On-Chain Finance:
All You Need is Zero-Knowledge

Arijit Dutta
Consensys Incorporated

NIT Calicut · Nomember 6, 2026

Consensys

MetaMask
Ethereum wallet
gateway for users to dApps and DeFi

Lineth
L2 rollup (zkEVM)
validity proofs, settled on Ethereum L1

Besu
Ethereum client
enterprise-grade, open-source execution client

users | scaling and proving | infrastructure

Wallet $\rightarrow$ rollup $\rightarrow$ client: the stack on which on-chain finance runs.

Composable Finance: Why We Build It at Consensys

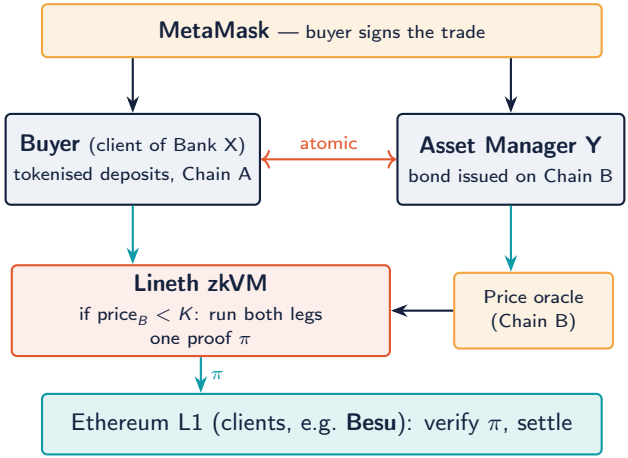
Scenario: a client of **Bank X** buys a tokenised bond from **Asset Manager Y**.

1. pay \$1M of tokenised deposits (**Chain A**)
2. deliver the bond token (**Chain B**)

Atomic: both legs happen, or neither.
Conditional: only if $\text{price}_B < K$.

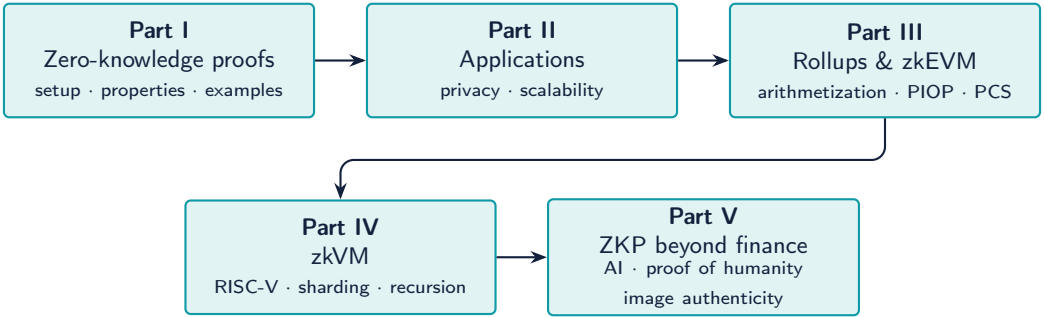
Today's problem

- No shared state; bridges and escrow add trust
- Settlement risk: one leg fails
- Positions must stay private



Composability: independent institutions and chains act as one system, *without* trusting each other.

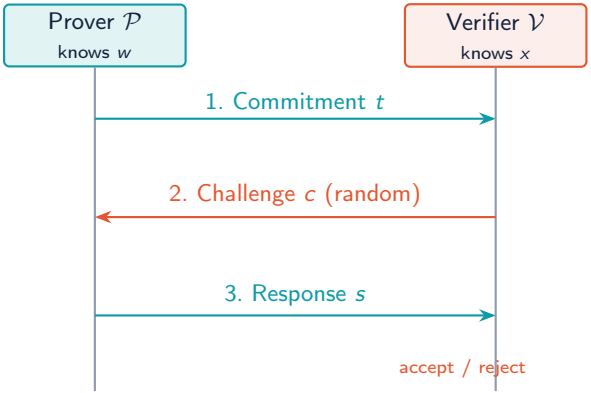
Roadmap


Part I
Zero-knowledge proofs
setup · properties · examples
Part II
Applications
privacy · scalability
Part III
Rollups & zkEVM
arithmetization · PIOP · PCS
Part IV
zkVM
RISC-V · sharding · recursion
Part V
ZKP beyond finance
AI · proof of humanity
image authenticity

