Privacy & Zero-Knowledge Proofs: From Theory to Application

Standalone Technical Lecture

Prof. Dr. Joerg Osterrieder

University Lecture Series

March 5, 2026

Lecture Roadmap

Learning Objectives

- Formalise ZK proof definitions and properties
- Compare SNARK, STARK, and Bulletproof systems
- Analyse privacy coin cryptography (Monero, Zcash)
- Evaluate ZK-rollup architectures and zkEVM types
- Assess future directions in ZK technology

Prerequisites

- Lessons 1–5: Blockchain fundamentals, smart contracts, DeFi
- L12 Tokenomics recommended for economic context

Durat

90 minutes — 5 sections — ˜55 frames — Prerequisite: Lessons 1–5

Table of Contents

Navig

through 5 sections from ZK foundations to advanced topics and future directions

Learning Objectives

By the end of this lecture, you will be able to:

1. **Explain** the formal definition of zero-knowledge proofs and the simulation paradigm
2. **Compare** proof systems (Groth16, PLONK, STARKs, Bulletproofs) on setup, size, and speed
3. **Analyze** privacy coin mechanisms (ring signatures, stealth addresses, shielded pools)
4. **Evaluate** ZK applications (rollups, identity, voting, confidential DeFi, zkML)
5. **Assess** hardware acceleration and post-quantum considerations for ZK systems

Bloom

taxonomy levels: Remember → Understand → Apply → Analyze → Evaluate → Create

Section 1: ZK Proof Foundations & Mathematical Basics

Zero-knowledge concepts, interactive proofs, and mathematical foundations

Historical Context: The GMR Paper (1985)

The Breakthrough

Goldwasser, Micali, and Rackoff (1985) posed a revolutionary question: *"Can we prove a statement is true without revealing* **why** *it is true?"* Their paper "The Knowledge Complexity of Interactive Proof Systems" introduced the concept of **zero-knowledge proofs**.

Impact

This work earned the 2012 Turing Award and laid the foundation for all modern ZK systems used in blockchain today.

- **1985** GMR paper: ZK defined
- **1986** Fiat-Shamir heuristic
- **1991** IP = PSPACE
- **2012** Turing Award
- **2013** Pinocchio (SNARKs)
- **2016** Zcash launches
- **2018** STARKs paper

Goldw

Micali & Rackoff showed that proof is about more than truth – it's about knowledge transfer

Formal Definition of Zero-Knowledge Proofs

Interactive Proof System (P, V)

For a language $L \in \mathrm{NP}$ with witness relation R:

1. Completeness

If $x \in L$ and the prover P has a valid witness w:

$$\Pr[V \text{ accepts} \mid (x, w) \in R] \geq 1 - \varepsilon$$

An honest prover always convinces the verifier.

2. Soundness

If $x \notin L$, for *any* cheating prover P^*:

$$\Pr[V \text{ accepts} \mid P^*] \leq \varepsilon$$

No cheating prover can convince the verifier.

3. Zero-Knowledge

There exists a **simulator** S such that:

$$\mathrm{View}_V(P(x, w)) \approx S(x)$$

The verifier's "view" of the real interaction is *computationally indistinguishable* from a simulation that doesn't know w.

Key Insight

The simulator produces fake transcripts that look identical to real ones. If the verifier can't tell the difference, it learned **nothing** from the real interaction.

The three properties – completeness, soundness, zero-knowledge – form the foundation of all ZK systems

The Simulation Paradigm

Real World

Transcript: (a, e, z)
Prover uses witness w

Simulated World

Simulator S

Produces transcript (a', e', z')
WITHOUT knowing w!

$(a, e, z) \approx (a', e', z')$
Computationally indistinguishable

Perfect ZK

Distributions are **identical**. Information-theoretically secure.

Statistical ZK

Distributions are **statistically close**. Negligible difference.

Computational ZK

No PPT algorithm can **distinguish** them. Relies on hardness.

The simulation paradigm is the gold standard for proving zero-knowledge – if it can be simulated, nothing was learned

Commitment Schemes

Pedersen Commitment

Let $\mathbb{G}$ be a group of prime order q with generators g, h:

$$C = g^v \cdot h^r$$

where v is the committed value and $r \xleftarrow{\$} \mathbb{Z}_q$ is randomness.

Properties

- **Perfectly Hiding:** For any v, the commitment C is uniformly distributed (information-theoretic)
- **Computationally Binding:** Opening to a different value requires solving DLog (breaking $\log_g h$)

Homomorphic Property

$$C(v_1, r_1) \cdot C(v_2, r_2) = C(v_1 + v_2, r_1 + r_2)$$

Enables arithmetic on committed values!

Scheme	Hiding	Binding
Pedersen	Perfect	Computational
Hash-based	Computational	Perfect
ElGamal	Perfect	Computational

Impossibility Result

A commitment scheme cannot be both *perfectly hiding* and *perfectly binding* simultaneously. One property must rely on computational assumptions.

Vector Pedersen Commitment

For vector $\vec{v} = (v_1, \ldots, v_n)$:

$$C = h^r \cdot \prod_{i=1}^{n} g_i^{v_i}$$

Commits to n values in a single group element.

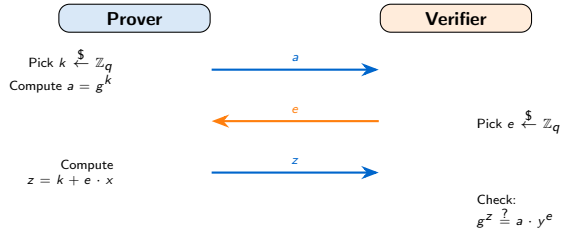
Peder

commitments are the building block of Bulletproofs, Monero's RingCT, and many ZK systems

Sigma Protocols: Schnorr's Protocol

Goal

Prove knowledge of discrete logarithm x such that $y = g^x$ without revealing x.



Verification Equation

$$g^z = g^{k+ex} = g^k \cdot g^{ex} = a \cdot (g^x)^e = a \cdot y^e$$

Properties

- **Complete:** Honest prover always passes
- **Special Sound:** Two valid transcripts with same a but different e yield x
- **SHVZK:** Special Honest-Verifier ZK via simulation

Fiat-Shamir Transform

Replace verifier's random e with $e = H(a \| m)$:

$$\text{Interactive} \xrightarrow{\text{Fiat-Shamir}} \text{Non-Interactive}$$

This is how digital signatures (EdDSA) work!

Schn

protocol is the simplest sigma protocol – it underlies EdDSA signatures and many ZK constructions

Polynomial Commitments (KZG)

Kate-Zaverucha-Goldberg Scheme

Commit to polynomial $p(x) = \sum_{i=0}^{d} c_i x^i$ using elliptic curve pairings.

Setup (Trusted)

Secret $s \in \mathbb{F}_p$, publish structured reference string:

$$\text{SRS} = \{[1]_1, [s]_1, [s^2]_1, \ldots, [s^d]_1, [1]_2, [s]_2\}$$

where $[x]_1 = x \cdot G_1$ (group $\mathbb{G}_1$ element).

Commitment

$$C = [p(s)]_1 = \sum_{i=0}^{d} c_i \cdot [s^i]_1$$

A single group element commits to the entire polynomial.

Evaluation Proof

To prove $p(z) = v$, compute quotient:

$$q(x) = \frac{p(x) - v}{x - z}$$

Proof: $\pi = [q(s)]_1$

Verification (Pairing Check)

$$e(C - [v]_1,\ [1]_2) \stackrel{?}{=} e(\pi,\ [s - z]_2)$$

This checks that $p(z) = v$ using a single pairing equation. Verification is $O(1)$.

Trusted Setup Required

If s is known, the committer can forge proofs. Multi-party ceremony ensures s is destroyed.

