zkEVM design, optimization and applications

Guest Lecturer: Ye Zhang

Zero Knowledge Proofs

Instructors: Dan Boneh, Shafi Goldwasser, Dawn Song, Justin Thaler, Yupeng Zhang
What is Scroll?

A scaling solution for Ethereum
What is Scroll?

An EVM-equivalent zk-Rollup
Outline

• Background & motivation
• Build a zkEVM from scratch
• Interesting research problems
• Other applications using zkEVM
The diagram of Layer 1 blockchain
The diagram of Layer 1 blockchain
The diagram of Layer 1 blockchain

Layer 1

Layer 1:

- \( TX_1 \)
- \( TX_2 \)
- \( \ldots \)
- \( TX_n \)
The diagram of Layer 1 blockchain

Layer 1
The diagram of Layer 1 blockchain

Layer 1

TX₁
TX₂
......
TXₙ

EVM
State Machine
Stack
Memory
Storage

root
The diagram of Layer 1 blockchain

Layer 1

- Secure
- Decentralized
- Expensive 😞
- Slow 😞
Zk-Rollup

Layer 1

Layer 2

data $\pi$

prover

$TX_1$

$TX_2$

.....

$TX_n$
However, ...

Layer 1

Layer 2

\[ TX_1 \]
\[ TX_2 \]
\[ \ldots \]
\[ TX_n \]
However, ...

Layer 1

Layer 2

Prover1

Prover2

\( TX_1 \)

\( TX_2 \)

\( TX_n \)

\( \pi \)
Scroll: a native zkEVM solution

Layer 1

Layer 2

data $\pi$

$TX_1$

$TX_2$

......

$TX_n$

zkEVM

- Developer friendly
- Composability
Scroll: a native zkEVM solution

Layer 1

Layer 2

- Developer friendly
- Composability
- Hard to build 😞
- Large proving overhead 😞
Scroll: a native zkEVM solution

Layer 1

Layer 2

$TX_1$

$TX_2$

……

$TX_n$

zkEVM

- Polynomial commitment
- Lookup + Custom gate
- Hardware acceleration
- Recursive proof
zkEVM flavors (by Justin Drake)

- **Language level**
  Transpile an EVM-friendly language (Solidity or Yul) to a SNARK-friendly VM which differs from the EVM. This is the approach of Matter Labs and Starkware.

- **Bytecode level**
  Interpret EVM bytecode directly, though potentially producing different state roots than the EVM, e.g. if certain implementation-level data structures are replaced with SNARK-friendly alternatives. This is the approach taken by Scroll, Hermez, and Consensys.

- **Consensus level**
  Target full equivalence with EVM as used by Ethereum L1 consensus. That is, it proves validity of L1 Ethereum state roots. This is part of the "zk-SNARK everything" roadmap for Ethereum.
zkEVM flavors (by Justin Drake)

- **Language level**
  Transpile an EVM-friendly language (Solidity or Yul) to a SNARK-friendly VM which differs from the EVM. This is the approach of Matter Labs and Starkware.

- **Bytecode level**
  Interpret EVM bytecode directly, though potentially producing different state roots than the EVM, e.g. if certain implementation-level data structures are replaced with SNARK-friendly alternatives. This is the approach taken by Scroll, Hermez, and Consensys.

- **Consensus level**
  Target full equivalence with EVM as used by Ethereum L1 consensus. That is, it proves validity of L1 Ethereum state roots. This is part of the "zk-SNARK everything" roadmap for Ethereum.
Outline

• Background & motivation
• Build a zkEVM from scratch
• Interesting research problems
• Other applications using zkEVM
The workflow of zero-knowledge proof

**Program**

```python
def hcf(x, y):
    if x > y:
        smaller = y
    else:
        smaller = x
    for i in range(1, smaller + 1):
        if((x % i == 0) and (y % i == 0)):
            hcf = i
    return hcf
```

**Constraints**

- \( x \times x = \text{var1} \)
- \( \text{var1} \times x = y \)
- \( (y+x) \times 1 = \text{var2} \)
- \( (\text{var2}+5) \times 1 = \text{out} \)

**Proof**

- R1CS
- Plonkish
- AIR
- Polynomial IOP
- PCS
The workflow of zero-knowledge proof

Program

Constraints

Proof

Ethereum Virtual Machine (EVM)

State Machine

Stack

Memory

Storage

bytecode

R1CS

Plonkish

AIR

x * x == var1
var1 * x == y
(y+x) * 1 == var2
(var2+5) * 1 == out

Polynomial IOP

+ PCS

π
The workflow of zero-knowledge proof

Program

Constraints

Proof

Ethereum Virtual Machine (EVM)

State Machine

Stack

Memory

Storage

bytecode

R1CS

Plonkish

AIR

\begin{align*}
\text{x * x} &= \text{var1} \\
\text{var1 * x} &= \text{y} \\
\text{(y+x) * 1} &= \text{var2} \\
\text{(var2+5) * 1} &= \text{out}
\end{align*}

Plonk IOP

+ KZG
Let’s start with R1CS

<table>
<thead>
<tr>
<th>$w_1$</th>
<th>$w_2$</th>
<th>$w_3$</th>
<th>$w_4$</th>
<th>$w_5$</th>
<th>......</th>
<th>$w_{n-1}$</th>
<th>$w_n$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Let’s start with R1CS

\[
\begin{array}{cccccc}
\text{\(w_1\)} & \text{\(w_2\)} & \text{\(w_3\)} & \text{\(w_4\)} & \text{\(w_5\)} & \cdots & \text{\(w_{n-1}\)} & \text{\(w_n\)} \\
\hline
\end{array}
\]

\[ (a_1w_1 + \cdots + a_nw_n) \times (b_1w_1 + \cdots + b_nw_n) = (c_1w_1 + \cdots + c_nw_n) \]
Let’s start with R1CS

\[(a_1w_1 + \cdots + a_n w_n) \times (b_1w_1 + \cdots + b_n w_n) = (c_1w_1 + \cdots + c_n w_n)\]

\[(2w_1 + 1) \times (3w_1 + 4w_2) = (w_{n-2} + 2)\]

\[(w_3 + 2) \times (w_4) = (w_n + 1)\]

\[\vdots \]

\[\vdots \]
Let’s start with R1CS

I know a vector \{input, va, vb, vc, \ldots\} that satisfies all those constraints
### Plonkish Arithmetization

