Week 6

Lesson 21 - Formal Verification

Lesson 22 - Risc zero

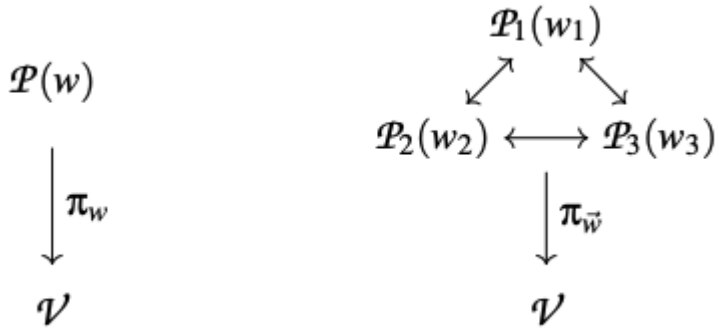
Lesson 23 - Advanced Plonk / Scroll

Lesson 24 - Review

Today's Topics

- Current Research Areas
- Course Review
- Where to go from here

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 $(pp, x, P_1(w_1) \dots P_N(w_N))$ -> 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

Poseidon Merkle Trees in Hardware

See Article

Many ZKPs in production now, however, use alternative hash functions designed for recursive ZKP verification, such as Poseidon, which is ~5x slower than Keccak on modern CPUs.
Arithmetizaion-friendly hash functions require substantial computational resources. For example, FPGA implementations of Keccak can be attained using a few thousand FPGA slices, orders of magnitude smaller than Poseidon.
The design in the article improves on this with a 1.53x speedup compared to a CPU implementation when running the Polygon zkEVM benchmark.

The same team have created the Binius SNARK to take advantage of hardware acceleration.
See article and zkpodcast episode

Common Misconceptions about SNARKS

For details see article

Terminology

- Using “ZK” to mean “succinct”
- Variations of the term “succinct”
- SNARKs vs. STARKs

SNARK design

- Only the Plonk back-end can support “Plonkish circuits,” and only STARK-based back-ends can support AIR
- SNARKs targeting R1CS cannot support lookup arguments
- Plonk is faster for the prover than Groth16
- STARKs rely on fewer or weaker assumptions than ECC alternatives
- Assuming that deployments of the same polynomial commitment scheme or SNARK have similar runtimes
- Characterising FRI and STARK proofs as in the dozens of KBs

Sumcheck protocol

The sumcheck protocol was an early ('90s) idea which is finding new applications.
It is an interactive process between prover and verifier, to allow the prover to show they know the sum of a multivariate polynomial.
It is interesting in the oracle aspects of the protocol, and the fact that the verifier only needs to know the polynomial at the very end.

In addition to the sumcheck proving the sum, we can also use it to check that a vector is all zero elements, so it provides a useful tool

Process

From Paper

The prover wants to prove that she knows H the sum of a multivariate polynomial g

$$H := \sum_{b_1 \in \{0,1\}} \sum_{b_2 \in \{0,1\}} \dots \sum_{b_v \in \{0,1\}} g(b_1, \dots, b_v).$$

We assume here that the verifier has oracle access to g, i.e. can evaluate $g(r1, \dots, rv)$ for a randomly chosen vector $(r1, \dots, rv) \in Fv$ with a single query to an oracle

The process has v rounds, each round is linked to the previous, that is we say that if the 1st round is correct, then we will accept the next one etc.

Description of Sum-Check Protocol.

