Lesson 16 - Extra Topics / trends / review

Upcoming Events

A good guide to web3 events generally is Crypto Nomads

zkSummit - Athens - April 10th
zkHack Krakow - May 17-19

Noir Example Projects

See this list of resources

- Anonymous proof of token ownership on Aztec (for token-gated access)
  - Sequi
  - Cyclone

Governance - MeloCafe
Circuits - Anonymous on-chain voting

Verification cost

From article

The verification costs for SNARKs can vary, but are well-understood and often quite good. For example, PlonK proofs cost about 290,000 gas to verify on Ethereum, while StarkWare's proofs cost about 5 million gas.

From Plonk benchmark article

	PLONK	Groth16
MiMC Prover Time	5.6s	16.5s
SHA-256 Prover Time	6.6s	1.4s
Verifier Gas Cost	223k	203k
Proof Size	0.51kb	0.13kB

Also see Security and Performance article
Reducing verification cost research

Proof generation benchmarks

Miden and Risc Zero benchmarks
Benchmarking SNARKS

Caulk

= Lookup Arguments in Sublinear Time
Caulk can be used for membership proofs and lookup arguments and outperforms all existing alternatives in prover time by orders of magnitude.

See Paper
See zk study club Video
See Slides

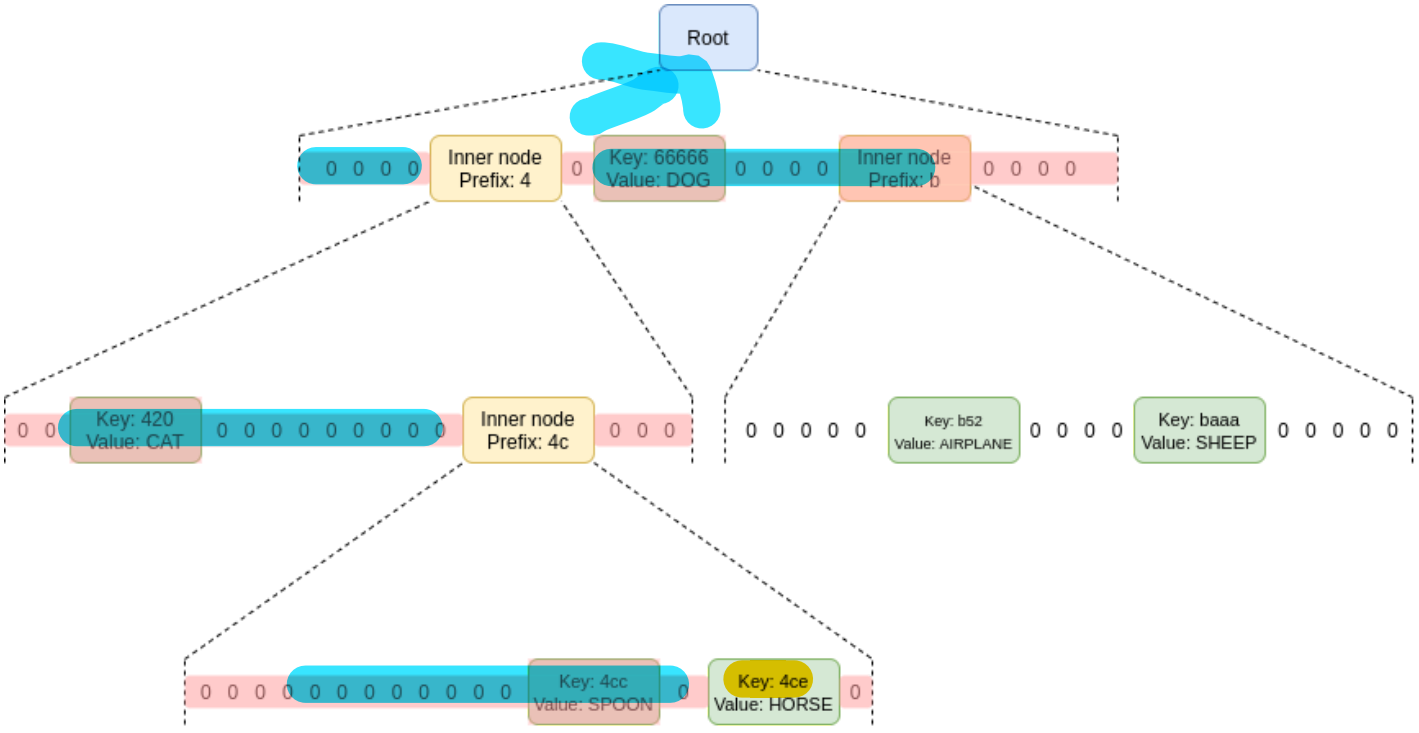
Verkle trees

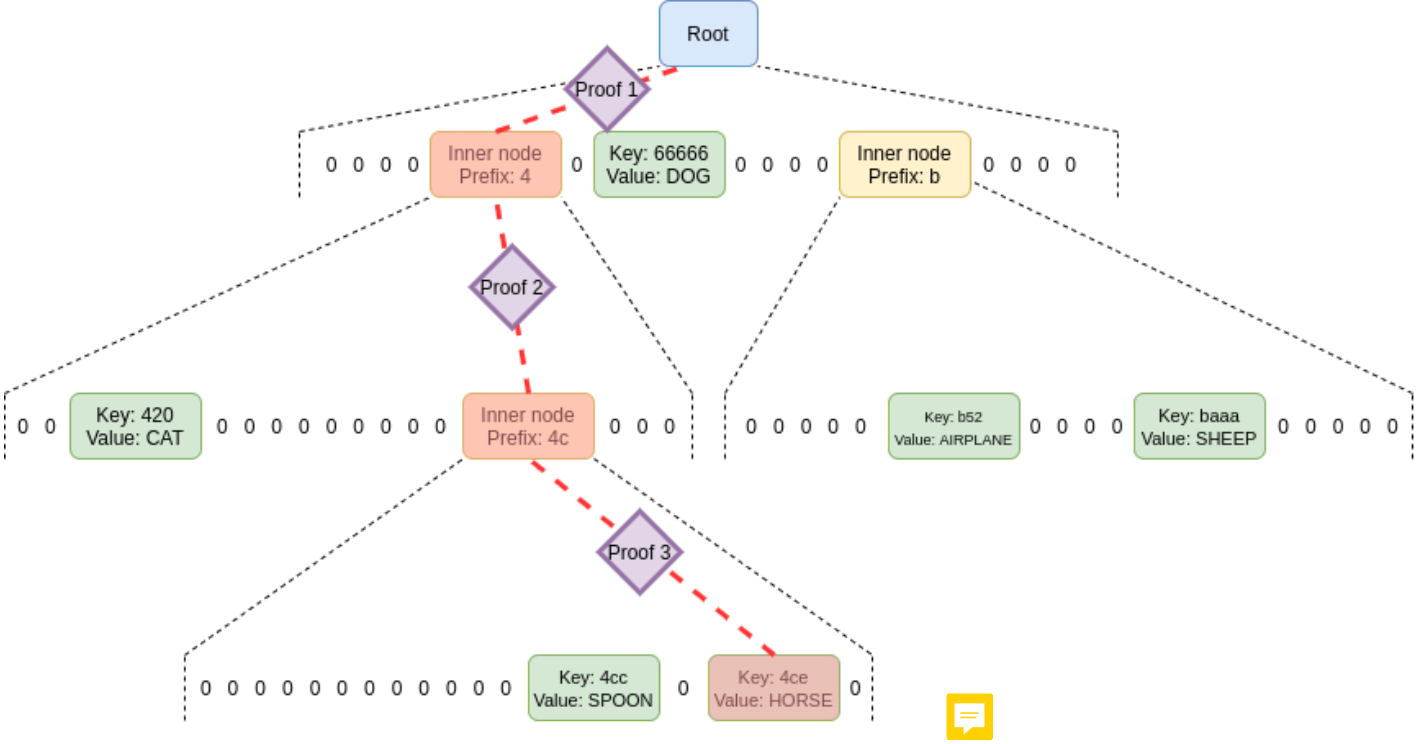
https://vitalik.eth.limo/general/2021/06/18/verkle.html

See article
and Ethereum Cat Herders Videos

Like merkle trees, you can put a large amount of data into a Verkle tree, and make a short proof ("witness") of any single piece, or set of pieces, of that data that can be verified by someone who only has the root of the tree.
What Verkle trees provide, however, is that they are much more efficient in proof size. If a tree contains a billion pieces of data, making a proof in a traditional binary Merkle tree would require about 1 kilobyte, but in a Verkle tree the proof would be less than 150 bytes.
Verkle trees replace hash commitments with vector commitments or better still a polynomial commitment.
Polynomial commitments give us more flexibility that lets us improve efficiency, and the simplest and most efficient vector commitments available are polynomial commitments.