<table>
<thead>
<tr>
<th>$a_0$</th>
<th>$a_1$</th>
<th>$a_2$</th>
<th>$a_3$</th>
<th>$a_4$</th>
<th>$T_0$</th>
<th>$T_1$</th>
<th>$T_2$</th>
<th>$T_3$</th>
<th>$T_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Table 1

- **witness**

#### Table 2

- **Table 1**
- **Table 2**
### Plonkish Arithmetization

<table>
<thead>
<tr>
<th></th>
<th>(a_0)</th>
<th>(a_1)</th>
<th>(a_2)</th>
<th>(a_3)</th>
<th>(a_4)</th>
<th>(T_0)</th>
<th>(T_1)</th>
<th>(T_2)</th>
<th>(T_3)</th>
<th>(T_4)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>input</strong></td>
<td>(\text{input}_0)</td>
<td>(\text{input}_1)</td>
<td>(\text{input}_2)</td>
<td><strong>output</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(va_1)</td>
<td>(vb_1)</td>
<td>(vc_1)</td>
<td></td>
<td>(vd_1)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(va_2)</td>
<td>(vb_2)</td>
<td>(vc_2)</td>
<td></td>
<td>(vd_2)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(va_3)</td>
<td>(vb_3)</td>
<td>(vc_3)</td>
<td></td>
<td>(vd_3)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(va_4)</td>
<td>(vb_4)</td>
<td>(vc_4)</td>
<td></td>
<td><strong>......</strong></td>
<td>(vd_4)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(va_5)</td>
<td>(vb_5)</td>
<td>(vc_5)</td>
<td></td>
<td></td>
<td>(vd_5)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(va_6)</td>
<td>(vb_6)</td>
<td>(vc_6)</td>
<td></td>
<td></td>
<td>(vd_6)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(va_7)</td>
<td>(vb_7)</td>
<td>(vc_7)</td>
<td></td>
<td></td>
<td>(vd_7)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**witness**

**Table 1**

**Table 2**
Plonkish Arithmetization – Custom gate

\[ va_3 \times vb_3 \times vc_3 - vb_4 = 0 \]

<table>
<thead>
<tr>
<th>( a_0 )</th>
<th>( a_1 )</th>
<th>( a_2 )</th>
<th>( a_3 )</th>
<th>( a_4 )</th>
<th>( T_0 )</th>
<th>( T_1 )</th>
<th>( T_2 )</th>
<th>( T_3 )</th>
<th>( T_4 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>input_0</td>
<td>input_1</td>
<td>input_2</td>
<td>output</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( va_1 )</td>
<td>( vb_1 )</td>
<td>( vc_1 )</td>
<td>( vd_1 )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( va_2 )</td>
<td>( vb_2 )</td>
<td>( vc_2 )</td>
<td>( vd_2 )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( va_3 )</td>
<td>( vb_3 )</td>
<td>( vc_3 )</td>
<td>( vd_3 )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( va_4 )</td>
<td>( vb_4 )</td>
<td>( vc_4 )</td>
<td>( vd_4 )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( va_5 )</td>
<td>( vb_5 )</td>
<td>( vc_5 )</td>
<td>( vd_5 )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( va_6 )</td>
<td>( vb_6 )</td>
<td>( vc_6 )</td>
<td>( vd_6 )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( va_7 )</td>
<td>( vb_7 )</td>
<td>( vc_7 )</td>
<td>( vd_7 )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( va_6 )</td>
<td>( vb_6 )</td>
<td>( vc_6 )</td>
<td>( vd_6 )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( va_7 )</td>
<td>( vb_7 )</td>
<td>( vc_7 )</td>
<td>( vd_7 )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

witness  Table 1  Table 2
Plonkish Arithmetization – Custom gate

<table>
<thead>
<tr>
<th>a₀</th>
<th>a₁</th>
<th>a₂</th>
<th>a₃</th>
<th>a₄</th>
<th>T₀</th>
<th>T₁</th>
<th>T₂</th>
<th>T₃</th>
<th>T₄</th>
</tr>
</thead>
<tbody>
<tr>
<td>input₀</td>
<td>input₁</td>
<td>input₂</td>
<td>output</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>va₁</td>
<td>vb₁</td>
<td>vc₁</td>
<td>vd₁</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>va₂</td>
<td>vb₂</td>
<td>vc₂</td>
<td>vd₂</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>va₃</td>
<td>vb₃</td>
<td>vc₃</td>
<td>vd₃</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>va₄</td>
<td>vb₄</td>
<td>vc₄</td>
<td>vd₄</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>va₅</td>
<td>vb₅</td>
<td>vc₅</td>
<td>vd₅</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>va₆</td>
<td>vb₆</td>
<td>vc₆</td>
<td>vd₆</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>va₇</td>
<td>vb₇</td>
<td>vc₇</td>
<td>vd₇</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>va₆</td>
<td>vb₆</td>
<td>vc₆</td>
<td>vd₆</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>va₇</td>
<td>vb₇</td>
<td>vc₇</td>
<td>vd₇</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[ va₃ \times vb₃ \times vc₃ - vb₄ = 0 \]

- High degree
- More customized

**Table 1**

**Table 2**

**witness**
### Plonkish Arithmetization – Custom gate

<table>
<thead>
<tr>
<th>a₀</th>
<th>a₁</th>
<th>a₂</th>
<th>a₃</th>
<th>a₄</th>
<th>T₀</th>
<th>T₁</th>
<th>T₂</th>
<th>T₃</th>
<th>T₄</th>
</tr>
</thead>
<tbody>
<tr>
<td>input₀</td>
<td>input₁</td>
<td>input₂</td>
<td>output</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>va₁</td>
<td>vb₁</td>
<td>vc₁</td>
<td>vd₁</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>va₂</td>
<td>vb₂</td>
<td>vc₂</td>
<td>vd₂</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>va₃</td>
<td>vb₃</td>
<td>vc₃</td>
<td>vd₃</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>va₄</td>
<td>vb₄</td>
<td>vc₄</td>
<td>vd₄</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>va₅</td>
<td>vb₅</td>
<td>vc₅</td>
<td>vd₅</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>va₆</td>
<td>vb₆</td>
<td>vc₆</td>
<td>vd₆</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>va₇</td>
<td>vb₇</td>
<td>vc₇</td>
<td>vd₇</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>va₈</td>
<td>vb₈</td>
<td>vc₈</td>
<td>vd₈</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[ vb₁ \times vc₁ + vc₂ - vc₃ = 0 \]