- Fix an $H \in \mathbb{F}$.
- In the first round, $\mathcal{P}$ sends the univariate polynomial

$$g_1(X_1) := \sum_{(x_2,\ldots,x_v)\in\{0,1\}^{v-1}} g(X_1, x_2, \ldots, x_v).$$

  $\mathcal{V}$ checks that g_1 is a univariate polynomial of degree at most $\deg_1(g)$, and that $H = g_1(0) + g_1(1)$, rejecting if not.
- $\mathcal{V}$ chooses a random element $r_1 \in \mathbb{F}$, and sends r_1 to $\mathcal{P}$.
- In the jth round, for $1 < j < v$, $\mathcal{P}$ sends to $\mathcal{V}$ the univariate polynomial

$$g_j(X_j) = \sum_{(x_{j+1},\ldots,x_v)\in\{0,1\}^{v-j}} g(r_1, \ldots, r_{j-1}, X_j, x_{j+1}, \ldots, x_v).$$

  $\mathcal{V}$ checks that g_j is a univariate polynomial of degree at most $\deg_j(g)$, and that $g_{j-1}(r_{j-1}) = g_j(0) + g_j(1)$, rejecting if not.
- $\mathcal{V}$ chooses a random element $r_j \in \mathbb{F}$, and sends r_j to $\mathcal{P}$.
- In Round v, $\mathcal{P}$ sends the univariate polynomial

$$g_v(X_v) = g(r_1, \ldots, r_{v-1}, X_v)$$

  to $\mathcal{V}$. $\mathcal{V}$ checks that g_v is a univariate polynomial of degree at most $\deg_v(g)$, rejecting if not, and also checks that $g_{v-1}(r_{v-1}) = g_v(0) + g_v(1)$.
- $\mathcal{V}$ chooses a random element $r_v \in \mathbb{F}$ and evaluates $g(r_1, \ldots, r_v)$ with a single oracle query to g. $\mathcal{V}$ checks that $g_v(r_v) = g(r_1, \ldots, r_v)$, rejecting if not.
- If $\mathcal{V}$ has not yet rejected, $\mathcal{V}$ halts and accepts.

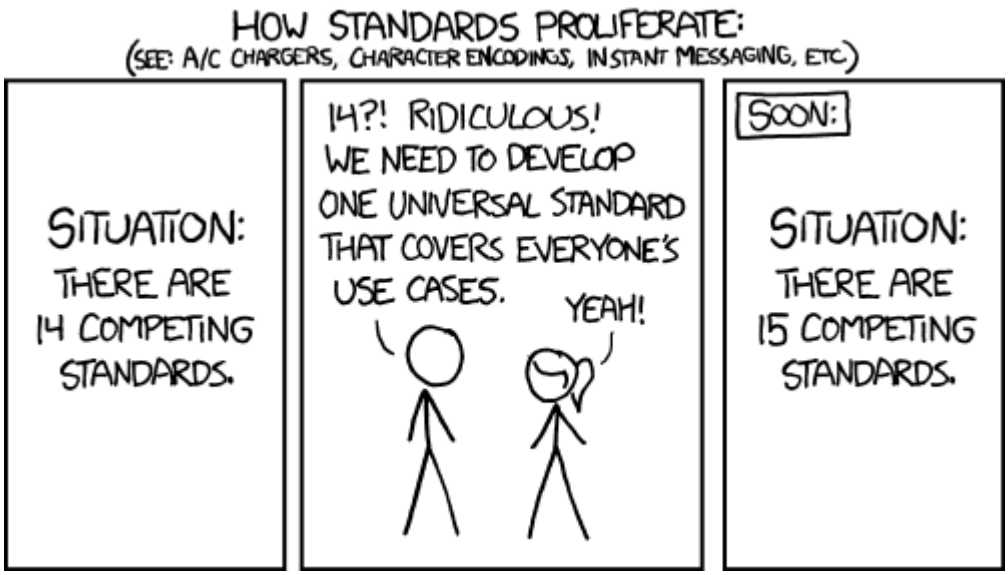
Cost of the protocol

There is one round in the sum-check protocol for each of the v variables of g. The total communication is
$\sum_{i=1}^{v} deg_i(g) + 1$
$= v + \sum_{i=1}^{v} deg_i(g)$ field elements.
In particular, if $deg_i(g) = O(1)$ for all j, then the communication cost is $O(v)$ field elements.