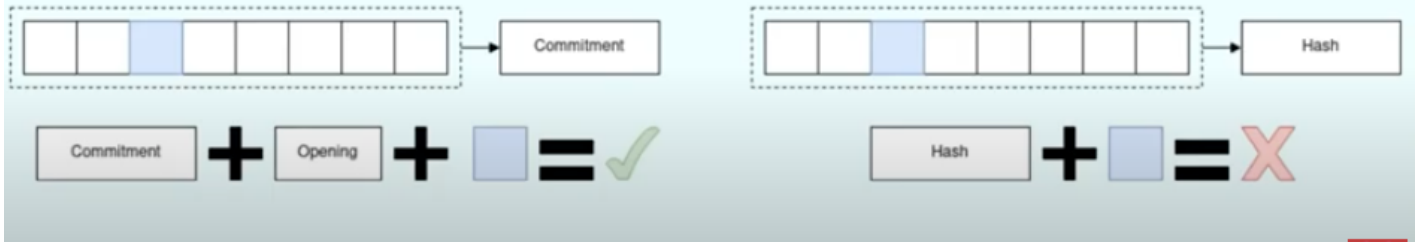
The number of nodes needed in a merkle proof is much greater than in a verkle proof





Vector commitments vs. Hash

- Vector commitments: existence of an “opening”, a small payload that allow for the verification of a portion of the source data without revealing it all.
- Hash : verifying a portion of the data = revealing the whole data.



Proof sizes

Merkle

Leaf data +
15 sibling
32 bytes each
for each level (~7)

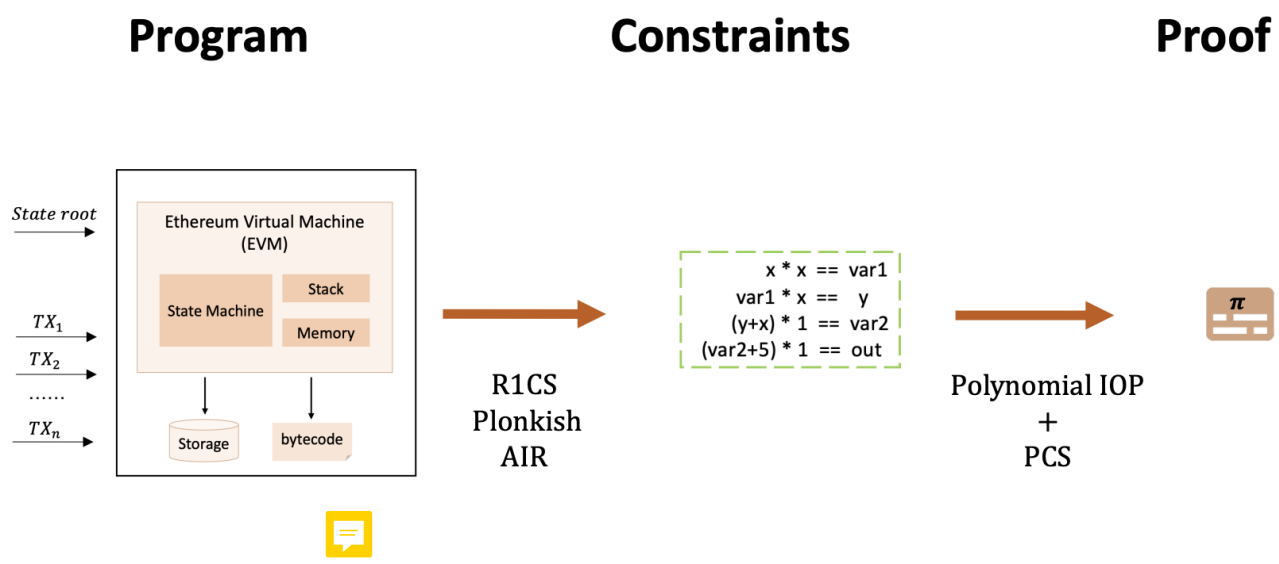
= ~3.5MB for 1K leaves

Verkle

Leaf data +
commitment + value + index
32 + 32 + 1 bytes
for ~4 levels
+ small constant-size data

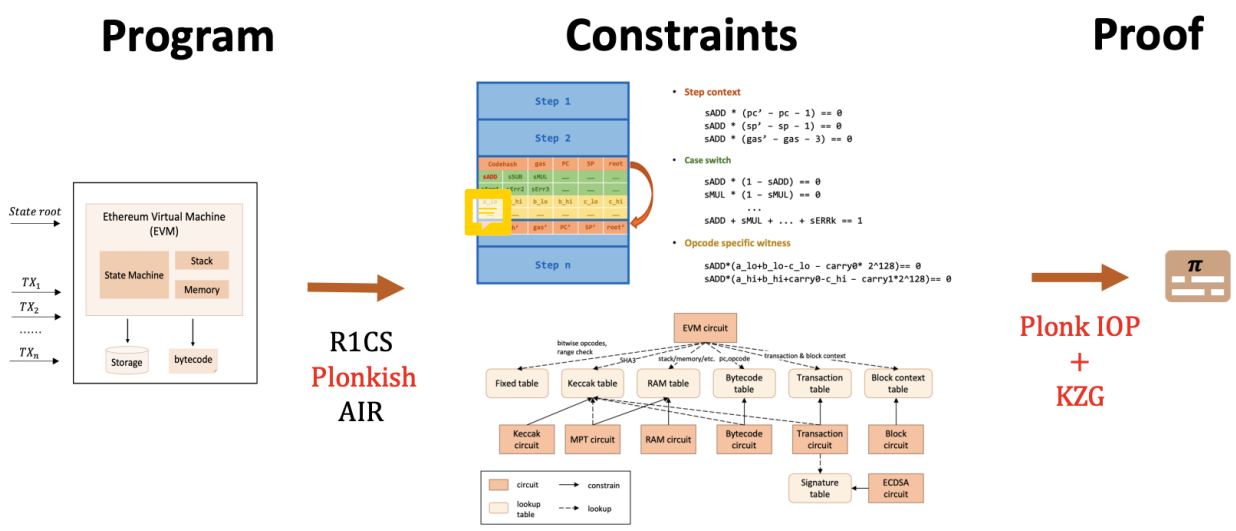
= ~ 150K for 1K leaves

Scroll overall process



The workflow of zero-knowledge proof

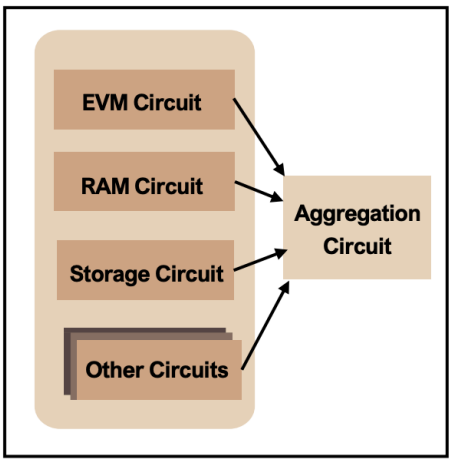
Scroll



Two-layer architecture

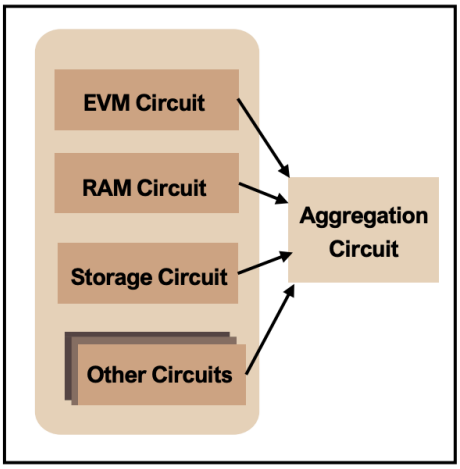
Scroll

zkEVM



- The first layer needs to handle large computation
  - Custom gate, Lookup support (“expressive”, customized)
  - Hardware friendly prover (parallelizable, low peak memory)
  - The verification circuit is small
  - Transparent or Universal trusted setup
- Some promising candidates
  - Plonky2/Starky /eSTARK
  - Halo2/Halo2-KZG
  - New IOP without FFTs (i.e. HyperPlonk, Plonk without FFT)
  - If Spartan/Virgo/... (sumcheck based) or Nova can support Plonkish

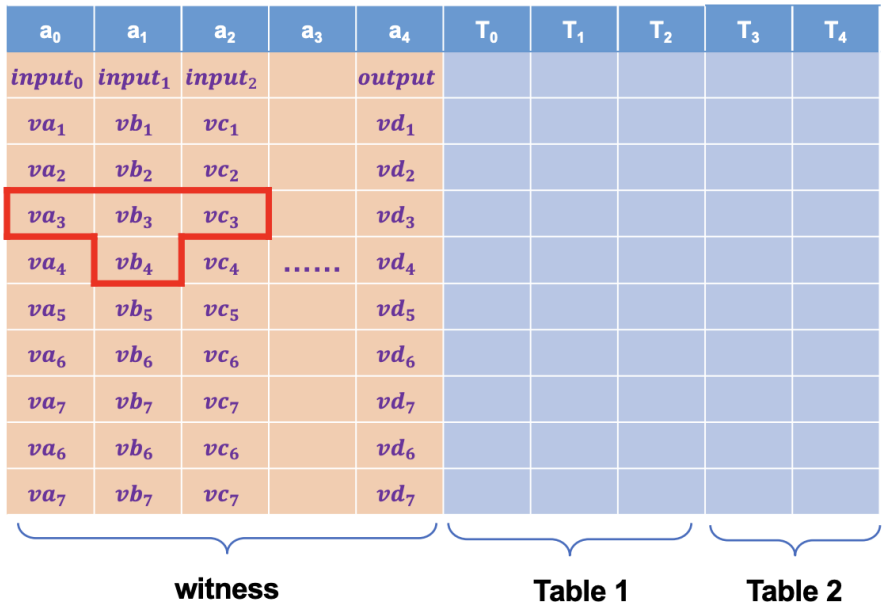
zkEVM



- **The first layer is Halo2-KZG** (Poseidon hash transcript)
  - Custom gate, Lookup support
  - Good enough prover performance (GPU prover)
  - The verification circuit is “small”
  - Universal trusted setup
- **The second layer is Halo2-KZG** (Keccak hash transcript)
  - Custom gate, Lookup support (express non-native efficiently)
  - Good enough prover performance (GPU prover)
  - The final verification cost can be configured to be really small