KZG commitments power PLONK, EIP-4844 (proto-danksharding), and most modern SNARK systems

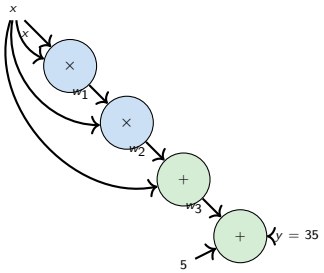
From Computation to Circuits

Any computation can be represented as a circuit of addition ($+$) and multiplication ($\times$) gates over a finite field $\mathbb{F}_p$.

Example

Prove: $x^3 + x + 5 = 35$ (without revealing $x = 3$).
Introduce intermediate variables:

$$w_1 = x \times x = 9$$
$$w_2 = w_1 \times x = 27$$
$$w_3 = w_2 + x = 30$$
$$y = w_3 + 5 = 35$$



Circuit size = number of multiplication gates. This circuit has 2 multiplication gates $\Rightarrow$ 2 constraints.

Arith

circuits are the universal representation – any NP statement can be expressed as a circuit over $\mathbb{F}_p$

From Circuits to R1CS

Rank-1 Constraint System

Each multiplication gate becomes a constraint:

$$\vec{a_i} \cdot \vec{s} \times \vec{b_i} \cdot \vec{s} = \vec{c_i} \cdot \vec{s}$$

where $\vec{s} = (1, x, w_1, w_2, w_3, y)$ is the witness vector and $\vec{a_i}, \vec{b_i}, \vec{c_i}$ are coefficient vectors.

Matrix Form

$\vec{s} = (1, x, w_1, w_2, w_3, y)$

$A \cdot \vec{s} \circ B \cdot \vec{s} = C \cdot \vec{s}$

where A, B, C are sparse matrices encoding the left input, right input, and output of each gate.

Example Constraints ($x^3 + x + 5 = 35$)

Gate 1: $x \times x = w_1$

Gate 2: $w_1 \times x = w_2$

Gate 3: $(w_2 + x) \times 1 = w_3$

Gate 4: $(w_3 + 5) \times 1 = y$

Why R1CS Matters

R1CS is the intermediate representation between:

- High-level programs (Circom, Noir)
- Low-level proof systems (Groth16, PLONK)

The compiler chain: Code $\rightarrow$ Circuit $\rightarrow$ R1CS $\rightarrow$ QAP $\rightarrow$ Proof.

R1CS

reduces any computation to a system of quadratic equations – the standard input format for SNARKs

QAP: Quadratic Arithmetic Programs

From R1CS to Polynomials

Transform each column of A, B, C matrices into polynomials via **Lagrange interpolation**:

$$A_j(x),\ B_j(x),\ C_j(x) \quad \text{for } j = 1, \ldots, m$$

where m is the number of witness variables.

QAP Equation

Define:

$$A(x) = \sum_j s_j \cdot A_j(x)$$

$$B(x) = \sum_j s_j \cdot B_j(x)$$

$$C(x) = \sum_j s_j \cdot C_j(x)$$

The R1CS is satisfied iff:

$$\boxed{A(x) \cdot B(x) - C(x) = H(x) \cdot Z(x)}$$

where $Z(x) = \prod_{i=1}^{n}(x - \omega^i)$ is the vanishing polynomial.

Key Insight

Checking n constraints in R1CS reduces to checking **one polynomial equation**. This is what makes SNARKs succinct!

Why Polynomials?

- Polynomials of degree d are determined by $d + 1$ points
- Two different polynomials of degree d agree on at most d points
- Checking at a random point: soundness error $\leq d/|\mathbb{F}|$ (Schwartz-Zippel)

Schwartz-Zippel Lemma

If $p(x) \not\equiv 0$ and $\deg(p) = d$:

$$\Pr_{r \leftarrow \mathbb{F}}[p(r) = 0] \leq \frac{d}{|\mathbb{F}|}$$

QAP

converts constraint checking into polynomial identity testing – the mathematical heart of SNARKs

Information-Theoretic vs Computational Security

Information-Theoretic Security

- Secure against **unbounded** adversaries
- No computational assumptions needed
- Security holds even with infinite computing power
- Example: one-time pad, Pedersen hiding

Computational Security

- Secure against **PPT** (probabilistic polynomial-time) adversaries
- Relies on hardness assumptions (DLog, pairings)
- Could be broken with sufficient computation
- Example: RSA, ECDSA, most ZK systems

System	ZK Type	Soundness
Schnorr	SHVZK (comp.)	Comp.
Groth16	Comp. ZK	Comp.
Bulletproofs	Perf. SHVZK	Comp.
STARKs	Comp. ZK	Info-theoretic

Post-Quantum Implications

- Computationally secure systems based on DLog/pairings are broken by quantum computers
- STARKs use hash functions only $\Rightarrow$ post-quantum secure
- SNARKs need pairing-friendly curves $\Rightarrow$ quantum-vulnerable

The security model determines what adversaries a system can withstand – critical for long-term protocol design

Section 1 Summary: ZK Foundations

Key Concepts

1. ZK proofs: completeness, soundness, zero-knowledge
2. Simulation paradigm proves zero-knowledge property
3. Pedersen commitments: perfectly hiding, computationally binding
4. Schnorr protocol: 3-move sigma protocol for DLog
5. KZG: polynomial commitment via pairings

Mathematical Toolkit

$$C_{\text{Pedersen}} = g^v \cdot h^r \tag{1}$$

$$\text{Schnorr verify:} \quad g^z = a \cdot y^e \tag{2}$$

$$C_{\text{KZG}} = [p(s)]_1 \tag{3}$$

$$\text{R1CS:} \quad \vec{a} \cdot \vec{s} \times \vec{b} \cdot \vec{s} = \vec{c} \cdot \vec{s} \tag{4}$$

$$\text{QAP:} \quad A \cdot B - C = H \cdot Z \tag{5}$$

Next Section

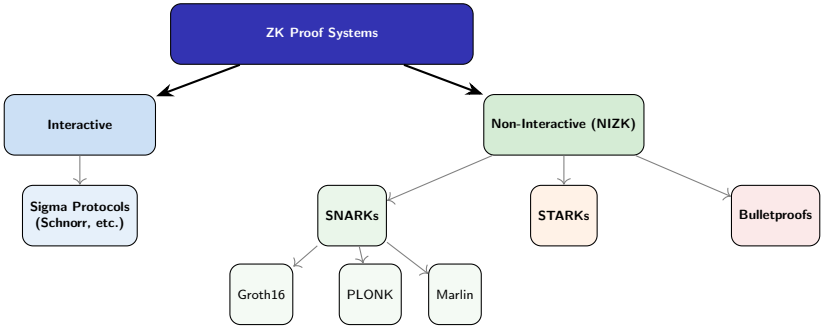
$\Rightarrow$ **Proof Systems:** Groth16, PLONK, STARKs, Bulletproofs – how they work and how they compare.

These mathematical primitives are the building blocks for all modern proof systems covered next

Section 2: Proof Systems — SNARKs, STARKs, Bulletproofs

Comparing proof systems: setup, verification, proof size, and trade-offs

Taxonomy of ZK Proof Systems



Key Differentiator	SNARKs	STARKs	Bulletproofs	Trade-off
Trusted Setup	Yes	No	No	Trust vs transparency
Proof Size	~200B	~45KB	~1.5KB	Bandwidth cost
Verification	O(1)	$O(\log^2 n)$	O(n)	On-chain cost
Post-Quantum	No	Yes	No	Future-proofing

No single system dominates – each makes different trade-offs between trust, size, and quantum resistance

Groth16: The Classic SNARK

Core Idea (Groth, 2016)

Given a QAP: $A(x) \cdot B(x) - C(x) = H(x) \cdot Z(x)$
The prover produces a proof $\pi = (A, B, C) \in \mathbb{G}_1 \times \mathbb{G}_2 \times \mathbb{G}_1$ (3 group elements, ~200 bytes).

