}_r[x] \longrightarrow C_g = \sum_{i=0}^{n-2} g_i \cdot (s^i G_1)$$

(Verify) $$e(C_f, G_2) = e(C_g, s \cdot G_2 - z \cdot G_2) \cdot e(G_1, G_2)^c$$

The Inner Product Argument (IPA)

Bootle, Cerulli, Chaidos, Groth, Petit (2016)



Inner Product Argument

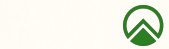
- The Inner Product Argument (**IPA**) is a discrete logarithm-based polynomial commitment scheme relying on Pedersen's commitments.

- The original construction is from *"Efficient Zero-Knowledge Arguments for Arithmetic Circuits in the Discrete Log Setting"* by Bootle et *al.*, and shows *succinctly* that:

$$\langle \vec{a}, \vec{b} \rangle = \sum_i a_i b_i = c$$

- Unlike KZG, it does not require a trusted setup and can be instantiated over any group where the discrete logarithm is hard.

- Here, we describe an optimized construction based on Bünz et *al.*'s *"Bulletproofs"* paper, as further improved by the `halo2` team.



Inner Product Argument - Overview

The IPA iteratively verifies that a commitment to $f(x)$ is consistent with respect to an evaluation challenge z. Its main steps are:

- ***Split & Fold:*** At each iteration, the Prover provides elements L and R that represent partial commitments. The Verifier replies by sending a challenge α that the Prover uses to form a new, smaller inner product of half the size.

- ***Final Check*** : After a logarithmic number of Split & Fold steps (i.e., $\log(n)$ steps for a polynomial of degree n), the inner product reduces to a single value, which the Verifier can check directly.

Inner Product Argument - Setup

- We work over an elliptic curve $E(\mathbb{F}_p)$ of prime order r.
- The degree bound n must be a power of 2, i.e., $n = 2^k$ for some $k \geq 1$.
- We randomly pick $n + 1$ generators of E:

$$G_0, \ldots, G_{n-1}, H$$

Inner Product Argument - Commit

- Given a polynomial

$$f(x) = \sum_{i=0}^{n-1} f_i x^i$$

the Prover picks a random blinding factor bf and computes a commitment to f as:

$$C_f = \left(\sum_{i=0}^{n-1} f_i \cdot G_i \right) + bf \cdot H$$

Inner Product Argument - Prove

- The Verifier picks and sends to the Prover a $z \in \mathbb{F}_r$. The Prover replies with the value $c = f(z)$.
- The Verifier randomly picks and sends to the Prover a random generator U of E.
- To show this value is correct with respect to C_f, the Prover and Verifier engage in the following interactive protocol.
- The protocol can be made non-interactive through the Fiat-Shamir heuristic.

Inner Product Argument - Prove

Initialize $A = [f_0, \ldots, f_{n-1}]$, $B = [1, z, \ldots, z^{n-1}]$, $G = [G_0, \ldots, G_{n-1}]$, $P = C_f$, $b = bf$ and proceed iteratively until $|A| = |B| = |G| = 1$:

- **(Prover & Verifier)** Let A^{Lo}, B^{Lo}, G^{Lo} and A^{Hi}, B^{Hi}, G^{Hi} be the first and second halves of A, B, G, respectively.

- **(Prover)** Picks random scalars $\lambda, \rho \in \mathbb{F}_r$, computes

$$L = \langle A^{Lo}, G^{Hi} \rangle + \langle A^{Lo}, B^{Hi} \rangle \cdot U + \lambda \cdot H$$
$$R = \langle A^{Hi}, G^{Lo} \rangle + \langle A^{Hi}, B^{Lo} \rangle \cdot U + \rho \cdot H$$

  and sends L, R to the Verifier.

Inner Product Argument - Prove

- **(Verifier)** Picks a random scalar $\alpha \in \mathbb{F}_r$ and sends it to the Prover.