- High degree
- More customized
Plonkish Arithmetization – Custom gate

<table>
<thead>
<tr>
<th></th>
<th>A0</th>
<th>A1</th>
<th>A2</th>
<th>A3</th>
<th>A4</th>
<th>T0</th>
<th>T1</th>
<th>T2</th>
<th>T3</th>
<th>T4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Va1</td>
<td>Vb1</td>
<td>Vc1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Va2</td>
<td>Vb2</td>
<td>Vc2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Va3</td>
<td>Vb3</td>
<td>Vc3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Va4</td>
<td>Vb4</td>
<td>Vc4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Va5</td>
<td>Vb5</td>
<td>Vc5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Va6</td>
<td>Vb6</td>
<td>Vc6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Va7</td>
<td>Vb7</td>
<td>Vc7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Va8</td>
<td>Vb8</td>
<td>Vc8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Va9</td>
<td>Vb9</td>
<td>Vc9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Va10</td>
<td>Vb10</td>
<td>Vc10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Va11</td>
<td>Vb11</td>
<td>Vc11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Va12</td>
<td>Vb12</td>
<td>Vc12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Va13</td>
<td>Vb13</td>
<td>Vc13</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Va14</td>
<td>Vb14</td>
<td>Vc14</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Va15</td>
<td>Vb15</td>
<td>Vc15</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[ vc_1 + va_2 \cdot vb_4 - vc_4 = 0 \]

- High degree
- More customized
# Plonkish Arithmetization – Permutation

<table>
<thead>
<tr>
<th>a₀</th>
<th>a₁</th>
<th>a₂</th>
<th>a₃</th>
<th>a₄</th>
<th>T₀</th>
<th>T₁</th>
<th>T₂</th>
<th>T₃</th>
<th>T₄</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>input</strong>₀</td>
<td><strong>input</strong>₁</td>
<td><strong>input</strong>₂</td>
<td><strong>output</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>va₁</th>
<th>vb₁</th>
<th>vc₁</th>
<th>vd₁</th>
</tr>
</thead>
<tbody>
<tr>
<td>va₂</td>
<td>vb₂</td>
<td>vc₂</td>
<td>vd₂</td>
</tr>
<tr>
<td>va₃</td>
<td>vb₃</td>
<td>vc₃</td>
<td>vd₃</td>
</tr>
<tr>
<td>va₄</td>
<td>vb₄</td>
<td>vc₄</td>
<td>vd₄</td>
</tr>
<tr>
<td>va₅</td>
<td>vb₅</td>
<td>vc₅</td>
<td>vd₅</td>
</tr>
<tr>
<td>va₆</td>
<td>vb₆</td>
<td>vc₆</td>
<td>vd₆</td>
</tr>
<tr>
<td>va₇</td>
<td>vb₇</td>
<td>vc₇</td>
<td>vd₇</td>
</tr>
</tbody>
</table>

\[
 vb₄ = vc₆ = vb₆ = va₆
\]

![Witness Diagram](image)
### Plonkish Arithmetization – Lookup argument

<table>
<thead>
<tr>
<th>a₀</th>
<th>a₁</th>
<th>a₂</th>
<th>a₃</th>
<th>a₄</th>
<th>T₀</th>
<th>T₁</th>
<th>T₂</th>
<th>T₃</th>
<th>T₄</th>
</tr>
</thead>
<tbody>
<tr>
<td>input₀</td>
<td>input₁</td>
<td>input₂</td>
<td>output</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>va₁</td>
<td>vb₁</td>
<td>vc₁</td>
<td>vd₁</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>va₂</td>
<td>vb₂</td>
<td>vc₂</td>
<td>vd₂</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>va₃</td>
<td>vb₃</td>
<td>vc₃</td>
<td>vd₃</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>va₄</td>
<td>vb₄</td>
<td>vc₄</td>
<td>vd₄</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>va₅</td>
<td>vb₅</td>
<td>vc₅</td>
<td>vd₅</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>va₆</td>
<td>vb₆</td>
<td>vc₆</td>
<td>vd₆</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>va₇</td>
<td>vb₇</td>
<td>vc₇</td>
<td>vd₇</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>va₆</td>
<td>vb₆</td>
<td>vc₆</td>
<td>vd₆</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>va₇</td>
<td>vb₇</td>
<td>vc₇</td>
<td>vd₇</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Table 1

#### Table 2

\[(va₇, vb₇, vc₇) ∈ (T₀, T₁, T₂)\]
Plonkish Arithmetization – Lookup argument

<table>
<thead>
<tr>
<th>a₀</th>
<th>a₁</th>
<th>a₂</th>
<th>a₃</th>
<th>a₄</th>
<th>T₀</th>
<th>T₁</th>
<th>T₂</th>
<th>T₃</th>
<th>T₄</th>
</tr>
</thead>
<tbody>
<tr>
<td>input₀</td>
<td>input₁</td>
<td>input₂</td>
<td>output</td>
<td>0000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>va₁</td>
<td>vb₁</td>
<td>vc₁</td>
<td>vd₁</td>
<td>0001</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>va₂</td>
<td>vb₂</td>
<td>vc₂</td>
<td>vd₂</td>
<td>0010</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>va₃</td>
<td>vb₃</td>
<td>vc₃</td>
<td>vd₃</td>
<td>0011</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>va₄</td>
<td>vb₄</td>
<td>vc₄</td>
<td>vd₄</td>
<td>0100</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>va₅</td>
<td>vb₅</td>
<td>vc₅</td>
<td>vd₅</td>
<td>0101</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>va₆</td>
<td>vb₆</td>
<td>vc₆</td>
<td>vd₆</td>
<td>......</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>va₇</td>
<td>vb₇</td>
<td>vc₇</td>
<td>vd₇</td>
<td>1101</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>va₆</td>
<td>vb₆</td>
<td>vc₆</td>
<td>vd₆</td>
<td>1110</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>va₇</td>
<td>vb₇</td>
<td>vc₇</td>
<td>vd₇</td>
<td>1111</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