Metric	Value
Proof size	3 group elements (~192B)
Verification	3 pairings (~1ms)
Prover time	$O(n \log n)$
CRS size	$O(n)$
Security	Comp. (q-type assumptions)

Verification Equation

$$e(A, B) = e(\alpha, \beta) \cdot e\left(\sum_i x_i \cdot [L_i]_1,\ [\gamma]_2\right) \cdot e(C, \delta)$$

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

Used By

- Zcash (Sapling circuit)
- Tornado Cash
- Filecoin
- Many early ZK projects

Per-Circuit Setup

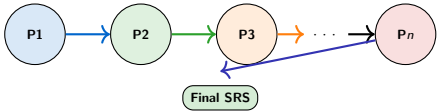
Groth16 requires a **new trusted setup for each circuit**. Change the program ⇒ new ceremony required.

Groth16 remains the most efficient SNARK for verification – but per-circuit setup is a major drawback

The Trusted Setup Ceremony

Multi-Party Computation (MPC)

1. Each participant i samples random τ_i
2. Computes $\text{SRS}_i = \text{SRS}_{i-1}^{\tau_i}$
3. Destroys τ_i (the "toxic waste")
4. Passes SRS_i to next participant



Security Guarantee

1-of-n honest: If *at least one* participant honestly destroys their τ_i, the setup is secure.

The Risk

If ALL participants collude (know combined τ), they can:

- Forge proofs for false statements
- Counterfeit tokens in Zcash
- Completely break soundness

Powers of Tau

Zcash Sprout: 6 participants (2016)
Zcash Sapling: 90 participants (2018)
Perpetual Powers of Tau: 1000+ (ongoing)

"Don

trust, verify" – except for the trusted setup, where you must trust at least one participant was honest

PLONK: Universal SNARKs

Key Innovation

Universal SRS: One trusted setup works for *all circuits* up to a maximum size. No per-circuit ceremony!

Gate Equation

PLONK uses a single arithmetic gate:

$$q_L \cdot a + q_R \cdot b + q_O \cdot c + q_M \cdot a \cdot b + q_C = 0$$

where q_L, q_R, q_O, q_M, q_C are selector polynomials.

- Addition: $q_L = q_R = 1, q_O = -1, q_M = 0$
- Multiplication: $q_M = 1, q_O = -1, q_L = q_R = 0$

Copy Constraints

Wiring between gates is enforced via **permutation polynomials**:

$$\sigma : \{1, \ldots, 3n\} \to \{1, \ldots, 3n\}$$

Grand product argument checks:

$$\prod_i \frac{f(i)}{g(\sigma(i))} = 1$$

Advantages over Groth16

- Universal setup (reusable)
- Updatable (new participants can strengthen)
- Custom gates for efficiency
- Used by: Aztec, Scroll, Polygon zkEVM

PLON

(2019) made universal SNARKs practical – one ceremony rules them all

STARKs: Transparent Proofs

Key Properties

- **No trusted setup** – uses only hash functions (public randomness)
- **Post-quantum secure** – no elliptic curves or pairings
- **Scalable prover:** quasi-linear $O(n \log n)$
- Proof size: $O(\log^2 n)$ – larger than SNARKs

Metric	Value
Proof size	45–200 KB
Verification	$O(\log^2 n)$
Prover time	$O(n \log n)$
Setup	None (transparent)
Quantum safe	Yes

Architecture

1. **Arithmetization:** Encode computation as polynomial constraints (AIR – Algebraic Intermediate Representation)
2. **Low Degree Testing:** Prove polynomial has low degree via FRI
3. **Commitment:** Merkle tree over evaluations (hash-based)

Used By

- StarkNet (Ethereum L2)
- StarkEx (dYdX, Sorare)
- Cairo programming language
- Polygon Miden

Trade-off

Larger proofs $\Rightarrow$ higher on-chain cost. Mitigated by STARK-to-SNARK wrapping ("proof compression").

STAR

trade proof size for transparency and quantum resistance – invented by Eli Ben-Sasson (StarkWare)

FRI Protocol Deep Dive

Fast Reed-Solomon IOP (FRI)

Goal: Prove that a function $f : D \to \mathbb{F}$ is "close to" a polynomial of degree $< d$.

Folding Procedure

Split polynomial into even and odd parts:

$$p(x) = p_{\text{even}}(x^2) + x \cdot p_{\text{odd}}(x^2)$$

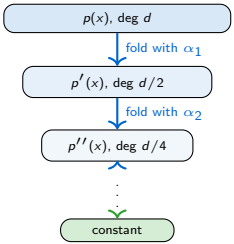
Given random α from verifier:

$$p'(y) = p_{\text{even}}(y) + \alpha \cdot p_{\text{odd}}(y)$$

New polynomial p' has **half the degree**!

Recursive Folding

Repeat $\log_2 d$ times: $d \to d/2 \to d/4 \to \cdots \to 1$
Final check: is p' a constant polynomial?



Verification

- Merkle path checks on evaluations
- Consistency checks between layers
- Total: $O(\log^2 d)$ field operations

FRI

is the polynomial commitment scheme for STARKs – it replaces pairings with hash-based Merkle proofs

Bulletproofs: No Trusted Setup, Small Proofs

Inner Product Argument

Prove knowledge of vectors $\vec{a}, \vec{b} \in \mathbb{F}^n$ such that:

$$\langle \vec{a}, \vec{b} \rangle = c \quad \text{and} \quad P = \vec{g}^{\vec{a}} \cdot \vec{h}^{\vec{b}} \cdot u^c$$

Recursive Halving

Split vectors: $\vec{a} = (\vec{a}_L, \vec{a}_R)$, $\vec{b} = (\vec{b}_L, \vec{b}_R)$
Compute cross-terms L, R and fold with challenge x:

$$\vec{a}' = x \cdot \vec{a}_L + x^{-1} \cdot \vec{a}_R$$
$$\vec{b}' = x^{-1} \cdot \vec{b}_L + x \cdot \vec{b}_R$$

After $\log_2 n$ rounds: vectors reduce to scalars.

Proof Size

$2 \log_2 n$ group elements $\approx O(\log n)$
For $n = 64$ (range proof): ≈ 672 bytes.

Range Proofs

Prove $v \in [0, 2^n)$ without revealing v:

- Express v in binary: $v = \sum_i b_i \cdot 2^i$
- Prove each $b_i \in \{0, 1\}$
- Inner product argument on binary decomposition

Used By

- Monero (confidential amounts)
- Mimblewimble (Grin, Beam)

Bulle

no trusted setup, logarithmic proof size, but linear verification – ideal for range proofs in privacy coins

Proof Systems Comparison

Property	Groth16	PLONK	STARKs	Bulletproofs
Trusted Setup	Per-circuit	Universal	None	None
Proof Size	˜200B	˜400B	45–200KB	$O(\log n)$
Verification	O(1), 3 pairings	O(1), 1 pairing	$O(\log^2 n)$	$O(n)$
Prover Time	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	$O(n^2)$
Quantum Safe	No	No	Yes	No
Assumptions	q-type	AGM + ROM	CRHF only	DLog
Used By	Zcash, Filecoin	Aztec, Scroll	StarkNet, dYdX	Monero, Grin

Choosing a System

- Minimise on-chain cost → Groth16 or PLONK
- No trust assumptions → STARKs
- Range proofs only → Bulletproofs
- Future-proof → STARKs

Hybrid Approaches

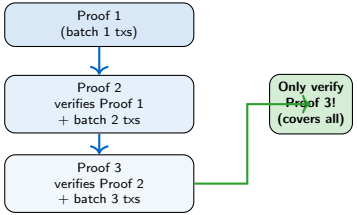
- **STARK-to-SNARK:** Prove STARK, wrap in SNARK for small on-chain proof (Polygon zkEVM)
- **Recursive SNARKs:** Prove verification of previous proof (Nova, Halo)

The "proof system wars" are driving rapid innovation – hybrid approaches combine the best of each

Recursive Proofs & Proof Composition

Concept

A **recursive proof** proves that another proof is valid inside itself. The verifier of proof n becomes part of the circuit for proof $n + 1$.



IVC (Incrementally Verifiable Computation)

$$\pi_n = \text{Prove}(\text{Verify}(\pi_{n-1}) \wedge F(s_{n-1}) = s_n)$$

Each step proves both the computation AND the validity of the previous proof.