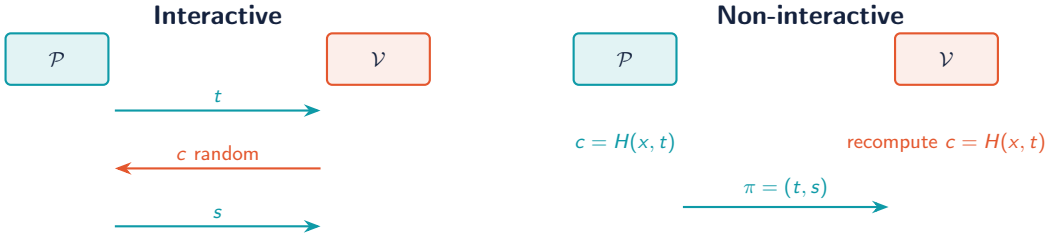
ZKP: The Setup

- **Prover** $\mathcal{P}$: knows a secret witness w
- **Verifier** $\mathcal{V}$: knows only the public statement x
- Goal: convince $\mathcal{V}$ that "$\exists\, w$ with $R(x, w) = 1$" while revealing *nothing else*

- Classic 3-move pattern:
  1. **Commitment** — fresh randomness from Prover
  2. **Challenge** — verifier's random coin
  3. **Response** — depends on secret + challenge



ZKP: Non-Interactive Proofs via Fiat–Shamir



- **Hash function**: $y = H(x)$, random, one-way, collision resistance
- **Fiat–Shamir heuristic**: replace the verifier's random challenge by a hash of the transcript so far
- Proof π is a single string, *publicly verifiable*—exactly what a smart contract needs
- Security in the random-oracle model; always hash the *statement* too

ZKP: Properties

Completeness

If the statement is true and both parties are honest, the verifier **always accepts**.

Soundness

If the statement is false, no cheating prover can make the verifier accept, except with **negligible probability**.

Zero-Knowledge

The verifier learns **nothing beyond the truth** of the statement: its view can be **simulated** without w.

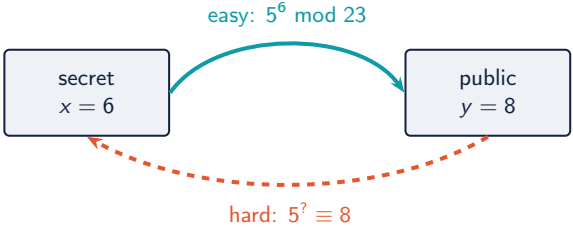
Stronger variant: **proof of knowledge**—an extractor can recover w from any successful prover.

ZKP: A Small Example — One-Way Functions

- **One-way function**: easy to compute, hard to invert
- **Discrete logarithm**: $\mathbb{G} = \langle g \rangle$ of prime order q

$$f(x) = g^x, \qquad x \in \mathbb{Z}_q$$

- Forward: square-and-multiply, $O(\log q)$
- Backward (find x from y): no known efficient algorithm



toy parameters $p = 23$, $g = 5$;
real systems use 256-bit elliptic-curve groups

- **Claim to prove**: "I know x such that $g^x = y$"
- Hard to compute, *easy to verify*

ZKP: A Small Example — Sigma Protocol (Schnorr)

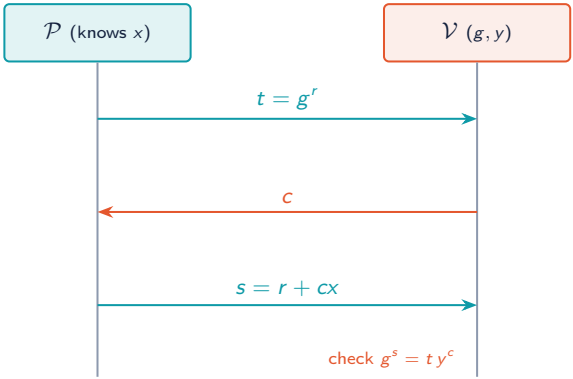
Public: $g,\ y = g^x$ **Prover knows:** x