The running time of the verifier over the entire execution of the protocol is proportional to the total communication, plus the cost of a single oracle query to g to compute $g(r1, \ldots, rv)$.

Standards

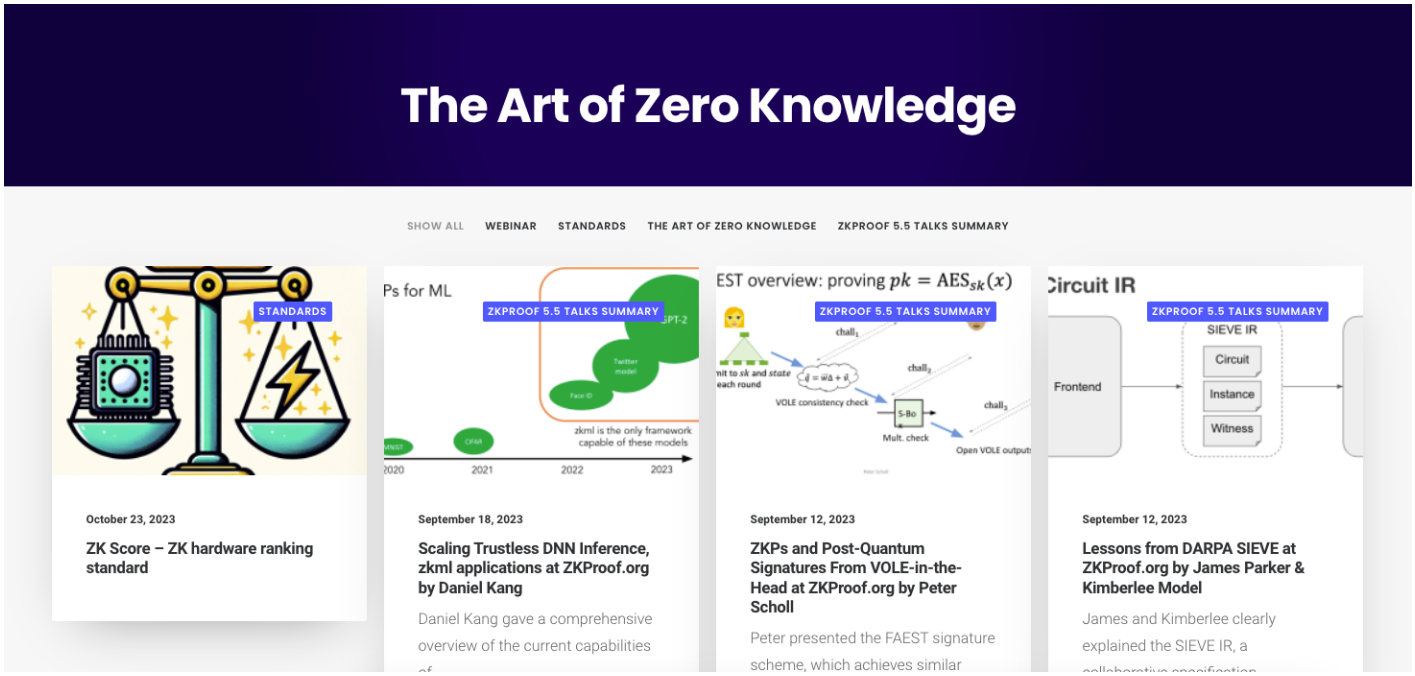
oblig xkcd



ZKproof.org

Community events

Blog



Working Groups

See groups

- Plonkish constraint systems
- Fiat Shamir compiler
- Sigma protocols

Community

See details

zkML updates

Zero Gravity Update

See PSE article

Axiom mainnet alpha

See article

Useful tools from PSE team

See post

Other projects

- Gensyn - decentralised compute
- Ritual Net - Open AI Infastructure

zkEVM Community edition

From the Ethereum PSE team, this has been used as the basis for other projects, such as Scroll.