Nova Folding Scheme

Instead of full recursion, **fold** instances together:

- No expensive verification inside the circuit
- Linear-time folding operation
- Final proof at the end only
- Dramatically reduces prover cost

Applications

Blockchain light clients, rollup proof aggregation, long-running computations.

proofs enable unbounded computation verification with constant-size proofs – the holy grail of ZK

ZK Proof Verification in Solidity

```solidity
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.20;

contract Groth16Verifier {
    // Verification key (set during deployment)
    uint256[2] public alfa1;    // G1 point
    uint256[2][2] public beta2; // G2 point
    uint256[2][2] public gamma2;
    uint256[2][2] public delta2;

    struct Proof {
        uint256[2] a;    // G1
        uint256[2][2] b; // G2
        uint256[2] c;    // G1
    }

    // Verify a Groth16 proof via ecPairing precompile
    function verifyProof(
        Proof memory proof,
        uint256[] memory pubInputs
    ) public view returns (bool) {
        // Compute linear combination of public inputs
        uint256[2] memory vk_x = computeLinearCombination(pubInputs);
        // Pairing check: e(A, B) == e(alpha, beta) * e(vk_x, gamma) * e(C, delta)
        // Uses EVM precompile at address 0x08
        return pairingCheck(proof, vk_x);
    }

    function pairingCheck(Proof memory p, uint256[2] memory vk_x)
        internal view returns (bool) {
        // Encode 4 pairing pairs and call ecPairing precompile
        // Returns true if product of pairings equals 1 in G_T
        assembly { /* pairing precompile call at 0x08 */ }
        return true; // simplified
    }
}
```

Section 2 Summary: Proof Systems

Key Concepts

1. Groth16: smallest proofs, per-circuit setup
2. PLONK: universal setup, custom gates
3. STARKs: transparent, post-quantum, hash-based
4. Bulletproofs: no setup, log-size, range proofs
5. FRI: degree testing via polynomial folding
6. Recursive proofs: verify proofs inside proofs

Design Decision Guide

- On-chain verification cost critical? $\rightarrow$ SNARK
- No trust assumptions? $\rightarrow$ STARK
- Quantum resistance needed? $\rightarrow$ STARK
- Privacy coin amounts? $\rightarrow$ Bulletproofs
- Flexibility + reusable setup? $\rightarrow$ PLONK

Next Section

$\Rightarrow$ **Privacy Coins:** How Monero and Zcash use these proof systems for transaction privacy.

The proof system landscape is evolving rapidly – new systems like Lasso and HyperNova push boundaries further

Section 3: Privacy Coins

Monero, Zcash, and privacy-preserving transaction mechanisms

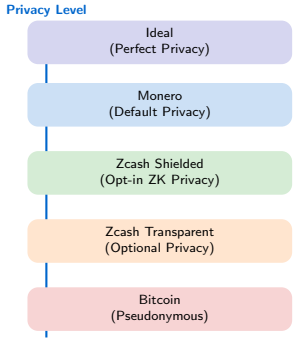
The Privacy Spectrum in Blockchain

Pseudonymity (Bitcoin, Ethereum)

- Addresses are **pseudonyms**, not identities
- All transactions publicly visible on-chain
- Chain analysis can **link** addresses to identities
- Heuristics: common input ownership, change detection
- Once one address deanonymized $\Rightarrow$ transaction graph exposed

Anonymity (Privacy Coins)

- **Sender** privacy: who sent the transaction?
- **Receiver** privacy: who received it?
- **Amount** privacy: how much was transferred?
- **Unlinkability**: can transactions be connected?



Chain analysis firms (Chainalysis, Elliptic) can deanonymize most Bitcoin transactions

Ring Signatures: Hiding the Sender

Definition (Rivest, Shamir, Tauman 2001)

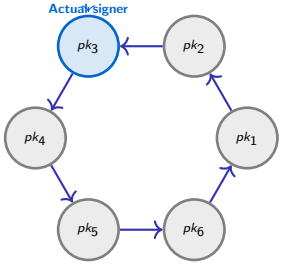
A **ring signature** allows a signer to produce a signature on behalf of an ad-hoc group ("ring") without revealing *which* member signed.

Construction: Given ring $R = \{pk_1, \ldots, pk_n\}$ and signer's secret key sk_i:

1. Choose random values s_j for all $j \neq i$
2. Compute partial signatures: $e_j = H(m \| s_j)$ for $j \neq i$
3. Solve for s_i such that the ring "closes":

$$\prod_{j=1}^{n} g^{s_j} \cdot pk_j^{e_j} = 1 \pmod{p}$$

4. Output $\sigma = (e_1, s_1, \ldots, s_n)$



Verifier knows *someone* in the ring signed, but cannot determine *who*

Key Properties

- **Unforgeability**: Only ring members can sign
- **Anonymity**: Signer indistinguishable among ring
- **No setup**: No group manager needed

Ring signatures provide plausible deniability – any ring member could have been the signer

Stealth Addresses: Hiding the Receiver

Dual-Key Stealth Address Protocol (DKSAP)

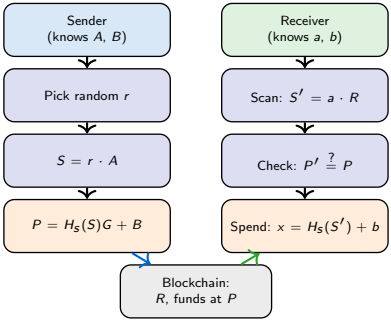
Receiver publishes two public keys: (A, B) where $A = aG$, $B = bG$.

Sending:

1. Sender generates random $r \xleftarrow{\$} \mathbb{Z}_q$
2. Computes shared secret: $S = r \cdot A = r \cdot a \cdot G$
3. Derives one-time key: $P = H_s(S) \cdot G + B$
4. Publishes ephemeral key $R = r \cdot G$ on-chain
5. Sends funds to address derived from P

Receiving:

1. Receiver scans R values, computes $S' = a \cdot R$
2. Checks if $P' = H_s(S') \cdot G + B$ matches output
3. Spends with $x = H_s(S') + b$



Each transaction creates a unique one-time address – no two transactions share the same destination

RingCT: Hiding the Amount

Pedersen Commitments for Amounts

Commit to amount v with blinding factor r:

$$C = r \cdot G + v \cdot H$$

where G, H are independent generators (nobody knows $\log_G H$).

Homomorphic property ensures balance:

$$\sum C_{\text{in}} - \sum C_{\text{out}} = r_\Delta \cdot G + 0 \cdot H$$

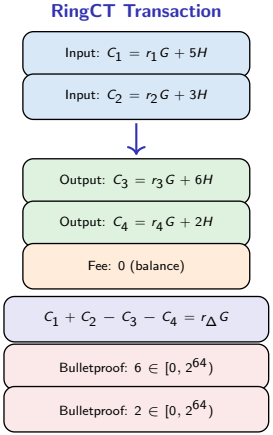
If amounts balance, the difference commits to zero.

Range Proofs (Critical)

Must prove $v \in [0, 2^{64})$ to prevent:

- Negative amounts (creating money from nothing)
- Overflow attacks

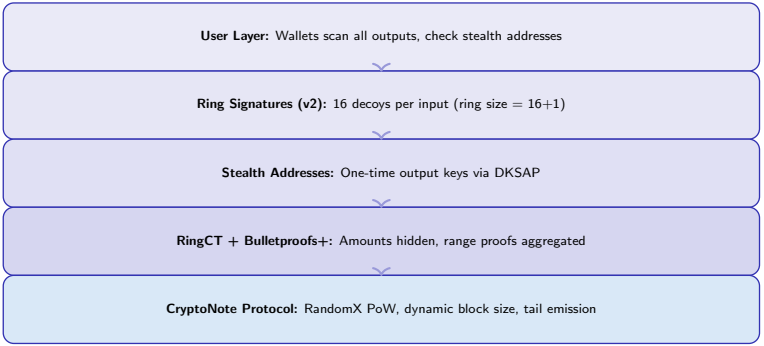
Monero uses **Bulletproofs** for range proofs:
$\mathcal{O}(\log n)$ proof size vs $\mathcal{O}(n)$ for naïve approach.