Scroll custom gate

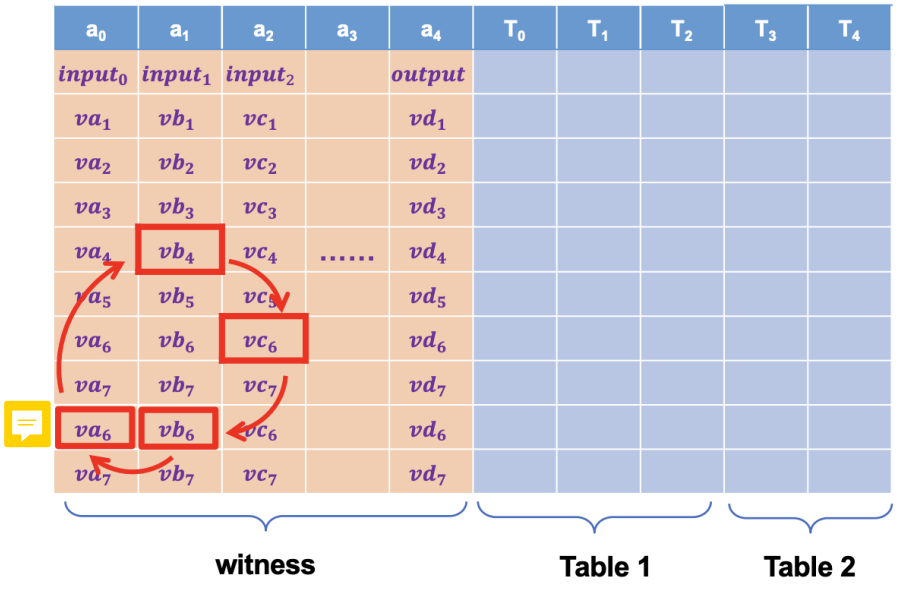
Taken from presentation



$$va_3 * vb_3 * vc_3 - vb_4 = 0$$

Plonkish Arithmetization – Permutation

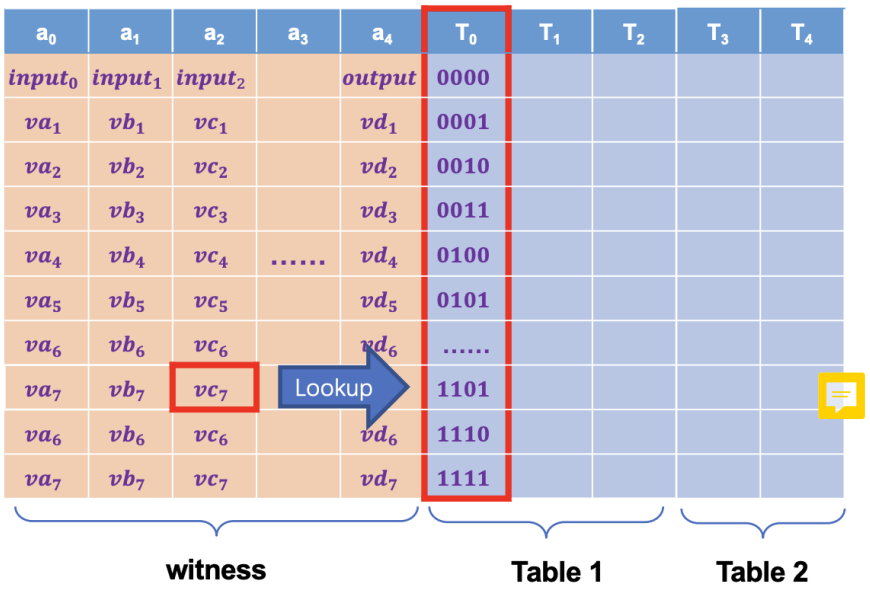
Scroll



$$vb_4 = vc_6 = vb_6 = va_6$$

Plonkish Arithmetization – Lookup argument

Scroll



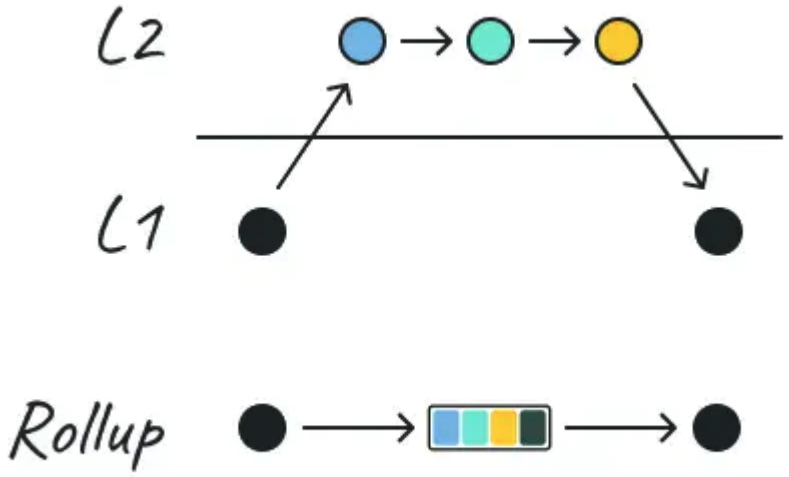
$vc_7 \in [0, 15]$

Current directions

L2 implementations

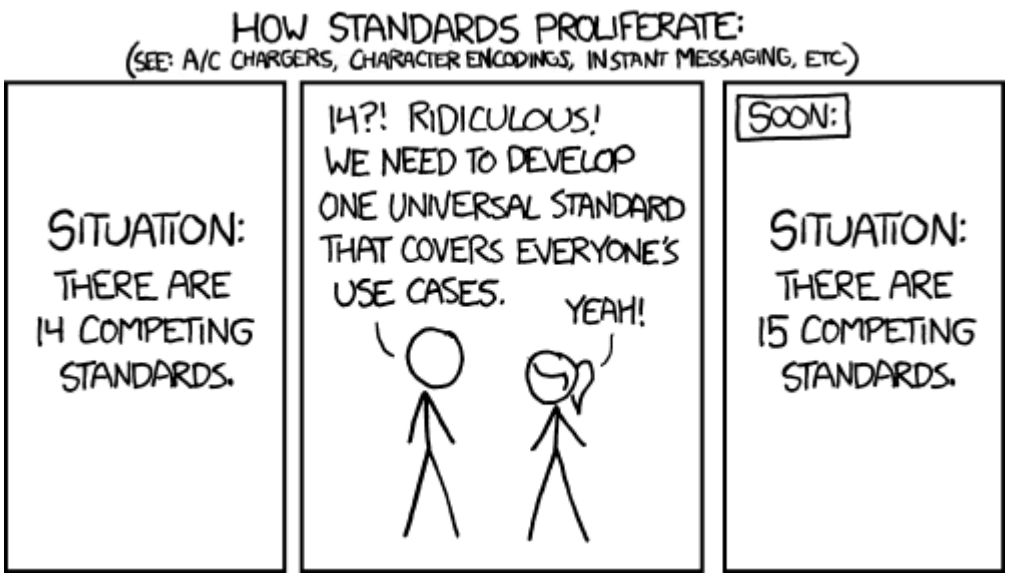
- Starknet
- ZKSync
- Polygon
- Aztec
  Also inter L2 communication - Mangata