- **(Prover & Verifier)** Compute:

$$G = [\alpha^{-1} \cdot G_i^{Lo} + \alpha \cdot G_i^{Hi} \mid G_i^{Lo} \in G^{Lo}, G_i^{Hi} \in G^{Hi}]$$

$$P = \alpha^2 \cdot L + P + \alpha^{-2} \cdot R$$

- **(Prover)** Computes:

$$A = [\alpha \cdot A_i^{Lo} + \alpha^{-1} \cdot A_i^{Hi} \mid A_i^{Lo} \in A^{Lo}, A_i^{Hi} \in A^{Hi}]$$

$$B = [\alpha^{-1} \cdot B_i^{Lo} + \alpha \cdot B_i^{Hi} \mid B_i^{Lo} \in B^{Lo}, B_i^{Hi} \in B^{Hi}]$$

$$b = \alpha^2 \cdot \lambda + b + \alpha^{-2} \cdot \rho$$

Inner Product Argument - Verify

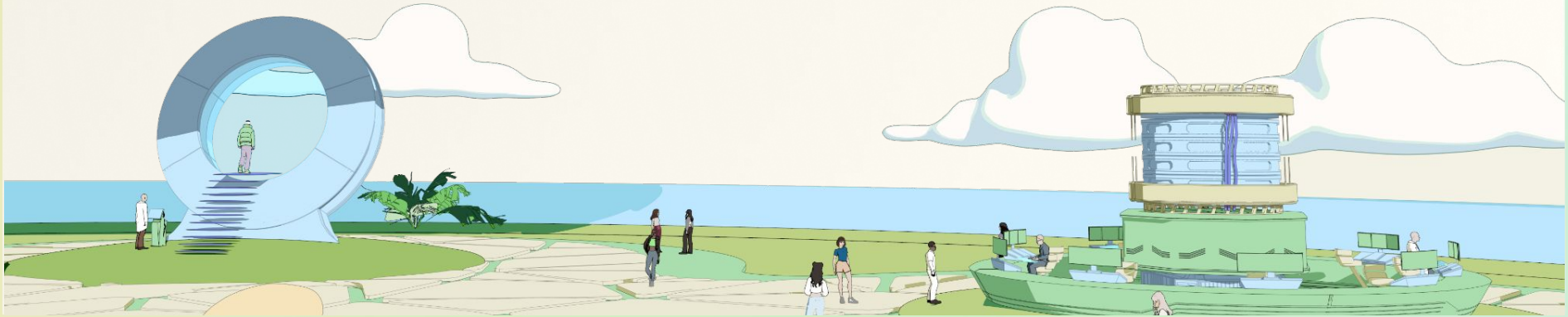
- **(Prover)** When $|A| = |B| = |G| = 1$, the Prover sends A, B and b to the Verifier.

- **(Verifier)** Checks if

$$P + c \cdot U \stackrel{?}{=} A \cdot G + b \cdot H + (A \cdot B) \cdot U$$

Note that the Verifier can compute the verification values P and G using their own round challenges α and the input C_f.

Coding time!



Inner Product Argument - Summary

$$A = [f_0, \ldots, f_{n-1}] \ , \ B = [1, z, \ldots, z^{n-1}] \ , \ G = [G_0, \ldots, G_{n-1}] \ , \ P = C_f \ , \ b = bf$$

(P)

$$\lambda, \rho \in \mathbb{F}_r \quad \rightarrow \quad \begin{aligned} L &= \langle A^{Lo}, G^{Hi} \rangle + \langle A^{Lo}, B^{Hi} \rangle \cdot U + \lambda \cdot H \\ R &= \langle A^{Hi}, G^{Lo} \rangle + \langle A^{Hi}, B^{Lo} \rangle \cdot U + \rho \cdot H \end{aligned}$$

(P&V)

$$\alpha \in \mathbb{F}_r \quad \rightarrow \quad \begin{aligned} G &= [\alpha^{-1} \cdot G_i^{Lo} + \alpha \cdot G_i^{Hi} \mid G_i^{Lo} \in G^{Lo}, G_i^{Hi} \in G^{Hi}] \\ P &= \alpha^2 \cdot L + P + \alpha^{-2} \cdot R \end{aligned}$$

(P)

$$\begin{aligned} A &= [\alpha \cdot A_i^{Lo} + \alpha^{-1} \cdot A_i^{Hi} \mid A_i^{Lo} \in A^{Lo}, A_i^{Hi} \in A^{Hi}] \\ B &= [\alpha^{-1} \cdot B_i^{Lo} + \alpha \cdot B_i^{Hi} \mid B_i^{Lo} \in B^{Lo}, B_i^{Hi} \in B^{Hi}] \\ b &= \alpha^2 \cdot \lambda + b + \alpha^{-2} \cdot \rho \end{aligned}$$

(V)

$$P + c \cdot U = A \cdot G + b \cdot H + (A \cdot B) \cdot U$$

Fast Reed-Solomon IOP of Proximity (FRI)

Ben-Sasson, Bentov, Horesh, Riabzev (2018)