1. **Commit**: pick $r \leftarrow \mathbb{Z}_q$, send $t = g^r$
2. **Challenge**: $\mathcal{V}$ sends $c \leftarrow \mathbb{Z}_q$
3. **Response**: $s = r + c \cdot x \bmod q$

Verify: $g^s \stackrel{?}{=} t \cdot y^c$

Non-interactive (Fiat–Shamir)

$c = H(g, y, t)$, proof $\pi = (t, s)$



Why Schnorr Satisfies the Three Properties

Completeness

$$
\begin{aligned}
g^s &= g^{r+cx} \\
&= g^r \, (g^x)^c \\
&= t \, y^c \ \checkmark
\end{aligned}
$$

Soundness (special)

Two accepting transcripts (t, c, s), (t, c', s') with $c \neq c'$ give

$$x = \frac{s - s'}{c - c'} \bmod q$$

so a prover who can answer two challenges *knows* x.

Zero-Knowledge (HVZK)

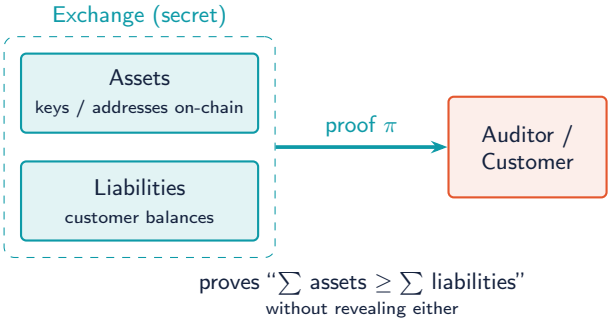
Simulator, without x: pick c, s at random and set

$t = g^s \, y^{-c}$.

Same distribution as a real transcript.

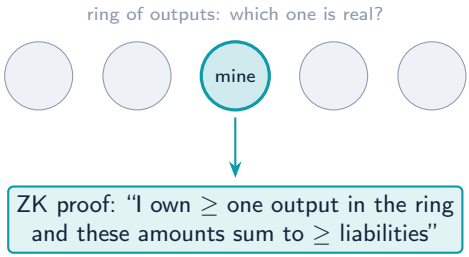
ZKP: Proof of Reserve

- **Prover**: centralised crypto exchange
- **Verifier**: auditor / customer
- **Statement**: total assets $\geq$ total customer liabilities
- **Why privacy matters** for the exchange:
  - addresses and holdings reveal strategy
  - publishing wallets invites attacks
  - customer balances are confidential



Proof of Reserve on Privacy-Preserving Blockchains

- **Monero**: ring signatures, stealth addresses, hidden amounts
- An output's owner is hidden inside a *ring* of decoys
- Publishing keys or signing a message is no longer an option—the proof of ownership must itself be **zero-knowledge**
- Must not leak *which* outputs the exchange owns (or spent)

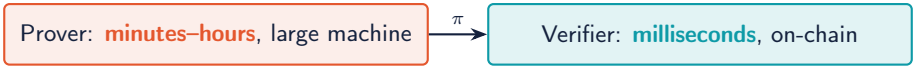


Protocols: MProve | MProve+ | Nummatus | Revelio

ZKP: Proof of Computation

- **Hard to compute, easy to verify**
- Prover runs a (huge) computation $y = F(x)$ and outputs a short proof π
- Verifier checks π in time *much smaller* than re-running F
- Two families:
  - **SNARK**: Succinct Non-interactive ARgument of Knowledge
  - **STARK**: Scalable Transparent ARgument of Knowledge

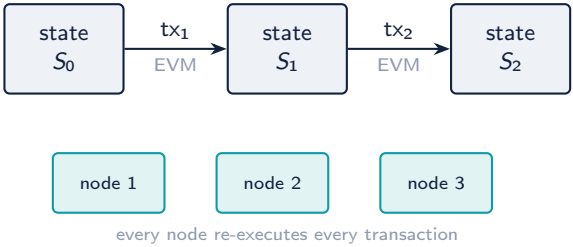
	SNARK	STARK
Setup	often trusted / universal	transparent
Proof	small (bytes–KB)	larger (10s–100s KB)
Verify	very fast	fast (poly-log)
Crypto	curves, pairings	hashes, post-quantum



