Zero-Knowledge Proofs: An Introduction

Wenping Hou

July 7, 2025

Outline

Outline

What is a Zero-Knowledge Proof?

A **zero-knowledge proof (ZKP)** is an interactive (or non-interactive) protocol in which a *prover* P convinces a *verifier* V that a statement $x \in L$ (for some NP language L) holds, while revealing *nothing* beyond the validity of x.
Typical uses:

- Password-less authentication (prove knowledge of secret key).
- Private compliance proofs (age, solvency, credentials).
- Blockchain validity & privacy (zk-rollups, shielded transfers).

Two flavours: **interactive** (multi-round) and **non-interactive** (Fiat–Shamir or setup CRS).

Completeness, Soundness, Simulator   Zero Knowledge

Completeness If P is honest and knows a witness w, V accepts.

Soundness Without valid witness, any cheating P^* convinces V with probability $\leq \varepsilon(\lambda)$.

Zero Knowledge There exists a **Simulator** S that, without access to w, produces a transcript indistinguishable from a real interaction. Thus V gains no knowledge beyond "x is true."

Simulator intuition. S *rewinds* the verifier, guesses (or programs) its random challenges, and forges consistent messages. If V cannot tell whether it talked to P or S, the protocol is zero-knowledge.

Color-Blind Ball Example — Problem & Protocol

Problem. Prover holds two visually identical balls, one red and one green. Verifier is color-blind and doubts they differ.
Goal. Convince verifier the balls have different colors *without* revealing which is which (zero knowledge).

Interactive protocol (repeat k rounds).

1. **Commit** — P hands both balls to V.
2. **Challenge** — V hides the balls behind his back and with prob. 1/2 swaps them.
3. **Response** — V shows the balls; P states "swapped / not swapped."
4. **Verification** — V accepts if answer correct.

Analysis. If colors differ, success prob. $= 1$. If identical, $\Pr[\text{cheat}] = 2^{-k}$. V learns nothing about the actual colors—their positions reveal no extra information—so the protocol is zero-knowledge.

Every NP Language Has a ZK Proof (GMW '86)

Under the existence of one-way functions, *any* NP statement admits an interactive ZKP. The high-level construction:

- Reduce the NP relation to *Circuit-SAT*.
- Use bit-commitments + random challenges so that a simulator can rewind.

Practical systems (SNARKs/STARKs) build on this foundation to achieve succinctness and efficiency.

Graph 3-Coloring — Problem Statement

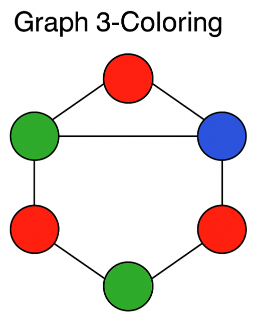
Decision problem. Given an undirected graph $G = (V, E)$, does there exist a mapping $\chi : V \to \{\mathrm{r}, \mathrm{g}, \mathrm{b}\}$ such that $\chi(u) \neq \chi(v)$ for all $(u, v) \in E$?
Fact. 3-Coloring is NP-complete; witness = a proper coloring.

For k rounds repeat:

1. **Commit** — Prover randomly permutes color labels (r g b) and commits each vertex color (e.g. envelopes or Pedersen commitments).
2. **Challenge** — Verifier selects random edge (u, v).
3. **Response** — Prover opens colors of u and v.
4. **Check** — Verifier ensures revealed colors differ.

Soundness. Cheating probability $(2/3)^k$.

Zero knowledge. Fresh permutation each round hides global coloring; simulator commits with random colors then programs the challenge edge via rewind.



Σ Protocol

Three-move structure for relation $R(x, w)$:

1. *Commit* $t \leftarrow \mathrm{P}(x, w)$
2. *Challenge* $c \leftarrow \mathrm{V}$ (uniform in $\mathcal{C}$)
3. *Response* $s \leftarrow \mathrm{P}(t, c, w)$

Key properties

- *Special soundness*: two accepting transcripts with identical t but $c \neq c' \Rightarrow$ extractor computes w.

Sigma protocols are building blocks for many practical ZK systems (e.g. Schnorr, Plonk).

