KZG Polynomial Commitment Scheme on zk-SNARKs Construction and Its Implementation

Mohammad Ferry Husnil Arif

Faculty of Computer Science
Universitas Indonesia

mohammad.ferry@ui.ac.id

Final Project Defense
June 2025

Outline

What are Zero-Knowledge Proofs?

Definition

A cryptographic protocol where a prover convinces a verifier of a statement's truth **without revealing any additional information**

What are Zero-Knowledge Proofs?

Definition

A cryptographic protocol where a prover convinces a verifier of a statement's truth **without revealing any additional information**

Three Fundamental Properties:

- **Completeness**: Valid proofs always accepted

What are Zero-Knowledge Proofs?

Definition

A cryptographic protocol where a prover convinces a verifier of a statement's truth **without revealing any additional information**

Three Fundamental Properties:

- **Completeness**: Valid proofs always accepted
- **Soundness**: Invalid proofs always rejected

What are Zero-Knowledge Proofs?

Definition

A cryptographic protocol where a prover convinces a verifier of a statement's truth **without revealing any additional information**

Three Fundamental Properties:

- **Completeness**: Valid proofs always accepted
- **Soundness**: Invalid proofs always rejected
- **Zero-Knowledge**: Nothing revealed beyond truth

What are Zero-Knowledge Proofs?

Definition

A cryptographic protocol where a prover convinces a verifier of a statement's truth **without revealing any additional information**

Three Fundamental Properties:

- **Completeness**: Valid proofs always accepted
- **Soundness**: Invalid proofs always rejected
- **Zero-Knowledge**: Nothing revealed beyond truth

Applications:

- Privacy-preserving cryptocurrencies (Zcash)

What are Zero-Knowledge Proofs?

Definition

A cryptographic protocol where a prover convinces a verifier of a statement's truth **without revealing any additional information**

Three Fundamental Properties:

- **Completeness**: Valid proofs always accepted
- **Soundness**: Invalid proofs always rejected
- **Zero-Knowledge**: Nothing revealed beyond truth

Applications:

- Privacy-preserving cryptocurrencies (Zcash)
- Secure healthcare data management

What are Zero-Knowledge Proofs?

Definition

A cryptographic protocol where a prover convinces a verifier of a statement's truth **without revealing any additional information**

Three Fundamental Properties:

- **Completeness**: Valid proofs always accepted
- **Soundness**: Invalid proofs always rejected
- **Zero-Knowledge**: Nothing revealed beyond truth

Applications:

- Privacy-preserving cryptocurrencies (Zcash)
- Secure healthcare data management
- Confidential financial auditing

What are Zero-Knowledge Proofs?

Definition

A cryptographic protocol where a prover convinces a verifier of a statement's truth **without revealing any additional information**

Three Fundamental Properties:

- **Completeness**: Valid proofs always accepted
- **Soundness**: Invalid proofs always rejected
- **Zero-Knowledge**: Nothing revealed beyond truth

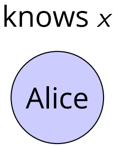
Applications:

- Privacy-preserving cryptocurrencies (Zcash)
- Secure healthcare data management
- Confidential financial auditing
- Legal verification frameworks

Simple Example: Hash Preimage

The Problem:

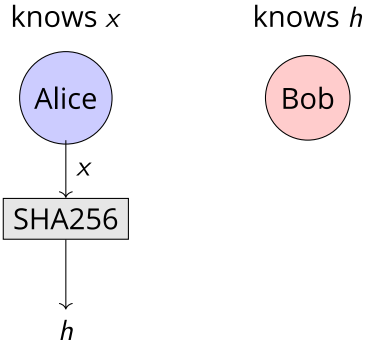
- Alice knows a password x



Simple Example: Hash Preimage

The Problem:

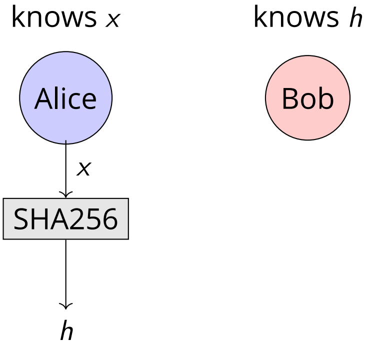
- Alice knows a password x
- Bob knows the hash $h = \text{SHA256}(x)$



Simple Example: Hash Preimage

The Problem:

- Alice knows a password x
- Bob knows the hash $h = \text{SHA256}(x)$
- Alice wants to prove she knows x **without revealing it**



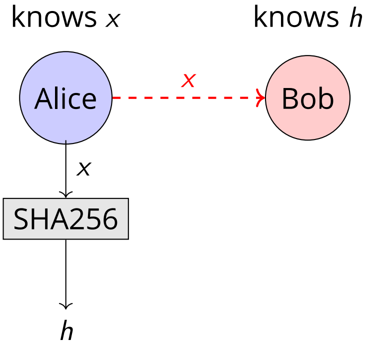
Simple Example: Hash Preimage

The Problem:

- Alice knows a password x
- Bob knows the hash $h = \text{SHA256}(x)$
- Alice wants to prove she knows x **without revealing it**

Without Zero-Knowledge:

✗ Alice sends x to Bob



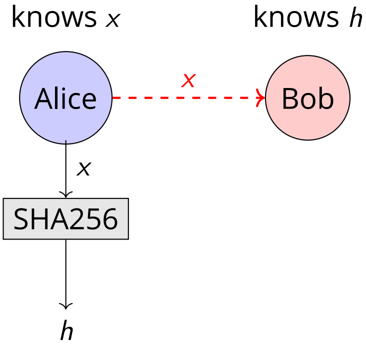
Simple Example: Hash Preimage

The Problem:

- Alice knows a password x
- Bob knows the hash $h = \text{SHA256}(x)$
- Alice wants to prove she knows x **without revealing it**

Without Zero-Knowledge:

- ✗ Alice sends x to Bob
- ✗ Bob learns the password!



Simple Example: Hash Preimage

The Problem:

- Alice knows a password x
- Bob knows the hash $h = \text{SHA256}(x)$
- Alice wants to prove she knows x **without revealing it**

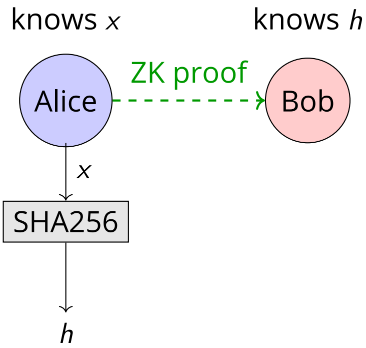
Without Zero-Knowledge:

✗ Alice sends x to Bob

✗ Bob learns the password!

With Zero-Knowledge:

✓ Alice proves knowledge of x



Simple Example: Hash Preimage

The Problem:

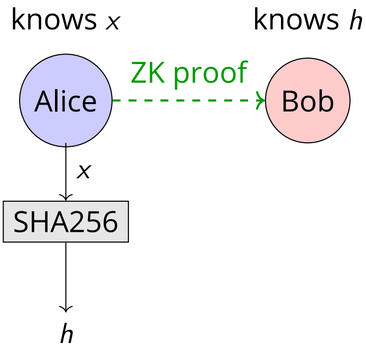
- Alice knows a password x
- Bob knows the hash $h = \text{SHA256}(x)$
- Alice wants to prove she knows x **without revealing it**

Without Zero-Knowledge:

- ✗ Alice sends x to Bob
- ✗ Bob learns the password!

With Zero-Knowledge:

- ✓ Alice proves knowledge of x
- ✓ Bob learns nothing about x



The Educational Challenge

High-level overviews

Lack mathematical rigor

The Educational Challenge

High-level overviews

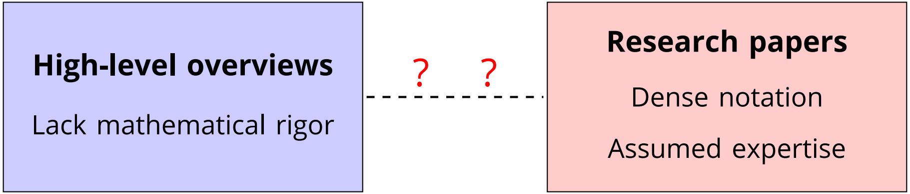
Lack mathematical rigor

Research papers

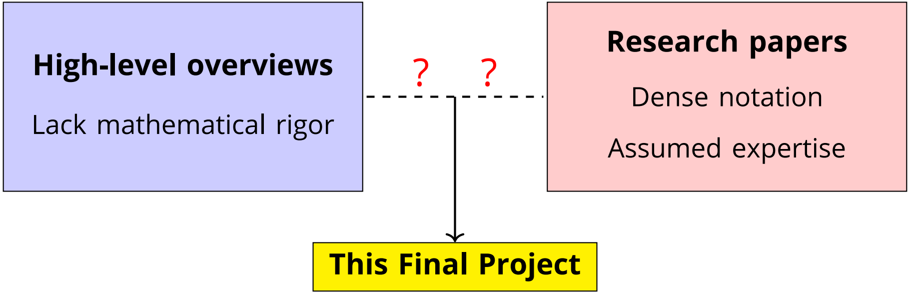
Dense notation

Assumed expertise

The Educational Challenge

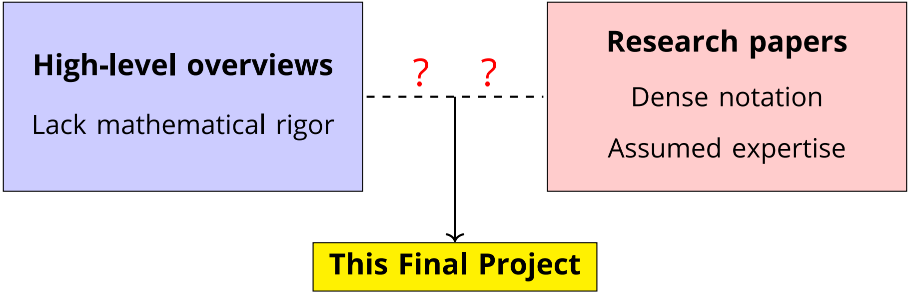


The Educational Challenge



Accessible exposition + Concrete examples + Working implementations

The Educational Challenge



Accessible exposition + Concrete examples + Working implementations

Why this matters: Growing importance in blockchain, privacy technologies, and secure computation

Final Project Objectives

1. **Make advanced cryptography accessible** to undergraduate students

Final Project Objectives

1. **Make advanced cryptography accessible** to undergraduate students
2. **Provide complete mathematical exposition** with concrete examples

Final Project Objectives

1. **Make advanced cryptography accessible** to undergraduate students
2. **Provide complete mathematical exposition** with concrete examples
3. **Deliver working SageMath implementations** for hands-on learning

Final Project Objectives

1. **Make advanced cryptography accessible** to undergraduate students
2. **Provide complete mathematical exposition** with concrete examples
3. **Deliver working SageMath implementations** for hands-on learning
4. **Bridge theory to practice** in cryptographic education

Final Project Objectives

1. **Make advanced cryptography accessible** to undergraduate students
2. **Provide complete mathematical exposition** with concrete examples
3. **Deliver working SageMath implementations** for hands-on learning
4. **Bridge theory to practice** in cryptographic education

Core Focus

KZG polynomial commitment scheme and its application in two prominent zk-SNARKs protocols: Marlin and Plonk

From Theory to Practice: zk-SNARKs

From Theory to Practice: zk-SNARKs

From Theory to Practice: zk-SNARKs

What makes zk-SNARKs special:

- **S**uccinct: Proofs are tiny (few hundred bytes)

From Theory to Practice: zk-SNARKs

What makes zk-SNARKs special:

- **S**uccinct: Proofs are tiny (few hundred bytes)
- **N**on-interactive: No back-and-forth communication needed

From Theory to Practice: zk-SNARKs

What makes zk-SNARKs special:

- **S**uccinct: Proofs are tiny (few hundred bytes)
- **N**on-interactive: No back-and-forth communication needed
- **AR**gument of **K**nowledge: Prover must know the witness

The zk-SNARKs Ecosystem

Framework	Frontend	Language	Proof System
Arkworks	Self-contained	Rust	Groth16, Marlin, GM17, Plonk
Gnark	Self-contained	Go	Groth16, Plonk (KZG, FRI)
Hyrax	None	Python	Hyrax
LEGOSnark	None	C++	Brakedown-like
LibSNARK	xJsnark	Java, C++	Groth16, Pinocchio, GGPR
Zokrates	Self-contained	Zokrates DSL	Groth16, GM17, Marlin, Nova
Mirage	None	Java	Pinocchio-like
PySNARK	Self-contained	Python	Groth16
SnarkJS	Circom	JavaScript, Circom DSL	Groth16, Plonk (via WASM)
Rapidsnark	Circom	JavaScript, Circom DSL	Groth16
Spartan	None	Rust	Spartan
Aurora (libiop)	None	C++	Aurora
Fractal (libiop)	None	C++	Fractal
Virgo	None	Python	Virgo
Noir	Self-contained	Rust (Noir DSL)	Any ACIR-compatible system
Dusk-PLONK	None	Rust	PLONK
Halo2	None (Rust API)	Rust	PLONK-like

The zk-SNARKs Ecosystem

Framework	Frontend	Language	Proof System
Arkworks	Self-contained	Rust	Groth16, Marlin, GM17, Plonk
Gnark	Self-contained	Go	Groth16, Plonk (KZG, FRI)
Hyrax	None	Python	Hyrax
LEGOSnark	None	C++	Brakedown-like
LibSNARK	xJsnark	Java, C++	Groth16, Pinocchio, GGPR
Zokrates	Self-contained	Zokrates DSL	Groth16, GM17, Marlin, Nova
Mirage	None	Java	Pinocchio-like
PySNARK	Self-contained	Python	Groth16
SnarkJS	Circom	JavaScript, Circom DSL	Groth16, Plonk (via WASM)
Rapidsnark	Circom	JavaScript, Circom DSL	Groth16
Spartan	None	Rust	Spartan
Aurora (libiop)	None	C++	Aurora
Fractal (libiop)	None	C++	Fractal
Virgo	None	Python	Virgo
Noir	Self-contained	Rust (Noir DSL)	Any ACIR-compatible system
Dusk-PLONK	None	Rust	PLONK
Halo2	None (Rust API)	Rust	PLONK-like

Adapted from “Zero-Knowledge Proof Frameworks: A Survey” by Sheybani et al. (2025)

Performance Comparison

Protocol	Proof Size	Prover	Verifier	Setup
Groth16	$2\mathbb{G}_1 + 1\mathbb{G}_2$	$O(n \log n)$	$O(\lvert \mathbb{x} \rvert)$	Circuit-specific
Marlin	$8\mathbb{F}_q + 13\mathbb{G}_1$	$O(n \log n)$	$O(\lvert \mathbb{x} \rvert + \log n)$	Universal
Plonk	$6\mathbb{F}_q + 9\mathbb{G}_1$	$O(n \log n)$	$O(\lvert \mathbb{x} \rvert + \log n)$	Universal

Performance Comparison

Protocol	Proof Size	Prover	Verifier	Setup
Groth16	$2\mathbb{G}_1 + 1\mathbb{G}_2$	$O(n \log n)$	$O(\lvert\mathbb{x}\rvert)$	Circuit-specific
Marlin	$8\mathbb{F}_q + 13\mathbb{G}_1$	$O(n \log n)$	$O(\lvert\mathbb{x}\rvert + \log n)$	Universal
Plonk	$6\mathbb{F}_q + 9\mathbb{G}_1$	$O(n \log n)$	$O(\lvert\mathbb{x}\rvert + \log n)$	Universal

Typical element sizes:

- $\mathbb{F}_q$ element: 32 bytes
- $\mathbb{G}_1$ element: 32 bytes (compressed)
- $\mathbb{G}_2$ element: 64 bytes (compressed)

Performance Comparison

Protocol	Proof Size	Prover	Verifier	Setup
Groth16	$2\mathbb{G}_1 + 1\mathbb{G}_2$	$O(n \log n)$	$O(\|\mathbb{x}\|)$	Circuit-specific
Marlin	$8\mathbb{F}_q + 13\mathbb{G}_1$	$O(n \log n)$	$O(\|\mathbb{x}\| + \log n)$	Universal
Plonk	$6\mathbb{F}_q + 9\mathbb{G}_1$	$O(n \log n)$	$O(\|\mathbb{x}\| + \log n)$	Universal

Typical element sizes:

- $\mathbb{F}_q$ element: 32 bytes
- $\mathbb{G}_1$ element: 32 bytes (compressed)
- $\mathbb{G}_2$ element: 64 bytes (compressed)

Key Insight

Trade small efficiency loss for huge flexibility gain!

The Universal SRS Advantage

Groth16
Circuit-specific setup

The Universal SRS Advantage

Groth16
Circuit-specific setup

Marlin/Plonk
Universal setup

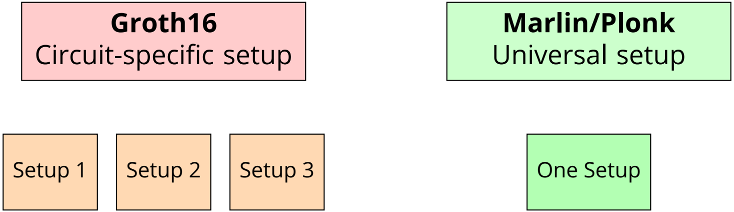
The Universal SRS Advantage

Groth16
Circuit-specific setup

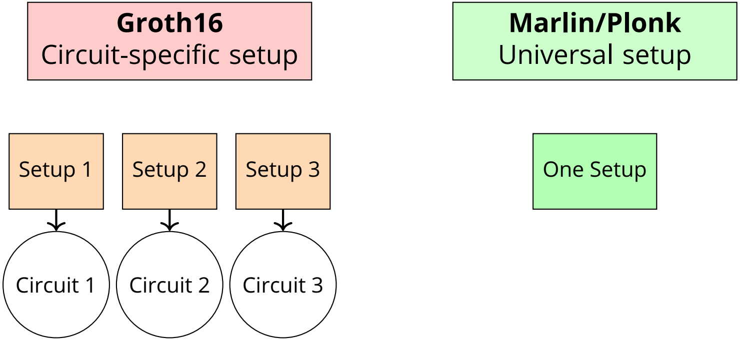
Marlin/Plonk
Universal setup

Setup 1
Setup 2
Setup 3

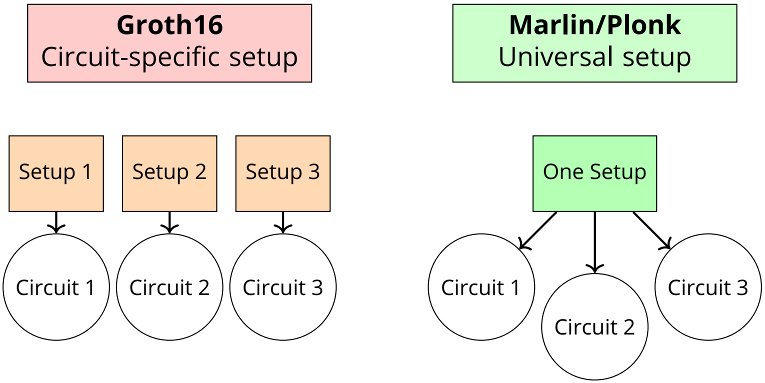
The Universal SRS Advantage



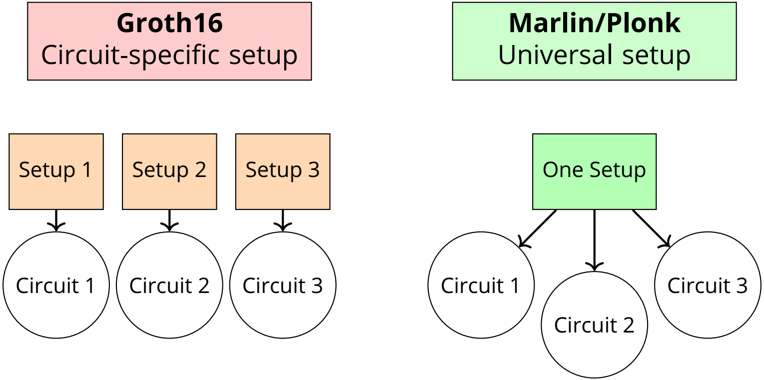
The Universal SRS Advantage



The Universal SRS Advantage



The Universal SRS Advantage



Key Advantage

Enabled by: KZG polynomial commitment scheme with updatable SRS!

Mathematical Foundations

Three Essential Algebraic Structures

Three Essential Algebraic Structures

1. **Finite Fields**

Mathematical Foundations

Three Essential Algebraic Structures

1. **Finite Fields**
2. **Elliptic Curves**

Mathematical Foundations

Three Essential Algebraic Structures

1. **Finite Fields**
2. **Elliptic Curves**
3. **Bilinear Pairings**

Mathematical Foundations

Three Essential Algebraic Structures

1. **Finite Fields**
2. **Elliptic Curves**
3. **Bilinear Pairings**

Building blocks for modern cryptography

Finite Fields

Definition

A field $(F, +, \cdot)$ is a commutative ring with unity where every non-zero element has a multiplicative inverse

Finite Fields

Definition

A field $(F, +, \cdot)$ is a commutative ring with unity where every non-zero element has a multiplicative inverse

Theorem

For any prime q, the integers modulo q with operations $+$ and $\cdot$ form a finite field, denoted $\mathbb{F}_q$

Finite Fields

Definition

A field $(F, +, \cdot)$ is a commutative ring with unity where every non-zero element has a multiplicative inverse

Theorem

For any prime q, the integers modulo q with operations $+$ and $\cdot$ form a finite field, denoted $\mathbb{F}_q$

Example in $\mathbb{F}_7 = \{0, 1, 2, 3, 4, 5, 6\}$:

- Addition: $5 + 4 = 9 \equiv 2 \pmod 7$

Finite Fields

Definition

A field $(F, +, \cdot)$ is a commutative ring with unity where every non-zero element has a multiplicative inverse

Theorem

For any prime q, the integers modulo q with operations $+$ and $\cdot$ form a finite field, denoted $\mathbb{F}_q$

Example in $\mathbb{F}_7 = \{0, 1, 2, 3, 4, 5, 6\}$:

- Addition: $5 + 4 = 9 \equiv 2 \pmod 7$
- Multiplication: $3 \cdot 5 = 15 \equiv 1 \pmod 7$

Finite Fields

Definition

A field $(F, +, \cdot)$ is a commutative ring with unity where every non-zero element has a multiplicative inverse

Theorem

For any prime q, the integers modulo q with operations $+$ and $\cdot$ form a finite field, denoted $\mathbb{F}_q$

Example in $\mathbb{F}_7 = \{0, 1, 2, 3, 4, 5, 6\}$:

- Addition: $5 + 4 = 9 \equiv 2 \pmod 7$
- Multiplication: $3 \cdot 5 = 15 \equiv 1 \pmod 7$
- Inverse: $3^{-1} = 5$ since $3 \cdot 5 \equiv 1 \pmod 7$

Finite Fields

Definition

A field $(F, +, \cdot)$ is a commutative ring with unity where every non-zero element has a multiplicative inverse

Theorem

For any prime q, the integers modulo q with operations $+$ and $\cdot$ form a finite field, denoted $\mathbb{F}_q$

Example in $\mathbb{F}_7 = \{0, 1, 2, 3, 4, 5, 6\}$:

- Addition: $5 + 4 = 9 \equiv 2 \pmod 7$
- Multiplication: $3 \cdot 5 = 15 \equiv 1 \pmod 7$
- Inverse: $3^{-1} = 5$ since $3 \cdot 5 \equiv 1 \pmod 7$

In cryptography: $q \approx 2^{256}$ for security

Elliptic Curves

Definition

An elliptic curve over $\mathbb{F}_q$ is the set of points (x, y) satisfying:

$$y^2 = x^3 + ax + b$$

plus a point at infinity $\mathcal{O}$

Elliptic Curves

Definition

An elliptic curve over $\mathbb{F}_q$ is the set of points (x, y) satisfying:

$$y^2 = x^3 + ax + b$$

plus a point at infinity $\mathcal{O}$

Group structure:

- Points form an abelian group

Definition

An elliptic curve over $\mathbb{F}_q$ is the set of points (x, y) satisfying:

$$y^2 = x^3 + ax + b$$

plus a point at infinity $\mathcal{O}$

Group structure:

- Points form an abelian group
- Identity: $\mathcal{O}$

Elliptic Curves

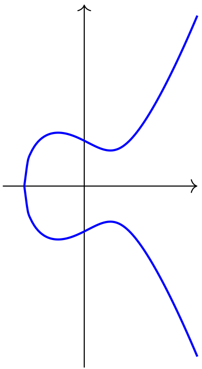
Definition

An elliptic curve over $\mathbb{F}_q$ is the set of points (x, y) satisfying:

$$y^2 = x^3 + ax + b$$

plus a point at infinity $\mathcal{O}$

Point addition



Group structure:

- Points form an abelian group
- Identity: $\mathcal{O}$

Elliptic Curves

Definition

An elliptic curve over $\mathbb{F}_q$ is the set of points (x, y) satisfying:

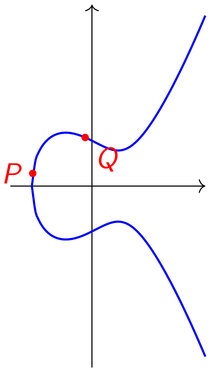
$$y^2 = x^3 + ax + b$$

plus a point at infinity $\mathcal{O}$

Group structure:

- Points form an abelian group
- Identity: $\mathcal{O}$

Point addition



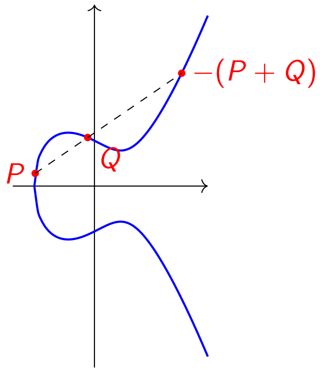
Definition

An elliptic curve over $\mathbb{F}_q$ is the set of points (x, y) satisfying:

$$y^2 = x^3 + ax + b$$

plus a point at infinity $\mathcal{O}$

Point addition



Group structure:

- Points form an abelian group
- Identity: $\mathcal{O}$

Elliptic Curves

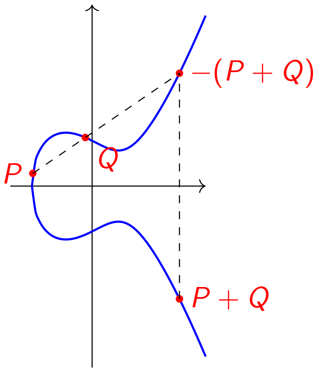
Definition

An elliptic curve over $\mathbb{F}_q$ is the set of points (x, y) satisfying:

$$y^2 = x^3 + ax + b$$

plus a point at infinity $\mathcal{O}$

Point addition



Group structure:

- Points form an abelian group
- Identity: $\mathcal{O}$

Bilinear Pairings

Definition

A bilinear pairing is a map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ where:

Bilinear Pairings

Definition

A bilinear pairing is a map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ where:

- $\mathbb{G}_1, \mathbb{G}_2$: cyclic groups of prime order q (usually elliptic curve groups)

Bilinear Pairings

Definition

A bilinear pairing is a map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ where:

- $\mathbb{G}_1, \mathbb{G}_2$: cyclic groups of prime order q (usually elliptic curve groups)
- $\mathbb{G}_T$: cyclic group of order q (usually in $\mathbb{F}^*_{q^k}$)

Bilinear Pairings

Definition

A bilinear pairing is a map $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ where:

- $\mathbb{G}_1, \mathbb{G}_2$: cyclic groups of prime order q (usually elliptic curve groups)
- $\mathbb{G}_T$: cyclic group of order q (usually in $\mathbb{F}^*_{q^k}$)

Bilinearity property:

$$e(aP, bQ) = e(P, Q)^{ab} \quad \text{for all } a, b \in \mathbb{F}_q$$

Bilinear Pairings

Definition

A bilinear pairing is a map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ where:

- $\mathbb{G}_1, \mathbb{G}_2$: cyclic groups of prime order q (usually elliptic curve groups)
- $\mathbb{G}_T$: cyclic group of order q (usually in $\mathbb{F}_{q^k}^*$)

Bilinearity property:

$$e(aP, bQ) = e(P, Q)^{ab} \quad \text{for all } a, b \in \mathbb{F}_q$$

Properties:

- Non-degenerate: $e(P, Q) \neq 1$ for generators P, Q

Bilinear Pairings

Definition

A bilinear pairing is a map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ where:

- $\mathbb{G}_1, \mathbb{G}_2$: cyclic groups of prime order q (usually elliptic curve groups)
- $\mathbb{G}_T$: cyclic group of order q (usually in $\mathbb{F}^*_{q^k}$)

Bilinearity property:

$$e(aP, bQ) = e(P, Q)^{ab} \quad \text{for all } a, b \in \mathbb{F}_q$$

Properties:

- Non-degenerate: $e(P, Q) \neq 1$ for generators P, Q
- Efficiently computable

Bilinear Pairings

Definition

A bilinear pairing is a map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ where:

- $\mathbb{G}_1, \mathbb{G}_2$: cyclic groups of prime order q (usually elliptic curve groups)
- $\mathbb{G}_T$: cyclic group of order q (usually in $\mathbb{F}_{q^k}^*$)

Bilinearity property:

$$e(aP, bQ) = e(P, Q)^{ab} \quad \text{for all } a, b \in \mathbb{F}_q$$

Properties:

- Non-degenerate: $e(P, Q) \neq 1$ for generators P, Q
- Efficiently computable
- Examples: Weil pairing, Tate pairing

zk-SNARKs Architecture

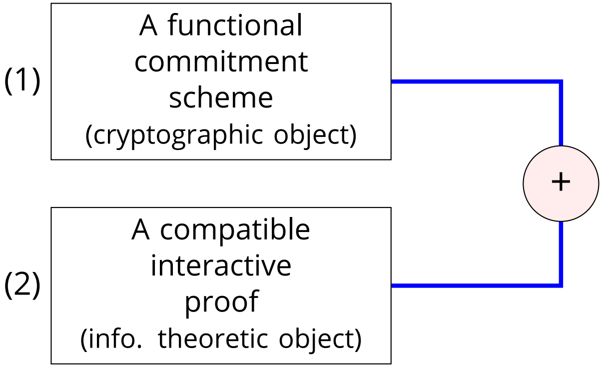
(1) A functional commitment scheme (cryptographic object)

zk-SNARKs Architecture

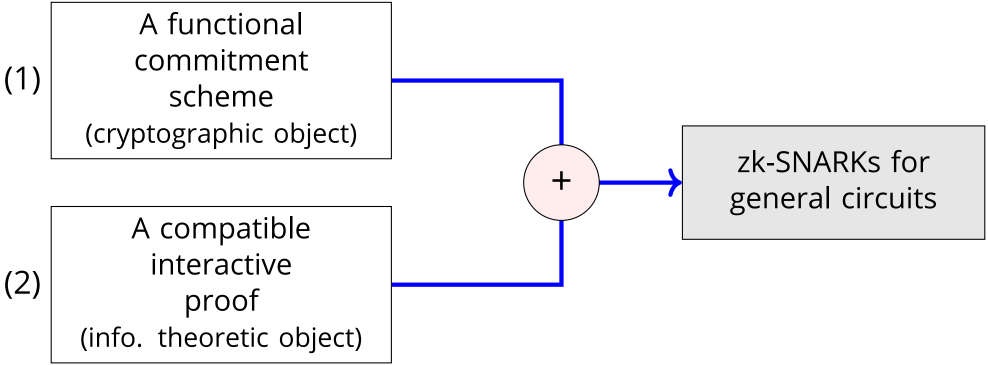
(1) A functional commitment scheme (cryptographic object)

(2) A compatible interactive proof (info. theoretic object)

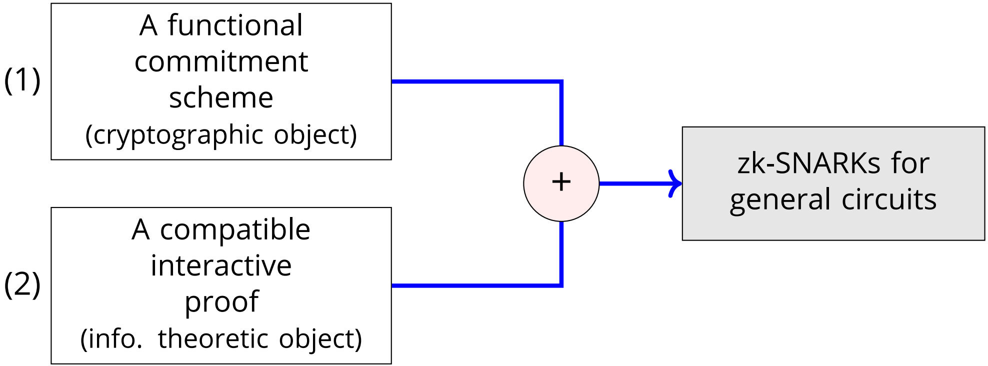
zk-SNARKs Architecture



zk-SNARKs Architecture



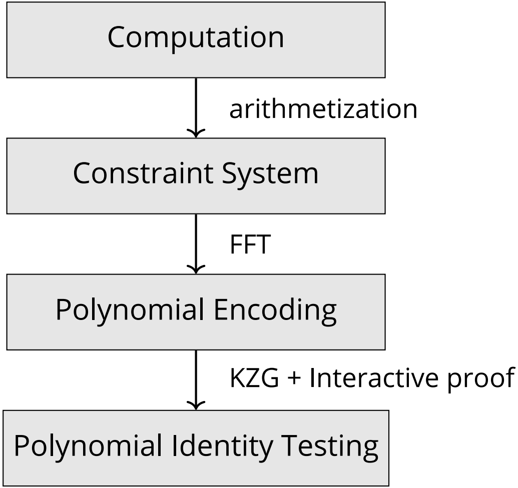
zk-SNARKs Architecture



In this work

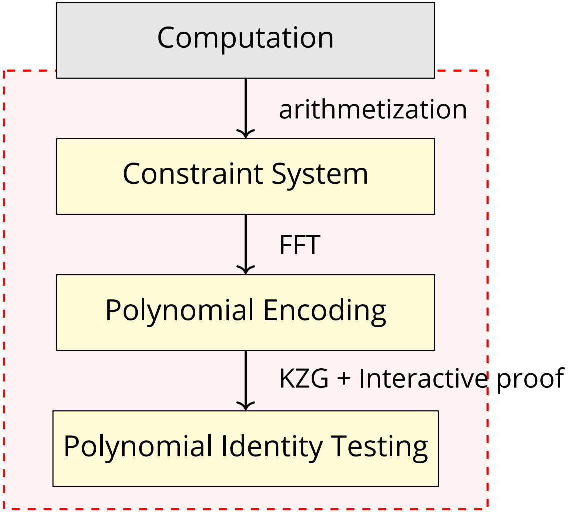
KZG (same for both) + Different interactive proofs (Marlin vs Plonk)