This series of articles describes the design
articles

L2 Security

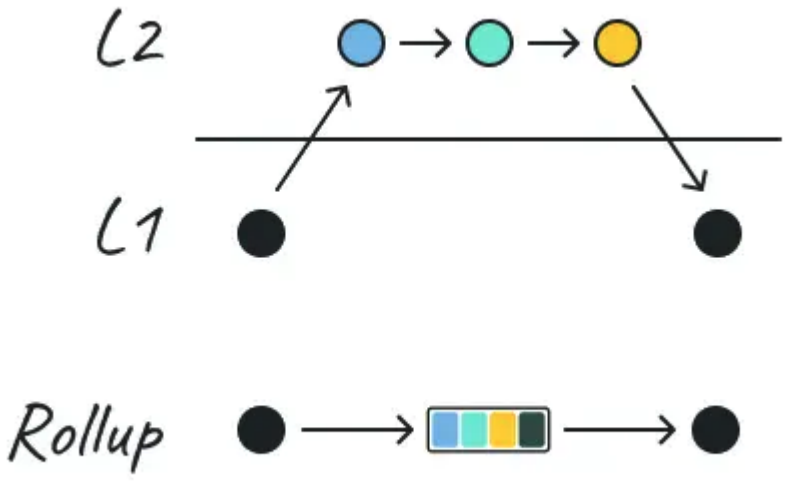
See article by PSE team describing the benefits of crowdsourcing security with audit competitions.
It provides examples of L2s that have been audited.

L2 Communication

In addition to the zkStack and OP Stack .

Magenta project
Involves
Starknet ZK-Rollups]to make use of state compression.
To get decentralized sequencing of Starkware ZK-Rollups for Ethereum on Substrate, the Madara community sequencer is used.

see article



L2 Types

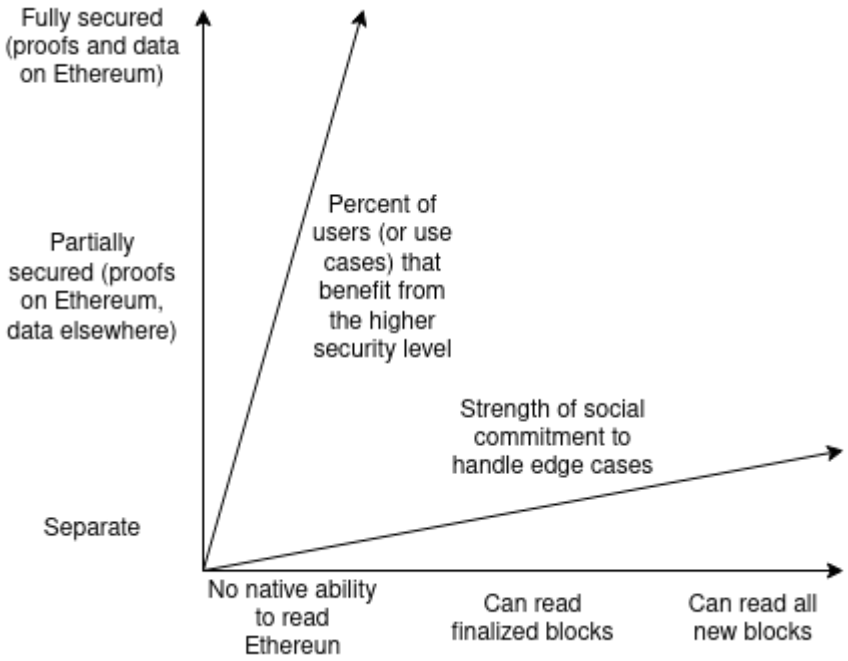
See article

From article

"There are two key dimensions to "connectedness to Ethereum":

1. Security of withdrawing to Ethereum
2. Security of reading Ethereum

These are both important, and have different considerations. There is a spectrum in both cases:"



Parallel EVM

There is much hype around this idea, see article about SEI blockchain.

