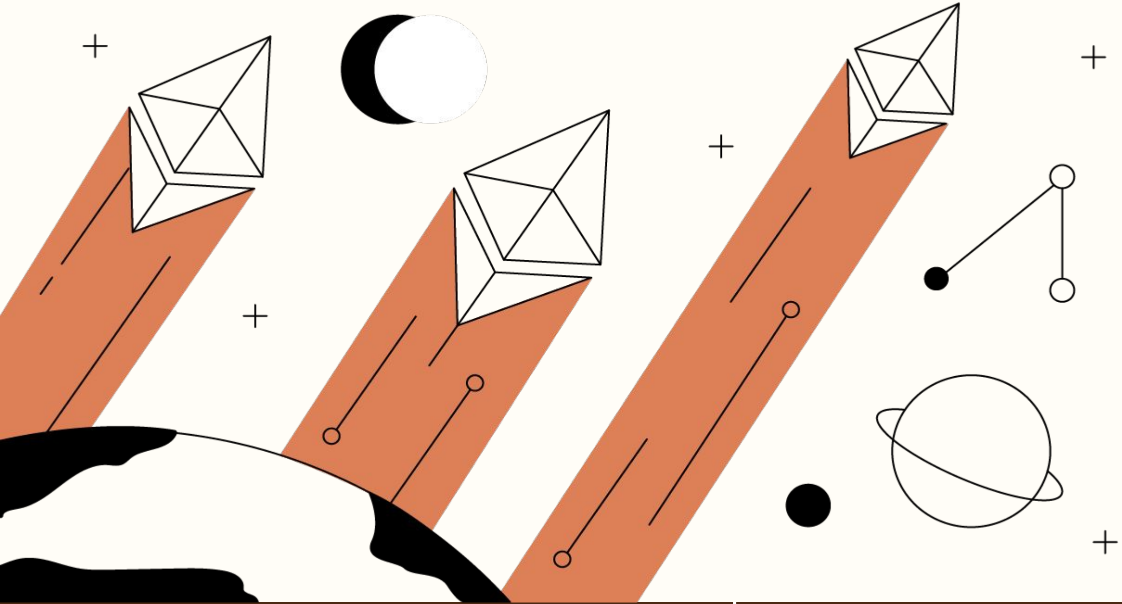
Recursive Composition of Zero-Knowledge SNARK Proofs in zkEVM

Tal Derei

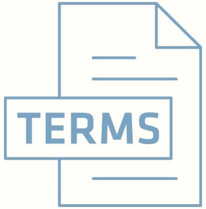
Agenda



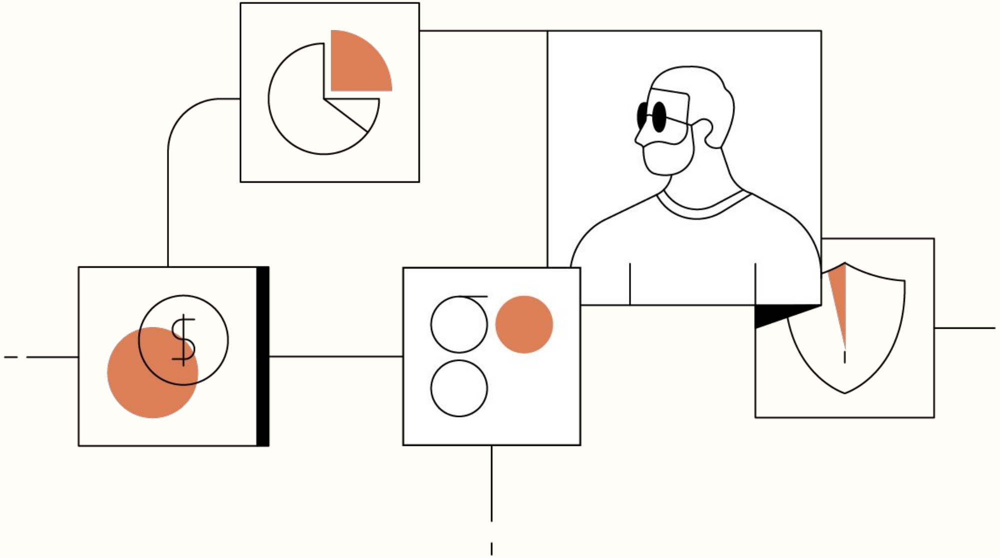
1. **Recursive zk-SNARKs + zkEVM Recap**
2. **General Background: Technology Stack and Building Blocks**
3. **Polygon-Zero: zkEVM Implementation**