Schnorr Protocol

Group assumption. Finite cyclic group $\mathbb{Z}_p^*$ of order q with generator g (Discrete-Log hard). Secret $x \in \mathbb{Z}_q^*$, public key $y = g^x \ mod\ p$.

Goal proof of knowledge of x without releasing any more information

Protocol steps

1. **Commit** — Prover picks $r \leftarrow \mathbb{Z}_q^*$, sends $t = g^r \ mod\ p$.
2. **Challenge** — Verifier samples $c \leftarrow \mathbb{Z}_q^*$ uniformly.
3. **Response** — Prover returns $s = r + cx \bmod q$.
4. **Verify** — Accept if $g^s = t\, y^c \ mod\ p$.

Schnorr — Completeness · Soundness · Simulator (HVZK)

Completeness — $g^x = g^{r+cx} = t\, y^c$.
Soundness — Probability that P^* cheats an honest verifier: $1/q$ - negligible.
Special soundness — Two valid (t, c, s) & $(t, c', s') \Rightarrow$
$x = (s - s')(c - c')^{-1} \bmod q$.
Simulator for HVZK

1. Choose $s \leftarrow \mathbb{Z}_q$ *first*.
2. Choose $c \leftarrow \mathbb{Z}_q$ *second*.
3. Compute $t = g^s y^{-c}$ so the verify equation holds.

Thus (t, c, s) indistinguishable from a real transcript when verifier is *honest* (uniform c). If verifier chooses c adaptively, indistinguishability no longer holds protocol offers honest-verifier ZK only.

Fiat–Shamir Transform — From Interactive to Non-interactive

Idea. Replace verifier's random challenge with the output of a cryptographic hash:

$$c = H(\text{statement} \parallel t)$$

Why it works.

- In the *Random Oracle Model*, H behaves like a public, unpredictable coin flip, giving the prover a way to sample c deterministically yet pseudo-randomly.
- Anyone can recompute c turns 3-move Sigma protocol into a single message proof / signature.
- Security reduction: forging a non-interactive proof implies breaking the original interactive protocol or programming the random oracle as hard as the underlying assumption (DLog).

Example. Applying Fiat–Shamir to Schnorr yields the widely-deployed Schnorr digital signature scheme (Taproot, EdDSA-variants).

Outline

zk-SNARK Definition

A **zk-SNARK** is a **Succinct**, **Non-interactive**, **Argument** of **Knowledge** that is also **Zero-Knowledge**.

- **Succinct** – proofs are tiny and verification time is poly-logarithmic (often constant) in the size of the computation.
- **Non-interactive** – a single message proof (no back-and-forth); achieved through a trusted CRS or Fiat–Shamir.
- **Argument of Knowledge** – if a prover convinces the verifier, there exists an extractor that can recover a valid witness (computational soundness).
- **Zero-Knowledge** – the proof reveals nothing about the witness beyond the fact that it exists and satisfies the statement.

Why zk-SNARKs?

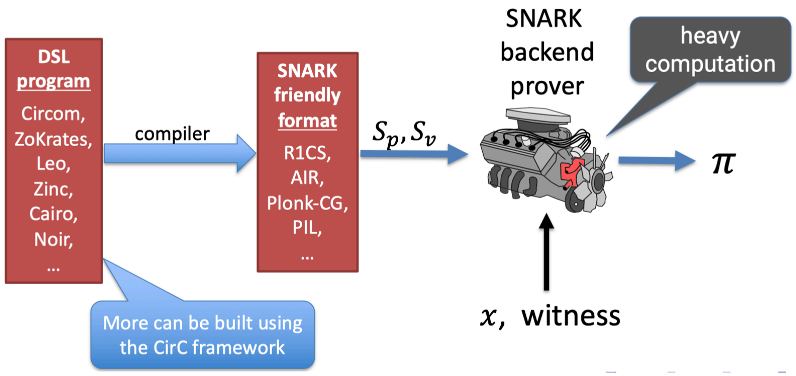
- **Scalable verification**: quickly verify large computations without re-execution – essential for blockchains and verifiable cloud computation.
- **Privacy**: proofs reveal nothing about the witness (e.g. private transactions in Zcash).
- **Applications**: zk-Rollups, private asset proofs, compliant confidentiality, verifiable machine learning, ...