FRI - Introduction

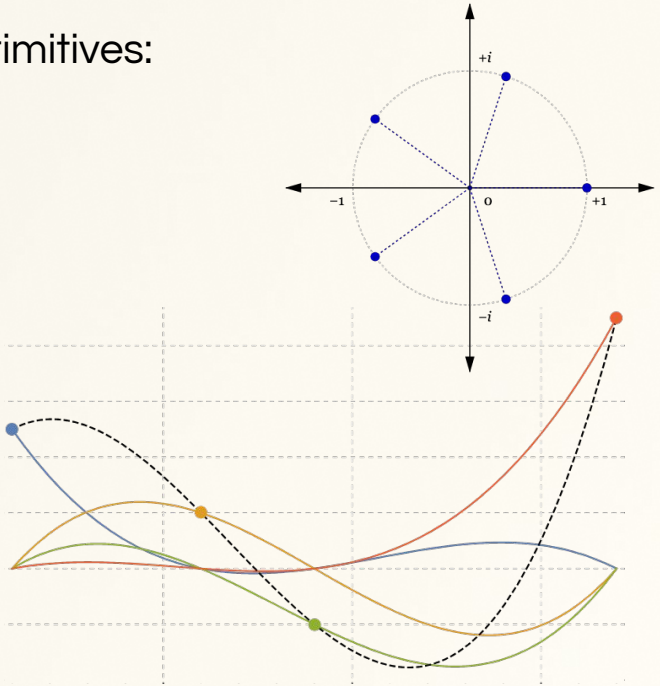
- FRI is an efficient Interactive Oracle Proof of Proximity (IOP).
- It is used to prove that a function is *close* to a low-degree polynomial.
- Does not require a trusted setup and is post-quantum resistant.
- FRI plays a crucial role in STARKs.

FRI - Background

FRI relies on different cryptographic concepts and primitives:

- Lagrange Interpolation;
- Roots of Unity;
- Reed-Solomon Codes;
- Merkle Trees.

We will quickly recap these key concepts.



FRI - Background: Lagrange Interpolation

- Lagrange interpolation is used to find a polynomial that passes through a given set of points.

- Given points $(x_0, y_0), (x_1, y_1), \ldots, (x_{N-1}, y_{N-1}) \in \mathbb{F}r \times \mathbb{F}r$ the Lagrange interpolation polynomial is:

$$L(x) = \sum_{i=0}^{N-1} y_i \prod_{\substack{0 \le j < N \\ j \neq i}} \frac{x - x_j}{x_i - x_j}$$

- **Key Observation** : If we interpolate $\{(x_i, f(x_i))\}_{i=0,\ldots,N-1}$ where $f(x) \in \mathbb{F}_r[x]$ is a polynomial with $\deg(f) < n \le N$, then $L(x) \equiv f(x)$ in $\mathbb{F}_r[x]$.

FRI - Background: Roots of Unity

- Any element in $\mathbb{F}_r$ that satisfies the equation $x^N = 1$ is said N-th root of unity.
- When $N|r-1$, the set of N-th roots of unity in $\mathbb{F}_r$ forms a cyclic group under multiplication: there exists an element $\omega \in \mathbb{F}r$ such that its powers $\{\omega^i\}_{i=0,\dots,N-1}$ generate all N-th roots of unity (*primitive N-th root of unity)*.
- In FRI, we're particularly interested in scalar fields $\mathbb{F}_r$ having a primitive 2^m-th root of unity with $m > 1$ small.
- **Key Observation:** If ω is a 2^m-th root of unity, then ω^2 is a primitive 2^{m-1}-th root of unity.

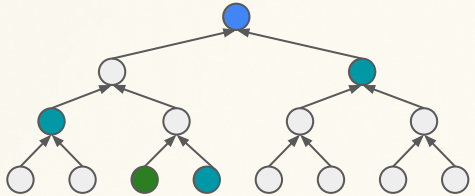
FRI - Background: Reed-Solomon Code

- Reed-Solomon (RS) codes are error-correcting codes.
- Every *codeword* of a RS code is the evaluation of **a** polynomial f of degree less than n over a certain ordered fixed domain D of size $N \geq n$.
- In FRI, D is $\{\omega^i\}_{i=0,\ldots,N-1}$, where ω is a fixed primitive N-th root of unity in $\mathbb{F}_r$. Thus, codewords are of the form

$$(f(1), f(\omega), \ldots, f(\omega^{N-1}))$$

- **Key Observation** : If f has low-degree (i.e., $\deg f < n$), then the Lagrange interpolation of its D-evaluations (or codeword) corresponds to a polynomial of degree less than n.