The Computation Pipeline

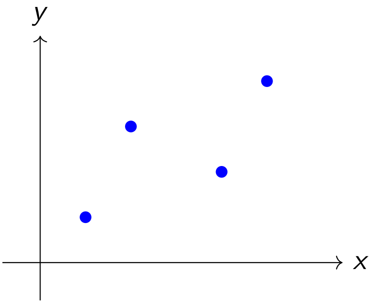


The Computation Pipeline

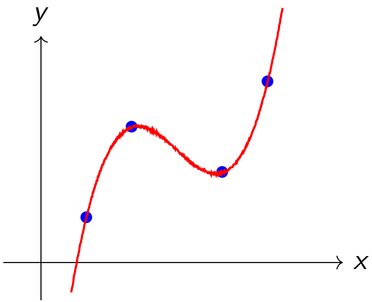
Scope of this final project



Why Polynomials?

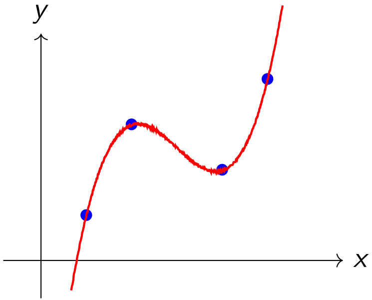


Why Polynomials?



Polynomial through points

Why Polynomials?

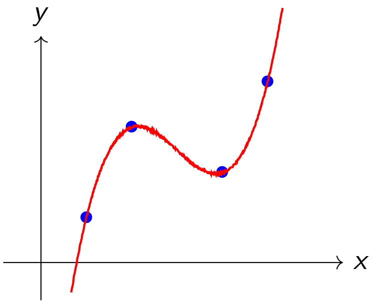


Polynomial through points

Unique Interpolation

- n points uniquely determine degree $n - 1$ polynomial

Why Polynomials?

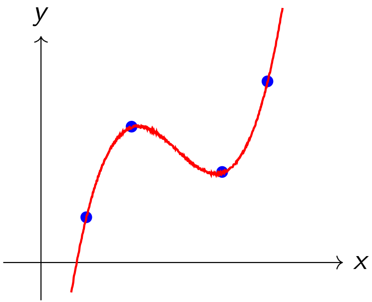


Polynomial through points

Unique Interpolation

- n points uniquely determine degree $n-1$ polynomial
- Can encode n values as single object

Why Polynomials?



Polynomial through points

Unique Interpolation

- n points uniquely determine degree $n-1$ polynomial
- Can encode n values as single object
- Efficient algorithms (FFT) with $O(n \log n)$

The Schwartz-Zippel Lemma

Lemma

Let $\mathbb{F}_q$ be a finite field and $f \in \mathbb{F}_q[X_1, X_2, \ldots, X_n]$ be a non-zero polynomial of total degree at most d. Then

$$\Pr[f(r) = 0 \mid r \leftarrow\!\$\ \mathbb{F}_q^n] \leq \frac{d}{|\mathbb{F}_q|}$$

The Schwartz-Zippel Lemma

Lemma

Let $\mathbb{F}_q$ be a finite field and $f \in \mathbb{F}_q[X_1, X_2, \ldots, X_n]$ be a non-zero polynomial of total degree at most d. Then

$$\Pr[f(r) = 0 \mid r \leftarrow\$ \mathbb{F}_q^n] \leq \frac{d}{|\mathbb{F}_q|}$$

Example: Consider $f(x, y) = x^3 + xy^2 - 2y^3 - 2x + y$ in $\mathbb{F}_7[x, y]$
Total degree: $d = 3$

The Schwartz-Zippel Lemma

Lemma

Let $\mathbb{F}_q$ be a finite field and $f \in \mathbb{F}_q[X_1, X_2, \ldots, X_n]$ be a non-zero polynomial of total degree at most d. Then

$$\Pr[f(r) = 0 \mid r \leftarrow\$ \mathbb{F}_q^n] \leq \frac{d}{|\mathbb{F}_q|}$$

Example: Consider $f(x, y) = x^3 + xy^2 - 2y^3 - 2x + y$ in $\mathbb{F}_7[x, y]$
Total degree: $d = 3$
Roots in $\mathbb{F}_7^2$: $(0, 0), (0, 2), (0, 5), (2, 6), (3, 0), (4, 0), (5, 1)$

The Schwartz-Zippel Lemma

Lemma

Let $\mathbb{F}_q$ be a finite field and $f \in \mathbb{F}_q[X_1, X_2, \ldots, X_n]$ be a non-zero polynomial of total degree at most d. Then

$$\Pr[f(r) = 0 \mid r \leftarrow\$ \mathbb{F}_q^n] \leq \frac{d}{|\mathbb{F}_q|}$$

Example: Consider $f(x, y) = x^3 + xy^2 - 2y^3 - 2x + y$ in $\mathbb{F}_7[x, y]$
Total degree: $d = 3$
Roots in $\mathbb{F}_7^2$: $(0, 0), (0, 2), (0, 5), (2, 6), (3, 0), (4, 0), (5, 1)$
Probability calculation:

- Total points in $\mathbb{F}_7^2$: $7^2 = 49$

The Schwartz-Zippel Lemma

Lemma

Let $\mathbb{F}_q$ be a finite field and $f \in \mathbb{F}_q[X_1, X_2, \ldots, X_n]$ be a non-zero polynomial of total degree at most d. Then

$$\Pr[f(r) = 0 \mid r \leftarrow\$ \mathbb{F}_q^n] \leq \frac{d}{|\mathbb{F}_q|}$$

Example: Consider $f(x, y) = x^3 + xy^2 - 2y^3 - 2x + y$ in $\mathbb{F}_7[x, y]$
Total degree: $d = 3$
Roots in $\mathbb{F}_7^2$: $(0, 0), (0, 2), (0, 5), (2, 6), (3, 0), (4, 0), (5, 1)$
Probability calculation:

- Total points in $\mathbb{F}_7^2$: $7^2 = 49$
- Number of roots: 7

The Schwartz-Zippel Lemma

Lemma

Let $\mathbb{F}_q$ be a finite field and $f \in \mathbb{F}_q[X_1, X_2, \ldots, X_n]$ be a non-zero polynomial of total degree at most d. Then

$$\Pr[f(r) = 0 \mid r \leftarrow\!\$\ \mathbb{F}_q^n] \leq \frac{d}{|\mathbb{F}_q|}$$

Example: Consider $f(x, y) = x^3 + xy^2 - 2y^3 - 2x + y$ in $\mathbb{F}_7[x, y]$
Total degree: $d = 3$
Roots in $\mathbb{F}_7^2$: $(0, 0), (0, 2), (0, 5), (2, 6), (3, 0), (4, 0), (5, 1)$
Probability calculation:

- Total points in $\mathbb{F}_7^2$: $7^2 = 49$
- Number of roots: 7
- $\Pr[f(r) = 0] = \frac{7}{49} = \frac{1}{7} \leq \frac{3}{7} = \frac{d}{|\mathbb{F}_7|}$ ✓

The Schwartz-Zippel Lemma

Lemma

Let $\mathbb{F}_q$ be a finite field and $f \in \mathbb{F}_q[X_1, X_2, \ldots, X_n]$ be a non-zero polynomial of total degree at most d. Then

$$\Pr[f(r) = 0 \mid r \leftarrow\$ \mathbb{F}_q^n] \leq \frac{d}{|\mathbb{F}_q|}$$

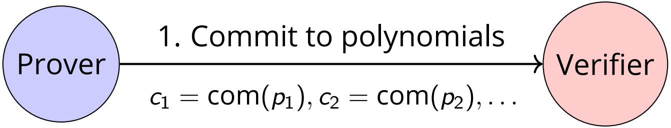
Example: Consider $f(x, y) = x^3 + xy^2 - 2y^3 - 2x + y$ in $\mathbb{F}_7[x, y]$
Total degree: $d = 3$
Roots in $\mathbb{F}_7^2$: $(0, 0), (0, 2), (0, 5), (2, 6), (3, 0), (4, 0), (5, 1)$
Probability calculation:

- Total points in $\mathbb{F}_7^2$: $7^2 = 49$
- Number of roots: 7
- $\Pr[f(r) = 0] = \frac{7}{49} = \frac{1}{7} \leq \frac{3}{7} = \frac{d}{|\mathbb{F}_7|}$ ✓

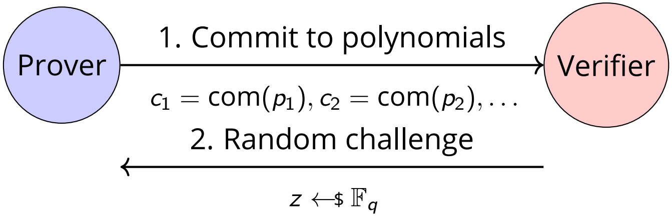
In practice: $q \approx 2^{256}$, so probability $\leq \frac{d}{2^{256}}$ is negligible!