see article

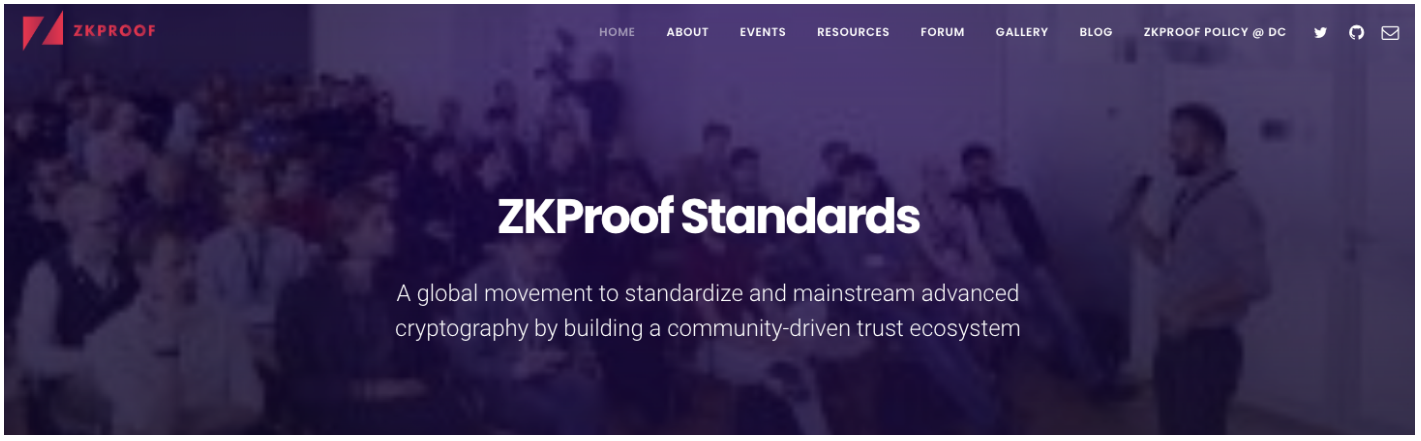


Standards

oblig xkcd

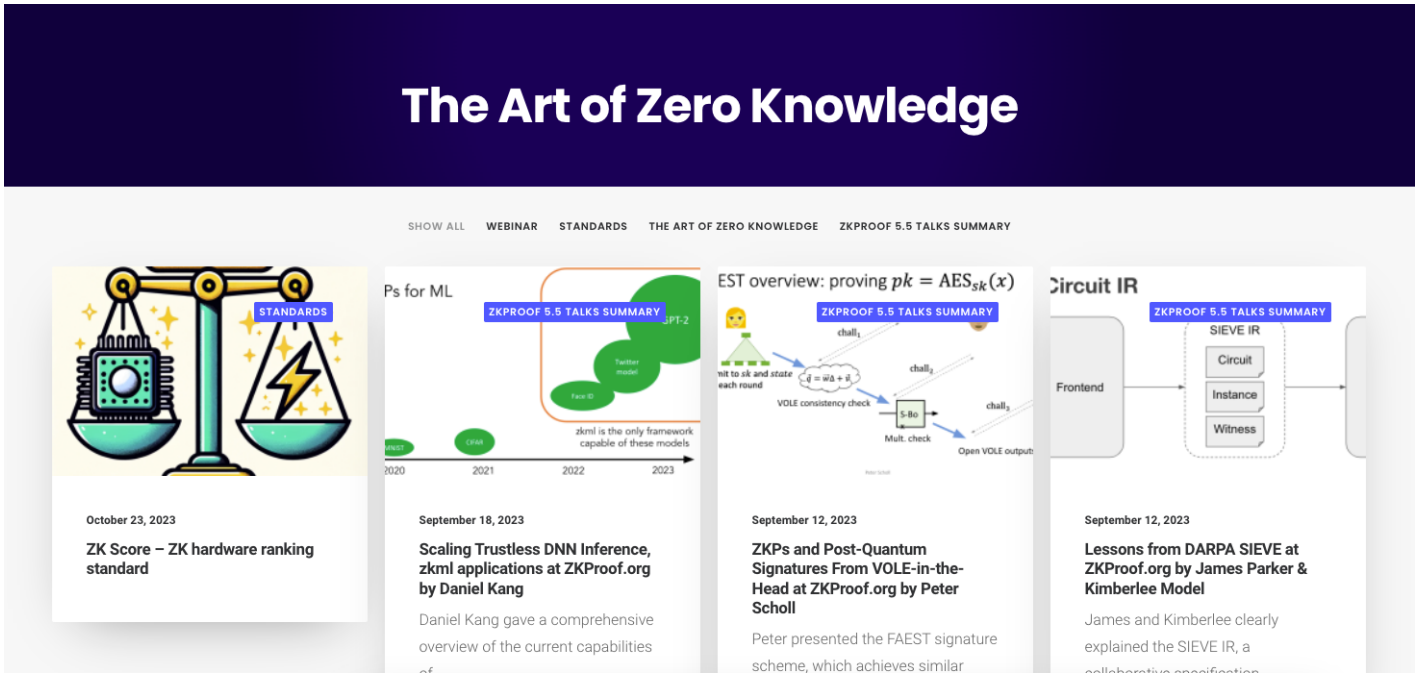


ZKproof.org



Community events

Blog



Working Groups

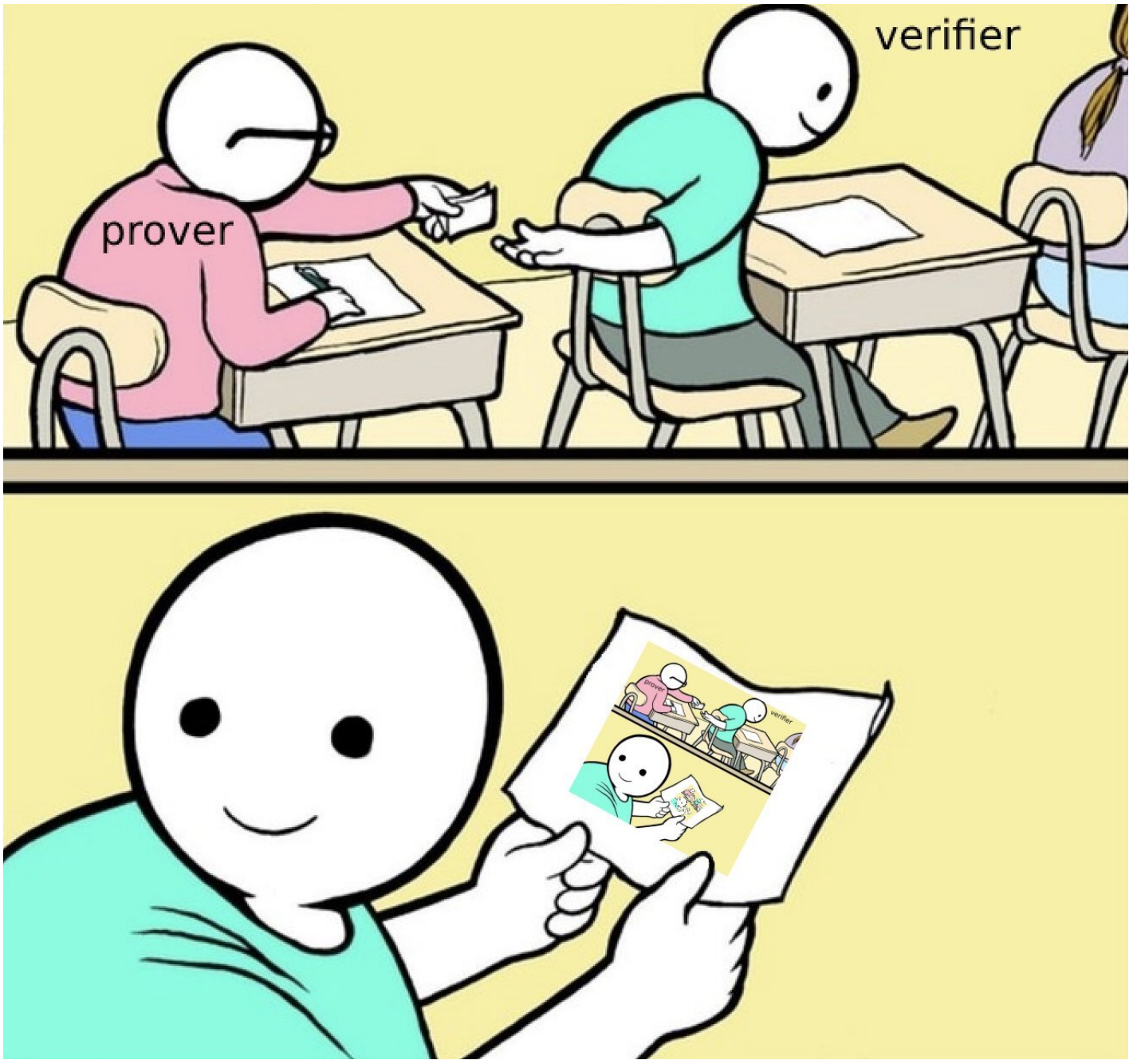
See groups

- Plonkish constraint systems
- Fiat Shamir compiler
- Sigma protocols

Community

See details

Recursion overview



The ability to create recursive proofs is a very powerful technique.
Proof recursion is at the heart of the Mina blockchain - a succinct blockchain.