The Ethereum Virtual Machine (EVM)

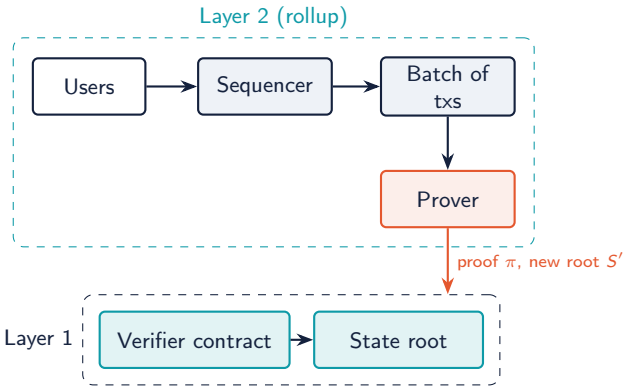
- A **state machine**: global state S = accounts, balances, contract storage
- Transaction tx executes bytecode (opcodes) and moves $S \to S'$
- Every full node re-executes every transaction

  $\Rightarrow$ **throughput is limited** by the weakest validator

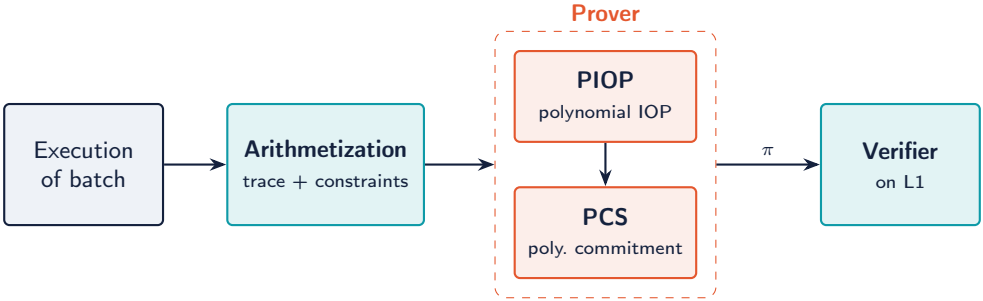


Rollups and Layer 2: Scalability and Privacy

- Move execution **off-chain**; keep security **on-chain**
- **Scalability**: L1 verifies one succinct proof instead of re-executing thousands of transactions
- **Privacy**: zero-knowledge lets the proof hide transaction data



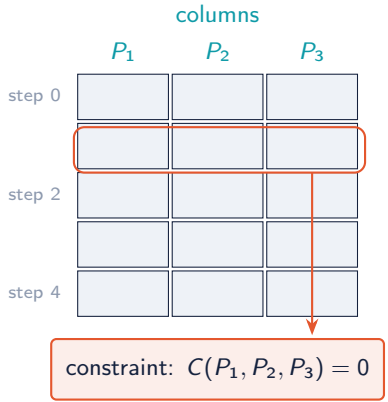
Components of a (ZK) Rollup



- **Arithmetization**: express "the execution was correct" as polynomial constraints
- **PIOP**: reduce constraint checking to a few random polynomial evaluations
- **PCS**: commit to polynomials and prove those evaluations succinctly

Arithmetization

- **Trace**: table recording the whole execution, one row per step
- **Column**: one variable / register across all steps
- **Constraint**: polynomial equation every row (or consecutive rows) must satisfy
- Each column becomes a polynomial $P_j(X)$ over a domain H (interpolation)
- "Execution is correct" $\iff$ constraint polynomials vanish on H



Arithmetization Example: Fibonacci

Columns A, B; row i holds (F_i, F_{i+1})

i	A	B
0	0	1
1	1	1
2	1	2
3	2	3
4	3	5
5	5	8
6	8	**13**

Boundary constraints

$$A_0 = 0, \quad B_0 = 1, \quad B_6 = 13 \text{ (public output)}$$

Transition constraints (all consecutive rows)