Polynomial Identity Testing Protocol

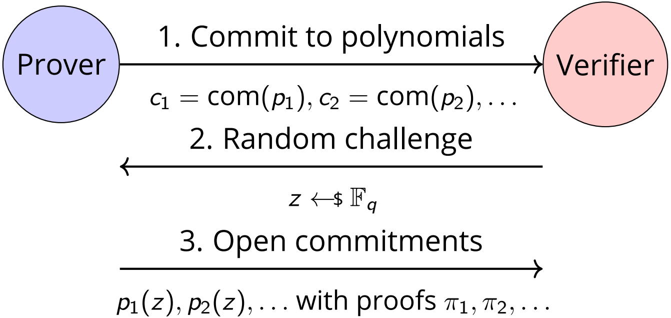
Polynomial Identity Testing Protocol



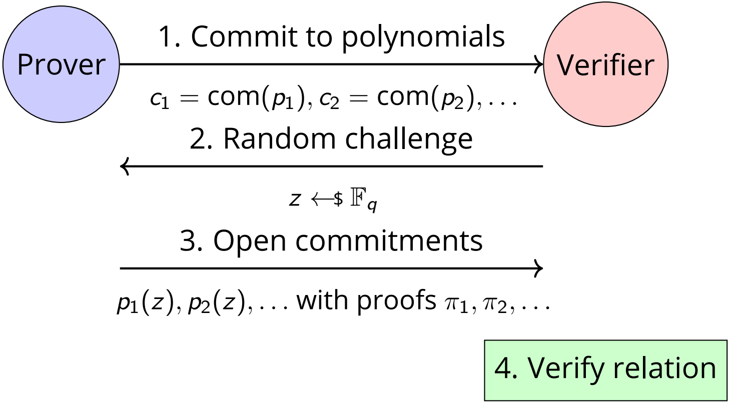
Polynomial Identity Testing Protocol



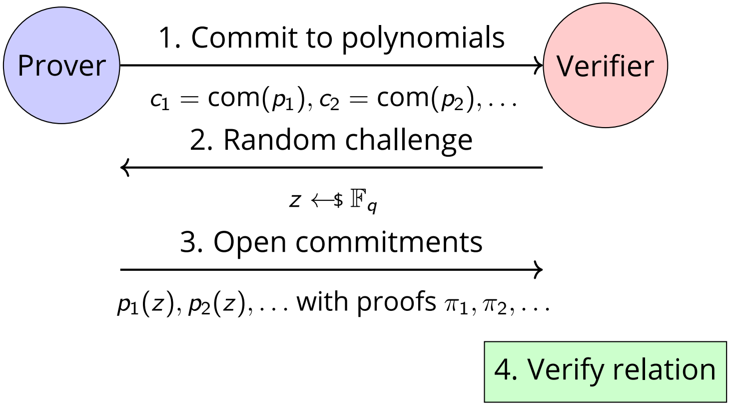
Polynomial Identity Testing Protocol



Polynomial Identity Testing Protocol



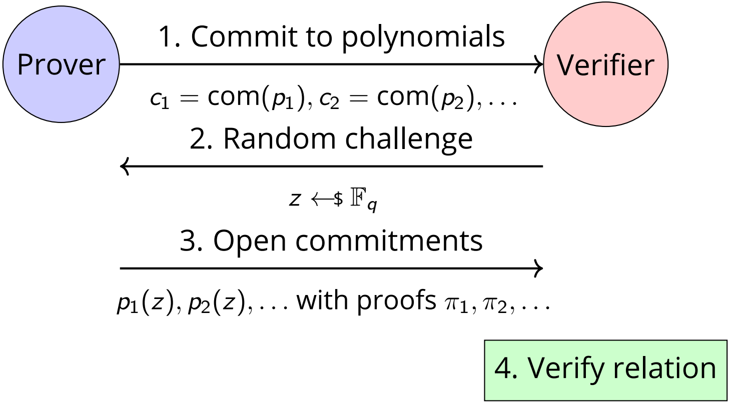
Polynomial Identity Testing Protocol



Security guarantees:

- **Binding**: Cannot change polynomials after commitment

Polynomial Identity Testing Protocol

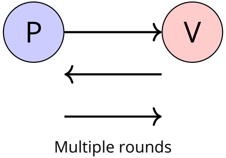


Security guarantees:

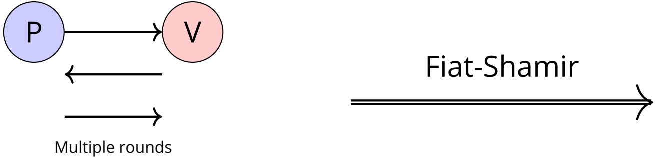
- **Binding**: Cannot change polynomials after commitment
- **Soundness**: Schwartz-Zippel ensures false claims fail with probability $\geq 1 - \frac{d}{|\mathbb{F}_q|}$

Interactive to Non-Interactive

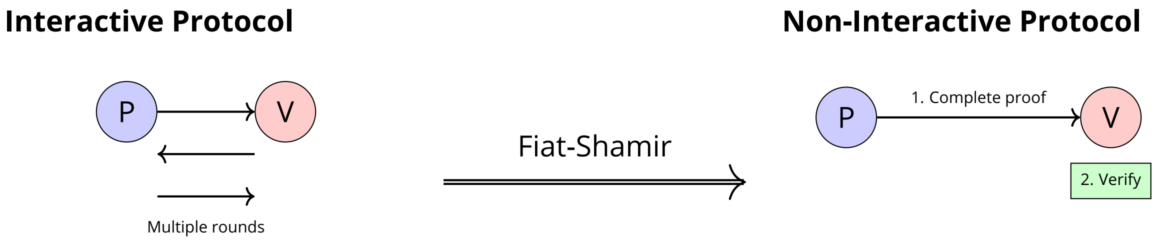
Interactive Protocol



Interactive Protocol



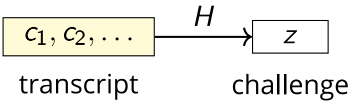
Interactive to Non-Interactive



Interactive to Non-Interactive

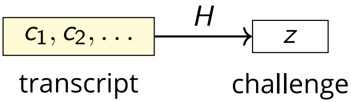
Interactive Protocol **Non-Interactive Protocol**

Key transformation: Replace verifier's random challenges with hash function



Interactive Protocol **Non-Interactive Protocol**

Key transformation: Replace verifier's random challenges with hash function



Result

Single proof string that can be verified by anyone - perfect for blockchain!

KZG Construction Overview

Setup Phase (Trusted)

Generate powers of secret x in $\mathbb{F}_q$:

$$\text{SRS} = \{G_1, xG_1, x^2 G_1, \ldots, x^d G_1, G_2, xG_2\}$$

Secret x is destroyed after setup!

KZG Construction Overview

Setup Phase (Trusted)

Generate powers of secret x in $\mathbb{F}_q$:

$$\mathrm{SRS} = \{G_1, xG_1, x^2 G_1, \ldots, x^d G_1, G_2, xG_2\}$$

Secret x is destroyed after setup!

KZG Construction Overview

Setup Phase (Trusted)

Generate powers of secret x in $\mathbb{F}_q$:

$$\text{SRS} = \{G_1, xG_1, x^2 G_1, \ldots, x^d G_1, G_2, xG_2\}$$

Secret x is destroyed after setup!

Key Operations:

1. **Commit**: For polynomial $p(X) = \sum a_i X^i$

$$C = p(x) G_1 = \sum a_i (x^i G_1)$$

KZG Construction Overview

Setup Phase (Trusted)

Generate powers of secret x in $\mathbb{F}_q$:

$$\text{SRS} = \{G_1, xG_1, x^2 G_1, \ldots, x^d G_1, G_2, xG_2\}$$

Secret x is destroyed after setup!

Key Operations:

1. **Commit**: For polynomial $p(X) = \sum a_i X^i$

$$C = p(x) G_1 = \sum a_i (x^i G_1)$$

2. **Open**: Prove $p(z) = v$ by showing $(p(X) - v)$ divisible by $(X - z)$

KZG Construction Overview

Setup Phase (Trusted)

Generate powers of secret x in $\mathbb{F}_q$:

$$\text{SRS} = \{G_1, xG_1, x^2 G_1, \ldots, x^d G_1, G_2, xG_2\}$$

Secret x is destroyed after setup!

Key Operations:

1. **Commit**: For polynomial $p(X) = \sum a_i X^i$

$$C = p(x) G_1 = \sum a_i (x^i G_1)$$

2. **Open**: Prove $p(z) = v$ by showing $(p(X) - v)$ divisible by $(X - z)$
3. **Verify**: Check using bilinear pairing

Committing to Polynomials

Example: $p(X) = 2X^2 + 3X + 5$

a_0	5
a_1	3
a_2	2

Example: $p(X) = 2X^2 + 3X + 5$

Using SRS powers:

$$\begin{aligned} C &= p(x)G_1 \\ &= (2x^2 + 3x + 5)G_1 \\ &= 2(x^2 G_1) + 3(xG_1) + 5G_1 \end{aligned}$$

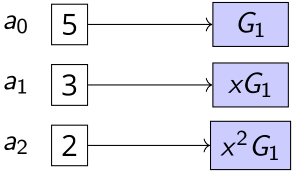
a_0	5	G_1
a_1	3	xG_1
a_2	2	$x^2 G_1$

Committing to Polynomials

Example: $p(X) = 2X^2 + 3X + 5$

Using SRS powers:

$$\begin{aligned} C &= p(x)G_1 \\ &= (2x^2 + 3x + 5)G_1 \\ &= 2(x^2 G_1) + 3(xG_1) + 5G_1 \end{aligned}$$

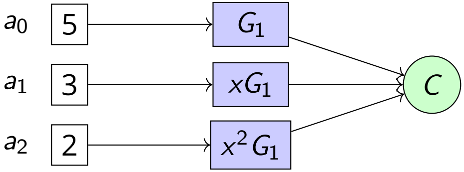


Committing to Polynomials

Example: $p(X) = 2X^2 + 3X + 5$

Using SRS powers:

$$\begin{aligned} C &= p(x)G_1 \\ &= (2x^2 + 3x + 5)G_1 \\ &= 2(x^2 G_1) + 3(xG_1) + 5G_1 \end{aligned}$$

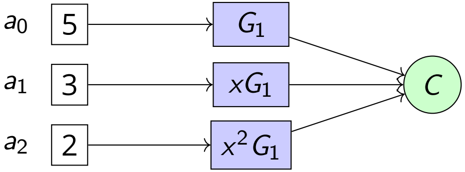


Committing to Polynomials

Example: $p(X) = 2X^2 + 3X + 5$

Using SRS powers:

$$\begin{aligned} C &= p(x)G_1 \\ &= (2x^2 + 3x + 5)G_1 \\ &= 2(x^2 G_1) + 3(xG_1) + 5G_1 \end{aligned}$$



Result: Single group element C!

Opening at Evaluation Point

Theorem

For $p \in \mathbb{F}_q[X]$ and $z, v \in \mathbb{F}_q$,

$$p(z) = v \iff (X - z) \text{ divides } (p(X) - v)$$

Opening at Evaluation Point

Theorem

For $p \in \mathbb{F}_q[X]$ and $z, v \in \mathbb{F}_q$,

$$p(z) = v \iff (X - z) \text{ divides } (p(X) - v)$$

Goal: Prove that committed polynomial p satisfies $p(z) = v$

Opening at Evaluation Point

Theorem

For $p \in \mathbb{F}_q[X]$ and $z, v \in \mathbb{F}_q$,

$$p(z) = v \iff (X - z) \text{ divides } (p(X) - v)$$

Goal: Prove that committed polynomial p satisfies $p(z) = v$

Protocol:

1. Compute witness polynomial: $w(X) = \frac{p(X)-v}{X-z}$

Opening at Evaluation Point

Theorem

For $p \in \mathbb{F}_q[X]$ and $z, v \in \mathbb{F}_q$,

$$p(z) = v \iff (X - z) \text{ divides } (p(X) - v)$$

Goal: Prove that committed polynomial p satisfies $p(z) = v$

Protocol:

1. Compute witness polynomial: $w(X) = \frac{p(X)-v}{X-z}$
2. Create proof: $\pi = w(x)G_1$ using SRS

Opening at Evaluation Point

Theorem

For $p \in \mathbb{F}_q[X]$ and $z, v \in \mathbb{F}_q$,

$$p(z) = v \iff (X - z) \text{ divides } (p(X) - v)$$

Goal: Prove that committed polynomial p satisfies $p(z) = v$

Protocol:

1. Compute witness polynomial: $w(X) = \frac{p(X)-v}{X-z}$
2. Create proof: $\pi = w(x)G_1$ using SRS
3. Proof size: Just one group element!