What we are trying to achieve using a common circuit is

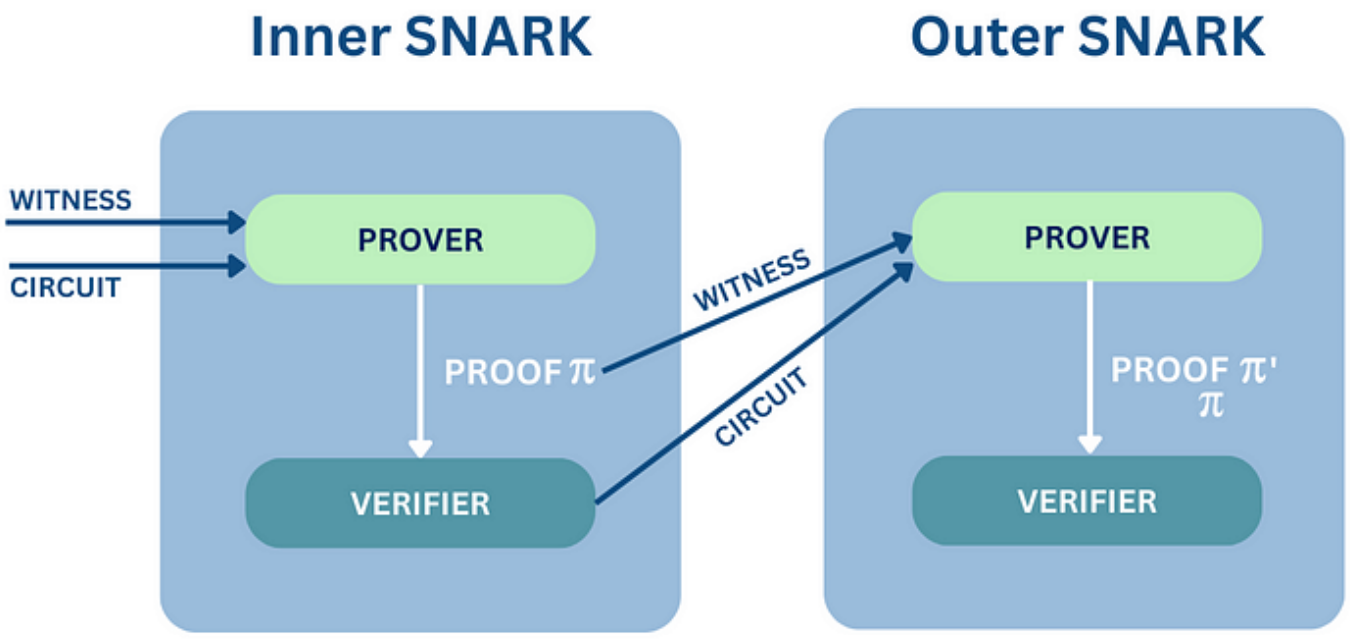
- Step 1 : have proof A as an input to the circuit that will produce proof B
- Step 2 : Proof B is the input to proof C ...

The how of recursion

See video from Stanford MOOC

One approach is to include a verifier within your circuit so that the verification of a previous proof can be part of the current proof.

From Article



A problem with this approach is if the verifier becomes too large, then the recursion will 'blow up' and become infeasible.

To do this we use cycles of elliptic curves. The curves are specially chosen to have complementary properties.

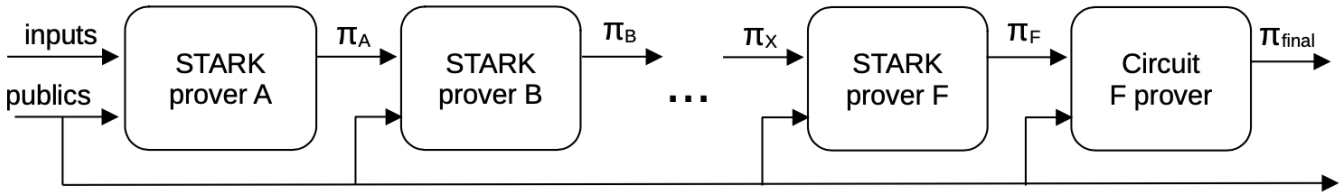
These curves are referred to as “tick” and “tock” within the Mina source code.

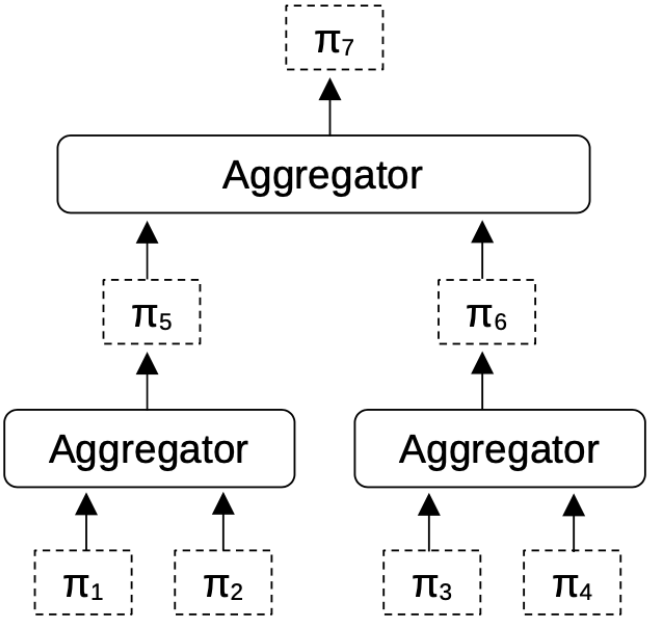
Tock is used to prove the verification of a Tick proof and outputs a Tick proof.
Tick is used to prove the verification of a Tock proof and outputs a Tock proof.
In other words,

- Provetock(Verify(Tick))=Tickproof
- Provetick(Verify(Tock))=Tockproof

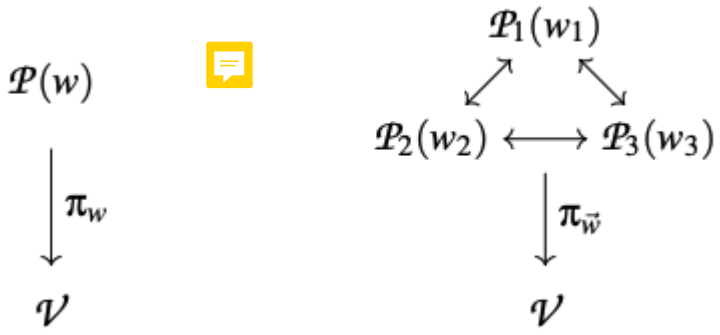
Proof Aggregation

We often want to combine proofs together, for example if we have proofs for multiple circuits in a zkEVM.





Collaborative SNARKS



The paper generalises from public proofs about a secret w held by a single party, to public proofs about a secret $\vec{w}$ = $(w1, \ldots, wN)$ distributed among N parties, where party i has w_i , for $i \in [N]$, as in Fig. 1.
The proof generation process should reveal nothing new about $\vec{w}$ to a coalition of parties, other than the validity of $\vec{w}$

In terms of zero knowledge, each prover is now concerned, not only that the verifier may learn the witness, but also that the other provers may learn their witness.

A prover will know how many other provers there are, but they don't know if any of the others have distinct identities.

What does soundness mean in this setting ?

If the provers collectively pooled all their information, they could construct the witnesses, this is a slightly weaker definition than each prover knowing one witness.

The verifier doesn't interact with each prover separately, it receives one message from all of them.

The processes mirrors that of single prover zkSNARKS

- Setup - gives pp (public parameters)
- Prove - takes ($pp, x, P_1(w_1) \ldots P_N(w_N)$) -> π
  x is the public input, P is the prover, w is the witness
- Verify(x, pp, π) -> {0,1}

The proving process is given secret sharing of the witnesses and the R1CS details. Its goal is to produce the proof.

A generic way of implementing this would give a 10^6 slowdown over native evaluation.

Usual problems for single proofs

- ECC
- FTTs

MPC Bottlenecks

- Polynomial divisions
- Partial products
- Merkle tree evaluations

These have been solved (Merkle tree to some extent)

The solution from Ozdemir and Boneh uses
SPDZ and GSZ MPC protocols
and Groth16 / Marlin / Plonk

Use Case

On chain dark pool Renegade uses collaborative SNARKS
See Documentation

Fundamentally, Renegade simply consists of a p2p gossip network of many independent relayers that constantly handshake and perform MPCs with each other as new orders enter the system. Relayers never custody assets, and are merely given view access to the wallet in order to compute pairwise MPCs.

The MPC computes *matching engine execution*. That is, given the two orders (each held privately by different relayers), the two parties will compute a MPC that implements matching engine execution on those two orders.
This allows for full anonymity, as no information whatsoever is leaked about the order in advance of the MPC.
After the MPC, the parties only learn what tokens were swapped; if there was no match between the orders, then no additional information is leaked.

By wrapping zero-knowledge proof generation inside of a MPC algorithm, collaborative SNARKs allow for the relayers to collaboratively prove a particular NP statement, VALID MATCH MPC.
This statement essentially claims that given the publicly-known commitments to order information and a public commitment to a matches tuple, both traders do indeed know valid input orders."

Resources

Paper
Repo
Paper

Projects emphasising privacy

Zama.ai

See Docs
See White paper
Zama have created a fhEVM incorporating FHE into the operation of the EVM.
Specifically

- An fhEVM, which integrates Zama's open source FHE library TFHE-rs nto a typical EVM, exposing homomorphic operations as precompiled contracts.
- A decryption mechanism based on distributed threshold protocols that prevents misuse by malicious smart contracts, yet is flexible and non- intrusive to honest smart-contract developers.
- A Solidity library that makes it easy for smart-contract developers to use encrypted data in their contracts, without any changes to compilation tools.

Users can create a private blockchain uses this EVM as the execution layer.
It would be interesting to see this combined with other chains via some off the interoperability mechanisms we have seen.

Aztec

See Site
Execution is performed off chain, and state is encrypted and private to its owner.

Namada

See Docs
See Vision
Namada is a protocol which supports multi-chain asset-agnostic privacy
Namada uses the CometBFT BFT consensus algorithm. Namada enables multi-asset private transfers for any native or non-native asset using a multi-asset shielded pool derived from the Sapling circuit.

Obscuro

See Docs
This is a L2, where all transactions and the internal state of application contracts are encrypted and hidden. Execution is performed by a bespoke encrypted EVM.