vc₇ ∈ [0, 15]

witness Table 1 Table 2
### Table 1

<table>
<thead>
<tr>
<th>a_0</th>
<th>a_1</th>
<th>a_2</th>
<th>a_3</th>
<th>a_4</th>
<th>T_0</th>
<th>T_1</th>
<th>T_2</th>
<th>T_3</th>
<th>T_4</th>
</tr>
</thead>
<tbody>
<tr>
<td>input_0</td>
<td>input_1</td>
<td>input_2</td>
<td>output</td>
<td></td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>va_1</td>
<td>vb_1</td>
<td>vc_1</td>
<td>vd_1</td>
<td></td>
<td>0000</td>
<td>0001</td>
<td>0001</td>
<td></td>
<td></td>
</tr>
<tr>
<td>va_2</td>
<td>vb_2</td>
<td>vc_2</td>
<td>vd_2</td>
<td></td>
<td>0000</td>
<td>0010</td>
<td>0010</td>
<td></td>
<td></td>
</tr>
<tr>
<td>va_3</td>
<td>vb_3</td>
<td>vc_3</td>
<td>vd_3</td>
<td></td>
<td>0000</td>
<td>0011</td>
<td>0011</td>
<td></td>
<td></td>
</tr>
<tr>
<td>va_4</td>
<td>vb_4</td>
<td>vc_4</td>
<td>vd_4</td>
<td></td>
<td>0000</td>
<td>0100</td>
<td>0100</td>
<td></td>
<td></td>
</tr>
<tr>
<td>va_5</td>
<td>vb_5</td>
<td>vc_5</td>
<td>vd_5</td>
<td></td>
<td>0000</td>
<td>0101</td>
<td>0101</td>
<td></td>
<td></td>
</tr>
<tr>
<td>va_6</td>
<td>vb_6</td>
<td>vc_6</td>
<td>vd_6</td>
<td></td>
<td>......</td>
<td>......</td>
<td>......</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Lookup**

| va_7 | vb_7 | vc_7 | vd_7 | | 1111 | 1101 | 0010 |     |     |
| va_6 | vb_6 | vc_6 | vd_6 | | 1111 | 1110 | 0001 |     |     |
| va_7 | vb_7 | vc_7 | vd_7 | | 1111 | 1111 | 0000 |     |     |

**Table 2**

\[ vc_7 \in [0, 15] \]

\[ va_7 \oplus vb_7 = vc_7 \]
### Plonkish Arithmetization – Lookup argument

<table>
<thead>
<tr>
<th>a₀</th>
<th>a₁</th>
<th>a₂</th>
<th>a₃</th>
<th>a₄</th>
<th>T₀</th>
<th>T₁</th>
<th>T₂</th>
<th>T₃</th>
<th>T₄</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>input₀</strong></td>
<td><strong>input₁</strong></td>
<td><strong>input₂</strong></td>
<td>output</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>va₁</strong></td>
<td><strong>vb₁</strong></td>
<td><strong>vc₁</strong></td>
<td><strong>vd₁</strong></td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>va₂</strong></td>
<td><strong>vb₂</strong></td>
<td><strong>vc₂</strong></td>
<td><strong>vd₂</strong></td>
<td>0000</td>
<td>0010</td>
<td>0010</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>va₃</strong></td>
<td><strong>vb₃</strong></td>
<td><strong>vc₃</strong></td>
<td><strong>vd₃</strong></td>
<td>0000</td>
<td>0011</td>
<td>0011</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>va₄</strong></td>
<td><strong>vb₄</strong></td>
<td><strong>vc₄</strong></td>
<td><strong>vd₄</strong></td>
<td>0000</td>
<td>0100</td>
<td>0100</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>va₅</strong></td>
<td><strong>vb₅</strong></td>
<td><strong>vc₅</strong></td>
<td><strong>vd₅</strong></td>
<td>0000</td>
<td>0101</td>
<td>0101</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>va₆</strong></td>
<td><strong>vb₆</strong></td>
<td><strong>vc₆</strong></td>
<td><strong>vd₆</strong></td>
<td>......</td>
<td>......</td>
<td>......</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>va₇</strong></td>
<td><strong>vb₇</strong></td>
<td><strong>vc₇</strong></td>
<td></td>
<td><strong>vc₇</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>va₆</strong></td>
<td><strong>vb₆</strong></td>
<td><strong>vc₆</strong></td>
<td><strong>vd₆</strong></td>
<td>1111</td>
<td>1101</td>
<td>0010</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>va₇</strong></td>
<td><strong>vb₇</strong></td>
<td><strong>vc₇</strong></td>
<td><strong>vd₇</strong></td>
<td>1111</td>
<td>1110</td>
<td>0001</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[ vc₇ \in [0, 15] \]

\[ va₇ \oplus vb₇ = vc₇ \]

**RAM operation**

- **witness**
- **Table 1**
- **Table 2**
## Plonkish Arithmetization – Constraints

<table>
<thead>
<tr>
<th>$\text{input}_0$</th>
<th>$\text{input}_1$</th>
<th>$\text{input}_2$</th>
<th>$\text{output}$</th>
<th>$T_0$</th>
<th>$T_1$</th>
<th>$T_2$</th>
<th>$T_3$</th>
<th>$T_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$va_1$</td>
<td>$vb_1$</td>
<td>$vc_1$</td>
<td>$vd_1$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$va_2$</td>
<td>$vb_2$</td>
<td>$vc_2$</td>
<td>$vd_2$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$va_3$</td>
<td>$vb_3$</td>
<td>$vc_3$</td>
<td>$vd_3$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$va_4$</td>
<td>$vb_4$</td>
<td>$vc_4$</td>
<td>$vd_4$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$va_5$</td>
<td>$vb_5$</td>
<td>$vc_5$</td>
<td>$vd_5$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$va_6$</td>
<td>$vb_6$</td>
<td>$vc_6$</td>
<td>$vd_6$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$va_7$</td>
<td>$vb_7$</td>
<td>$vc_7$</td>
<td>$vd_7$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$va_6$</td>
<td>$vb_6$</td>
<td>$vc_6$</td>
<td>$vd_6$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$va_7$</td>
<td>$vb_7$</td>
<td>$vc_7$</td>
<td>$vd_7$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 1