Opening at Evaluation Point

Theorem

For $p \in \mathbb{F}_q[X]$ and $z, v \in \mathbb{F}_q$,

$$p(z) = v \iff (X - z) \text{ divides } (p(X) - v)$$

Goal: Prove that committed polynomial p satisfies $p(z) = v$

Protocol:

1. Compute witness polynomial: $w(X) = \frac{p(X)-v}{X-z}$
2. Create proof: $\pi = w(x)G_1$ using SRS
3. Proof size: Just one group element!

Example: If $p(X) = X^2 + 2X + 1$ and claiming $p(3) = 16$:

$$w(X) = \frac{X^2 + 2X + 1 - 16}{X - 3} = \frac{X^2 + 2X - 15}{X - 3} = X + 5$$

Verification via Pairing

Verification Equation

$$e(C - vG_1, G_2) \stackrel{?}{=} e(\pi, xG_2 - zG_2)$$

Verification via Pairing

Verification Equation

$$e(C - vG_1, G_2) \stackrel{?}{=} e(\pi, xG_2 - zG_2)$$

Why this works:

$$\text{LHS} = e((p(x) - v)G_1, G_2) = e(w(x)(x - z)G_1, G_2) = e(w(x)G_1, (x - z)G_2) = \text{RHS}$$

Verification via Pairing

Verification Equation

$$e(C - vG_1, G_2) \stackrel{?}{=} e(\pi, xG_2 - zG_2)$$

Why this works:

$$\text{LHS} = e((p(x) - v)G_1, G_2) = e(w(x)(x - z)G_1, G_2) = e(w(x)G_1, (x - z)G_2) = \text{RHS}$$

Verifier efficiency:

- Just 2 pairing operations

Verification Equation

$$e(C - vG_1, G_2) \stackrel{?}{=} e(\pi, xG_2 - zG_2)$$

Why this works:

$$\text{LHS} = e((p(x) - v)G_1, G_2) = e(w(x)(x - z)G_1, G_2) = e(w(x)G_1, (x - z)G_2) = \text{RHS}$$

Verifier efficiency:

- Just 2 pairing operations
- Independent of polynomial degree

Verification via Pairing

Verification Equation

$$e(C - vG_1, G_2) \stackrel{?}{=} e(\pi, xG_2 - zG_2)$$

Why this works:

$$\text{LHS} = e((p(x) - v)G_1, G_2) = e(w(x)(x - z)G_1, G_2) = e(w(x)G_1, (x - z)G_2) = \text{RHS}$$

Verifier efficiency:

- Just 2 pairing operations
- Independent of polynomial degree
- Constant time verification!

Security Properties

1. **Completeness** ✓

Security Properties

1. **Completeness** ✓
   - Honest prover always succeeds

Security Properties

1. **Completeness** ✓
    - Honest prover always succeeds
    - Straightforward from construction

Security Properties

1. **Completeness** ✓
   - Honest prover always succeeds
   - Straightforward from construction
   - If $p(z) = v$, then verification equation holds

Security Properties

1. Completeness ✓

- Honest prover always succeeds
- Straightforward from construction
- If $p(z) = v$, then verification equation holds

2. Evaluation Binding

- Cannot open to two different values at same point

Security Properties

1. Completeness ✓

- Honest prover always succeeds
- Straightforward from construction
- If $p(z) = v$, then verification equation holds

2. Evaluation Binding

- Cannot open to two different values at same point
- Based on Strong Diffie-Hellman (SDH) assumption

Security Properties

1. Completeness ✓

- Honest prover always succeeds
- Straightforward from construction
- If $p(z) = v$, then verification equation holds

2. Evaluation Binding

- Cannot open to two different values at same point
- Based on Strong Diffie-Hellman (SDH) assumption
- Breaking requires solving hard problem

Security Properties

1. Completeness ✓

- Honest prover always succeeds
- Straightforward from construction
- If $p(z) = v$, then verification equation holds

2. Evaluation Binding

- Cannot open to two different values at same point
- Based on Strong Diffie-Hellman (SDH) assumption
- Breaking requires solving hard problem

Remark

Complete zk-SNARKs actually need a stronger property than evaluation binding: **extractability**. This ensures any valid commitment corresponds to an actual polynomial (as required in the Marlin paper).

Proving Evaluation Binding

Strong Diffie-Hellman (SDH) Assumption

Given $\{G_1, xG_1, x^2G_1, \ldots, x^dG_1, G_2, xG_2\}$, hard to compute:

$$\left(c, \frac{1}{x+c}G_1\right) \text{ for any } c \in \mathbb{F}_q$$

Proof idea: If adversary breaks binding $\Rightarrow$ can break SDH
Suppose adversary outputs (C, z, v, v', π, π') with $v \neq v'$
Both proofs verify:

$$e(C - vG_1, G_2) = e(\pi, xG_2 - zG_2)$$
$$e(C - v'G_1, G_2) = e(\pi', xG_2 - zG_2)$$

Subtracting:

$$e((v' - v)G_1, G_2) = e(\pi - \pi', xG_2 - zG_2)$$

If $\pi \neq \pi'$**:** Can extract $\frac{1}{x-z}G_1 = \frac{\pi-\pi'}{v'-v} \Rightarrow$ Breaks SDH!

Constraint System

Marlin (R1CS)

Constraint equation:

$$\boxed{Az \circ Bz = Cz}$$

Where:

- $A, B, C \in \mathbb{F}_q^{n \times n}$ are constraint matrices
- $z = (x, w) \in \mathbb{F}_q^n$ is the assignment vector
- x are public inputs, w are witness values
- $\circ$ denotes entry-wise product

Plonk

Gate constraint:

$$\boxed{\begin{aligned} q_L \cdot z_{a_i} + q_R \cdot z_{b_i} + q_O \cdot z_{c_i} \\ + q_M \cdot (z_{a_i} \cdot z_{b_i}) + q_C = 0 \end{aligned}}$$

Where:

- $q_L, q_R, q_O, q_M, q_C \in \mathbb{F}_q^n$ are selectors
- $z = (x, w) \in \mathbb{F}_q^m$ is wire assignment
- x are public inputs, w are witness values
- $a, b, c \in [m]^n$ are wire indices
- Additional copy constraints via σ

Example Problem Definition

Polynomial Evaluation Problem

Prove knowledge of secret $X \in \mathbb{F}_{23}$ such that:

$$Y = X^3 + 2X + 5$$

where $Y = 15$ is public and $X = 3$ is the witness.

Example Problem Definition

Polynomial Evaluation Problem

Prove knowledge of secret $X \in \mathbb{F}_{23}$ such that:

$$Y = X^3 + 2X + 5$$

where $Y = 15$ is public and $X = 3$ is the witness.

Computation trace:

- $w_1 = X = 3$

Example Problem Definition

Polynomial Evaluation Problem

Prove knowledge of secret $X \in \mathbb{F}_{23}$ such that:

$$Y = X^3 + 2X + 5$$

where $Y = 15$ is public and $X = 3$ is the witness.

Computation trace:

- $w_1 = X = 3$
- $w_2 = X^2 = 9$

Example Problem Definition

Polynomial Evaluation Problem

Prove knowledge of secret $X \in \mathbb{F}_{23}$ such that:

$$Y = X^3 + 2X + 5$$

where $Y = 15$ is public and $X = 3$ is the witness.

Computation trace:

- $w_1 = X = 3$
- $w_2 = X^2 = 9$
- $w_3 = X^3 = 4$ (note: $27 \bmod 23 = 4$)

Example Problem Definition

Polynomial Evaluation Problem

Prove knowledge of secret $X \in \mathbb{F}_{23}$ such that:

$$Y = X^3 + 2X + 5$$

where $Y = 15$ is public and $X = 3$ is the witness.

Computation trace:

- $w_1 = X = 3$
- $w_2 = X^2 = 9$
- $w_3 = X^3 = 4$ (note: $27 \bmod 23 = 4$)
- $w_4 = 2X = 6$

Example Problem Definition

Polynomial Evaluation Problem

Prove knowledge of secret $X \in \mathbb{F}_{23}$ such that:

$$Y = X^3 + 2X + 5$$

where $Y = 15$ is public and $X = 3$ is the witness.

Computation trace:

- $w_1 = X = 3$
- $w_2 = X^2 = 9$
- $w_3 = X^3 = 4$ (note: $27 \bmod 23 = 4$)
- $w_4 = 2X = 6$
- $w_5 = X^3 + 2X = 10$

Example Problem Definition

Polynomial Evaluation Problem

Prove knowledge of secret $X \in \mathbb{F}_{23}$ such that:

$$Y = X^3 + 2X + 5$$

where $Y = 15$ is public and $X = 3$ is the witness.

Computation trace:

- $w_1 = X = 3$
- $w_2 = X^2 = 9$
- $w_3 = X^3 = 4$ (note: $27 \bmod 23 = 4$)
- $w_4 = 2X = 6$
- $w_5 = X^3 + 2X = 10$
- $Y = w_5 + 5 = 15$ ✓

Example Problem Definition

Polynomial Evaluation Problem

Prove knowledge of secret $X \in \mathbb{F}_{23}$ such that:

$$Y = X^3 + 2X + 5$$

where $Y = 15$ is public and $X = 3$ is the witness.

Computation trace:

- $w_1 = X = 3$
- $w_2 = X^2 = 9$
- $w_3 = X^3 = 4$ (note: $27 \bmod 23 = 4$)
- $w_4 = 2X = 6$
- $w_5 = X^3 + 2X = 10$
- $Y = w_5 + 5 = 15 \checkmark$

Marlin R1CS Encoding

Assignment vector: $z = [1, 15, 3, 9, 4, 6]$

Marlin R1CS Encoding

Assignment vector: $z = [1, 15, 3, 9, 4, 6]$

- Public inputs: $[1, Y]$

Marlin R1CS Encoding

Assignment vector: $z = [1, 15, 3, 9, 4, 6]$

- Public inputs: $[1, Y]$
- Witness values: $[X, X^2, X^3, 2X]$

Marlin R1CS Encoding

Assignment vector: $z = [1, 15, 3, 9, 4, 6]$

- Public inputs: $[1, Y]$
- Witness values: $[X, X^2, X^3, 2X]$

Constraints:

1. $w_1 \cdot w_1 = w_2$ (computing X^2)

Marlin R1CS Encoding

Assignment vector: $z = [1, 15, 3, 9, 4, 6]$

- Public inputs: $[1, Y]$
- Witness values: $[X, X^2, X^3, 2X]$

Constraints:

1. $w_1 \cdot w_1 = w_2$ (computing X^2)
2. $w_2 \cdot w_1 = w_3$ (computing X^3)

Marlin R1CS Encoding

Assignment vector: $z = [1, 15, 3, 9, 4, 6]$

- Public inputs: $[1, Y]$
- Witness values: $[X, X^2, X^3, 2X]$

Constraints:

1. $w_1 \cdot w_1 = w_2$ (computing X^2)
2. $w_2 \cdot w_1 = w_3$ (computing X^3)
3. $2 \cdot w_1 = w_4$ (computing $2X$)

Marlin R1CS Encoding

Assignment vector: $z = [1, 15, 3, 9, 4, 6]$

- Public inputs: $[1, Y]$
- Witness values: $[X, X^2, X^3, 2X]$

Constraints:

1. $w_1 \cdot w_1 = w_2$ (computing X^2)
2. $w_2 \cdot w_1 = w_3$ (computing X^3)
3. $2 \cdot w_1 = w_4$ (computing $2X$)
4. $(5 + w_3 + w_4) \cdot 1 = Y$ (final addition)

Marlin Constraint Matrices

$$A = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 2 & 0 & 0 & 0 & 0 & 0 \\ 5 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}, \quad B = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Marlin Constraint Matrices

$$A = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 2 & 0 & 0 & 0 & 0 & 0 \\ 5 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}, \quad B = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$C = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Marlin Constraint Matrices

$$A = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 2 & 0 & 0 & 0 & 0 & 0 \\ 5 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}, \quad B = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$C = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Note: Matrices are 4×6 (4 constraints, 6 variables). In practice, padded with zero rows to form square $n \times n$ matrices for polynomial encoding.