Penumbra

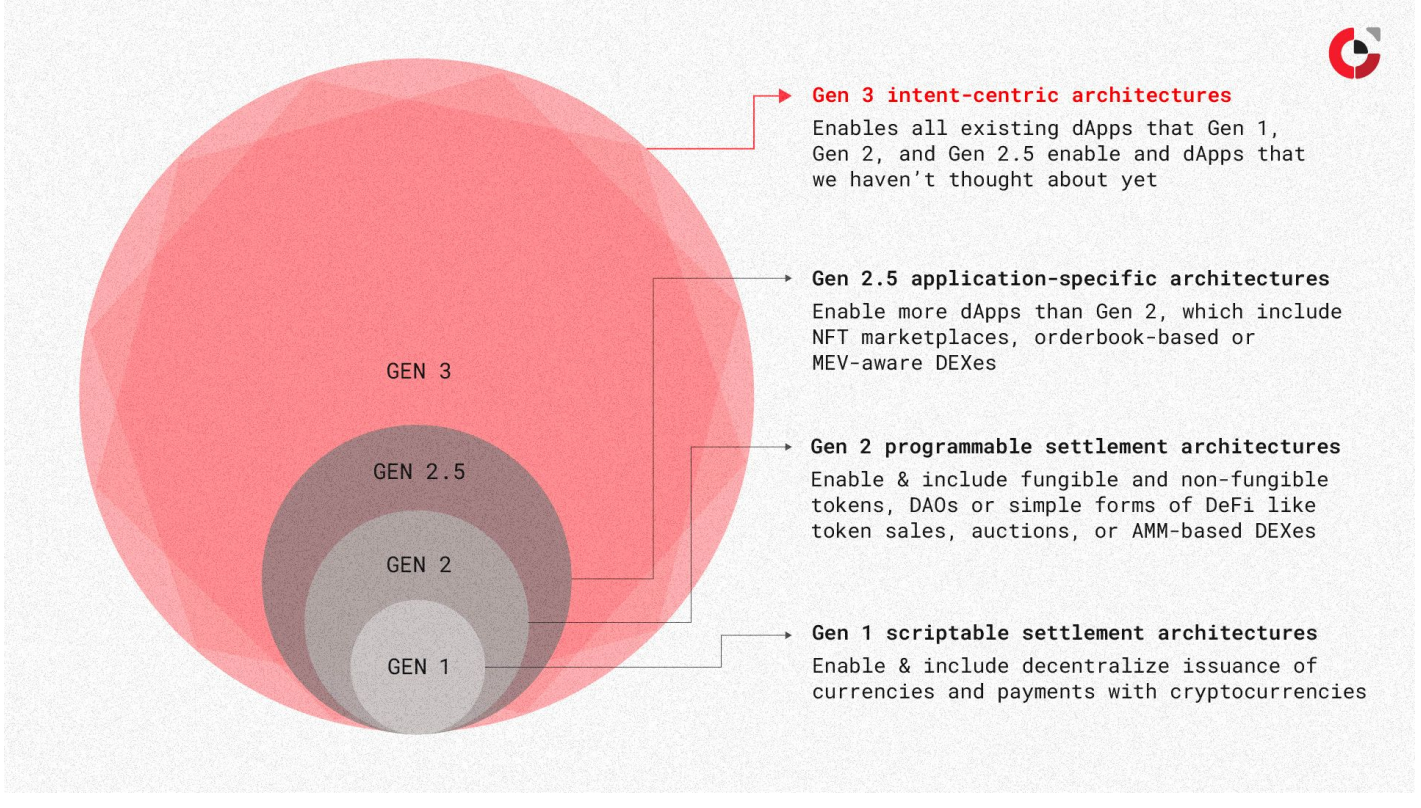
See Docs
Penumbra is a shielded, cross-chain network allowing private trading in crypto assets.
Penumbra records all value in a single multi-asset shielded pool based on the Zcash Sapling design, but allows private transactions in any kind of IBC (see IBC protocol) asse
Inbound IBC transfers shield value as it moves into the zone, while outbound IBC transfers unshield value.

Anoma

See paper
See research
Listen to ZK Podcast

See Blog

Anoma is designed around the concept of an Intent centric architecture



Anoma design features

1. **Intents**: Users supply transaction intentions, which are matched with counterparts.
2. **Privacy Measures**:
   Transaction and user details are kept confidential
3. **Cross-Chain Interactions**:
   Anoma supports cross chain interactions
4. **MTCS**:
   Multilateral Trade Credit Set-Off, this allows for the offsetting of multiple obligations simultaneously, optimising liquidity usage.
5. **Atomic Settlements**:
   Interaction across chains is atomic.

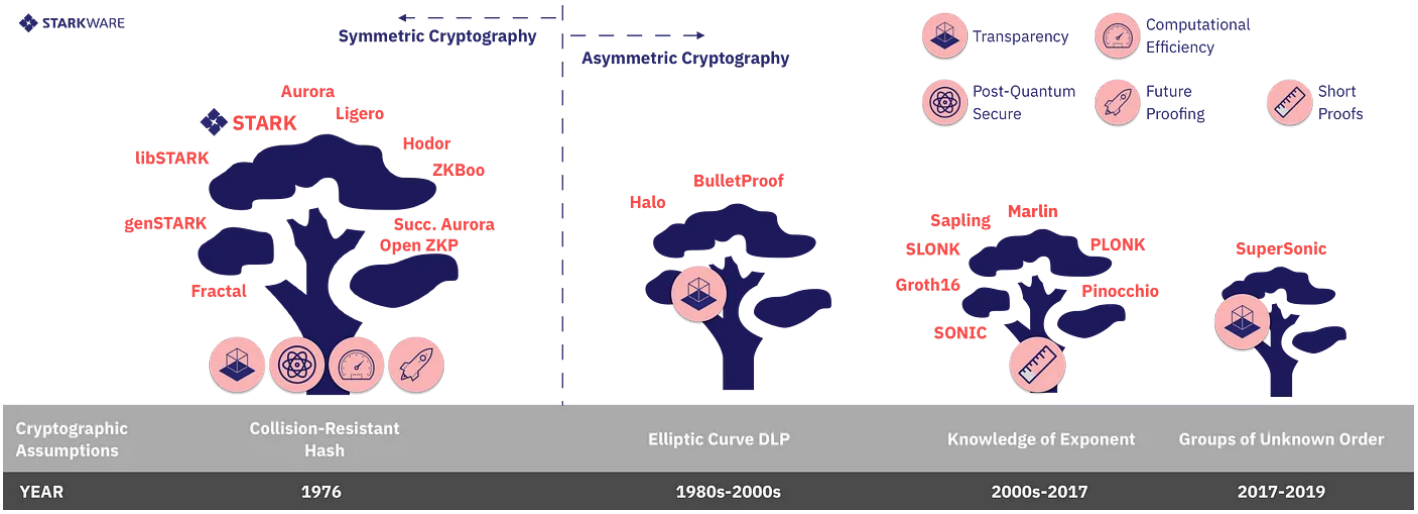
Useful talks from Devcon

ZKP Workshop video
Zk Application showcase video
The KZG Ceremony video
zkEVM Technical details video
Vampire SNARK from Nethermind video
Shielded Voting and Threshold encryption video
Self Sovereign Identity video
Are your ZKPs correct ? video
ZKP Performance video
Proving execution in zkEVM video
ZKP Introduction video
Autonomous Worlds workshop video 