Table 2

\[
\begin{align*}
vb_1 \times vc_1 + vc_2 - vc_3 &= 0 \\
va_3 \times vb_3 \times vc_3 - vb_4 &= 0 \\
vb_4 + vc_6 \times vb_6 - va_6 &= 0 \\
\cdots \\
vb_4 &= vc_6 = vb_6 = va_6 \\
\cdots \\
(va_7, vb_7, vc_7) &\in (T_0, T_1, T_2)
\end{align*}
\]
How should we choose “front-end”?

Computation

Ethereum Virtual Machine (EVM)

State root

TX₁

TX₂

......

TXₙ

State Machine

Stack

Memory

Storage

bytecode

State root

TX₁

TX₂

......

TXₙ
EVM word size is 256bit
- Efficient range proof
- EVM has zk-unfriendly opcodes
  - Efficient way to connect circuits
- Read & Write consistency
  - Efficient mapping
- EVM has a dynamic execution trace
  - Efficient on/off selectors
How should we choose “front-end”?

Computation

- EVM word size is 256bit
  - Efficient range proof
- EVM has zk-unfriendly opcodes
  - Efficient way to connect circuits
- Read & Write consistency
  - Efficient mapping
- EVM has a dynamic execution trace
  - Efficient on/off selectors

Ethereum Virtual Machine (EVM)

State Machine
Stack
Memory
Storage
bytecode

State root
TX₁
TX₂
······
TXₙ
• EVM word size is 256bit
  • Efficient range proof
• EVM has zk-unfriendly opcodes
  • Efficient way to connect circuits
• Read & Write consistency
  • Efficient mapping
• EVM has a dynamic execution trace
  • Efficient on/off selectors
What you need to prove

World State (t)

Transaction:

```javascript
{
  from: "0xEA674fdDe714fd979de3EdF0F56AA97168898eC8",
  to: "0xac03bb73b6a9e108530aff4df0772bd3481e5a",
  gasLimit: "21000",
  maxFeePerGas: "300",
  maxPriorityFeePerGas: "10",
  nonce: "0",
  value: "10000000000"
}
```
What you need to prove

World State (t)

Transaction:

```
from: "0xEA674fdDe714fd979de3EdF0F56AA971688898c8",
to: "0xac03bb73b6a9e108530aff4df5077c2b3d48165a",
gasLimit: "21000",
maxFeePerGas: "300",
maxPriorityFeePerGas: "0",
nonce: "0",
value: "10000000000"
```
What you need to prove

Transaction:
{
    from: "0xEA674fdDe714fd97db3EdF0F56AA9716B898ec8",
    to: "0xac03bb73b6a9e108530aff4df0077c2b34841e5a",
    gasLimit: "21000",
    maxFeePerGas: "300",
    maxPriorityFeePerGas: "0",
    nonce: "0",
    value: "10000000000"
}

Computation

Execution trace

<table>
<thead>
<tr>
<th>Step</th>
<th>Opcode</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>PUSH1 80</td>
</tr>
<tr>
<td>1</td>
<td>PUSH1 40</td>
</tr>
<tr>
<td>2</td>
<td>MSTORE</td>
</tr>
<tr>
<td>3</td>
<td>CALLVALUE</td>
</tr>
</tbody>
</table>

n RETURN

World State (t)

World State (t+1)
What you need to prove

World State (t)

Computation

Execution trace

<table>
<thead>
<tr>
<th>Step</th>
<th>Opcode</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>PUSH1 80</td>
</tr>
<tr>
<td>1</td>
<td>PUSH1 40</td>
</tr>
<tr>
<td>2</td>
<td>MSTORE</td>
</tr>
<tr>
<td>3</td>
<td>CALLVALUE</td>
</tr>
</tbody>
</table>

Transaction:
```json
from: "0xEA674fdDe714fd97bd3EdF056AA97168898e88", to: "0xac03bb73b6a9e108530aff4df5077c2b3d48e1e5a", gasLimit: "21000", maxFeePerGas: "300", maxPriorityFeePerGas: "10", nonce: "0", value: "10000000000"
```

World State (t+1)
What you need to prove

Transaction:
```
{  
  from: "0xEA674fdDe714fd979de3EdF0F56AA9716B898ec8",  
  to: "0xac03bb73b6a9e108530aff4df077c2b3d481e5a",  
  gasLimit: "21000",  
  maxFeePerGas: "300",  
  maxPriorityFeePerGas: "10",  
  nonce: "0",  
  value: "10000000000"  
}
```

zkEVM

World State (t)

Computation

Execution trace

<table>
<thead>
<tr>
<th>Step</th>
<th>Opcode</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>PUSH1 80</td>
</tr>
<tr>
<td>1</td>
<td>PUSH1 40</td>
</tr>
<tr>
<td>2</td>
<td>MSTORE</td>
</tr>
<tr>
<td>3</td>
<td>CALLVALUE</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
<tr>
<td>n</td>
<td>RETURN</td>
</tr>
</tbody>
</table>

World State (t+1)
World State (t)

Computation

Transaction:

```json
{
  from: "0xEA674fdDe714fd979de3EdF0F56AA9716B898ec8",
  to: "0xac03bb73b6a9e108530aff4df5077c2b3d481ea5a",
  gasLimit: "21000",
  gasPrice: "300",
  maxFeePerGas: "10",
  nonce: "0",
  gasUsed: "0",
  input: "0x0000000000000000000000000000000000000000000000000000000000000000",
  value: "10000000000"
}
```

zkEVM

Execution trace

<table>
<thead>
<tr>
<th>Step</th>
<th>Opcode</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>PUSH1 80</td>
</tr>
<tr>
<td>1</td>
<td>PUSH1 40</td>
</tr>
<tr>
<td>2</td>
<td>MSTORE</td>
</tr>
<tr>
<td>3</td>
<td>CALLVALUE</td>
</tr>
</tbody>
</table>