Plonk Circuit Encoding

Wire assignment: $z = [1, 2, 5, 15, 3, 9, 4, 6, 10]$

Wire assignment: $z = [1, 2, 5, 15, 3, 9, 4, 6, 10]$

- Wires 1-4: public inputs $[1, 2, 5, Y]$

Plonk Circuit Encoding

Wire assignment: $z = [1, 2, 5, 15, 3, 9, 4, 6, 10]$

- Wires 1-4: public inputs $[1, 2, 5, Y]$
- Wires 5-9: witness values $[X, X^2, X^3, 2X, X^3 + 2X]$

Plonk Circuit Encoding

Wire assignment: $z = [1, 2, 5, 15, 3, 9, 4, 6, 10]$

- Wires 1-4: public inputs $[1, 2, 5, Y]$
- Wires 5-9: witness values $[X, X^2, X^3, 2X, X^3 + 2X]$

Constraints:

1-4. Constant gates: $z_1 = 1, z_2 = 2, z_3 = 5, z_4 = 15$

Plonk Circuit Encoding

Wire assignment: $z = [1, 2, 5, 15, 3, 9, 4, 6, 10]$

- Wires 1-4: public inputs $[1, 2, 5, Y]$
- Wires 5-9: witness values $[X, X^2, X^3, 2X, X^3 + 2X]$

Constraints:

1-4. Constant gates: $z_1 = 1, z_2 = 2, z_3 = 5, z_4 = 15$

5. $z_5 \cdot z_5 = z_6$ (computing X^2)

Plonk Circuit Encoding

Wire assignment: $z = [1, 2, 5, 15, 3, 9, 4, 6, 10]$

- Wires 1-4: public inputs $[1, 2, 5, Y]$
- Wires 5-9: witness values $[X, X^2, X^3, 2X, X^3 + 2X]$

Constraints:

1-4. Constant gates: $z_1 = 1, z_2 = 2, z_3 = 5, z_4 = 15$

5. $z_5 \cdot z_5 = z_6$ (computing X^2)

6. $z_6 \cdot z_5 = z_7$ (computing X^3)

Plonk Circuit Encoding

Wire assignment: $z = [1, 2, 5, 15, 3, 9, 4, 6, 10]$

- Wires 1-4: public inputs $[1, 2, 5, Y]$
- Wires 5-9: witness values $[X, X^2, X^3, 2X, X^3 + 2X]$

Constraints:

1-4. Constant gates: $z_1 = 1, z_2 = 2, z_3 = 5, z_4 = 15$

5. $z_5 \cdot z_5 = z_6$ (computing X^2)
6. $z_6 \cdot z_5 = z_7$ (computing X^3)
7. $z_5 \cdot z_2 = z_8$ (computing $2X$)

Plonk Circuit Encoding

Wire assignment: $z = [1, 2, 5, 15, 3, 9, 4, 6, 10]$

- Wires 1-4: public inputs $[1, 2, 5, Y]$
- Wires 5-9: witness values $[X, X^2, X^3, 2X, X^3 + 2X]$

Constraints:

1-4. Constant gates: $z_1 = 1, z_2 = 2, z_3 = 5, z_4 = 15$

5. $z_5 \cdot z_5 = z_6$ (computing X^2)
6. $z_6 \cdot z_5 = z_7$ (computing X^3)
7. $z_5 \cdot z_2 = z_8$ (computing $2X$)
8. $z_7 + z_8 = z_9$ (computing $X^3 + 2X$)

Plonk Circuit Encoding

Wire assignment: $z = [1, 2, 5, 15, 3, 9, 4, 6, 10]$

- Wires 1-4: public inputs $[1, 2, 5, Y]$
- Wires 5-9: witness values $[X, X^2, X^3, 2X, X^3 + 2X]$

Constraints:

1-4. Constant gates: $z_1 = 1, z_2 = 2, z_3 = 5, z_4 = 15$

5. $z_5 \cdot z_5 = z_6$ (computing X^2)
6. $z_6 \cdot z_5 = z_7$ (computing X^3)
7. $z_5 \cdot z_2 = z_8$ (computing $2X$)
8. $z_7 + z_8 = z_9$ (computing $X^3 + 2X$)
9. $z_9 + z_3 = z_4$ (final addition)

Plonk Selector Vectors and Wire Indices

Selector vectors and wire indices:

	G1	G2	G3	G4	G5	G6	G7	G8	G9
q_L	1	1	1	1	0	0	0	1	1
q_R	0	0	0	0	0	0	0	1	1
q_O	0	0	0	0	-1	-1	-1	-1	-1
q_M	0	0	0	0	1	1	1	0	0
q_C	-1	-2	-5	-15	0	0	0	0	0
a	1	2	3	4	5	6	5	7	9
b	0	0	0	0	5	5	2	8	3
c	0	0	0	0	6	7	8	9	4

Plonk Selector Vectors and Wire Indices

Selector vectors and wire indices:

	G1	G2	G3	G4	G5	G6	G7	G8	G9
q_L	1	1	1	1	0	0	0	1	1
q_R	0	0	0	0	0	0	0	1	1
q_O	0	0	0	0	-1	-1	-1	-1	-1
q_M	0	0	0	0	1	1	1	0	0
q_C	-1	-2	-5	-15	0	0	0	0	0
a	1	2	3	4	5	6	5	7	9
b	0	0	0	0	5	5	2	8	3
c	0	0	0	0	6	7	8	9	4

Copy constraints: Permutation σ ensures wire consistency $\sigma = (1)(2,16)(3,18)(4,27)(5,7,14,15)(6,23)(8,24)(9,26)(10,11,12,13,19,20,21,22)(17,25)$

Polynomial Encoding

Marlin **Plonk**

Polynomial Encoding

Marlin

Witness polynomials:

- $\hat{w}(X)$ - shifted witness

Plonk

Marlin

Witness polynomials:

- $\hat{w}(X)$ - shifted witness
- $\hat{z}(X)$ - full assignment

Plonk

Polynomial Encoding

Marlin

Witness polynomials:

- $\hat{w}(X)$ - shifted witness
- $\hat{z}(X)$ - full assignment
- $\hat{z}_A(X), \hat{z}_B(X), \hat{z}_C(X)$ - linear combinations

Plonk

Polynomial Encoding

Marlin

Witness polynomials:

- $\hat{w}(X)$ - shifted witness
- $\hat{z}(X)$ - full assignment
- $\hat{z}_A(X), \hat{z}_B(X), \hat{z}_C(X)$ - linear combinations

Matrix polynomials:

- $\hat{\text{row}}_{M^*}(X), \hat{\text{col}}_{M^*}(X), \hat{\text{val}}_{M^*}(X)$

Plonk

Polynomial Encoding

Marlin

Plonk

Witness polynomials:

- $\hat{w}(X)$ - shifted witness
- $\hat{z}(X)$ - full assignment
- $\hat{z}_A(X), \hat{z}_B(X), \hat{z}_C(X)$ - linear combinations

Matrix polynomials:

- $\hat{\text{row}}_{M^*}(X), \hat{\text{col}}_{M^*}(X), \hat{\text{val}}_{M^*}(X)$
- For $M \in \{A, B, C\}$

Polynomial Encoding

Marlin

Witness polynomials:

- $\hat{w}(X)$ - shifted witness
- $\hat{z}(X)$ - full assignment
- $\hat{z}_A(X), \hat{z}_B(X), \hat{z}_C(X)$ - linear combinations

Matrix polynomials:

- $\hat{\text{row}}_{M^*}(X), \hat{\text{col}}_{M^*}(X), \hat{\text{val}}_{M^*}(X)$
- For $M \in \{A, B, C\}$

Plonk

Polynomial Encoding

Marlin

Witness polynomials:

- $\hat{w}(X)$ - shifted witness
- $\hat{z}(X)$ - full assignment
- $\hat{z}_A(X), \hat{z}_B(X), \hat{z}_C(X)$ - linear combinations

Matrix polynomials:

- $\hat{\text{row}}_{M^*}(X), \hat{\text{col}}_{M^*}(X), \hat{\text{val}}_{M^*}(X)$
- For $M \in \{A, B, C\}$

Plonk

Wire polynomials:

- $a(X), b(X), c(X)$ - left, right, output

Polynomial Encoding

Marlin

Witness polynomials:

- $\hat{w}(X)$ - shifted witness
- $\hat{z}(X)$ - full assignment
- $\hat{z}_A(X), \hat{z}_B(X), \hat{z}_C(X)$ - linear combinations

Matrix polynomials:

- $\hat{\text{row}}_{M^*}(X), \hat{\text{col}}_{M^*}(X), \hat{\text{val}}_{M^*}(X)$
- For $M \in \{A, B, C\}$

Plonk

Wire polynomials:

- $a(X), b(X), c(X)$ - left, right, output

Selector polynomials:

- $q_L(X), q_R(X), q_O(X)$ - linear

Polynomial Encoding

Marlin

Witness polynomials:

- $\hat{w}(X)$ - shifted witness
- $\hat{z}(X)$ - full assignment
- $\hat{z}_A(X), \hat{z}_B(X), \hat{z}_C(X)$ - linear combinations

Matrix polynomials:

- $\hat{\text{row}}_{M^*}(X), \hat{\text{col}}_{M^*}(X), \hat{\text{val}}_{M^*}(X)$
- For $M \in \{A, B, C\}$

Plonk

Wire polynomials:

- $a(X), b(X), c(X)$ - left, right, output

Selector polynomials:

- $q_L(X), q_R(X), q_O(X)$ - linear
- $q_M(X)$ - multiplication

Polynomial Encoding

Marlin

Witness polynomials:

- $\hat{w}(X)$ - shifted witness
- $\hat{z}(X)$ - full assignment
- $\hat{z}_A(X), \hat{z}_B(X), \hat{z}_C(X)$ - linear combinations

Matrix polynomials:

- $\hat{\text{row}}_{M^*}(X), \hat{\text{col}}_{M^*}(X), \hat{\text{val}}_{M^*}(X)$
- For $M \in \{A, B, C\}$

Plonk

Wire polynomials:

- $a(X), b(X), c(X)$ - left, right, output

Selector polynomials:

- $q_L(X), q_R(X), q_O(X)$ - linear
- $q_M(X)$ - multiplication
- $q_C(X)$ - constant

Polynomial Encoding

Marlin

Witness polynomials:

- $\hat{w}(X)$ - shifted witness
- $\hat{z}(X)$ - full assignment
- $\hat{z}_A(X), \hat{z}_B(X), \hat{z}_C(X)$ - linear combinations

Matrix polynomials:

- $\hat{\text{row}}_{M^*}(X), \hat{\text{col}}_{M^*}(X), \hat{\text{val}}_{M^*}(X)$
- For $M \in \{A, B, C\}$

Plonk

Wire polynomials:

- $a(X), b(X), c(X)$ - left, right, output

Selector polynomials:

- $q_L(X), q_R(X), q_O(X)$ - linear
- $q_M(X)$ - multiplication
- $q_C(X)$ - constant

Permutation polynomials:

- $S_{\sigma_1}(X), S_{\sigma_2}(X), S_{\sigma_3}(X)$

Polynomial Identity Testing

Marlin

Entry-wise product constraint:

$$\hat{z}_A(X)\hat{z}_B(X) - \hat{z}_C(X) = h_0(X)v_H(X)$$

First sumcheck relation:

$$s(X) + r(\alpha, X)\sum_M \eta_M \hat{z}_M(X) - t(X)\hat{z}(X) = h_1(X)v_H(X) + Xg_1(X)$$

Second sumcheck relation:

$$a(X) - b(X)q_2(X) = h_2(X)v_K(X)$$

Plonk

Gate constraint:

$$q_L(X)a(X) + q_R(X)b(X) + q_O(X)c(X) + q_M(X)a(X)b(X) + q_C(X) + (X) = h_0(X)v_H(X)$$

Permutation first:

$$L_1(X)(Z(X) - 1) = q_1(X)v_H(X)$$

Permutation second:

$$Z(X)f'(X) - g'(X)Z(X) = q_2(X)v_H(X)$$

Marlin vs Plonk: Performance Comparison

Metric	Marlin	Plonk
Constraint System	R1CS	Custom gates
SRS degree	$6m$	n
Proof size ($\mathbb{F}_q$)	8	**6**
Proof size ($\mathbb{G}_1$)	13	**9**
Prover v-MSM operations	11	**7**
Verifier field operations	$O(\ell + \log m)$	$O(\ell + \log n)$

Marlin vs Plonk: Performance Comparison

Metric	Marlin	Plonk
Constraint System	R1CS	Custom gates
SRS degree	$6m$	n
Proof size ($\mathbb{F}_q$)	8	**6**
Proof size ($\mathbb{G}_1$)	13	**9**
Prover v-MSM operations	11	**7**
Verifier field operations	$O(\ell + \log m)$	$O(\ell + \log n)$

m = sparse matrix domain, n = number of gates, ℓ = public inputs

Marlin vs Plonk: Performance Comparison

Metric	Marlin	Plonk
Constraint System	R1CS	Custom gates
SRS degree	$6m$	n
Proof size ($\mathbb{F}_q$)	8	**6**
Proof size ($\mathbb{G}_1$)	13	**9**
Prover v-MSM operations	11	**7**
Verifier field operations	$O(\ell + \log m)$	$O(\ell + \log n)$

m = sparse matrix domain, n = number of gates, ℓ = public inputs

Choose Marlin when:

- High fan-in addition gates

Choose Plonk when:

Marlin vs Plonk: Performance Comparison

Metric	Marlin	Plonk
Constraint System	R1CS	Custom gates
SRS degree	$6m$	n
Proof size ($\mathbb{F}_q$)	8	**6**
Proof size ($\mathbb{G}_1$)	13	**9**
Prover v-MSM operations	11	**7**
Verifier field operations	$O(\ell + \log m)$	$O(\ell + \log n)$

m = sparse matrix domain, n = number of gates, ℓ = public inputs

Choose Marlin when:

- High fan-in addition gates
- Existing R1CS circuits

Choose Plonk when:

Marlin vs Plonk: Performance Comparison

Metric	Marlin	Plonk
Constraint System	R1CS	Custom gates
SRS degree	$6m$	n
Proof size ($\mathbb{F}_q$)	8	**6**
Proof size ($\mathbb{G}_1$)	13	**9**
Prover v-MSM operations	11	**7**
Verifier field operations	$O(\ell + \log m)$	$O(\ell + \log n)$

m = sparse matrix domain, n = number of gates, ℓ = public inputs

Choose Marlin when:

- High fan-in addition gates
- Existing R1CS circuits

Choose Plonk when:

- General-purpose circuits

Marlin vs Plonk: Performance Comparison

Metric	Marlin	Plonk
Constraint System	R1CS	Custom gates
SRS degree	$6m$	n
Proof size ($\mathbb{F}_q$)	8	**6**
Proof size ($\mathbb{G}_1$)	13	**9**
Prover v-MSM operations	11	**7**
Verifier field operations	$O(\ell + \log m)$	$O(\ell + \log n)$

m = sparse matrix domain, n = number of gates, ℓ = public inputs

Choose Marlin when:

- High fan-in addition gates
- Existing R1CS circuits

Choose Plonk when:

- General-purpose circuits
- Smaller proof size critical

Marlin vs Plonk: Performance Comparison

Metric	**Marlin**	**Plonk**
Constraint System	R1CS	Custom gates
SRS degree	$6m$	n
Proof size ($\mathbb{F}_q$)	8	**6**
Proof size ($\mathbb{G}_1$)	13	**9**
Prover v-MSM operations	11	**7**
Verifier field operations	$O(\ell + \log m)$	$O(\ell + \log n)$

m = sparse matrix domain, n = number of gates, ℓ = public inputs

Choose Marlin when:

- High fan-in addition gates
- Existing R1CS circuits

Choose Plonk when:

- General-purpose circuits
- Smaller proof size critical

Both achieve universal & updatable SRS via KZG!

Implementation Architecture

Why SageMath?

Implementation Architecture

Why SageMath?

- Built-in finite field arithmetic: `GF(p)`

Implementation Architecture

Why SageMath?

- Built-in finite field arithmetic: `GF(p)`
- Native polynomial operations: `R.<x> = PolynomialRing(GF(p), 'x')`

Implementation Architecture

Why SageMath?

- Built-in finite field arithmetic: `GF(p)`
- Native polynomial operations: `R.<x> = PolynomialRing(GF(p), 'x')`
- Elliptic curve support: `EllipticCurve(GF(p), [a,b])`

Implementation Architecture

Why SageMath?

- Built-in finite field arithmetic: `GF(p)`
- Native polynomial operations: `R.<x> = PolynomialRing(GF(p), 'x')`
- Elliptic curve support: `EllipticCurve(GF(p), [a,b])`
- Used extensively in cryptography research

Implementation Architecture

Why SageMath?

- Built-in finite field arithmetic: `GF(p)`
- Native polynomial operations: `R.<x> = PolynomialRing(GF(p), 'x')`
- Elliptic curve support: `EllipticCurve(GF(p), [a,b])`
- Used extensively in cryptography research
- Educational clarity over performance

Contributions Summary

1. **Self-contained mathematical exposition**

Contributions Summary

1. **Self-contained mathematical exposition**
   - All concepts built from first principles

Contributions Summary

1. **Self-contained mathematical exposition**
    - All concepts built from first principles
    - Extensive worked examples over small fields

Contributions Summary

1. **Self-contained mathematical exposition**
   - All concepts built from first principles
   - Extensive worked examples over small fields
   - Clear progression from basics to advanced

Contributions Summary

1. **Self-contained mathematical exposition**
   - All concepts built from first principles
   - Extensive worked examples over small fields
   - Clear progression from basics to advanced

2. **Complete protocol implementations**

Contributions Summary

1. **Self-contained mathematical exposition**
   - All concepts built from first principles
   - Extensive worked examples over small fields
   - Clear progression from basics to advanced
2. **Complete protocol implementations**
   - Both Marlin and Plonk in SageMath

Contributions Summary

1. **Self-contained mathematical exposition**
   - All concepts built from first principles
   - Extensive worked examples over small fields
   - Clear progression from basics to advanced
2. **Complete protocol implementations**
   - Both Marlin and Plonk in SageMath
   - Following theoretical constructions exactly

Contributions Summary

1. **Self-contained mathematical exposition**
   - All concepts built from first principles
   - Extensive worked examples over small fields
   - Clear progression from basics to advanced

2. **Complete protocol implementations**
   - Both Marlin and Plonk in SageMath
   - Following theoretical constructions exactly
   - With extensive documentation

Contributions Summary

1. **Self-contained mathematical exposition**
   - All concepts built from first principles
   - Extensive worked examples over small fields
   - Clear progression from basics to advanced
2. **Complete protocol implementations**
   - Both Marlin and Plonk in SageMath
   - Following theoretical constructions exactly
   - With extensive documentation
3. **Comparative analysis**

Contributions Summary

1. **Self-contained mathematical exposition**
   - All concepts built from first principles
   - Extensive worked examples over small fields
   - Clear progression from basics to advanced
2. **Complete protocol implementations**
   - Both Marlin and Plonk in SageMath
   - Following theoretical constructions exactly
   - With extensive documentation
3. **Comparative analysis**
   - How different designs use same primitive (KZG)

Contributions Summary

1. **Self-contained mathematical exposition**
   - All concepts built from first principles
   - Extensive worked examples over small fields
   - Clear progression from basics to advanced
2. **Complete protocol implementations**
   - Both Marlin and Plonk in SageMath
   - Following theoretical constructions exactly
   - With extensive documentation
3. **Comparative analysis**
   - How different designs use same primitive (KZG)
   - Trade-offs in performance and complexity

Contributions Summary

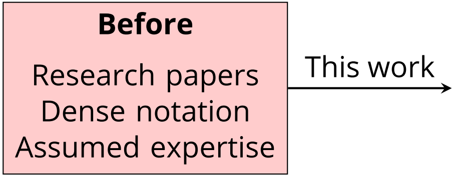
1. **Self-contained mathematical exposition**
   - All concepts built from first principles
   - Extensive worked examples over small fields
   - Clear progression from basics to advanced
2. **Complete protocol implementations**
   - Both Marlin and Plonk in SageMath
   - Following theoretical constructions exactly
   - With extensive documentation
3. **Comparative analysis**
   - How different designs use same primitive (KZG)
   - Trade-offs in performance and complexity
   - Guidance for protocol selection

Educational Impact

Before

Research papers
Dense notation
Assumed expertise

Educational Impact



Educational Impact

Before

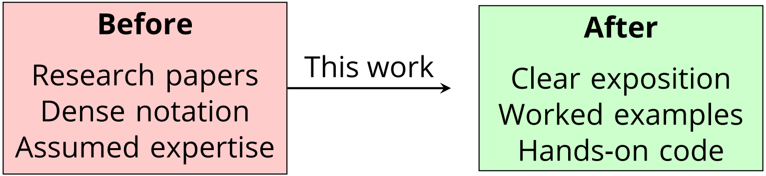
Research papers
Dense notation
Assumed expertise

→ This work →

After

Clear exposition
Worked examples
Hands-on code

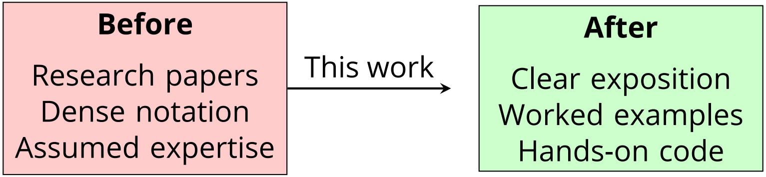
Educational Impact



Enables students to:

- Understand core concept

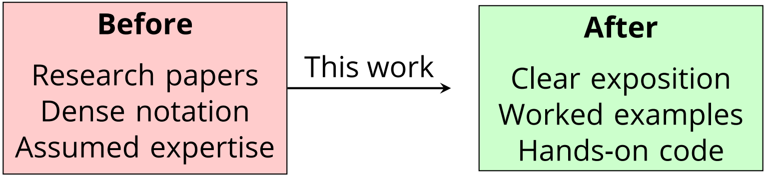
Educational Impact



Enables students to:

- Understand core concept
- Experiment with parameters and see effects

Educational Impact



Enables students to:

- Understand core concept
- Experiment with parameters and see effects
- Build foundation for advanced study in ZKPs

Thank You!

Questions?

mohammad.ferry@ui.ac.id