Basic Requirements

We need to construct a zk-SNARK protocol that satisfies:
- **Completeness**: An honest prover can always generate a valid proof
- **Soundness**: A malicious prover cannot generate a valid proof for a false statement
- **Zero-Knowledge**: The proof does not leak witness information
- **Succinctness**: Proof size and verification time are independent of circuit size

Common SNARK Template

- Convert program to arithmetic circuit
- Convert arithmetic circuit to polynomial
- Build an argument to prove something about the poly.
- Add on other features generically, e.g., zk



Rank-1 Constraint System

Each constraint j:

$$(\mathbf{a}_j \cdot \mathbf{w}) \times (\mathbf{b}_j \cdot \mathbf{w}) = (\mathbf{c}_j \cdot \mathbf{w}).$$

Witness vector $\mathbf{w} = (1, \text{inputs}, \text{aux})$ satisfies all m constraints $\iff$ circuit correct.

What is a constraint?
A SNARK verifier does not "run" a program—it checks that *output values satisfy predetermined relations*. A collection of such relations forms a **Rank-1 Constraint System (R1CS)**.

- **Booleanity:** enforce $a \in \{0, 1\}$ via $a(a-1) = 0$.
- **Constant equality:** enforce $a = 2$ via $(a-2) \cdot 1 = 0$.
- **Bit-range (4-bit) check:**
  $a = b_3 2^3 + b_2 2^2 + b_1 2^1 + b_0$, with $b_i(b_i - 1) = 0$.

Complex relations are decomposed into such quadratic constraints.

QAP Satisfiability

Given a QAP instance, we need to prove the existence of a witness such that:

$$A(X) \cdot B(X) - C(X) = H(X) \cdot t(X)$$

where:

$$A(X) = \sum_{i=0}^{n} a_i \cdot u_i(X) \tag{1}$$

$$B(X) = \sum_{i=0}^{n} a_i \cdot v_i(X) \tag{2}$$

$$C(X) = \sum_{i=0}^{n} a_i \cdot w_i(X) \tag{3}$$

Non-Interactive Scheme Design

To achieve non-interactivity, we need to "bake in" all necessary random challenges during the trusted setup.
Key Insight: Using the properties of bilinear pairings, we can perform polynomial operations in the exponent without exposing the polynomials themselves.

Naive Approach

For the QAP relation:

$$A(X) \cdot B(X) - C(X) = H(X) \cdot t(X)$$

The most straightforward idea is to verify:

$$A(\tau) \cdot B(\tau) - C(\tau) = H(\tau) \cdot t(\tau)$$

Based on bilinear pairings,

$$e(g^{A(\tau)}, g^{B(\tau)}) = e(g^{C(\tau)} \cdot g^{H(\tau) \cdot t(\tau)}, g)$$

Problem: This approach has serious soundness issues:

- The prover may not know the polynomials corresponding to $A(\tau)$, $B(\tau)$, $C(\tau)$
- Cannot guarantee that A, B, C have the correct structure

Structured Constraints

We need to constrain the form of A, B, C. Introduce the idea of knowledge extraction:
First Improvement: Using α-shift technique

$$\pi_A = g^{\alpha + A(\tau)} \quad (4)$$
$$\pi_B = g^{\beta + B(\tau)} \quad (5)$$

Verification:

$$e(\pi_A / g^{\alpha}, \pi_B / g^{\beta}) = e(g^{C(\tau)} \cdot g^{H(\tau) \cdot t(\tau)}, g)$$

This way, the prover must "know" $A(\tau)$, $B(\tau)$ to compute the correct π_A, π_B.

Problem Identification

Even with α-shift, the prover can still:

- Use incorrect coefficient combinations
- Not follow the linear structure of QAP

We need to ensure that $A(X)$, $B(X)$, $C(X)$ are indeed correct linear combinations of $u_i(X)$, $v_i(X)$, $w_i(X)$.

