What is this talk about?

- What is a ZK-EVM?
- Polygon’s zkEVM
- Differences
  - Storage - SMT
  - Memory
  - Zk-counters
  - Selfdestruct
  - Precompileds
- Other differences
Whats is a zk-EVM?
A ZK-EVM

- Definition: a virtual machine that executes smart contracts in a way that is compatible with zero-knowledge-proof computation

- Purpose: Scale Ethereum

- How? use ZK-SNARK technology to make cryptographic proofs of execution of Ethereum-like transactions.
The ZK-EVM “dilemma”

* Can verify an environment that looks exactly like Ethereum, and even the Ethereum chain itself
* Can scale the Ethereum L1, and not just rollups
* Maximaly easy for rollups because you can share infrastructure (incl execution clients)
* Takes a very long time to generate proofs

1. Can verify an environment that looks exactly like Ethereum, but with minor changes (e.g. state tree) that don't touch the application layer
   * Fully compatible with almost all Ethereum apps
   * Can share most infrastructure
   * Takes a long time to generate proofs

2. Modify the EVM only by changing gas costs
   * Makes it faster to generate proofs
   * Introduces a few incompatibilities

2.5

3. Compiles contracts written in Solidity, Vyper or other high-level langs to a specialized VM, and proves that
   * Not compatible with some Ethereum apps
   * Can't share a lot of infrastructure
   * Fastest proof generation time, saves costs and reduces centralization risks

Source: https://vitalik.eth.limo/general/2022/08/04/zkevm.html
polygon zkEVM
Fully EVM-equivalent

New state A = New state B
Polygon zkEVM

- **zkasm**: defines the steps
- **PIL**: verifies the correctness of the steps
zkEVM vs EVM
Storage

EVM
- Key: ethAddr
- Root state trie
  - Ethereum account:
    - Nonce
    - Balance
    - storageRoot
    - codeHash

zkEVM
- Key: ethAddr | balance
- Key: ethAddr | nonce
- Root state trie
  - Ethereum account:
    - Balance
    -Nonce
  - Ethereum account:
    - codeHash
  - Ethereum account:
    - storage value
<table>
<thead>
<tr>
<th>EVM</th>
<th>zkEVM</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x20</td>
<td>0x20</td>
</tr>
<tr>
<td>0x21</td>
<td>0x21</td>
</tr>
<tr>
<td>0x22</td>
<td>0x22</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>0x3f</td>
<td></td>
</tr>
<tr>
<td>0x40</td>
<td></td>
</tr>
<tr>
<td>0x41</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>0x5f</td>
<td></td>
</tr>
<tr>
<td>0x60</td>
<td></td>
</tr>
<tr>
<td>0x61</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

8 bits

256 bits
Example MLOAD

- The easiest case: `MLOAD 32 bytes && offset%32 == 0`
  
  \[
  \text{offset} = 0x20 \\
  \text{memory}[0x20] = [0x20, 0x21, 0x22, 0x23, \ldots, 0x3e, 0x3f] \\
  \]

- `MLOAD 32 bytes && offset%32 != 0`
  
  \[
  \text{offset} = 0x23 \rightarrow \text{memory}[0x23, 0x43] \\
  \text{memory}[0x20] = [0x20, 0x21, 0x22, 0x23, \ldots, 0x3e, 0x3f] \\
  \text{memory}[0x21] = [0x40, 0x41, 0x42, 0x43, \ldots, 0x5e, 0x5f] \\
  \]
Example MLOAD

- **MLOAD X bytes (X < 32 bytes) && offset%32 == 0**
  
  \[
  \text{offset} = 0x20, \quad \text{length} = 4
  \]
  
  \[
  \text{memory}[0x20] = [0x20, 0x21, 0x22, 0x23, \ldots, 0x3e, 0x3f]
  \]

- **MLOAD X bytes && offset%32 != 0 && offset%32+length < 32**
  
  \[
  \text{offset} = 0x23, \quad \text{length} = 4 \rightarrow [0x23, 0x26]
  \]
  
  \[
  \text{memory}[0x20] = [0x20, \ldots, 0x23, 0x24, 0x25, 0x26, \ldots, 0x3f]
  \]

- **MLOAD X bytes && offset%32 != 0 && offset%32+length > 32**
  
  \[
  \text{offset} = 0x3e, \quad \text{length} = 4 \rightarrow [0x3e, 0x41]
  \]
  
  \[
  \text{memory}[0x20] = [0x20, 0x21, 0x22, 0x23, \ldots, 0x3e, 0x3f]
  \]
  
  \[
  \text{memory}[0x21] = [0x40, 0x41, 0x42, 0x43, \ldots, 0x4e, 0x4f]
  \]
zk-counters

- Counters are a way to control that the total number of steps do not exceed the maximum polynomial size.
- If we go out of counters, then, there is an error in the processing of the batch (it is not an error of the user).
zk-counters example

```assembly
opEQ:

%MAX_CNT_BINARY - CNT_BINARY - 1 : JMPN(outOfCountersBinary)
%MAX_CNT_STEPS - STEP - 120 : JMPN(outOfCountersStep)

SP - 2 : JMPN(stackUnderflow)
SP - 1 ⇒ SP
$ ⇒ A : MLOAD(SP--)
$ ⇒ B : MLOAD(SP)
GAS - 3 ⇒ GAS : JMPN(outOfGas)
$ : EQ,MSTORE(SP++)
1024 - SP : JMPN(stackOverflow)
: JMP(readCode)
```

laisolizq, 8 months ago • First opcodes
- **EIP-4758**: Deactivate SELFDESTRUCT

- This EIP renames the SELFDESTRUCT opcode to SENDALL, and replaces its functionality. The new functionality is there to only send all Ether in the account to the caller.
## Precompileds

<table>
<thead>
<tr>
<th>Address</th>
<th>Name</th>
<th>Supported</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x01</td>
<td>ecRecover</td>
<td>Yes</td>
</tr>
<tr>
<td>0x02</td>
<td>SHA-256</td>
<td>Ongoing</td>
</tr>
<tr>
<td>0x03</td>
<td>RIPEMD-160</td>
<td>Ongoing</td>
</tr>
<tr>
<td>0x04</td>
<td>identity</td>
<td>Yes</td>
</tr>
<tr>
<td>0x05</td>
<td>modexp</td>
<td>Yes</td>
</tr>
<tr>
<td>0x06</td>
<td>ecAdd</td>
<td>Ongoing</td>
</tr>
<tr>
<td>0x07</td>
<td>ecMul</td>
<td>Ongoing</td>
</tr>
<tr>
<td>0x08</td>
<td>ecPairing</td>
<td>Ongoing</td>
</tr>
<tr>
<td>0x08</td>
<td>blake2f</td>
<td>Ongoing</td>
</tr>
</tbody>
</table>
Section 4

Other differences
Other differences

- **EXTCODEHASH**: returns hashContract from zkEVM tree, hashed with Poseidon

- **BLOCKHASH**: returns all previous block hashes (not just the last 256 blocks)

- **Memory limits**: 0x20000, which require 8.5 million GAS of memory expansion.

- **Pre EIP-155 and EIP-2718 TXS**: not supported yet, is our current priority
We are passing 97% of the Ethereum Test Suit

Final Remarks

- We are full-EVM equivalent
- Differences are addressed aiming at performance and equivalence
- We are on public Testnet!
  
public.zkevm-test.net
Thank you!

Ignasi Ramos
Protocol Team, Polygon zkEVM
ignasi@polygon.technology

@0xIgnasi
Appendix
Some assets.