LEHIGH UNIVERSITY

zkEVM

Tal Derei

Zero-Knowledge Virtual Machine

zkEVMs:

→ Opcode-Based Approach
→ Compiler-Based Approach

Zero-Knowledge Virtual Machine

- Opcode-Based Approach
  - Generally uses the **SAME** virtual machine (i.e. EVM) as L1 under the hood
  - Implements the full set of EVM opcodes DIRECTLY in ZK-Rollup
  - Bottom-Up Construction Approach: each opcode needs to be reconstructed for L2
    - Direct support for the existing set of EVM and solidity opcodes (i.e. rewriting all opcodes for L2)
    - Some opcodes more complex and expensive to compute on L2 (e.g. SHA256 or Keccak Hash Functions)
    - There are different opcodes, and a **state machine / circuits** associated for each opcode (i.e. storage, matrix multiplication, etc.)
    - ~140 opcodes
    - Examples:
      - **Scroll** designs sub circuits to prove the correctness for each opcode
      - **Polygon Hermez** rewrites each opcode using new assembly language and generates a proof for underlying state machine.

Zero-Knowledge Virtual Machine

- Compiler-Based Approach
  - Building a **newly designed VM** to maintain solidity compatibility
  - Not bound by EVM opcodes = more flexible to compile the code into a more zero-knowledge proof-friendly set of opcodes that are different from Ethereum
  - Examples:
    - StarkWare and ZkSync are building newly designed virtual machine and compiler different than EVM, and compile solidity to their VM target

Trade-Offs and Motivation

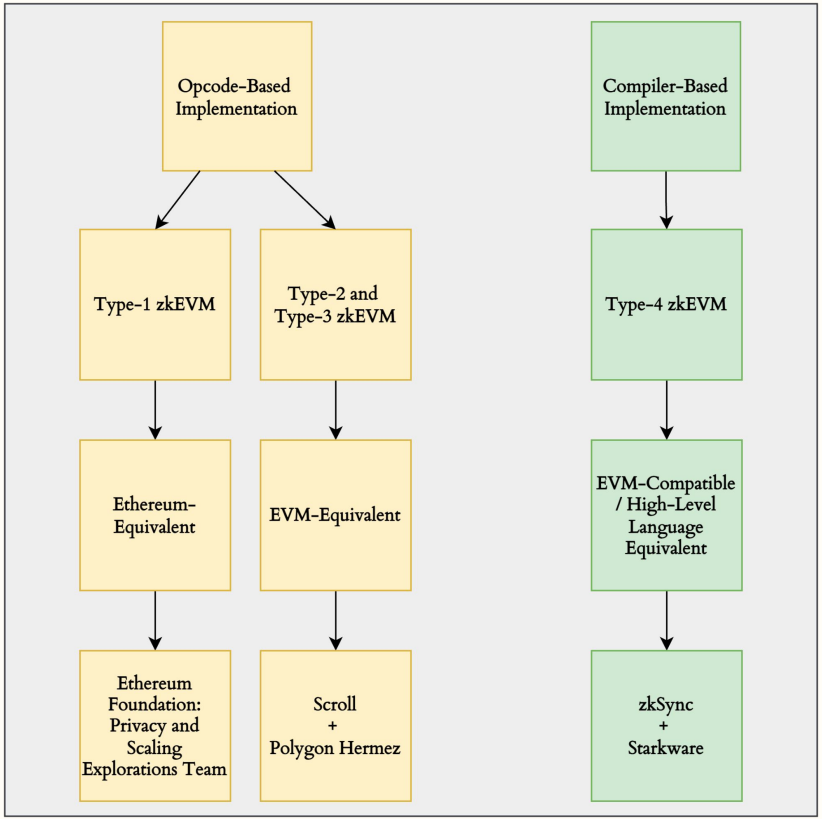
Opcode-Based Approach is **HARDER** to build and bound by Ethereum/EVM limitations!

- Tons of ZK unfriendly opcodes like sha3/kechack hash
- Inefficient data structures like EVM word sizes are 256 bytes and range checks everywhere which explodes your circuit size
- Data storage is also merkle-Patricia tree based, which is another overhead
- Can't define their own instruction set / opcodes

Q. Why was zkEVM thought to be impossible in the past, and now possible?

Recent breakthroughs: new proof systems, advanced circuit optimizations, hardware acceleration, and recursive proofs.

zkEVM Types


Opcode-Based Implementation
Compiler-Based Implementation
Type-1 zkEVM
Type-2 and Type-3 zkEVM
Type-4 zkEVM
Ethereum-Equivalent
EVM-Equivalent
EVM-Compatible / High-Level Language Equivalent
Ethereum Foundation: Privacy and Scaling Explorations Team
Scroll + Polygon Hermez
zkSync + Starkware

zkEVM Types

Can further decompose Opcode and Compiler-based approaches into the following types:

Type 1: Ethreum-Equivalence

- Follows the EVM specification defined in Ethereum yellow paper, e.g. Enshrined rollups

Type 2/3: EVM-Equivalence (i.e. bytecode compatible)

- EVM bytecode-level compatibility (transpile/interpret the **EVM bytecode** into your VM's bytecode)

Type 4: EVM-compatibility (i.e. language compatible)

- Translate high-level **Solidity/Vyper code** into your VM's native bytecode and then prove the validity of the execution trace in a circuit