$$A_{i+1} - B_i = 0$$
$$B_{i+1} - A_i - B_i = 0$$

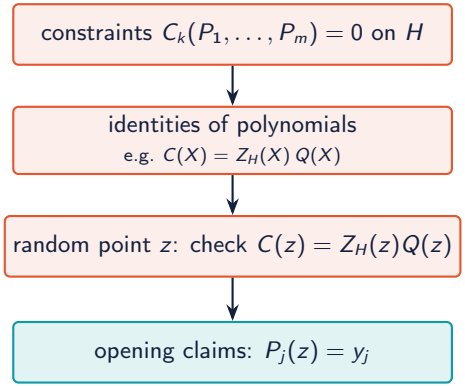
As polynomials, with ω a generator of H:

$$A(\omega X) - B(X) = 0, \qquad B(\omega X) - A(X) - B(X) = 0$$

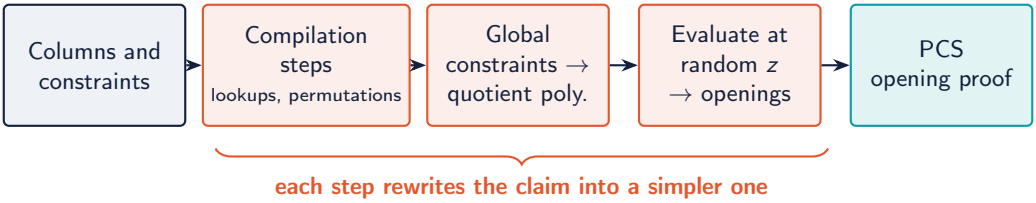
Program of 6 steps $\rightarrow$ 2 columns, 5 constraints

PIOP: Polynomial Interactive Oracle Proof

- $H = \{\omega, \omega^2, \cdots, \omega^n\}$, $Z_H(X) = (X-\omega)(X-\omega^2)\cdots(X-\omega^n)$
- Prover sends *polynomial oracles*; verifier queries them at points of its choice
- **Compiler step**: every constraint is compiled into a **polynomial opening constraint** $P(z) = y$
- Idea (Schwartz–Zippel): a nonzero low-degree polynomial is zero at a random point only with tiny probability
- Check a polynomial identity by checking it at *one* random z



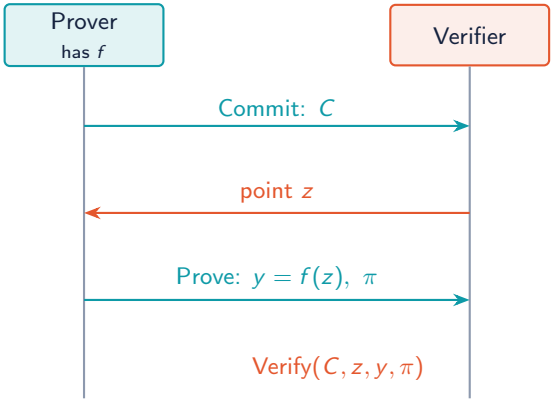
PIOP in Practice: Compilation Steps in Arcane (Consensys)



- Modular design: independent *compilers* are applied one after another to the protocol
- Each compiler removes one kind of constraint until only polynomial openings remain
- Last step hands the opening claims to the polynomial commitment scheme

PCS: Polynomial Commitment Scheme

- $\text{Commit}(f) \to C$ short, binding
- $\text{Prove}(f, z) \to (y, \pi)$ $y = f(z)$
- $\text{Verify}(C, z, y, \pi) \to 0/1$
- Properties: **binding**, **succinct** proof and verification, optionally **hiding**
- Instances: KZG, FRI, **IPA**, ...



zkEVM

Definition

A zkEVM produces a **proof of the state change** after processing a batch of transactions:
$\pi : S_{\text{old}} \xrightarrow{\text{batch}} S_{\text{new}}$

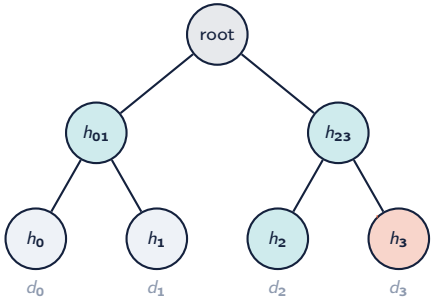
State management: accounts and storage live in a Merkle tree; every EVM read or write becomes one of four operations on it.