World State (t+1)

What you need to prove
EVM circuit

Execution trace

<table>
<thead>
<tr>
<th>step</th>
<th>opcode</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>PUSH1 80</td>
</tr>
<tr>
<td>2</td>
<td>PUSH1 40</td>
</tr>
<tr>
<td>3</td>
<td>ADD</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
<tr>
<td>n</td>
<td>RETURN</td>
</tr>
</tbody>
</table>

Step 1

Step 2

Step 3

Step n...
Step 1

Step 2

Step n

Execution trace

<table>
<thead>
<tr>
<th>step</th>
<th>opcode</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>PUSH1 80</td>
</tr>
<tr>
<td>2</td>
<td>PUSH1 40</td>
</tr>
<tr>
<td>3</td>
<td>ADD</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>n</td>
<td>RETURN</td>
</tr>
</tbody>
</table>
Execution trace

<table>
<thead>
<tr>
<th>step</th>
<th>opcode</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>PUSH1 80</td>
</tr>
<tr>
<td>2</td>
<td>PUSH1 40</td>
</tr>
<tr>
<td>3</td>
<td>ADD</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>n</td>
<td>RETURN</td>
</tr>
</tbody>
</table>

Step 1
- Codehash
- gas
- PC
- SP
- root

Step 2
- case switch

Step n
- opcode specific witness

Step context
- Codehash
- Gas left
- Program counter, Stack pointer
EVM circuit

Execution trace

<table>
<thead>
<tr>
<th>step</th>
<th>opcode</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>PUSH1 80</td>
</tr>
<tr>
<td>2</td>
<td>PUSH1 40</td>
</tr>
<tr>
<td>3</td>
<td>ADD</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>n</td>
<td>RETURN</td>
</tr>
</tbody>
</table>

Step 1
- Codehash
- gas
- PC
- SP
- root

Step 2
- sADD
- sSUB
- sMUL
- sErr1
- sErr2
- sErr3

opcode specific witness

Step n

- Step context
  - Codehash
  - Gas left
  - Program counter, Stack pointer
- Case switch
  - Select opcodes & error cases
  - Exactly one is switched on
### Execution trace

<table>
<thead>
<tr>
<th>step</th>
<th>opcode</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>PUSH1 80</td>
</tr>
<tr>
<td>2</td>
<td>PUSH1 40</td>
</tr>
<tr>
<td>3</td>
<td>ADD</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>n</td>
<td>RETURN</td>
</tr>
</tbody>
</table>

#### Step 1

- **Codehash**
- **gas**
- **PC**
- **SP**
- **root**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>sADD</td>
<td>......</td>
</tr>
<tr>
<td>sSUB</td>
<td>......</td>
</tr>
<tr>
<td>sMUL</td>
<td>......</td>
</tr>
<tr>
<td>sErr1</td>
<td>......</td>
</tr>
<tr>
<td>sErr2</td>
<td>......</td>
</tr>
<tr>
<td>sErr3</td>
<td>......</td>
</tr>
<tr>
<td>a_lo</td>
<td>......</td>
</tr>
<tr>
<td>a_hi</td>
<td>......</td>
</tr>
<tr>
<td>b_lo</td>
<td>......</td>
</tr>
<tr>
<td>b_hi</td>
<td>......</td>
</tr>
<tr>
<td>c_lo</td>
<td>......</td>
</tr>
<tr>
<td>c_hi</td>
<td>......</td>
</tr>
</tbody>
</table>

#### Step 2

- **Case switch**
  - Select opcodes & error cases
  - Exactly one is switched on

#### Step n

- **Opcode specific witness**
  - Extra witness used for opcodes
  - i.e. operands, carry, limbs, ...
**Step context**

- $sADD \cdot (pc' - pc - 1) = 0$
- $sADD \cdot (sp' - sp - 1) = 0$
- $sADD \cdot (gas' - gas - 3) = 0$

**Case switch**

- $sADD \cdot (1 - sADD) = 0$
- $sMUL \cdot (1 - sMUL) = 0$
- ... 
- $sADD + sMUL + ... + sERR_k = 1$

**Opcode specific witness**

- $sADD \cdot (a_{lo} + b_{lo} - c_{lo} - carry0 \cdot 2^{128}) = 0$
- $sADD \cdot (a_{hi} + b_{hi} + carry0 - c_{hi} - carry1 \cdot 2^{128}) = 0$
EVM circuit - ADD

Step 1

Step 2

- Codehash
- gas
- PC
- SP
- root

- sADD
- sSUB
- sMUL
- sErr1
- sErr2
- sErr3

- a_lo
- a_hi

- b_lo
- b_hi

- c_lo
- c_hi

... 

Step n

**Opcode specific witness**

<table>
<thead>
<tr>
<th>idx</th>
<th>tag</th>
<th>addr</th>
<th>R/W</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>STACK</td>
<td>1023</td>
<td>1</td>
<td>...</td>
</tr>
<tr>
<td>5</td>
<td>STACK</td>
<td>1022</td>
<td>0</td>
<td>word_a</td>
</tr>
<tr>
<td>6</td>
<td>STACK</td>
<td>1022</td>
<td>0</td>
<td>word_b</td>
</tr>
<tr>
<td>7</td>
<td>STACK</td>
<td>1023</td>
<td>1</td>
<td>word_c</td>
</tr>
</tbody>
</table>

... 

... 

... 

...
EVM circuit - ADD

- **Opcode specific witness**

<table>
<thead>
<tr>
<th>idx</th>
<th>tag</th>
<th>addr</th>
<th>R/W</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>STACK</td>
<td>1023</td>
<td>1</td>
<td>...</td>
</tr>
<tr>
<td>5</td>
<td>STACK</td>
<td>1022</td>
<td>0</td>
<td>word_a</td>
</tr>
<tr>
<td>6</td>
<td>STACK</td>
<td>1023</td>
<td>0</td>
<td>word_b</td>
</tr>
<tr>
<td>7</td>
<td>STACK</td>
<td>1023</td>
<td>1</td>
<td>word_c</td>
</tr>
</tbody>
</table>

... STACK ...