Second Improvement: Introduce linear combination check
For each i, include in the preprocessing parts:

$$g^{\beta u_i(\tau) + \alpha v_i(\tau) + w_i(\tau)}$$

This way, the prover must use consistent coefficients $\{a_i\}$.

Public Input Handling

The first ℓ variables are public, and the verifier should be able to use them directly.
Third Improvement: Separate verification

$$e(g^{A(\tau)}, g^{B(\tau)}) = e(g^{C(\tau)}, g) \cdot e(g^{H(\tau)}, g^{t(\tau)}) \quad (6)$$
$$\pi_A = g^{\alpha + A(\tau)} \quad (7)$$
$$\pi_B = g^{\beta + B(\tau)} \quad (8)$$

Introducing Grouping Parameters

Use different "grouping" parameters γ, δ to separate public and private parts:

- γ for public inputs
- δ for private inputs

Include in the setup:

$$g^{\frac{\beta u_i(\tau)+\alpha v_i(\tau)+w_i(\tau)}{\gamma}} \text{ (for public inputs)}$$

$$g^{\frac{\beta u_i(\tau)+\alpha v_i(\tau)+w_i(\tau)}{\delta}} \text{ (for private witness)}$$

Information Leakage Problem

The current construction still leaks information about the witness.
Fourth Improvement: Introduce randomization

$$\pi_A = g^{\alpha+A(\tau)+r\cdot\delta} \tag{9}$$

$$\pi_B = g^{\beta+B(\tau)+s\cdot\delta} \tag{10}$$

where r, s are random numbers.

Maintaining Verification Equation

After randomization, we need to adjust π_C to maintain the balance of the verification equation:
Derivation Process: Expanding the target equation:

$$(\alpha + A(\tau) + r\delta)(\beta + B(\tau) + s\delta) =?$$

The right side should equal:

$$\alpha\beta + [\text{public part}] + [\text{private part}] + [\text{randomization compensation terms}]$$

To balance the randomization terms, π_C must include:

$$A(\tau) \cdot s + B(\tau) \cdot r - r \cdot s \cdot \delta$$

Complete Form of Verification Equation

The verification equation decomposes into:

$$e(\pi_A, \pi_B) = e(g^\alpha, g^\beta) \cdot e(\mathrm{IC}, g^\gamma) \cdot e(\pi_C, g^\delta)$$

where:

$$\mathrm{IC} = \sum_{i=0}^{\ell} a_i \cdot g^{\frac{\beta u_i(\tau)+\alpha v_i(\tau)+w_i(\tau)}{\gamma}} \tag{11}$$

$$= g^{\frac{\sum_{i=0}^{\ell} a_i(\beta u_i(\tau)+\alpha v_i(\tau)+w_i(\tau))}{\gamma}} \tag{12}$$

- Public part: $\sum_{i=0}^{\ell} a_i \cdot (\beta u_i(\tau) + \alpha v_i(\tau) + w_i(\tau))$
- Private part: $\sum_{i=\ell+1}^{n} a_i \cdot (\beta u_i(\tau) + \alpha v_i(\tau) + w_i(\tau))$

Complete Form Derivation of π_C – Steps 1-3

Step 1: Start from verification equation

$$e(\pi_A, \pi_B) = e(g^\alpha, g^\beta) \cdot e(\mathrm{IC}, g^\gamma) \cdot e(\pi_C, g^\delta)$$

Step 2: Expand left side

$$e(g^{\alpha+A(\tau)+r\delta}, g^{\beta+B(\tau)+s\delta})$$

Step 3: Expand product

$$= e(g, g)^{(\alpha+A(\tau)+r\delta)(\beta+B(\tau)+s\delta)}$$

Complete Form Derivation of π_C – Steps 4-6

Step 4: Expand right side

$$e(g^\alpha, g^\beta) \cdot e(\mathrm{IC}, g^\gamma) \cdot e(\pi_C, g^\delta) \tag{13}$$

$$= e(g, g)^{\alpha\beta} \cdot e(g, g)^{\frac{\text{IC exponent}}{\gamma}} \cdot e(g, g)^{\frac{\pi_C \text{ exponent}}{\delta}} \tag{14}$$

Step 5: Match coefficients For the equation to hold, the exponent of π_C must satisfy a specific form.
Step 6: Utilize QAP relation Since $A(\tau) \cdot B(\tau) = C(\tau) + H(\tau) \cdot t(\tau)$, we can rearrange terms.

