Technical Details of the Opcode Compatible zkEVM

Introducing polygon zkEVM Testnet

Jordi Baylina
Co-founder and Technical Lead
Polygon Hermez
Introduction to zkRollup

- User
- L2 TX
- Sequencer
- Sequenced Batches
- Trusted State
- Trusted Implicit State
- Trusted Explicit State
- Prover
- Blockchain
Structure of the Proof

- zkEVM ROM
- zkASSEMBLY
- zkProcessor
- PIL

PROVER
- Multiple R/W
- 1 Access per CLOCK
- Paged for handling Ethereum CALL contexts
- 32 byte alignment sub stat machine.

- The Code that always execute the prover
- It cannot be modified.

- Sparse Merkle Tree
- Goldilocks Poseidon hash function
- Single tree for the system
- Hashes of the smart contract codes are in the tree.
EVM Processor

RAM

ROM

STORAGE

BINARY

Operations done byte to byte with a carry from a plookup table
- ADD
- SUB
- LT & SLT
- EQ
- AND
- OR
- XOR

ARITHMETIC

256 bits arithmetic operations
- $A \times B + C = D \times 2^{256} + E$
- Range check of inputs and outputs
- 32 CLOCKs per operation
- Includes EC addition formulas for ECDSA multiplication.

HASH

Binary circuit of ANDn and XOR
We use a plookup to do various circuits in parallel
We currently can do 468 keccakf's in the current circuit. ($N = 2^{23}$).
ZkASM-ROM

Some examples:

- Opcodes
- RLP Processing

- Ethereum Transaction processor
- FREE Input, the Transactions and the hash must match
- About 16 clocks per gas unit
- zkCounters to prevent the proof to fail (DoS)
opPUSH31:
31 => D
$ => B : MLOAD(isCreateContract)
0 - B : JMPN(opAuxPUSHB)
JMP(opAuxPUSHA)

opPUSH32:
32 => D
$ => B : MLOAD(isCreateContract)
0 - B : JMPN(opAuxPUSHB)
JMP(opAuxPUSHA)

opDUP1:

%MAX_CNT_STEPS - STEP - 120 : JMPN(outOfCounters)

SP - 1 => SP : JMPN(stackUnderflow)
$ => A : MLOAD(SP++)
1024 - SP : JMPN(stackOverflow)
A : MSTORE(SP++)
1024 - SP : JMPN(stackOverflow)
GAS-3 => GAS : JMPN(outOfGas)
JMP(readCode)

opDUP2:

%MAX_CNT_STEPS - STEP - 120 : JMPN(outOfCounters)

SP - 2 => SP : JMPN(stackUnderflow)
$ => A : MLOAD(SP)
Recursion and on-chain verification

Groth16/Plonk Proof that proves the recursive STARK

STARK proof that verifies a STARK proof that verifies N Stark block proofs

STARK proof that verifies N Stark block proofs

Groth16/Plonk VERIFIER SMART CONTRACT

STARK Prover

RECURSIVE STARK PROVER

Groth16/Plonk Proof that proves the recursive STARK

STARK proof that verifies a STARK proof that verifies N Stark block proofs

STARK proof that verifies N Stark block proofs

Ethereum Blockchain

RECURSIVE STARK PROVER

STARK proof of block #10

STARK proof of block #11

STARK proof of block #12

STARK proof of block #13

STARK proof of block #14

STARK proof of block #15
Time for the zkEVM Testnet DEMO

public.zkevm-test.net

1. Transfer goerliETH L1 -> L2
2. Deploy a simple smart contract.
3. Call the smart contract
4. Deploy a Withdraw L2 -> L1
A Scaling Solution fully compatible with the Ethereum Ecosystem

Our community of dApp Developers should not be able to notice any difference between developing on Ethereum L1 and Polygon zkEVM.

Our Commitment is Security and Zero friction for the dApp Developer and for Users to have transparent smart contract execution with off chain validity proofs.

The zkEVM design offers:

- Same tooling
- Same language (Solidity)
- Same gas model

Fast finality (Centralized Sequencer).

Maintaining Security backed by Ethereum.
It's a TESTNET not a showcase
Let's Test it!

Input and constructive feedback Welcomed!

Every problem we manage to find and fix in Testnet will be avoided in Mainnet

What to expect?

- Expect some restarts
- Expect bugs
- Expect that it will not be available for some periods
Prover Costs

- In a single CPU with 192 cores ($9/h in AWS), a more than 4M gas proof takes about 9 minutes to be generated
- Cost per TX is less than $0.007
- Margins we are working to improve:
  - Coding optimisations
  - Mathematical optimisations
  - GPU / FPGA
  - Design improvements
What is missing today

- Support for pre EIP-155 TXs and modern EIP-2718 Transaction types
- Support for SHA256, BLAKE and PAIRINGS In our roadmap for development
PREPARING FOR AUDIT

- Developing an Audit that serves all rollups and leveraging community effort
- Audit structure
  - Smart contracts: rollup and bridge
  - Arithmetization (PIL)
  - ROM
  - Proving System
- STARK
  - Recursive STARK
  - SNARK
- Tooling zkASM and PILCOM
- UI safety and General Infrastructure security
- Optimizing, bug fixing, testing, observing and ready to provide support for the public testnet
- Implementing the key elements and differences to become a type 2 rollup according to Vitalik’s classification
- Aggregating proofs
- Getting us ready for Audit
  a. We already open source all the repos.
  b. Next auditors’ training program.
- EIP-4844 Data availability in Danksharding.

What is the zkEVM team working on?

We are collaborating with EF and other rollups in a community effort for auditing Smart Contracts
Testing Updates

We are using Ethereum test suit to verify the zkEVM

The Goal is to maintain a high level of equivalence

We are passing 97% of the Ethereum test suite
zkEVM Roadmap

Open Source Code

Testnet

We are here

Jul 2022

Audit

Dec 2022

Mainnet

Q1 2023
Open Source
https://github.com/0xPolygonHermez

Core repos:
- zkevm-proverjs
- zkevm-rom
- zkevm-prover
- zkevm-node
- zkevm-contracts
- zkevm-bridge-service
- zkevm-bridge-ui
- zkevm-doc

zkEVM specific tools and libraries
- zkevm-commonjs
- zkasmcom
- zkevm-testvectors
- zkevm-storagerom

generic tools and libraries
- pilcom
- pil-stark
zkEVM is no longer a myth. It’s now here.
Thank you!

Jordi Baylina
Technical Lead and Co-founder
Polygon Hermez

@jbaylina