Terms

- **L1** = Layer-1 (Main Chain)
- **L2** = Layer-2 (ZK-Rollups)
- **ZK** = Zero-Knowledge
- **zk-SNARKs** = Succinct Non-Interactive Argument of Knowledge *Proofs*
- **zkEVM** = Zero-Knowledge Ethereum Virtual Machine

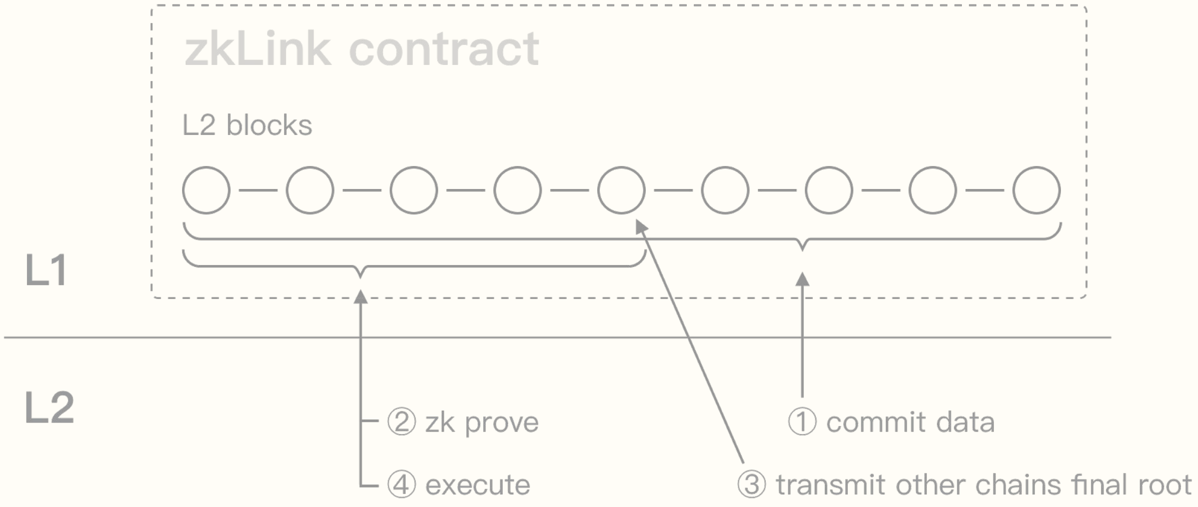


1. Recursive zk-SNARKs



zk-Rollups

- **zk-Rollups** = Layer 2 scaling solutions that move computation and state storage off-chain



zk-Rollups

BUT…

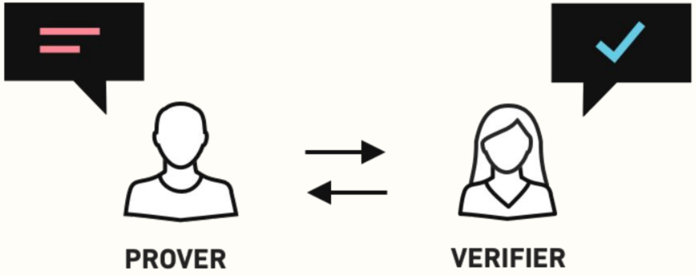
Generating proofs is resource and computationally <u>expensive</u>!