... MEMORY 0x40 ...

... STORAGE ...

Step 1

Step 2

Codehash | gas | PC | SP | root |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>sADD</td>
<td>sSUB</td>
<td>sMUL</td>
<td>......</td>
<td>......</td>
</tr>
<tr>
<td>sErr1</td>
<td>sErr2</td>
<td>sErr3</td>
<td>......</td>
<td>......</td>
</tr>
<tr>
<td>a_lo</td>
<td>a_hi</td>
<td>b_lo</td>
<td>b_hi</td>
<td>c_lo</td>
</tr>
<tr>
<td>......</td>
<td>......</td>
<td>......</td>
<td>......</td>
<td>......</td>
</tr>
</tbody>
</table>

 Ramirez circuit

...
EVM circuit - Hash

- Opcode specific witness

Hash lookup table

- Step 1
  - Step 2
  - step context
  - case switch
  - input \( \text{Hash}(\text{input}) \)
  - ...
EVM circuit - Hash

- Opcode specific witness

Step 1

Step 2

step context

case switch

input Hash(input)

... 

Step n

Hash lookup table

Hash circuit
The architecture of zkEVM circuits

EVM circuit

- circuit
- lookup table

| → | constrain |
| → | lookup |
The architecture of zkEVM circuits
The architecture of zkEVM circuits
The workflow of zero-knowledge proof

**Program**
- Ethereum Virtual Machine (EVM)
- State Machine
- Stack
- Memory
- Storage
- Bytecode

**Constraints**
- Step 1
  - Step context
    - sADD *(pc' - pc - 1) == 0
    - sADD *(sp' - sp - 1) == 0
    - sADD *(gas' - gas - 3) == 0
  - Case switch
    - sADD *(1 - sADD) == 0
    - sREJ *(1 - sREJ) == 0
    - sADD + sREJ + ... + sERRk == 1
  - Opcode specific witness
    - sADD*(a_lo+b_lo-c_lo-c_lo - carry)*2^128% == 0
    - sADD*(a_hi+b_hi+c_hi-c_hi - carry)*2^128% == 0

**Proof**
- R1CS
- Plonkish
- AIR
- Plonk IOP
- KZG
The proof system for zkEVM

zkEVM

- EVM Circuit
- RAM Circuit
- Storage Circuit
- Other Circuits
The proof system for zkEVM

zkEVM

- EVM Circuit
- RAM Circuit
- Storage Circuit
- Other Circuits
- Aggregation Circuit
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 second layer needs to be verifier efficient (in EVM)
  - Proof is efficiently verifiable on EVM (small proof, low gas cost)
  - Prove the verification circuit of the former layer efficiently
  - Ideally, hardware friendly prover
  - Ideally, transparent or universal trusted setup

- Some promising candidates
  - Groth16
  - Plonk with very few columns
    - KZG/Fflonk/Keccak FRI (larger code rate)
zkEVM

Two-layer architecture

- **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
• The first layer needs to be “expressive”
  • EVM circuit has 116 columns, 2496 custom gates, 50 lookups
  • Highest custom gate degree: 9
  • For 1M gas, EVM circuit needs $2^{18}$ rows (more gas, more rows)

• The second layer needs to aggregate proofs into one proof
  • Aggregation circuit has 23 columns, 1 custom gate, 7 lookups
  • Highest custom gate degree: 5
  • For aggregating EVM, RAM, Storage circuits, it needs $2^{25}$ rows
zkEVM

- Our GPU prover optimization
  - MSM, NTT and quotient kernel
  - Pipeline and overlap CPU and GPU computation
  - Multi-card implementation, memory optimization

- The Performance
  - For EVM circuit
    - CPU prover takes 270.5s, GPU prover takes 30s (9x speedup!)
  - For Aggregation circuit
    - CPU prover takes 2265s, GPU prover takes 149s (15x speedup!)
  - For 1M gas, first layer takes 2 minutes, second layer takes 3 minutes
Outline

- Background & motivation
- Build a zkEVM from scratch
- Interesting research problems
- Other applications using zkEVM
Step 1

Step 2

step context

case switch

opcode specific witness

...

Step n
• Break down 256-bit word into 32 8-bit limbs.

\[ A = a_0 + a_1 \times 256 + a_2 \times 256^2 + \cdots + a_{31} \times 256^{31} \]
• Break down 256-bit word into 32 8-bit limbs.

\[ A = a_0 + a_1 \times 256 + a_2 \times 256^2 + \cdots + a_{31} \times 256^{31} \]

• Encode EVM word using RLC (Random Linear Combination)

\[ A_{RLC} \equiv a_0 + a_1 \times \theta + a_2 \times \theta^2 + \cdots + a_{31} \times \theta^{31} \pmod{F_p} \]
• Break down 256-bit word into 32 8-bit limbs.

\[ A = a_0 + a_1 \times 256 + a_2 \times 256^2 + \cdots + a_{31} \times 256^{31} \]

• Encode EVM word using RLC (Random Linear Combination)

\[ A_{RLC} \equiv a_0 + a_1 \times \theta + a_2 \times \theta^2 + \cdots + a_{31} \times \theta^{31} \pmod{F_p} \]

• \( \theta \) should be computed after \( a_0, \ldots, a_{31} \) are fixed

  • Multi-phase prover: synthesis part of witness, derive witness
• Break down 256-bit word into 32 8-bit limbs.
  \[ A = a_0 + a_1 \times 256 + a_2 \times 256^2 + \cdots + a_{31} \times 256^{31} \]

• Encode EVM word using RLC (Random Linear Combination)
  \[ A_{RLC} \equiv a_0 + a_1 \times \theta + a_2 \times \theta^2 + \cdots + a_{31} \times \theta^{31} \pmod{F_p} \]

• \(\theta\) should be computed after \(a_0, \ldots, a_{31}\) are fixed
  • Multi-phase prover: synthesis part of witness, derive witness