RingC

was activated on Monero in January 2017 (mandatory since September 2017)

Monero: Privacy by Default



Key Design Choices

- **Mandatory** privacy (no transparent pool)
- Tail emission: 0.6 XMR/block forever
- ASIC-resistant mining (RandomX)
- Regular hard forks for upgrades

Tradeoffs

- Larger transactions ($\sim$2 KB vs 250 B for BTC)
- Wallet sync requires scanning all outputs
- No supply auditability (hidden amounts)
- Regulatory pressure (delisted from exchanges)

ring size increased from 4 (2014) to 11 (2018) to 16 (2022) – larger rings = stronger anonymity sets

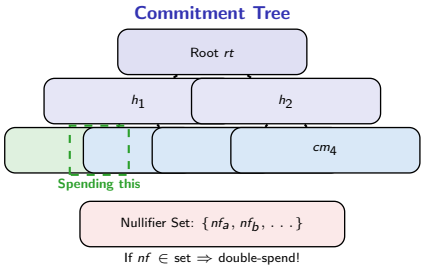
Zcash: Shielded Transaction Pools

Zcash Shielded Transaction Model

Zcash uses **zk-SNARKs** to prove transaction validity without revealing sender, receiver, or amount.

Note model (UTXO-like):

- A **note** (pk, v, ρ, rcm): owner, value, unique ID, randomness
- **Note commitment**: $cm = \text{COMM}(pk \| v \| \rho; rcm)$
- Posted to commitment tree (Merkle tree of all commitments)
- To spend: reveal **nullifier** $nf = \text{PRF}(sk, \rho)$
- ZK proof demonstrates:
  1. Note exists in commitment tree (Merkle path valid)
  2. Spender knows secret key for the note
  3. Nullifier correctly derived (prevents double-spend)
  4. Value balance: $\sum v_{\text{in}} = \sum v_{\text{out}} + v_{\text{fee}}$



Zcash processes ∼500K shielded transactions/month as of 2024 (Sapling + Orchard pools)

Zcash Protocol Evolution: Sprout → Sapling → Orchard

Feature	Sprout (2016)	Sapling (2018)	Orchard (2022)
Proof System	Groth16 (PGHR)	**Groth16**	**Halo 2**
Trusted Setup	Yes (ceremony)	Yes (Powers of Tau)	**No**
Proving Time	∼40 seconds	∼2.3 seconds	∼2.5 seconds
Memory Required	∼3 GB	∼40 MB	∼40 MB
Curve	BN-254	BLS12-381	Pallas/Vesta
Commitment Scheme	SHA-256 compress	Pedersen hash	Sinsemilla
Nullifier Derivation	SHA-256 based	PRF based	Poseidon
Note Encryption	ECIES	In-band	In-band
Action Model	JoinSplit (2-in, 2-out)	Spend + Output	**Action** (unified)
Unified Addresses	No	No	**Yes**

Key Improvement: Halo 2

Orchard eliminates trusted setup via **recursive proof composition** on a cycle of curves (Pallas/Vesta). No toxic waste.

Unified Addresses

Single address encodes receivers for transparent, Sapling, and Orchard pools. Wallet automatically routes to most private available pool.

Zcash

Sprout ceremony (2016) involved 6 participants who each had to destroy their toxic waste independently

Monero vs Zcash: A Detailed Comparison

Dimension	Monero (XMR)	Zcash (ZEC)
Privacy Model	Default (mandatory)	Opt-in (transparent + shielded)
Cryptographic Basis	Ring signatures + stealth addresses	zk-SNARKs (Halo 2)
Trusted Setup	None required	Eliminated in Orchard
Transaction Size	$\sim$2 KB (with Bulletproofs+)	$\sim$2 KB (shielded)
Verification Time	Fast (no pairings)	Moderate (pairing-based)
Anonymity Set	Ring of 16+1 per input	All shielded notes in pool
Supply Auditable?	No (amounts hidden)	Yes (turnstile mechanism)
Scalability	Moderate (pruning hard)	Better (nullifier set compact)
Regulatory Status	Delisted on many exchanges	Listed on major exchanges
Governance	Community-driven	Zcash Foundation + ECC
Mining	RandomX (CPU-friendly)	Equihash (GPU/ASIC)
Adoption	Strong darknet market use	Growing DeFi integration

Monero Advantage

Mandatory privacy means *all* users contribute to the anonymity set. No "opt-in problem" where shielded users stand out.

Zcash Advantage

Stronger cryptographic guarantees: anonymity set is *entire shielded pool*, not just a ring of 16. Supply is auditable via turnstile.

The "anonymity set" debate: Monero's is always 16+1 per transaction; Zcash's is all shielded notes but only $\sim$15% of ZEC is shielded

Other Privacy Solutions in the Ecosystem

Ethereum Privacy

Tornado Cash (2019–2022):

- Smart contract mixer using Groth16 proofs
- Deposit fixed ETH denominations into pool
- Withdraw to new address with ZK proof
- OFAC sanctioned August 2022
- Developer arrested (Netherlands)

Railgun:

- On-chain privacy system for ERC-20 tokens
- Shield/unshield with ZK proofs
- Private transfers, swaps, NFT interactions

Privacy-Focused L1/L2

Aztec Network:

- ZK-rollup with native privacy
- Noir language for private smart contracts
- Private + public state in same contract

Other Approaches:

- **Secret Network**: Encrypted smart contracts (TEE-based, not ZK)
- **Mina Protocol**: Succinct blockchain (22 KB state proof), Kimchi ZK system
- **Penumbra**: Private DEX on Cosmos, shielded staking
- **Namada**: Multi-asset shielded pool for any chain

The

OFAC sanctions on Tornado Cash (2022) raised fundamental questions about whether code can be sanctioned

Privacy Mixer: Solidity Pattern

```solidity
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.20;

interface IVerifier {
    function verifyProof(
        uint[2] calldata a,
        uint[2][2] calldata b,
        uint[2] calldata c,
        uint[4] calldata input
    ) external view returns (bool);
}

contract PrivacyMixer {
    IVerifier public verifier;
    uint256 public denomination;
    mapping(bytes32 => bool) public commitments;
    mapping(bytes32 => bool) public nullifiers;
    bytes32[] public commitmentLeaves;

    event Deposit(bytes32 indexed commitment,
                  uint32 leafIndex);
    event Withdrawal(address to,
                     bytes32 nullifierHash);

    constructor(address _verifier, uint256 _denom) {
        verifier = IVerifier(_verifier);
        denomination = _denom;
    }

    function deposit(bytes32 _commitment)
        external payable
    {
        require(msg.value == denomination,
                "Wrong amount");
        require(!commitments[_commitment],
                "Duplicate");
        commitments[_commitment] = true;
        uint32 idx = uint32(commitmentLeaves.length);
```

```solidity
    function withdraw(
        uint[2] calldata _a,
        uint[2][2] calldata _b,
        uint[2] calldata _c,
        bytes32 _nullifierHash,
        address payable _recipient,
        bytes32 _root
    ) external {
        require(!nullifiers[_nullifierHash],
                "Already spent");
        // Public inputs: root, nullifier,
        //   recipient, denomination
        uint[4] memory input = [
            uint256(_root),
            uint256(_nullifierHash),
            uint256(uint160(address(_recipient))),
            denomination
        ];
        require(verifier.verifyProof(
            _a, _b, _c, input),
            "Invalid proof");
        nullifiers[_nullifierHash] = true;
        _recipient.transfer(denomination);
        emit Withdrawal(_recipient,
                        _nullifierHash);
    }
}
```

Section 3 Summary: Privacy Coins

Privacy Primitives

Primitive	Protects
Ring Signatures	Sender identity
Stealth Addresses	Receiver identity
RingCT / Pedersen	Transaction amounts
Nullifiers	Double-spend prevention
Bulletproofs	Range validity
zk-SNARKs	Everything (Zcash)

Monero
Default privacy
Ring sigs + stealth

Zcash
Opt-in ZK privacy
Halo 2 (no setup)

Tornado Cash
Ethereum mixer
SNARKs on EVM

Aztec
Private L2
ZK-rollup + privacy

Key Insight

Privacy is not binary – it's a **spectrum**. Each system makes different tradeoffs between anonymity set size, performance, trusted setup, and regulatory compatibility.