Complete Form Derivation of π_C – Final Form

Step 7: Rearrange Using the QAP relation and coefficient matching:

$$\alpha B(\tau) + A(\tau)\beta = \sum_{i=0}^{n} a_i(\alpha v_i(\tau) + \beta u_i(\tau))$$

Step 8: Final form

$$\pi_C = g^{\frac{\sum_{i=\ell+1}^{n} a_i(\beta u_i(\tau)+\alpha v_i(\tau)+w_i(\tau))+H(\tau)\cdot t(\tau)+A(\tau)\cdot s+B(\tau)\cdot r-r\cdot s\cdot\delta}{\delta}}$$

Note that the sign adjustment here is to match the actual verification equation.

Why Choose This Grouping Method?

Role of γ grouping:

- Separates public inputs from private parts
- Allows verifier to compute public part independently
- Provides additional algebraic structure constraints

Role of δ grouping:

- Provides “carrier” for randomization
- Ensures zero-knowledge property
- Maintains balance of verification equation

Why Are Cross Terms Necessary?

The necessity of cross terms $A(\tau) \cdot s + B(\tau) \cdot r - r \cdot s \cdot \delta$:
Mathematical Necessity: When we add randomization terms $r\delta, s\delta$ to π_A, π_B, the expansion of the verification equation naturally produces cross terms. To maintain equation balance, these terms must be compensated in π_C.
Algebraic Consistency: Without these cross terms, the verification equation would fail because the left side has $(r\delta) \cdot (s\delta) = rs\delta^2$ terms, while the right side has no corresponding terms.

Why Is This Construction Optimal?

1. **Minimizes group operations**: Only needs 3 group elements
2. **Minimizes pairing count**: Only needs 3 pairing operations
3. **Maximizes security**: Provides complete zero-knowledge and knowledge extraction guarantees
4. **Maximizes efficiency**: Verification time is independent of circuit size

Groth16 Protocol – Setup

Trusted Setup Generation:

- Choose random numbers $\alpha, \beta, \gamma, \delta, \tau$
- **Proving Key**:
  - $\{g^{\tau^i}\}_{i=0}^{d}$, $\{g^{\alpha\tau^i}\}_{i=0}^{d}$, $\{g^{\beta\tau^i}\}_{i=0}^{d}$
  - $\{g^{\frac{\beta u_i(\tau)+\alpha v_i(\tau)+w_i(\tau)}{\delta}}\}_{i=\ell+1}^{n}$
  - $g^{\frac{t(\tau)}{\delta}}$
- **Verification Key**:
  - $g^{\alpha}, g^{\beta}, g^{\gamma}, g^{\delta}$
  - $\{g^{\frac{\beta u_i(\tau)+\alpha v_i(\tau)+w_i(\tau)}{\gamma}}\}_{i=0}^{\ell}$

Destroy: α, β, γ, δ, τ must be destroyed.

Groth16 Protocol – Proving

Prover Computation:

1. Choose random numbers $r, s \in \mathbb{F}_q$
2. Compute polynomials $A(X), B(X), C(X), H(X)$
3. Generate proof:

$$\pi_A = g^{\alpha+A(\tau)+r\delta} \tag{15}$$
$$\pi_B = g^{\beta+B(\tau)+s\delta} \tag{16}$$
$$\pi_C = g^{\frac{C(\tau)+H(\tau)\cdot t(\tau)}{\delta}+A(\tau)\cdot s+B(\tau)\cdot r-rs\delta} \tag{17}$$

Proof Size: Only 3 group elements (π_A, π_B, π_C), independent of circuit size!