Course Review

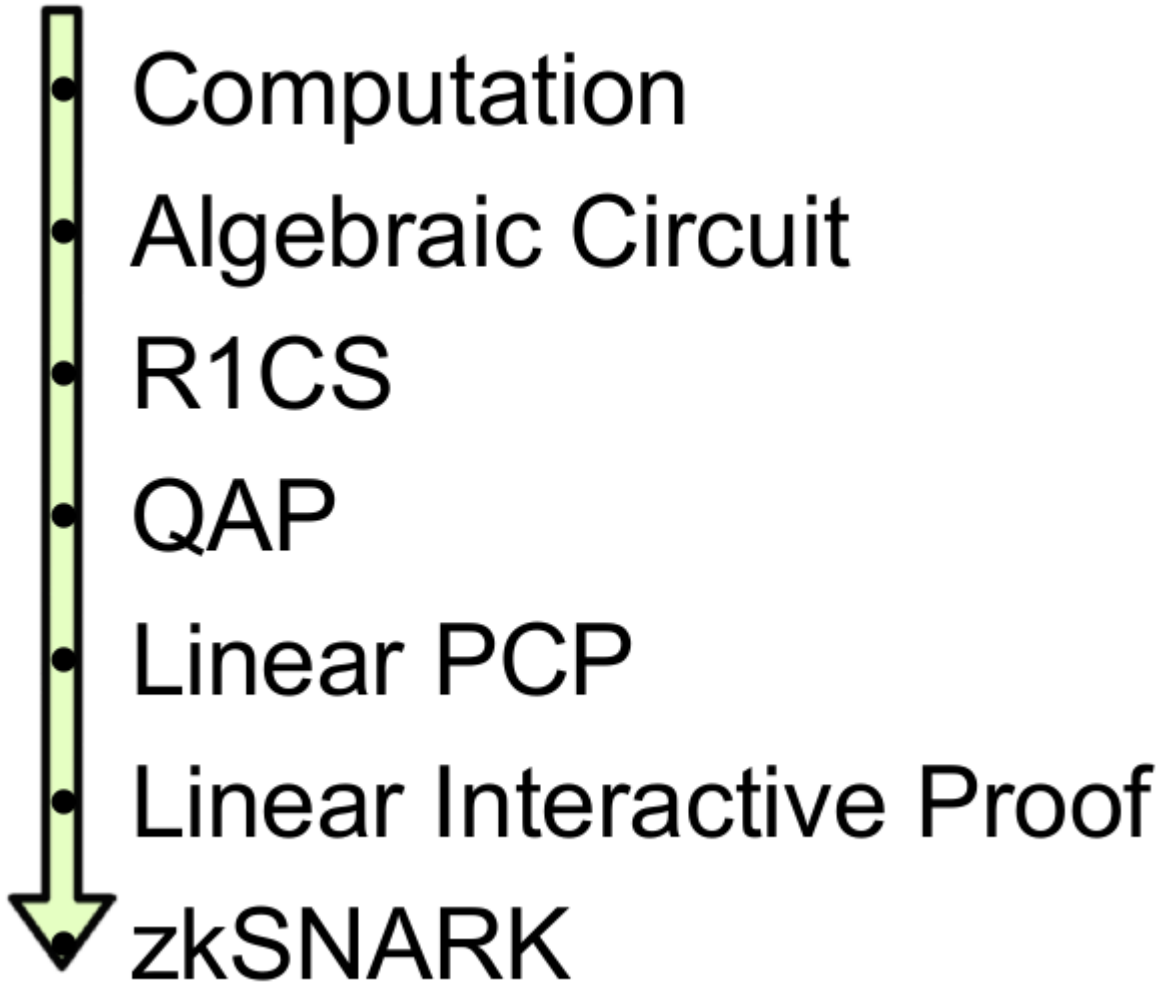
Lesson 1 & 2

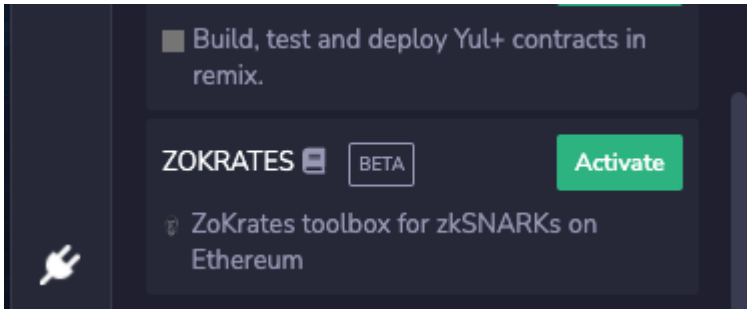
- Introductory maths & cryptography
- General ZKP theory



	SNARKs	STARKs	Bulletproofs
Algorithmic complexity: prover	O(N * log(N))	O(N * poly-log(N))	O(N * log(N))
Algorithmic complexity: verifier	~O(1)	O(poly-log(N))	O(N)
Communication complexity (proof size)	~O(1)	O(poly-log(N))	O(log(N))
- size estimate for 1 TX	Tx: 200 bytes, Key: 50 MB	45 kB	1.5 kb
- size estimate for 10.000 TX	Tx: 200 bytes, Key: 500 GB	135 kb	2.5 kb
Ethereum/EVM verification gas cost	~600k (Groth16)	~2.5M (estimate, no impl.)	N/A
Trusted setup required?	YES 😒	NO 😄	NO 😄
Post-quantum secure	NO 😒	YES 😄	NO 😒
Crypto assumptions	Strong 😒	Collision resistant hashes 😄	Discrete log 😏