Course Review

Lesson 1 - Introduction to Blockchain and Layer 1

- Decentralised Systems
- Blockchain timeline
- Layer 1 Theory
- Consensus on Ethereum

Lesson 2 - Why Scalability

- Introduction to scalability
- Possible approaches
- Layer 1 solutions
- Off chain (Layer 2) solutions
- State channels
- Rollups and Plasma chains
- Side chains
- Future directions

Lesson 3 - Introduction to Layer 2

- Off chain scaling introduction
- Bridges versus layer 2
- Rollups in more detail

- Introduction to zkEVM solutions
- Data availability
- (Proto) Danksharding

Lesson 4 - Maths and Cryptography

- Cryptography review
- Mathematical terminology
- Modular arithmetic
- Group and Field theory
- Introduction to polynomials
- Complexity theory
- Introduction to Zero Knowledge Proofs

Lesson 5 -Understanding and Analysing Layer 2

- Categorising L2s
- Analysis criteria
- Rollup stages
- Op Stack and ZK Stack summary

Lesson 6 - Agnostic Layer 2 Transaction Lifecycle

- L2 components and transaction lifecycle

- L2 rollup process
- L2 examples

Lesson 7 - Optimistic Rollups v ZK Rollups (Arbitrum)

- Optimisitic versus ZK Rollups
- Finality options
- SNARK overview
- STARK overview

Lesson 8 - Decentralised Sequencers

- Future of rollups
- Sequencers

Lesson 9 - L3s / Hyperchains

- Fractal scaling
- Data availability review
- L2 / L3 communication
- ZKStack / Hyperchains
- Hyperbridges
- Polygon CDK

Lesson 10 - Privacy in Layer 2 (Aztec)

- Privacy introduction
- Privacy techniques
- Privacy projects

Lesson 11 -What are ZK EVMs part 1

- zkVMs
- zkEVM introduction
- Rollup architecture
- Timeline / phases
- Workflow
- Scroll / Polygon / zkSync / Starknet / Risc Zero

Lesson 12 - What are ZK EVMs part 2

- EVM operations
- zkEVM architecture
- zkEVM design challenges
- Gates / selectors
- Benchmarks

Lesson 13 -What are ZK EVMs part 3

- Custom gates
- Proving systems overview
- Polynomial commitment schemes / KZG
- IOP introduction
- Polygon / Scroll proving systems

Lesson 14 - zkEVM security (Matter Labs)

Lesson 15 - Overview of Proving Systems

- General ZKP Theory
- SNARK, PLONK, STARK Overview
- Bulletproofs Overview
- STARK Overview
- Plonkish protocols
- Folding Schemes

Lesson 16 - SNARK implementation

- SNARK process overview

- Polynomial Checking
- Scroll example
- Fiat Shamir heuristic
- Comparison of schemes
- Verification cost

Lesson 17 -STARK implementation (Starkware)

- Reed Solomon codewords
- Computational integrity
- Stark arithmetisation
- FRI background

Lesson 18 - Plonk part 1 (Linea)

- Arithmetisation review / STARK Arithmetisation continued.
- Trusted Setup
- Circuit / QAP process in PLONK
- Plonkish protocols

Lesson 19 -Plonk part 2 / Boojum (Matter Labs)

- Using the Plonky2 library
- Polynomial permutation

- Hyperplonk
- Plonk implementation in python

Lesson 20 - ZKML

- Machine Learning background
- zkML Introduction
- Approaches to zkML
- EZKL
- Giza
- Hardware

Lesson 21 -Formal Verification

- Formal Verification overview
- Projects
- Examples of vulnerabilities
- Recursion / Folding Schemes

Lesson 22 - Risc Zero

- Impact of Inscriptions on L2 chains
- Risc Zero review
- Powdr and Polygon Miden review
- Walkthrough of example code