Control Flow:

Computation → Algebraic Circuit → R1CS → QAP → zk-SNARK

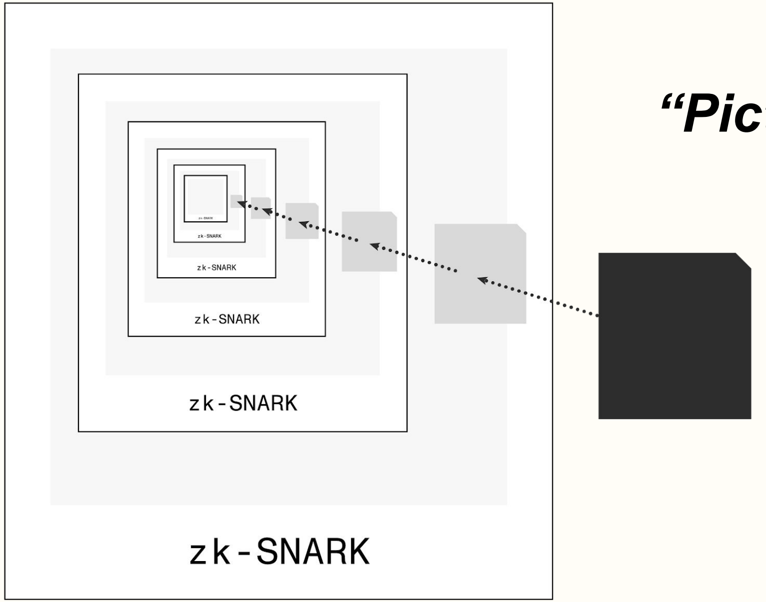
zk-SNARKs

- **SNARKs** = cryptographic <u>proof</u>
    - Enables a prover to prove a mathematical statement to a verifier with a *short proof* and *succinct verification* using zero knowledge techniques



Recursive zk-SNARKs

- **Recursive SNARKs** = Enables producing proofs that prove statements about previous proofs



"Picture of a Picture"

Recursive zk-SNARKs

Recursive zk-SNARK Program:

INPUTS:

→ Previous State
→ Proof of Previous State
→ New Transactions

→

OUTPUTS:

→ New State
→ Proof of New State

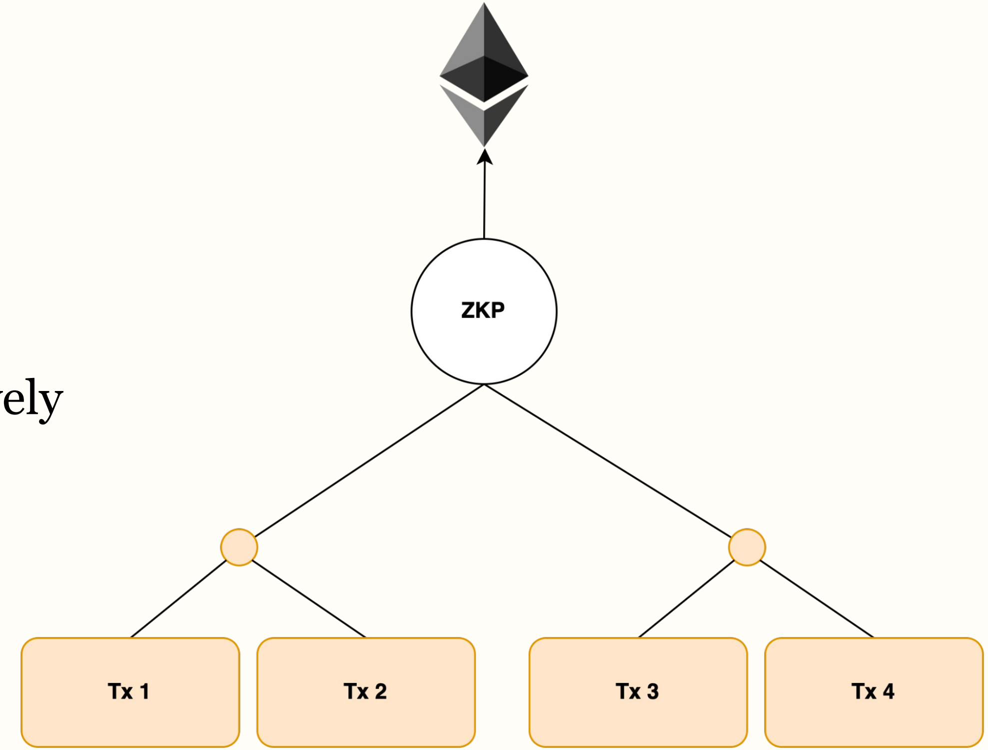
Recursive zk-SNARKs

SNARKs:

- ONE proof that verifies 10,000 transactions

Recursive SNARKs

- 10,000 proofs that each verify ONE transaction in parallel, and recursively aggregating them



Recursive zk-SNARKs

Batching and **Compactness** Techniques

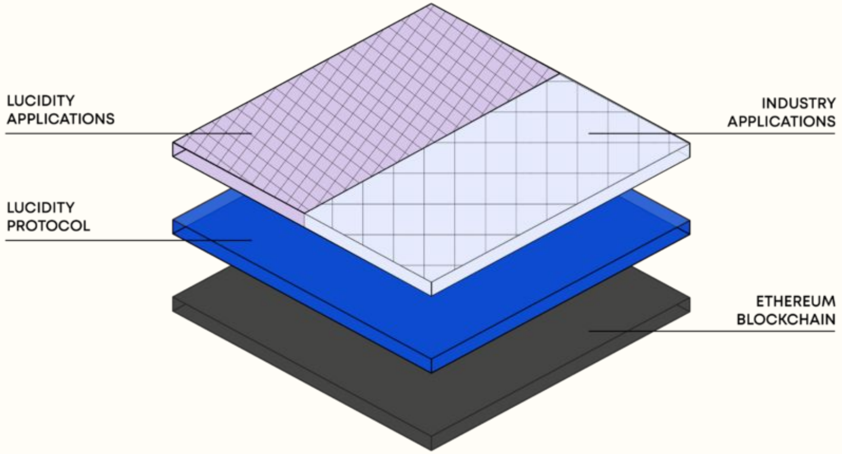
↓ *Snark-ify*

Tiny, **Fixed-Sized** Blockchain

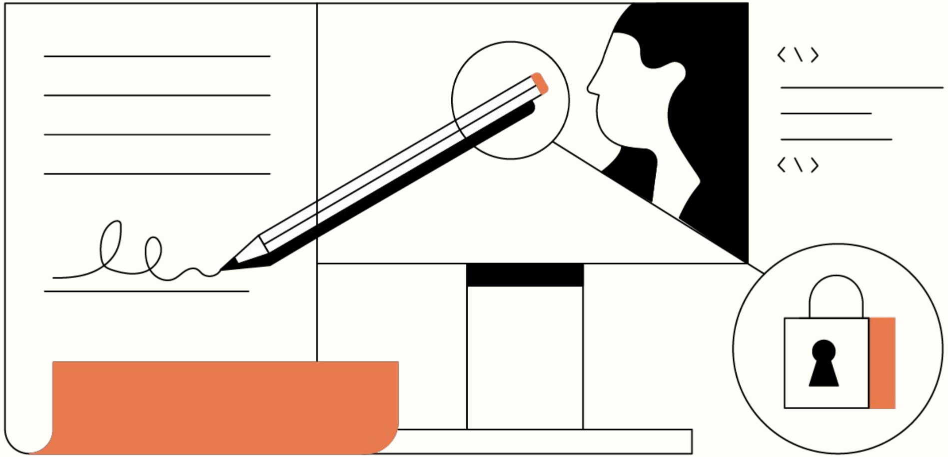
zkEVM

zkEVM: Zero-Knowledge Ethereum Virtual Machine

→ ZK-Rollup (*layer-2*) for Payments and **<u>Generic Smart Contracts</u>**!