zkSNARKS





Lesson 3

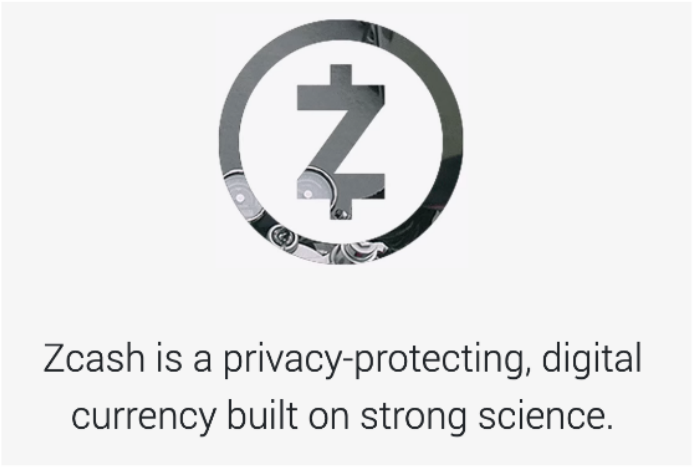
- ZKP use cases / rollups

Lessons 4 - 5

- Starknet / Cairo

Lesson 6

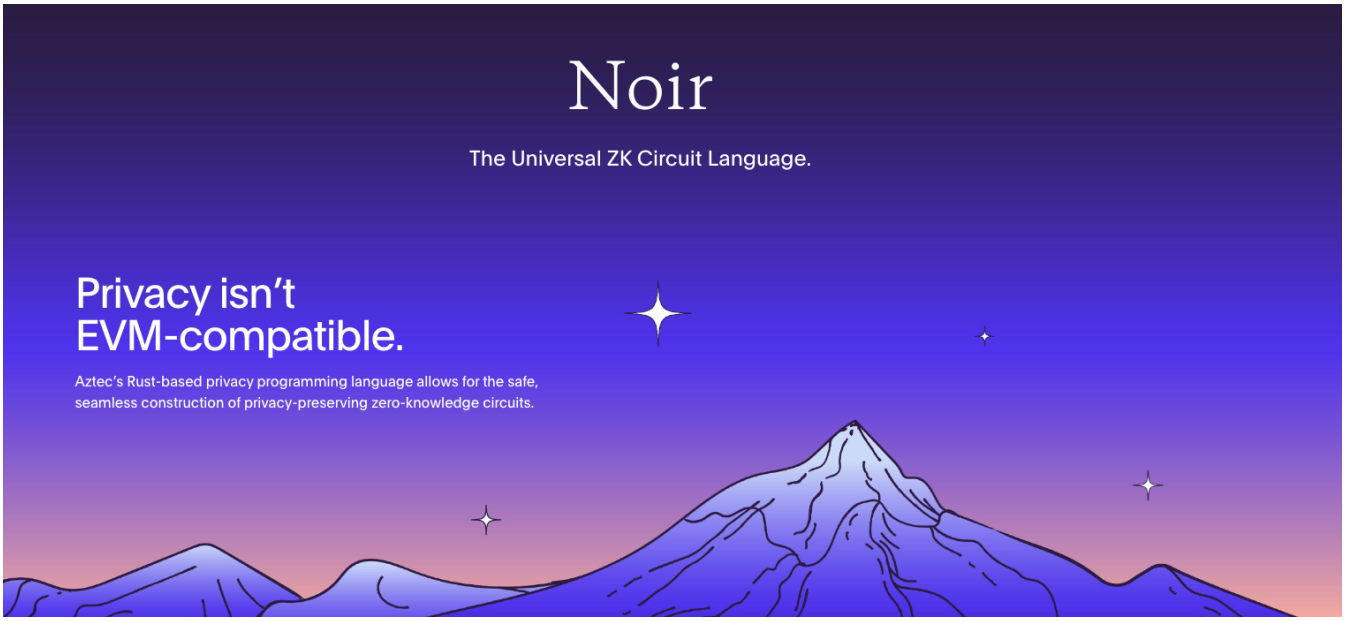
- Confidential tokens



Also Nightfall , ZKDai

Lesson 7

- Noir

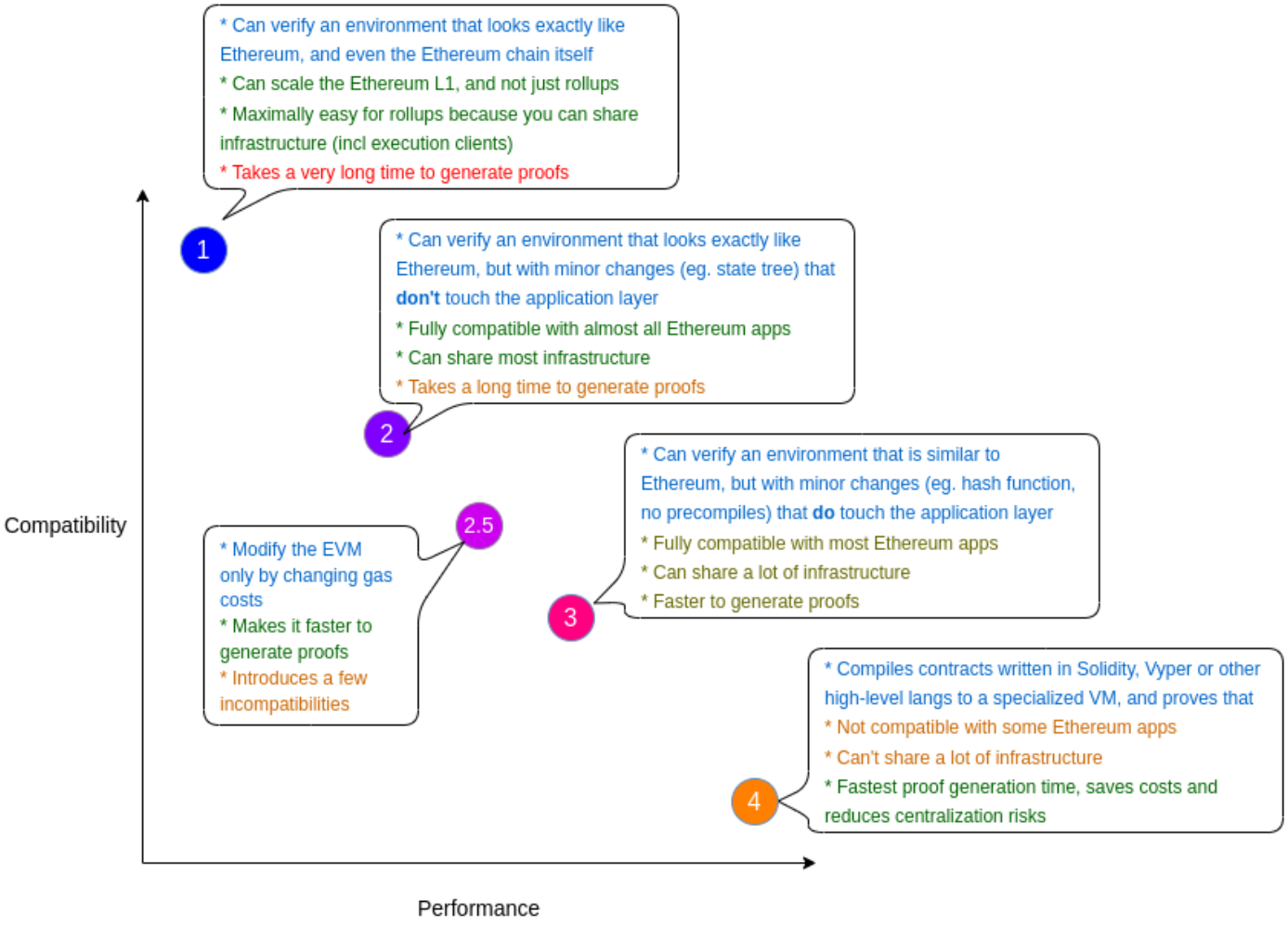


Lesson 8 - 9

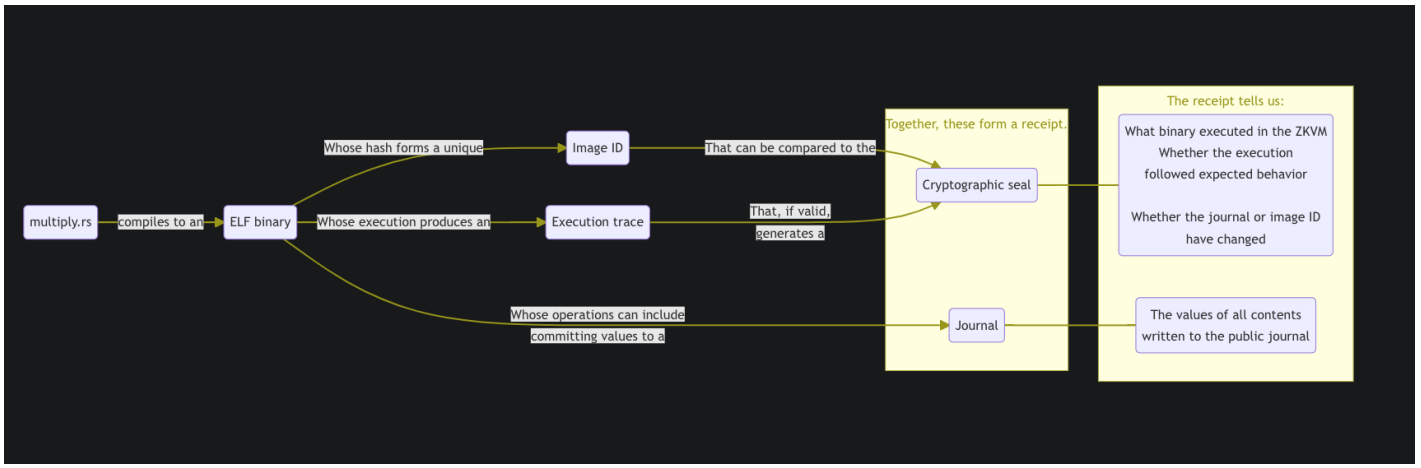
- Mina
- zkApps

Lesson 10

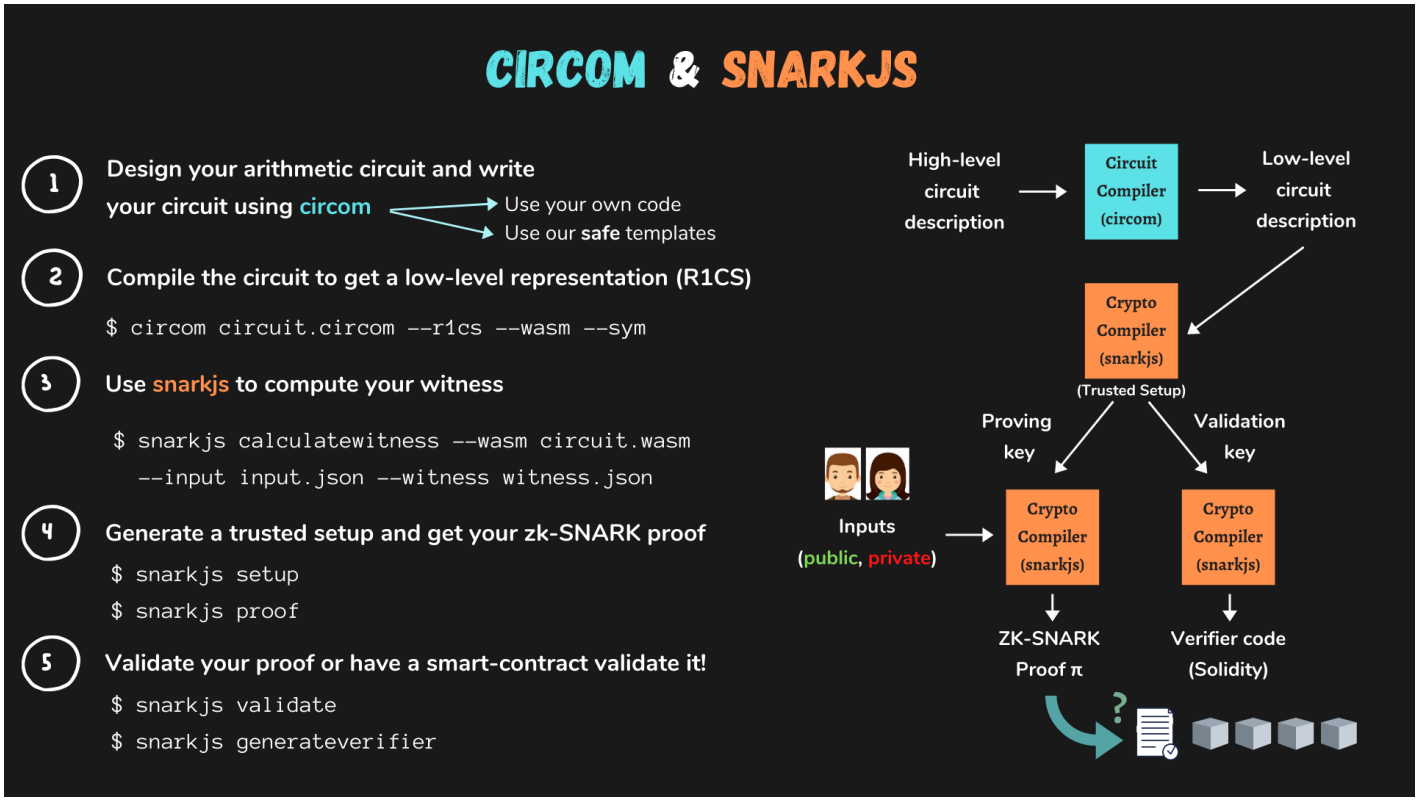
zkEVM solutions



Lesson 11



- Risc zero
- PLONK



- Circom

Lesson 12

- SNARK theory

Lesson 13

- STARK theory

Lesson 14

- Alternative technologies
- Voting systems

Lesson 15

- Identity Solutions
- zkML
- Oracles
- Halo2

Lesson 16

- Extra Topics / Community Information / trends / review

Developer Options

Writing zkp programs

Many languages are rust like

- Cairo
- Noir
- Leo (Aleo)
- Risc Zero

Others :

- Circom
- Snarky.JS (Mina)
- Lurk (Filecoin)
- Solidity (for zkEVM)
- Solidity + WARP -> Cairo

In addition to the DSL -> SNARK / STARK approach, there are other cryptographic techniques that could be useful, see the alternative techniques lesson.

There are many areas to get involved in

- L2 solutions
- Privacy / Identity projects
- ZK-ML
- zk Games
- Security / Auditing

At the protocol level

- STARKS
- SNARKS - PLONK derivatives
- zk VMs
- Recursive proofs

Builder Programs

Encode Hackathons
Mina Navigator Program
Aztec grants

Communities / Groups

Mina UK Meetup group
Starknet Meetup groups

Resources

Zero Knowledge Podcast
Starknet podcast
ZK Hack and ZK Study Club Videos
ZK Whiteboard sessions
Cairo mummies course
Berkley ZKP MOOC
Moonmath Manual
Thaler book
Alex Pinto's Blog
Extropy Foundation collection