Next Section

⇒ **ZK Applications:** Rollups, identity, voting, and private DeFi.

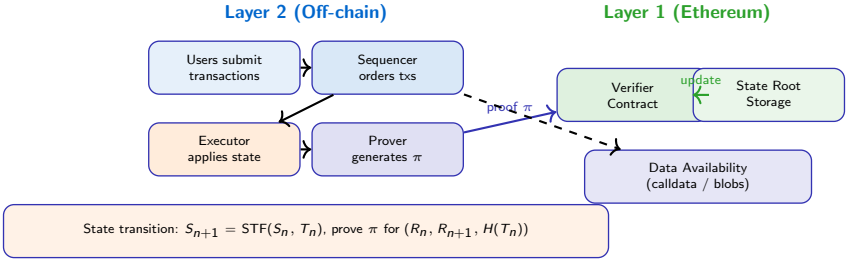
Priva

technology evolves rapidly – new solutions emerge as older ones face regulatory or technical challenges

Section 4: ZK Applications

ZK-rollups, identity systems, voting, and real-world deployments

ZK-Rollups: Architecture Overview



Why ZK-Rollups?

- **Scalability**: 1000–2000+ TPS vs Ethereum's $\sim$15 TPS
- **Security**: Inherits L1 security via validity proofs
- **Finality**: Instant once proof verified on L1

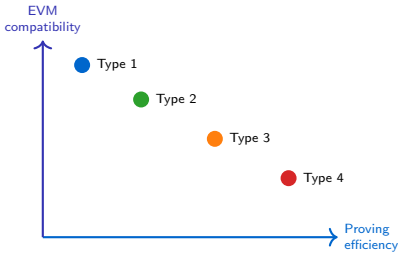
Data Availability (EIP-4844)

- **Calldata**: Expensive, permanently stored
- **Blobs**: Cheaper, pruned after $\sim$18 days
- Proto-danksharding reduces L2 costs by 10–100$\times$

ZK-rollups process transactions off-chain but post validity proofs on-chain – "don't trust, verify (efficiently)"

zkEVM: Types and Tradeoffs

Type	Description	Compatibility	Proving Cost	Projects
1	Fully Ethereum-equivalent	Exact consensus rules	Very high	Taiko
2	EVM-equivalent	Same bytecode, diff state	High	Scroll, Polygon zkEVM
2.5	EVM + gas adjustments	Minor gas differences	Moderate–High	–
3	Almost EVM-equivalent	Most opcodes supported	Moderate	–
4	Language-equivalent	Compiles Solidity to custom VM	Lower	zkSync Era



The Fundamental Tradeoff

Higher EVM compatibility ⇒ more expensive ZK proving.

- Type 1: Can verify Ethereum blocks natively
- Type 4: Fastest proofs but needs recompilation
- Most projects target Type 2–3 as sweet spot

Vitali Buterin's zkEVM taxonomy (2022) provides framework for comparing ZK-rollup approaches to EVM compatibility

ZK-Rollup: State Verification Mathematics

State Transition Function

$$S_{n+1} = \text{STF}(S_n, T_n)$$
$$R_n = \text{MerkleRoot}(S_n)$$
$$R_{n+1} = \text{MerkleRoot}(S_{n+1})$$

where S_n is the state at batch n, T_n is the transaction batch, R_n is the state root.

ZK Proof Statement

The prover generates π proving:

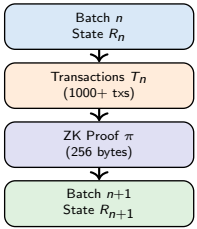
1. $\exists\ S_n$ with $\text{Root}(S_n) = R_n$
2. Applying T_n to S_n yields S_{n+1}
3. $\text{Root}(S_{n+1}) = R_{n+1}$
4. All transactions in T_n have valid signatures
5. All state updates follow protocol rules

Public inputs: $(R_n, R_{n+1}, H(T_n))$
Private witness: $(S_n, T_n,$ Merkle paths$)$

L1 Verification

The L1 verifier contract checks:

$$\text{Verify}(\text{vk}, \pi, [R_n, R_{n+1}, H(T_n)]) \stackrel{?}{=} \texttt{true}$$



Constant-size proof regardless of #txs

Compression Ratio

1000 transactions → single 256-byte proof.
L1 verification: ∼300K gas (constant).

The key insight: verification cost is constant regardless of how many transactions are in the batch

ZK-Rollup: L1 Verifier Contract

```solidity
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.20;

interface IZKVerifier {
    function verifyProof(
        uint[2] calldata a,
        uint[2][2] calldata b,
        uint[2] calldata c,
        uint[3] calldata publicInputs
    ) external view returns (bool);
}

contract ZKRollupVerifier {
    IZKVerifier public immutable zkVerifier;
    bytes32 public stateRoot;
    uint256 public batchNumber;
    address public sequencer;

    event BatchVerified(
        uint256 indexed batch,
        bytes32 oldRoot,
        bytes32 newRoot
    );

    modifier onlySequencer() {
        require(msg.sender == sequencer,
                "Not sequencer");
        _;
    }
```

```solidity
    constructor(
        address _verifier,
        bytes32 _genesisRoot,
        address _sequencer
    ) {
        zkVerifier = IZKVerifier(_verifier);
        stateRoot = _genesisRoot;
        sequencer = _sequencer;
    }

    function verifyBatch(
        uint[2] calldata _a,
        uint[2][2] calldata _b,
        uint[2] calldata _c,
        bytes32 _newRoot,
        bytes32 _txDataHash
    ) external onlySequencer {
        // Public inputs: old root, new root,
        //   transaction data hash
        uint[3] memory inputs = [
            uint256(stateRoot),
            uint256(_newRoot),
            uint256(_txDataHash)
        ];
        require(zkVerifier.verifyProof(
            _a, _b, _c, inputs),
            "Invalid batch proof");
        emit BatchVerified(
            batchNumber, stateRoot, _newRoot);
        stateRoot = _newRoot;
        batchNumber++;
    }
}
```

Simpl

L1 verifier – production systems include data availability posting, escape hatches, and timelock governance

Zero-Knowledge Identity: Selective Disclosure

The Problem

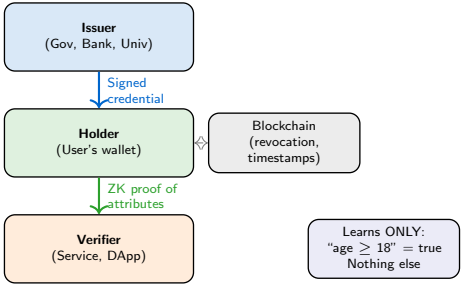
Traditional identity verification requires revealing *all* information:

- Buying alcohol → show full ID (name, address, DOB)
- KYC → upload passport, utility bills, selfies
- Credit check → expose full financial history

ZK Solution: Prove Properties, Not Data

With ZK proofs, prove *statements about* your data:

- "I am over 18" (without revealing age)
- "I am a citizen of EU" (without revealing country)
- "My credit score > 700" (without revealing score)
- "I am not on sanctions list" (without revealing identity)



W3C

Verifiable Credentials + ZK proofs enable self-sovereign identity where users control their own data

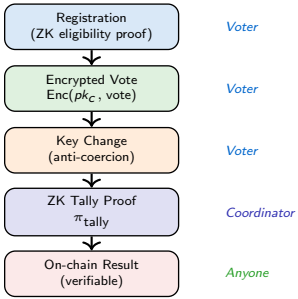
ZK Voting: Private, Verifiable Elections

Requirements for Secure Voting

1. **Eligibility**: Only authorized voters can vote
2. **Privacy**: No one learns how anyone voted
3. **Correctness**: Tally accurately reflects all votes
4. **Coercion resistance**: Cannot prove how you voted to a briber

MACI (Minimum Anti-Collusion Infrastructure)

1. Voters encrypt votes with coordinator's public key
2. Coordinator decrypts, tallies, generates ZK proof of correct tally
3. Voters can submit *key change* messages to invalidate earlier votes (anti-coercion)
4. Final tally verified on-chain via ZK proof



MACI was developed by the Ethereum Foundation and used for Gitcoin grants allocation and DAO governance

Private DeFi: Confidential Finance on-Chain

Why Private DeFi?