Groth16 Protocol – Verification

Verifier Check:

1. Parse proof (π_A, π_B, π_C) and public inputs $(a_0, a_1, \ldots, a_\ell)$
2. Compute IC $= \prod_{i=0}^{\ell}(g^{\frac{\beta u_i(\tau)+\alpha v_i(\tau)+w_i(\tau)}{\gamma}})^{a_i}$
3. Verify pairing equation:

$$e(\pi_A, \pi_B) = e(g^\alpha, g^\beta) \cdot e(\mathrm{IC}, g^\gamma) \cdot e(\pi_C, g^\delta)$$

Verification Cost: 3 pairing operations + linear work (public inputs)

Summary

The mathematical formula of Groth16 is not designed out of thin air, but through the following step-by-step optimization process:

1. **Basic requirements** $\rightarrow$ QAP satisfiability proof
2. **Non-interactivity** $\rightarrow$ Bilinear pairing verification
3. **Knowledge extraction** $\rightarrow$ α, β-shift techniques
4. **Structural constraints** $\rightarrow$ Linear combination check
5. **Input separation** $\rightarrow$ γ, δ grouping
6. **Zero-knowledge** $\rightarrow$ Randomization and cross-term compensation

Each step is designed to solve the problems left by the previous step, ultimately forming a mathematically complete and efficiency-optimal zk-SNARK protocol. This derivation process demonstrates how abstract security requirements are transformed into concrete mathematical constructions in modern cryptographic protocol design.

Outline

What are zk-STARKs?

- **Zero-Knowledge Scalable Transparent Arguments of Knowledge**
- Novel zero-knowledge proof system addressing key limitations of zk-SNARKs
- Introduced by Eli Ben-Sasson et al. to eliminate trusted setup requirements
- Built on hash functions and Reed-Solomon codes rather than elliptic curve cryptography

Key Innovation

Achieve transparency, quantum resistance, and scalability by trading off proof size

Comprehensive Comparison

Feature	zk-SNARKs	zk-STARKs
Trusted Setup	Required	Not Required
Proof Size	~128-256 bytes	~40-200 KB
Verification Time	Very Fast (ms)	Fast (log time)
Generation Time	Fast	Faster + Parallel
Quantum Security	Vulnerable	Resistant
Underlying Assumption	Elliptic Curve DLog	Hash + Reed-Solomon

Trade-offs Analysis

zk-SNARKs Advantages

- Extremely compact proofs
- Fast verification via pairings
- Mature technology
- Wide adoption

zk-STARKs Advantages

- Transparent setup
- Quantum resistant
- Highly parallelizable
- No trusted third party

zk-SNARKs Disadvantages

- Trusted setup ceremony
- Quantum vulnerability
- Transparency concerns

zk-STARKs Disadvantages

- 100-1000× larger proofs
- Higher verification cost
- Technical complexity

Arithmetic Intermediate Representation (AIR)

1. **Execution trace table** $\mathcal{T}$:
   - Rows: state at each time-step
   - Columns: registers
2. **Transition constraints**: Low-degree polynomial relations:

$$P_i(\mathcal{T}[j], \mathcal{T}[j+1]) = 0$$

3. **Boundary constraints**: Fix initial/final states
4. All constraints as *low-degree polynomials* over $\mathbb{F}_p$

Step	Reg 0	Reg 1	Reg 2
0	s_0	s_1	s_2
1	s_0'	s_1'	s_2'
2	s_0''	s_1''	s_2''

Boundary

Transition Constraints

Boundary

AIR enables efficient encoding of computational integrity proofs

Low-Degree Extension (LDE)

- Given trace table $\mathcal{T}$ of size n, interpolate each column to polynomial $f_i(X)$ of degree $< n$
- Evaluate f_i on expanded coset domain $\mathbb{F}_p \setminus D$ of size $m \gg n$
- Creates redundancy needed for error detection
- Commit to LDE values with Merkle tree (root R_1)

FRI Protocol (Fast Reed-Solomon IOP)

- **Goal**: Prove function $f : D \to \mathbb{F}_p$ is ε-close to some low-degree polynomial
- **Method**: Iterative domain folding:

$$f_{k+1}(X) = \frac{f_k(X) + f_k(X\omega)}{2}$$

- After $\log |D|$ rounds, degree becomes tiny (≤ 1)
- Prover sends final polynomial coefficients; verifier checks random spots via Merkle proofs