FRI - Background: Merkle Trees



A Merkle tree is used to commit to a large set of indexed values.

It consists of three main primitives:

- **Commit** : The Prover builds a binary tree where each leaf node represents an indexed input value, and each internal node is the hash of its children. He returns the root of such a tree.

- **Open**: To prove a leaf v is at a specific index i, the Prover provides a Merkle proof $\pi_{v,i}$, that is, a path of hashes from such leaf to the root.

- **Verify** : The Verifier checks the consistency of a Merkle proof $\pi_{v,i}$ with respect to a value v its tree index i, and root.

FRI - Overview

- The goal of FRI is to prove that a polynomial f has a bounded degree.
- To achieve this, the Prover iteratively commits to Merkle-tree evaluations of f and then *"folds"* the polynomial, halving its degree at each round.
- This process is interactive but can be made non-interactive using the Fiat-Shamir heuristic.
- After sufficient rounds, the degree bound on f can be verified by directly checking the degree of the final folded polynomial.

FRI - Overview

- A FRI *query* checks if a certain committed polynomial f has been folded consistently in a certain point.

- Multiple FRI queries to f will statistically ensure that it has indeed low degree.

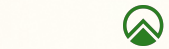
- It is conjectured that to reach a security level of λ bits, $\lceil \lambda / \log_2 N/n \rceil$ queries are required.

FRI - Setup

- We generate a finite field $\mathbb{F}_r$ such that $2^m | r-1$.
- We pick a primitive 2^m-th root of unity γ and square it until we obtain an N-th root of unity $\omega = \gamma^{2^m/N}$.
- We work over an RS code of length N, where codewords are evaluations over $D = \{\omega^i\}_i$ of polynomials of degree less than $n = 2^k \leq N$.
- We set a proof-verification trade-off output degree $2^d \leq n$.

FRI - Commitment Phase

- The main idea of the commitment phase is to iteratively:
  - ***Commit*** to all evaluations of f over a certain domain $D = \{\omega^i\}_i$ using Merkle Trees.
  - ***Split & Fold*** f using random challenges from the Verifier. In this step, the degree of f is halved, and the evaluation domain is updated to $D = \{\omega^{2i}\}_i$.
- When the folded f reaches the output degree 2^d, the Prover sends its evaluations in full to the Verifier, along with all the intermediate Merkle tree roots.



FRI - Commitment Phase

For round $0, \ldots, k-d-1$:

- **(Prover)** Computes the evaluations $f(\omega^i)_{i=0,\ldots,N-1}$ and sends to the Verifier the corresponding Merkle root.
- **(Verifier)** Picks a random folding challenge $\alpha \in \mathbb{F}_r$ and sends it to the Prover.
- **(Prover)** Splits $f(x) = f_{even}(x^2) + x \cdot f_{odd}(x^2)$. Folds it as $f^{fold}(x) = f_{even}(x) + \alpha \cdot f_{odd}(x)$. Computes the evaluations $f^{fold}(\omega^{2i})_{i=0,\ldots,N/2-1}$. If it is the last round, sends to the Verifier evaluations of f^{fold} in full; otherwise, sends the corresponding Merkle root.
- **(Prover)** Sets $f = f^{fold}$, $N = N/2$, $\omega = \omega^2$.

After receiving the final f^{fold} evaluations, the Verifier uses Lagrange interpolation to check that its degree is less than 2^d.

FRI - Query Phase

In a query, the Verifier asks the Prover to provide specific evaluations of the folded polynomials at each round, along with their corresponding Merkle proofs.

- **(Verifier)** Picks a random index $i \in [0, N/2 - 1]$ and sends it to the Prover.

- **(Prover)** For round $0, \ldots, k - d - 1$:
    - Retrieves the committed round values $f(\omega^i)$, $f(\omega^{N/2+i})$, $f^{fold}(\omega^{2i})$, computes their Merkle proofs $\pi_{f,i}$, $\pi_{f,N/2+i}$, $\pi_{f^{fold},i}$ with respect to each corresponding evaluation's tree root, and sends them to the Verifier.
    - Sets $N = N/2$ and $i = i \bmod N/2$.

FRI - Verify

Given a queried index i, the Verifier, for round $0, \ldots, k-d-1$:

- Checks the validity of the Merkle proofs $\pi_{f,i}$, $\pi_{f,N/2+i}$, $\pi_{ffold,i}$ with respect to the provided values $f(\omega^i)$, $f(\omega^{N/2+i})$, $f^{fold}(\omega^{2i})$ indexes i, $N/2+i$, i and roots, respectively.