Public DeFi exposes traders to:

- **Front-running**: Bots see pending trades in mempool
- **MEV extraction**: Miners/validators reorder transactions
- **Copy trading**: Competitors mirror your strategy
- **Information leakage**: Collateral ratios, positions visible

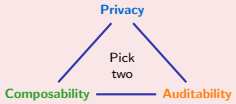
Protocol Examples

Protocol	Approach
Penumbra	Private DEX + staking
Aztec	ZK-rollup with privacy
Railgun	ERC-20 shielding
Renegade	Dark pool matching
Panther	Multi-chain privacy

ZK Solutions

- **Dark pools**: Private order books with ZK-proven fair matching
- **Confidential balances**: Hidden token amounts
- **Private lending**: Hide collateral ratios
- **Shielded swaps**: Trade without revealing intent

The Trilemma



MEV

(Maximal Extractable Value) cost Ethereum users $600M+ in 2023 – private DeFi aims to eliminate this

zkML: Verifiable Machine Learning

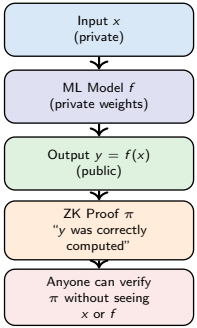
What is zkML?

Prove that a machine learning inference was computed correctly *without revealing*:

- The model weights (proprietary model)
- The input data (user privacy)
- Or both (full confidentiality)

Applications:

- **Verifiable AI**: Prove an AI model made a specific prediction
- **Private inference**: Use a model without revealing your data
- **Model marketplace**: Sell predictions without exposing weights
- **On-chain AI**: Smart contracts that verify ML outputs



Projects:
EZKL
Modulus Labs
Giza
Inference Labs

Current Limitations

- Large neural networks: proving is extremely expensive
- Current: can prove small models (decision trees, small NNs)
- Research frontier: efficient ZK circuits for matrix multiplication

EZKL

can generate ZK proofs for ONNX models – current practical limit is $\sim$10M parameter models

ZK Bridges: Trustless Cross-Chain Verification

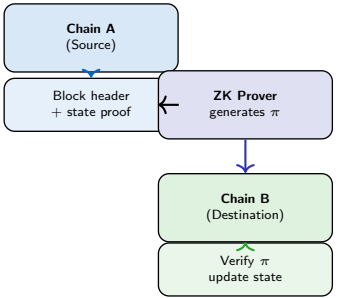
The Bridge Problem

Cross-chain bridges are the #1 target for hacks ($2B+ stolen 2021–2023). Traditional approaches:

- **Multisig**: Trust m-of-n validators (centralized)
- **Optimistic**: Assume valid, challenge period (slow)
- **Light client**: Run consensus verification (expensive)

ZK Bridge Solution

1. Generate ZK proof of Chain A's consensus (block headers, validator signatures)
2. Verify proof on Chain B (∼300K gas)
3. No trusted third parties, no challenge periods
4. Security reduces to ZK proof soundness



Security Comparison

Multisig: trust validators. ZK: trust math.

Ronin

Bridge hack ($625M), Wormhole ($320M), Nomad ($190M) – ZK bridges eliminate the trusted validator attack surface

ZK Coprocessors: Verifiable Off-Chain Computation

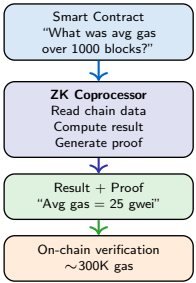
The Problem

Smart contracts are limited:

- Cannot access historical state cheaply
- Gas limits constrain computation complexity
- On-chain data is expensive to process
- Complex analytics impossible on-chain

ZK Coprocessor Pattern

1. Query historical blockchain data off-chain
2. Perform complex computation (aggregation, ML, etc.)
3. Generate ZK proof of correct computation
4. Smart contract verifies proof on-chain
5. Result available trustlessly on-chain



Projects:
Axiom
RISC Zero
Brevis
Herodotus

Use Cases

Dynamic NFT pricing, on-chain analytics, loyalty rewards based on history, insurance claims verification.

Think of it as: A trusted computation oracle, where "trust" comes from mathematics rather than reputation.

ZK coprocessors extend smart contract capabilities without sacrificing trustlessness – "SQL for blockchains"

Section 4 Summary: ZK Applications

ZK-Rollups
Scalability via validity proofs
1000+ TPS on Ethereum

ZK Identity
Selective disclosure
Prove attributes, not data

ZK Voting
Private ballots
Coercion-resistant (MACI)

Private DeFi
Prevent MEV/front-running
Dark pools, shielded swaps

ZK Infrastructure
Bridges, coprocessors, zkML
Trustless cross-chain

Common Pattern

All ZK applications follow the same structure:

1. **Compute** something privately off-chain
2. **Prove** correctness with a ZK proof
3. **Verify** the proof cheaply on-chain
4. **Trust** mathematics, not intermediaries

Open Challenges

- Proving costs still high for complex computations
- Developer tooling immature vs traditional development
- Regulatory uncertainty around privacy features
- UX gap: users shouldn't need to understand ZK

Next Section

$\Rightarrow$ **Advanced Topics:** Hardware acceleration, post-quantum, regulation, and the road ahead.

ZK technology is transitioning from "interesting cryptography" to "critical infrastructure" for Web3

Section 5: Advanced Topics & Future

Recursive proofs, cross-chain privacy, regulation, and emerging trends

Proving System Frontiers: Next-Generation ZK

Lookup-Based Systems

Lasso & Jolt (a16z Research, 2023):

- Replace arithmetic circuits with **lookup tables**
- Prover commits to table entries used
- Sumcheck protocol verifies consistency
- 2–10× faster proving for many operations
- Jolt: zkVM built entirely on Lasso lookups

Folding Schemes

Nova (Kothapalli, Setty, Tzialla 2022):

- IVC without recursive SNARKs
- "Fold" two instances into one
- **HyperNova**: Generalization to CCS (Customizable Constraint Systems)
- Enables efficient incremental computation

Binary & Small Field Systems

Binius (Irreducible, 2024):

- Operates over **binary fields** $\mathbb{F}_2$
- Native to how computers actually work
- No field arithmetic overhead
- Dramatic speedups for bitwise operations

Circle STARKs

Stwo (StarkWare, 2024):

- STARKs over **Mersenne prime** $p = 2^{31} - 1$
- Circle group structure for FFT-like operations
- 32-bit native arithmetic (CPU-friendly)
- Expected 10–100× speedup vs standard STARKs

Trend: Provers are getting faster at roughly 2×/year.

The ZK proving landscape is evolving rapidly – systems that don't exist today may become dominant within 2-3 years

Hardware Acceleration for ZK Proving

Why Hardware Matters

ZK proving is computationally intensive:

- Multi-Scalar Multiplication (MSM): $\sum_i s_i \cdot P_i$
- Number Theoretic Transform (NTT/FFT)
- Hash computations (Poseidon, Keccak)
- These operations dominate prover runtime

Platform	MSM Speed	Cost
CPU (single)	1× (baseline)	$
GPU (A100)	10–50×	$$
FPGA	20–100×	$$$
ASIC	100–1000×	$$$$

Prover Networks

Decentralized proving infrastructure:

- **Ingonyama**: GPU-accelerated MSM/NTT
- **Cysic**: Custom ZK ASIC development
- **Ulvetanna**: FPGA proving clusters
- **Gevulot**: Decentralized prover marketplace

Prover Economics

Current costs per proof:

- Simple transfer: $0.001–0.01
- zkEVM batch (1000 txs): $1–10
- Complex zkML: $10–100+

Goal: Real-time proving on consumer devices (phones, browsers) for client-side ZK.