• RLC is useful in many places
  • Compress EVM word into one value
  • Encode dynamic length input
  • Lookup layout optimization
• Break down 256-bit word into 32 8-bit limbs.

\[ A = a_0 + a_1 \times 256 + a_2 \times 256^2 + \cdots + a_{31} \times 256^{31} \]

• Encode EVM word using RLC (Random Linear Combination)

\[ A_{RLC} \equiv a_0 + a_1 \times \theta + a_2 \times \theta^2 + \cdots + a_{31} \times \theta^{31} \pmod{F_p} \]

• \( \theta \) should be computed after \( a_0, \ldots, a_{31} \) are fixed
  - Multi-phase prover: synthesis part of witness, derive witness

• RLC is useful in many places, remove it?
  - Compress EVM word into one value \( \rightarrow \) high, low for EVM word
  - Encode dynamic length input \( \rightarrow \) fixed chunk, dynamic times
  - Lookup layout optimization
### Execution trace

<table>
<thead>
<tr>
<th>step</th>
<th>opcode</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>PUSH1 80</td>
</tr>
<tr>
<td>2</td>
<td>PUSH1 40</td>
</tr>
<tr>
<td>3</td>
<td>ADD</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>n</td>
<td>RETURN</td>
</tr>
</tbody>
</table>

- **Step 1**
- **Step 2**
- **Step 3**
- ...
The execution trace is dynamic:
- enable different constraints
- permutation is not fixed
- hard to use standard gates

<table>
<thead>
<tr>
<th>step</th>
<th>opcode</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>PUSH1 80</td>
</tr>
<tr>
<td>2</td>
<td>PUSH1 40</td>
</tr>
<tr>
<td>3</td>
<td>ADD</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>n</td>
<td>RETURN</td>
</tr>
</tbody>
</table>
• The execution trace is dynamic
  → enable different constraints
  → permutation is not fixed
  → hard to use standard gates

• Better way to layout?
  • We have 2000+ custom gates
  • Different rotation to access cells
Execution trace

<table>
<thead>
<tr>
<th>step</th>
<th>opcode</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>PUSH1 80</td>
</tr>
<tr>
<td>2</td>
<td>PUSH1 40</td>
</tr>
<tr>
<td>3</td>
<td>ADD</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>n</td>
<td>RETURN</td>
</tr>
</tbody>
</table>

Step 1
Step 2
Step 3
Step n

Hash circuit
Precompile circuit

Circuit - Dynamic size
Circuit - Dynamic size

Execution trace

<table>
<thead>
<tr>
<th>step</th>
<th>opcode</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>PUSH1 80</td>
</tr>
<tr>
<td>2</td>
<td>PUSH1 40</td>
</tr>
<tr>
<td>3</td>
<td>ADD</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>n</td>
<td>RETURN</td>
</tr>
</tbody>
</table>

Step 1

Step 2

Step 3

...
Execution trace

<table>
<thead>
<tr>
<th>step</th>
<th>opcode</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>PUSH1 80</td>
</tr>
<tr>
<td>2</td>
<td>PUSH1 40</td>
</tr>
<tr>
<td>3</td>
<td>ADD</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>n</td>
<td>RETURN</td>
</tr>
</tbody>
</table>

- Some bad influences
  - i.e. Maximum number of Keccaks
- i.e. Mload is more costly (more rows)
  - i.e. Pay larger proving cost for padding
- Can we make zkEVM dynamic?
Prover – Hardware & Algorithm

- Our prover
  - GPU can make MSM & NTT really fast
    Bottleneck moves to witness generation & data copy
  - Need large CPU memory (1TB -> 300GB+)

- Hardware friendly prover?
  - Parallelizable & Low peak memory
  - Don’t ignore the witness generation
  - Run on cheap machines, more decentralized
• **Best way to compose different proof system?**
  • The first layer needs to be “expressive”
  • The second layer needs to be verifier efficient (in EVM)
  • **Should we move to smaller field?**
    (Breakdown/FRI with Goldilocks, Mersenne prime)
  • **Should we stick to EC-based constructions?**
    (SuperNova, Cyclic elliptic curve with fast MSM)
  • More options waiting for you ➔ Reach out to us!

PSE ZK-EVM circuits: 34,469 lines of code

34,469 lines of code are not going to be bug-free for a long long time.
The best way to audit zkEVM circuit?

(In general, VM circuit based on IR)

- Audit Manually
- Formal verification for some opcodes

PSE ZK-EVM circuits: 34,469 lines of code
Outline

- Background & motivation
- Build a zkEVM from scratch
- Interesting research problems
- Other applications using zkEVM
Applications – zkRollup

Layer 1

Layer 2

\[ TX_1 \quad TX_2 \quad \ldots \quad TX_n \]

zkEVM

- Prove n Txs on layer 2 are valid
- Verify proof in smart contract
Application - Enshrine blockchain

Layer 1

TX_1
TX_2
......
TX_n

Senders

Accept
Reject
Accept
Reject
Accept
Reject
Accept
Reject
Application - Enshrine blockchain

Layer 1

• Prove layer 1 block directly
Application - Enshrine blockchain

- Prove layer 1 block directly
- Recursive proof
- One proof for blockchain
Applications – Proof of exploit

- Prove I know a Tx that can change the state root to state root’
  
  (Prove I know a bug that can change your balance, etc)
Applications – Attestation ("zk oracle")

Layer 1

- Trustlessly read historic on-chain data (Need state proof of zkEVM)

ZK circuits

- Read state, compute and verify i.e. Axiom (a zk co-processor)

Verify proofs on-chain
Finally, ...

- **We are building cool things at Scroll!**
  - Scroll is a general purpose scaling solution for Ethereum based on zkRollup
  - Building a native zkEVM using very advanced circuit arithmetization + proof system
  - Building fast prover through hardware acceleration (GPU in production) + proof recursion
  - We are live on the testnet with a production-level robust infrastructure

- **There are a bunch of interesting problems to be solved!**
  - Protocol design and mechanism design
  - Zk engineer & research for practical efficiency
Thank you!

@yezhang1998