Lesson 23 - Scroll

Lesson 24

- Current Research Areas
- Course Review
- Where to go from here

Scroll Additional Resources

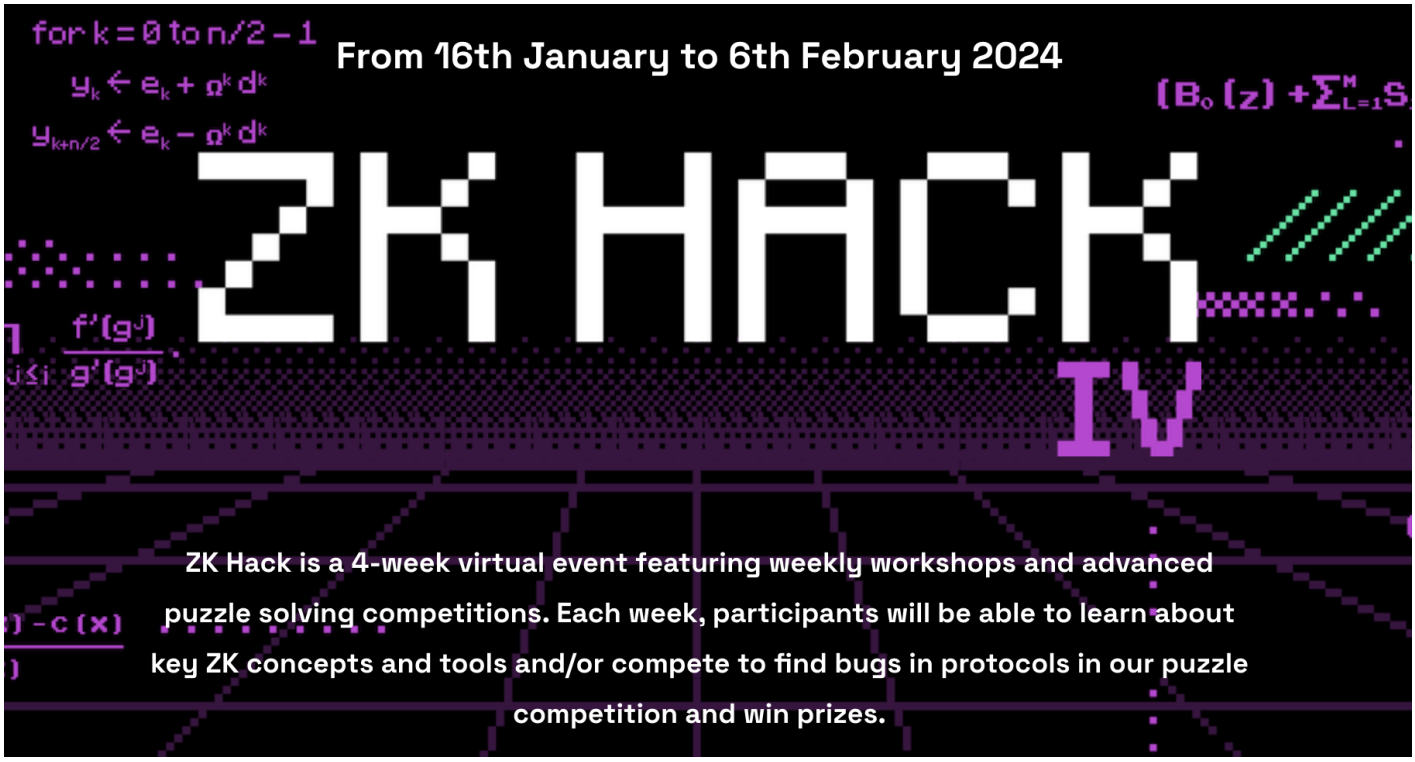
Witness generation with bus mapping crate

EVM circuit

Notes - no longer maintained

Where to go from here

ZKHack



Register

Encode Events and Bootcamps

See Events

Grants programs

Many, for example

Polygon

ZkSync

Mina

Moloch DAO

Extropy Resources

Discord Server

Invite

Medium articles

Writing games with Cairo

Mina and Kimchi

Twitter : @Extropy