Concrete Example: Fibonacci Sequence Verification

Problem

Prove knowledge of how to generate the n-th Fibonacci number without revealing intermediate computations

Execution Trace:

Step i	$t_{i,0}$	$t_{i,1}$
0	1	1
1	1	2
2	2	3
3	3	5
4	5	8
$\vdots$	$\vdots$	$\vdots$

Constraints:

- Transition: $t_{i+1,0} = t_{i,1}$
- Transition: $t_{i+1,1} = t_{i,0} + t_{i,1}$
- Boundary: $t_{0,0} = 1$, $t_{0,1} = 1$

Protocol Steps (1/2)

1. **Trace Construction**: Generate execution trace table $\mathcal{T}$
2. **Polynomial Interpolation**:
   - $f_0(X)$: interpolation of column 0
   - $f_1(X)$: interpolation of column 1
3. **Constraint Composition**: Create combined polynomial

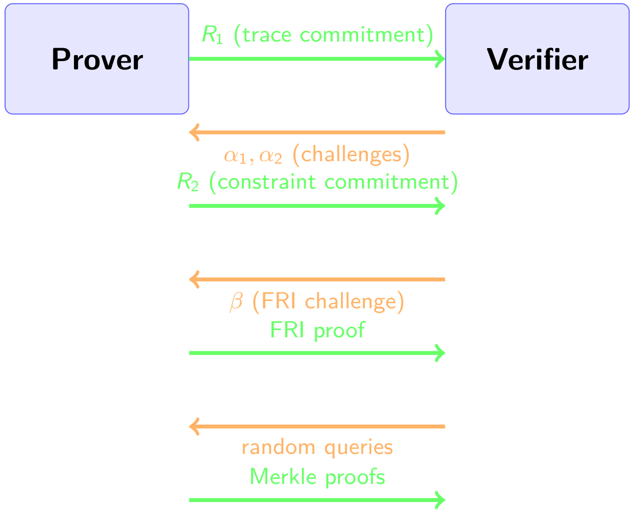
$$C(X) = \alpha_1 \cdot (f_0(\omega X) - f_1(X)) + \alpha_2 \cdot (f_1(\omega X) - f_0(X) - f_1(X))$$

   where α_1, α_2 are random challenges from verifier
4. **LDE & Commitment**:
   - Evaluate polynomials on extended domain
   - Commit via Merkle trees, send roots R_1, R_2

Protocol Steps (2/2)

5. **FRI Proof**: Prove constraint composition polynomial is low-degree
   - Iterative folding: $f_{k+1}(X) = [f_k(X) + f_k(-X)]/2$
   - Commit each round's polynomial
   - Send final polynomial coefficients
6. **Query Phase**:
   - Verifier sends random query positions
   - Prover provides Merkle proofs for all queries
7. **Verification**:
   - Check Merkle proof consistency
   - Verify FRI folding correctness
   - Validate constraint satisfaction at random points

Complete Protocol Flow



Security Guarantees

- **Completeness**: If prover knows correct Fibonacci sequence, protocol always accepts
- **Soundness**: If prover doesn't know correct sequence, protocol rejects with high probability
  - FRI ensures proximity to low-degree polynomials
  - Constraint checks bind trace to correct relation
  - Combined: guarantee execution integrity
- **Zero-Knowledge** (optional):
  - Default STARKs reveal hash commitments (not zero-knowledge)
  - Add random linear blinding + mask composition polynomial
  - Achieves zero-knowledge while preserving validity

Theoretical Complexity

- **Proof Generation**: $O(n \log n)$ where n is trace length
- **Proof Size**: $O(\log^2 n)$
- **Verification Time**: $O(\log^2 n)$
- **Parallelization**: Highly parallelizable across multiple cores/GPUs

Empirical Benchmarks

Example: SHA-256 Hash Chain

- Trace length: $n = 2^{20}$ steps
- **Proof Generation**: $\approx$ 6 minutes (single core)
- **Peak Memory**: 5 GB RAM
- **Proof Size**: 200 KB
- **Verification**: $\approx$ 3M gas (Ethereum EVM)