2. General Background: Technology Stack and Building Blocks



Terms

- **Polynomial Commitments**
- **FRI**
- **PLONK**

Polynomial Commitments

- **<u>Polynomial Commitments</u>** allow a committer to publish a value (*commitment*), while keeping the value hidden to others (*hiding*).
  - Committer commits to a polynomial P (i.e. bind original message with a polynomial)

PC Schemes	KZG10	IPA	FRI	DARKS
Low level tech	Pairing group	Discrete log group	Hash function	Unknown order group
Setup	**G1, G2** groups **g1, g2** generators **e** pairing function s_k secret value in F	**G** elliptic curve g^n independent elements in G	**H** hash function w unity root	**N** unknown order **g** random in N **q** large integer
Commitment	$(a_0 s^0 + \ldots + a_n s^n) g_1$	$a_0 g_0 + \ldots + a_n g_n$	$H(f(w^0), \ldots, f(w^n))$	$(a_0 q^0 + \ldots + a_d q^d) g$

Different ZKPs have different <u>Commitment Schemes</u>

Polynomial Commitment

→ committer commits to certain polynomial **P** (bind original message to polynomial)

→ committer proves the value of polynomial at certain point **Z** satisfies **P(Z)** through the proof WITHOUT revealing the polynomial

$$\mathbf{P(Z) = z}$$

Polynomial Commitment

A polynomial commitment is a sort of "hash" of some polynomial P(x) with the property that you can perform arithmetic checks on hashes.

```
Polynomial Commitments:
h_P = commit(P(x)) on P(x)
h_Q = commit(Q(x)) on Q(x)
h_R = commit(R(x)) on R(x)
```

Can show:

- If **P(x) + Q(x) = R(x)** *OR* **P(x) * Q(x) = R(x)**, you can generate a proof that proves this relation against h_P, h_Q, h_R
- If P(z) = a you can generate a proof (known as an "opening proof") that the evaluation of **P** at **z** is indeed **a**

Polynomial Protocol

Prover

cm(a(x))

cm(b(x))

cm(c(x))

r

a(r), opening(cm, r, a(r))

b(r), opening(cm, r, b(r))

c(r), opening(cm, r, c(r))

Verifier

Verify a(r)+b(r) = c(r)

Polynomial Commitment: Use-Cases

Use **polynomial commitments** instead of **Merkle trees**

- Replace current Merkle roots of block data (eg. of Eth2 shard blocks) <u>with</u> polynomial commitments
- Replace Merkle branches <u>with</u> opening proofs

FRI-Proofs

FRI-based zkSNARKs

- **FRI-Proofs**
  - Verify all the polynomial commitments based on hash functions
  - **Hashing** is the main mathematical operation

PLONK

PLONK is a zk-SNARK system / construction developed by **Aztec**

PLONK

PLONK

FRI-based Polynomial Commitments

aztec

PLONK

Proof Generation: *O(n log n)* for ALL zk-SNARKs

Differences:

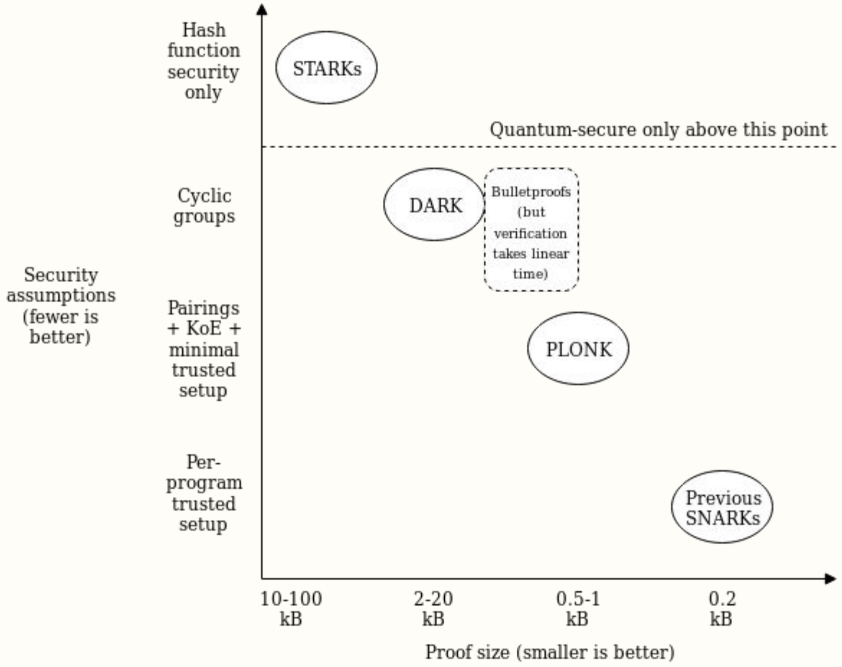
Proof size

Verification time

Type and size of the "reference string" of Trusted Setup

PLONK

Comparing zkSNARKs:



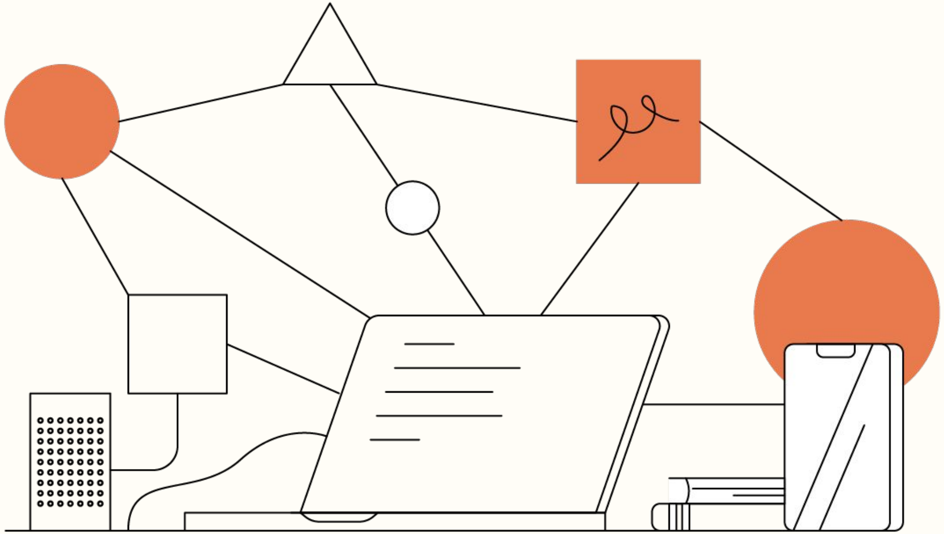
aztec

PLONK

PLONK Characteristics:

- Fast proof generation
- Constant-time verification
- Small proof sizes

3. **Polygon Zero (zkEVM)**



Polygon Zero

→ Polygon Zero
→ Tech Stack
→ Circuit Optimizations
→ Low-Level Optimizations

polygon

Polygon Zero

History

Pre-2014: recursive proofs are theoretical

2019: 2 minutes to a generate recursive proof

2020: 60s to generate a recursive proof

2021: …

Polygon Zero: *Recursive* SNARK-based zkEVM

Plonky 2 = PLONK + FRI + Magic (100x speedup)

Polygon Zero

- Natively Ethereum compatible (supports generic smart contracts)
- Constant 1M gas to verify plonky 2 proof on mainchain
- **170 ms** recursive proof generation on MacBook Pro

Polygon Zero

Polygon Zero:

The most <u>scalable</u> zkEVM,
powered by the plonky2 SNARK system

polygon

Tech Stack

Performance Is A Big Deal!