- Checks if the points

$$A = (\omega^i, f(\omega^i))$$
$$B = (\omega^{N/2+i}, f(\omega^{N/2+i}))$$
$$C = (\alpha, f^{fold}(\omega^{2i}))$$

  are collinear.

- Sets $N = N/2$, $\omega = \omega^2$, $i = i \bmod N/2$.

FRI - Verify

Why are A, B and C collinear?

- From the definition of f^{fold} we have:

$$f^{fold}(\omega^{2i}) = \frac{f(\omega^i) + f(-\omega^i)}{2} + \alpha \cdot \frac{f(\omega^i) - f(-\omega^i)}{2\omega^i}$$
$$= 2^{-1} \cdot \left((1 + \alpha \cdot \omega^{-i}) \cdot f(\omega^i) + (1 - \alpha \cdot \omega^{-i}) \cdot f(-\omega^i)\right)$$

- Note that $\omega^N = 1 \Rightarrow \omega^{N/2} = -1 \Rightarrow -\omega^i = \omega^{N/2+i}$.

- If we interpolate $A = (\omega^i, f(\omega^i))$ and $B = (\omega^{N/2+i}, f(\omega^{N/2+i}))$, we get the polynomial

$$2^{-1} \cdot \left((1 + x \cdot \omega^{-i}) \cdot f(\omega^i) + (1 - x \cdot \omega^{-i}) \cdot f(\omega^{N/2+i})\right)$$

  which passes through $C = (\alpha, f^{fold}(\omega^{2i}))$ at $x = \alpha$.

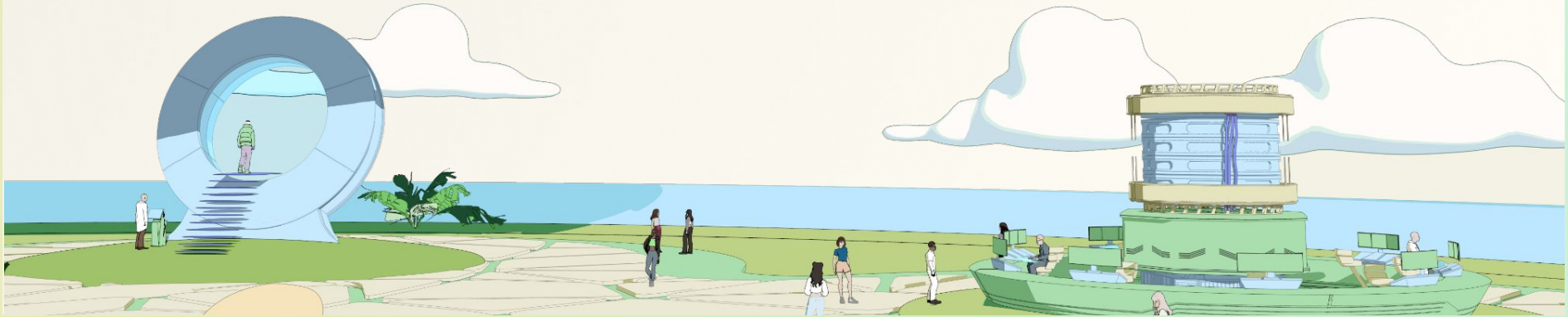
FRI - Polynomial Commitment Scheme

- How can the Verifier check that $f(z) = c$?
- If $f(x)$ is low-degree and $f(z) = c$, then $g(x) = \frac{f(x) - c}{x - z} \in \mathbb{F}_r[x]$ must be low-degree too.
- More precisely, if $f(x)$ is shown to have degree less than n, then $g(x)$ should result in a polynomial of degree less than $n - 1$.
- After Merke-tree committing to evaluations of $f(x)$, the Prover runs FRI over $g(x)$.
- The Verifier queries $g(x)$, but only in the first round asks the Prover to reveal and Merkle-tree open $f(x)$ at the same queried evaluation points ω^i in order to compute

$$g(\omega^i) = \frac{f(\omega^i) - c}{\omega^i - z}$$

and proceed to the next round.

Coding time!



Conclusions

- In this workshop, we explored different polynomial commitment schemes, a crucial component for achieving succinctness in modern zero-knowledge proof systems.
- More specifically, we described the KZG, IPA, and FRI polynomial commitment schemes, detailing their internal mechanics.
- We engaged in proof-of-concept implementations for each of these schemes.

Zircuit