Hardware acceleration is critical for ZK adoption – the "prover bottleneck" is the primary scalability constraint

Post-Quantum Zero-Knowledge Proofs

The Quantum Threat

Quantum computers threaten:

- **Elliptic curves**: Shor's algorithm breaks ECDLP
- **Pairing-based SNARKs**: Groth16, PLONK at risk
- **RSA**: Factoring becomes easy
- **Not threatened**: Hash functions, symmetric crypto

System	Pre-Q	Post-Q
Groth16	✓	×
PLONK	✓	×
Bulletproofs	✓	×
STARKs	✓	✓
Lattice SNARK	✓	✓

Post-Quantum ZK Approaches

- **STARKs**: Already post-quantum! Based on hash functions and algebraic coding theory (FRI)
- **Lattice-based commitments**: Replace Pedersen with lattice schemes (SIS/LWE)
- **Hash-based signatures**: Merkle trees + hash chains
- **Code-based**: Error-correcting code assumptions

NIST Impact

NIST PQC standards (2024):

- ML-KEM (CRYSTALS-Kyber): Key encapsulation
- ML-DSA (CRYSTALS-Dilithium): Digital signatures
- SLH-DSA (SPHINCS+): Hash-based signatures
- ZK community preparing migration paths

STAR

are naturally post-quantum since they rely only on collision-resistant hash functions, not elliptic curves

Regulatory & Ethical Dimensions of ZK Privacy

The Privacy–Compliance Tension

- **FATF Travel Rule**: VASPs must share sender/receiver info for transfers $>$ \$1,000
- **OFAC Sanctions**: Tornado Cash sanctioned (Aug 2022), developer arrested
- **AML/KYC**: Regulators demand identity verification
- **EU MiCA**: Limits anonymous crypto transfers

ZK as Middle Ground: Selective Disclosure

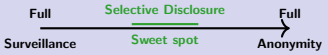
Compliance-friendly privacy:

- Prove "not on sanctions list" without revealing identity
- Prove "KYC'd by approved provider" without sharing documents
- Prove "tax-compliant" without exposing all transactions
- **Privacy Pools** (Buterin et al., 2023): Users prove association with compliant set

Arguments for Privacy

- Financial privacy is a **human right** (UN Declaration Art. 12)
- Businesses need confidentiality (trade secrets, salaries)
- Protection from surveillance and discrimination
- Cash transactions are private by default

The Spectrum



The Tornado Cash case (2022–2024) is a landmark legal battle over whether open-source code constitutes sanctionable "property"

ZK Developer Ecosystem & Tools

Tool	Language	Used By
Circom	DSL (JS-like)	iden3, Tornado
Noir	Rust-like	Aztec
Cairo	Rust-like	StarkNet
Halo2	Rust	Zcash, Scroll
SP1	Rust (any code)	RISC Zero
Gnark	Go	Consensys

Circom Example: Hash Preimage

```
pragma circom 2.1.6;
include "circomlib/poseidon.circom";

template HashPreimage() {
    signal input preimage;
    signal input hash;
    signal output valid;
    // Compute Poseidon hash of preimage
    component hasher = Poseidon(1);
    hasher.inputs[0] <== preimage;
    // Constrain: computed hash == public hash
    hash === hasher.out;
    valid <== 1;
}
component main {public [hash]}
    = HashPreimage();
```

Development Workflow

1. **Write circuit** in DSL (Circom, Noir, etc.)
2. **Compile** to arithmetic circuit (R1CS/PLONK)
3. **Trusted setup** (if needed) or use universal SRS
4. **Generate prover/verifier** keys
5. **Export Solidity verifier** contract
6. **Deploy** verifier to Ethereum
7. **Generate proofs** client-side or server-side

Emerging Trends

- **zkVM**: Prove any program (SP1, RISC Zero) – no circuit writing needed
- **Client-side proving**: Generate proofs in browser (WASM)
- **Formal verification**: Prove circuits are bug-free
- **ZK-as-a-Service**: APIs for proof generation

The

The Road Ahead: ZK as Infrastructure

Near-Term (2024–2026)

- zkEVM mainnet maturity (Type 1–2)
- Client-side proving on mobile devices
- ZK coprocessors for smart contract analytics
- Privacy pools for regulatory compliance
- Proof aggregation across rollups

Medium-Term (2026–2030)

- Universal ZK: prove *any* computation
- Real-time proving (sub-second for complex circuits)
- ZK-native identity infrastructure
- Cross-chain ZK interoperability standard
- Post-quantum migration for pairing-based systems

Long-Term Vision

- "**Verify, don't trust**" extends to everything
- ZK proofs as fundamental internet protocol
- Private-by-default computation everywhere
- Provable AI, provable elections, provable finance

2024: ZK-rollups go mainstream

2026: Client-side proving standard

2028: Universal ZK for all computation

2030+: ZK as internet layer

Open Problems

Proof composability, standardization, UX, regulatory clarity, formal verification of circuits.

"Zero-knowledge proofs are to computation what public-key cryptography was to communication" – paraphrasing Eli Ben-Sasson

Course Summary: Privacy & Zero-Knowledge Proofs

ZK Foundations (Section 1)

- Completeness, soundness, zero-knowledge
- Commitment schemes (Pedersen, KZG)
- Sigma protocols (Schnorr)
- Arithmetic circuits $\rightarrow$ R1CS $\rightarrow$ QAP

Proof Systems (Section 2)

- Groth16: 3-element proofs, trusted setup
- PLONK: Universal SRS, custom gates
- STARKs: Transparent, post-quantum, FRI
- Bulletproofs: No trusted setup, log-size

Privacy Coins (Section 3)

- Monero: Ring sigs + stealth + RingCT
- Zcash: zk-SNARKs, Halo 2, note model
- Privacy spectrum: mandatory vs opt-in

ZK Applications (Section 4)

- ZK-rollups: Scalability via validity proofs
- zkEVM: Types 1–4 compatibility tradeoff
- Identity, voting, private DeFi, zkML
- ZK bridges and coprocessors

Advanced Topics (Section 5)

- Frontier systems: Lasso, HyperNova, Binius
- Hardware acceleration: GPU/FPGA/ASIC
- Post-quantum: STARKs already safe
- Regulation: selective disclosure as middle ground

Key Takeaway

Zero-knowledge proofs transform the trust model of computation: from "trust the computer" to "verify the proof." This paradigm shift enables privacy, scalability, and interoperability simultaneously.

ZK proofs are the most important cryptographic primitive since public-key cryptography

Questions & Further Reading

Discussion Prompts

1. Should financial privacy be a *right* or a *privilege*? How does this change with blockchain?
2. Can ZK proofs solve the "privacy vs compliance" dilemma, or is it a false dichotomy?
3. Which zkEVM type will dominate, and why?
4. Is Monero's "privacy by default" approach superior to Zcash's "opt-in" model?
5. What are the implications of zkML for AI governance?

Essential Reading

- **ZKP.science**: Curated ZK proof resources
- **Vitalik's blog**: "An Incomplete Guide to Rollups", "The Different Types of ZK-EVMs"
- **IACR ePrint**: Latest cryptographic research
- **RareSkills ZK Book**: Practical ZK development guide
- **0xPARC**: ZK learning resources and grants

Hands-On Resources

- Circom tutorial: `docs.circom.io`
- Noir playground: `noir-lang.org`
- zkSNARK interactive demo: `zkp.science`
- StarkNet/Cairo: `starknet.io/learn`

"Priv

is not about having something to hide. It is about having something to protect." – Edward Snowden