Generating SNARK proofs are computationally expensive, supporting conventional code with features like:

- unbounded loops
- random access
- binary arithmetic
- arbitrary control flow

The EVM is particularly difficult because its word size is 256 bits (256-bit machine)

polygon

EVM Statistics

The average Ethereum transaction involves:

- ~3,400 total instructions
- ~1,200 bytes of memory
- ~24 storage operations
- ~5 contracts calls

Opcode	Avg Executions
ADD	149
AND	88
MUL	19
MULMOD	8
DIV	20
SLOAD	18
SHA3	13

polygon

SNARK Systems

	KZG based	Halo based	FRI based
Prover speed	Moderate	Moderate	Variable
Recursion cost	High	Moderate	Very low
Proof size	~1 kb	~20 kb	40~300 kb
Gas to verify	~300k	Very high	800k~5M
Security	Pairings	ECDLP	CRH

polygon

FRI “Blowup Factor” Parameter

Blowup Factor = how much redundancy we add to polynomial before we generate the commitment to it

Fast prover = small blowup factor (less data committed to it)

↓

Smaller blowup factor = reduces security of FRI-protocol

polygon

FRI “Blowup Factor” Parameter

Dilemma: fast proof generation OR smaller proofs

Solution: Recursion

- Take larger proof, shrink it by wrapping it in a recursive proof with a larger blowup factor. Think of this as a compression technique
- ~**45kb** proof size (Ethereum charges 16 gas \ byte)

polygon

FRI Field Size

Most SNARKs are encoded in a 256-bit prime field

256-bit multiplication is expensive!

Instead encode in a 64 bit field

This makes all our arithmetic much faster!

Circuit Optimizations

Proof Recursion

Naive zkSNARK (circuits) use **2^16 gates**

polygon

Hash Function

The FRI verifier uses a lot of hashes.

Currently they make up **75%** of the circuit.

Poseidon, a popular algebraic hash used by many other ZK teams.

polygon

Matrix Multiplication

Poseidon MDS matrix

$$
\begin{pmatrix}
1 & 1 & 2 & 1 & 8 & 32 & 4 & 256 \\
256 & 1 & 1 & 2 & 1 & 8 & 32 & 4 \\
4 & 256 & 1 & 1 & 2 & 1 & 8 & 32 \\
32 & 4 & 256 & 1 & 1 & 2 & 1 & 8 \\
8 & 32 & 4 & 256 & 1 & 1 & 2 & 1 \\
1 & 8 & 32 & 4 & 256 & 1 & 1 & 2 \\
2 & 1 & 8 & 32 & 4 & 256 & 1 & 1 \\
1 & 2 & 1 & 8 & 32 & 4 & 256 & 1
\end{pmatrix}
$$

polygon

Result

Recursive circuit uses only **2^12 gates**

Low-Level Optimizations

polygon

Prime Field

Prime field Fp, where p is prime number:

$$\mathbb{F}\,\mathbf{p}$$

polygon

Prime Field

Any computation that you want to prove with a zkSNARK has to be expressed as arithmetic operations on a finite field, known as a **prime field**

polygon

Prime Field

Prime Number: $2^{64} - 2^{32} + 1$

Vector Operations

Vector Instructions = 2x speedup over poseidon hash

8f80fc7d4bcb0c1b	424a14b932f990f5	18176ea4160abc5d	69728b5ab45d1034
d0d73217796c1e3	312c68a580dbac62	9bba9009a07f3f96	dc3b642b6f3666e2
9c8e6f9ec361cdfe	73767d5eb3d53d57	b3d1feadb689fbf3	45adef8723937715

polygon

Result

Lightning fast zero-knowledge proofs!

References

https://mirprotocol.org/blog/Introducing-Mir

https://blog.polygon.technology/polygon-takes-a-major-lead-in-zk-rollups-welcomes-mir-a-groundbreaking-zk-startup-in-a-400m-deal/

https://minaprotocol.com/blog/what-are-zk-snarks

https://ethresear.ch/t/using-polynomial-commitments-to-replace-state-roots/7095

Thank you!

SCALABLE SYSTEMS
& SOFTWARE
RESEARCH GROUP

https://sss.cse.lehigh.edu/