Insert	**Update**	**Read zero**	**Read non-zero**
key absent $\rightarrow$ present	change existing value	prove key is absent	prove key $\mapsto v \neq 0$

Each operation $\Rightarrow$ Merkle-path check (hash computations) $\Rightarrow$ constraints in the trace

Merkle Tree

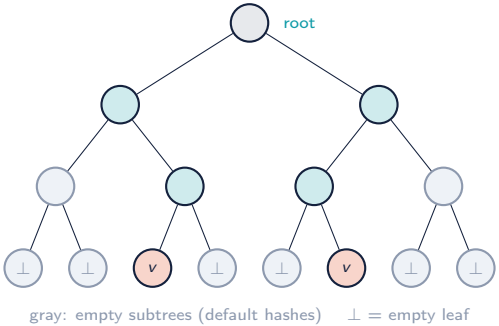
- Leaves: hashes of data items
- Each parent $= H(\text{left} \,\|\, \text{right})$
- **Root** commits to the whole set
- **Membership proof** for a leaf: the sibling hashes along its path, $\log n$ values
- Verifier recomputes the root and compares
- Prove: d_2 is present in the tree
- Proof: $\{h_3, h_{01}\}$
- Verify: $root \stackrel{?}{=} H(H(H(d_2), h_3), H_{01})$



prove $d_2 \in$ tree: reveal h_3 (sibling) and h_{01}

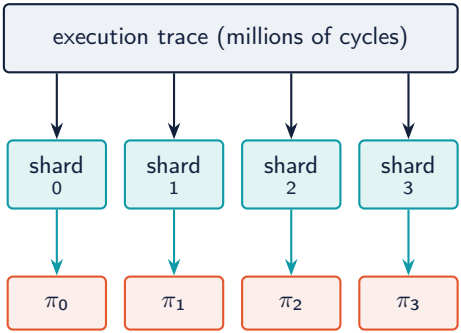
Sparse Merkle Tree

- **Definition**: Merkle tree of fixed depth d with 2^d leaves (e.g. $d = 256$), almost all of them empty
- Key = path from the root to a leaf
- Empty subtrees use precomputed default hashes
- Supports **non-membership** proofs: "key is absent / value 0"
- **Arithmetization**

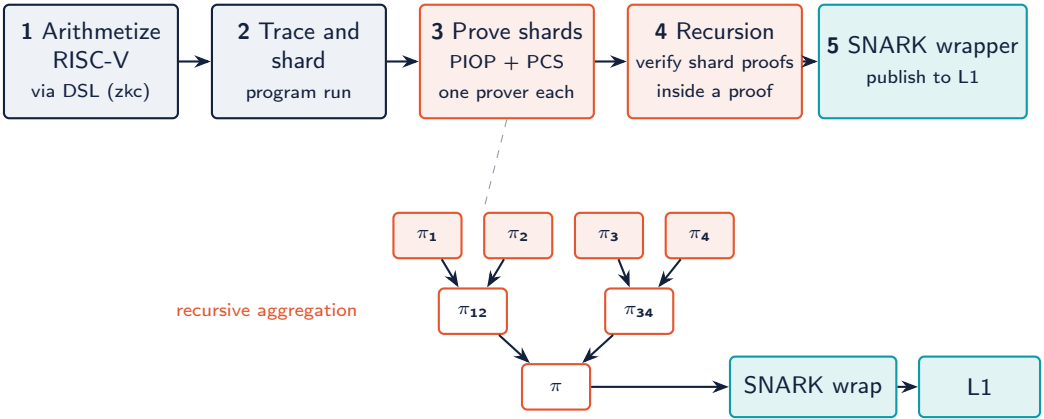


zkVM

- **VM**: executes programs written for a fixed instruction set
- **zkVM**: also outputs a proof that the execution was correct
- **RISC-V**: open, small, regular ISA; compile from Rust or C
- **Tracing**: one row per cycle (pc, instruction, registers, memory)
- **Sharding**: split a long trace into fixed-size shards proved in parallel



zkVM: End-to-End Steps



ZKP Beyond Finance

- **ZKP and AI**: zkML, provable inference
- Proof of humanity, AnonAadhar
- Image and video authenticity

Thank you for your attention

Questions?

Arijit Dutta · Consensys Incorporated