Fibonacci Example

- Trace length: $n = 2^{10}$
- **Generation**: $\sim$ 85 ms
- **Proof Size**: $\sim$ 60 KB
- **Verification**: $\sim$ 3 ms

Current Applications

- **Layer-2 Scaling Solutions**:
  - Starknet: Ethereum Layer-2 using STARKs
  - Polygon zkEVM: Hybrid SNARK-STARK architecture
- **Privacy-Preserving Computation**:
  - Private smart contracts in Cairo language
  - Verifiable outsourced computation
- **Blockchain Infrastructure**:
  - Recursive STARKs for proof-carrying data
  - Batch verification for transaction aggregation

What You Should Remember

- **Transparency**: STARKs eliminate trusted setup by using hash-based commitments and FRI
- **Quantum Resistance**: Built on hash functions rather than elliptic curve assumptions
- **Scalability**: Highly parallelizable proof generation with $O(n \log n)$ complexity
- **Trade-offs**: Larger proof sizes but logarithmic verification time
- **Complementary**: STARKs and SNARKs serve different use cases; choice depends on specific requirements

STARKs represent the future of transparent, quantum-resistant zero-knowledge proofs

Outline

Privacy Payments and Anonymous Transactions

- **Zcash Sapling Protocol**
  - Complete anonymity on public blockchain
  - Uses Groth16 zk-SNARKs with 192-byte proofs
  - Verification time: <10ms
  - Hides sender, receiver, and transaction amount
- **Tornado Cash Protocol**
  - Non-custodial privacy mixer using zk-SNARKs
  - Breaks transaction linkability through commitment-nullifier scheme
  - Fixed denomination pools (0.1, 1, 10, 100 ETH)

Layer 2 Scaling Solutions

- **zk-Rollups (Ethereum)**
  - Batch thousands of transactions into single proof
  - Reduces gas costs by 90-95%
  - Examples: Polygon zkEVM, zkSync Era
  - Maintains Ethereum's security while scaling to 2000+ TPS
- **StarkNet and Cairo**
  - Uses STARK proofs for transparent verification
  - No trusted setup required
  - Quantum-resistant security properties
  - Enables complex smart contract computations off-chain

Digital Identity Verification

- **Self-Sovereign Identity (SSI)**
  - Prove age without revealing exact birthdate
  - Verify citizenship without exposing personal details
  - Selective disclosure of credentials
  - Example: Proving "over 18" for services without showing ID
- **Professional Credentials**
  - Verify university degree without revealing institution
  - Prove work experience without exposing salary details
  - Medical license verification maintaining doctor privacy
  - Implementation: BBS+ signatures with ZK proofs

Biometric Authentication

- **Privacy-Preserving Biometrics**
  - Prove identity match without storing biometric data
  - Fingerprint verification using homomorphic encryption + ZK
  - Face recognition with zero-knowledge templates
  - Prevents biometric data breaches and misuse
- **Multi-Factor Authentication**
  - Combine multiple authentication factors in ZK proof
  - Prove possession of multiple credentials simultaneously
  - Examples: "Password + Biometric + Device" without revealing any private information

Regulatory Compliance

- **Anti-Money Laundering (AML)**
  - Prove transaction legitimacy without revealing amounts
  - Demonstrate compliance with transaction limits
  - Show fund sources without exposing transaction history
  - Example: Proving "funds not from sanctioned addresses"
- **Know Your Customer (KYC)**
  - Verify customer identity across multiple institutions
  - Prove creditworthiness without sharing financial details
  - Demonstrate regulatory compliance to auditors
  - Reduces redundant verification processes

Private Financial Operations

- **Confidential Trading**
  - Prove sufficient balance for trades without revealing amounts
  - Dark pool trading with privacy guarantees
  - Institutional portfolio management without exposure
  - Example: Proving "portfolio value > \$1M" for accredited investor status
- **Private Lending and Credit**
  - Prove creditworthiness without revealing income details
  - Demonstrate collateral ownership without asset disclosure
  - Private credit scoring using ZK-ML models
  - Cross-institutional credit verification

Outline

Questions & Discussion

Thank you for your attention!