zkEVM

DoHoon Kim

Abstraction

- Challenges of zkEVM
- High-level design of zkEVM
- Backend of zkEVM (Halo2)

Design Challenges of zkEVM

- EVM word size is 256-bits (mismatch with prime field)
- EVM has lots of “non-traditional” opcodes
- EVM storage relies on Keccak and MPT
- EVM is too-large

“Hard to design, too-big, and slow for proving”

How do we handle it?

- Modular SNARK (PLONK + PCS)
- Custom gates (TurboPLONK)
- Lookup arguments
- Hardware acceleration

What zkEVM wants to prove?

- Correct execution of program(=smart contract)
- How does smart contract executed on EVM?

What zkEVM wants to prove?

- First, load the bytecode from EVM storage
- Second, execute the bytecode
- Finally, write the execution result back to memory/storage

What zkEVM wants to prove?

- **First, load the bytecode from EVM storage**

  -> prove that loaded correctly

- Second, execute the bytecode
- Finally, write the execution result back to memory/storage

What zkEVM wants to prove?

- cryptographic accumulator
- Use merkle tree
- Use lookup table! (Plookup)

What zkEVM wants to prove?

- First, load the bytecode from EVM storage
- **Second, execute the bytecode**

  -> How to prove the execution of bytecode in static circuit?

- Finally, write the execution result back to memory/storage

What zkEVM wants to prove?

- How can we handle JUMP opcodes? (for-loop, while, if-else)
- Prover has to prove real execution trace (=”unrolled” bytecode)
- How can prover prove correctly unrolled?

What zkEVM wants to prove?

- We label each opcode with program counter and check the consistency

```python

def execution_proof(execution_opcodes):
    program_counter = 0

    for i, opcode in execution_opcodes:
        assert(program_counter == i)
        # execute the current opcode and get the next one to execute
        next_opcode = execute(opcode)
        program_counter = next_opcode
```

Brought from Barry Whitehat's hackmd: https://hackmd.io/5vKFGDcITKuNPHSRT8H9jA

What zkEVM wants to prove?

- First, load the bytecode from EVM storage
- Second, execute the bytecode
- **Finally, write the execution result back to memory/storage**

-> How to prove execution results of opcodes are correct?

What zkEVM wants to prove?

- We have to read/execute/write per opcode
- Prover have to prove about EVM stack in ADD opcode

ADD -> POP, POP, PUSH

What zkEVM wants to prove?

- We separates the proof into 2 parts - state proof, EVM proof
- State proof proves about R/W
- EVM proof proves about correctness of execution result of opcode

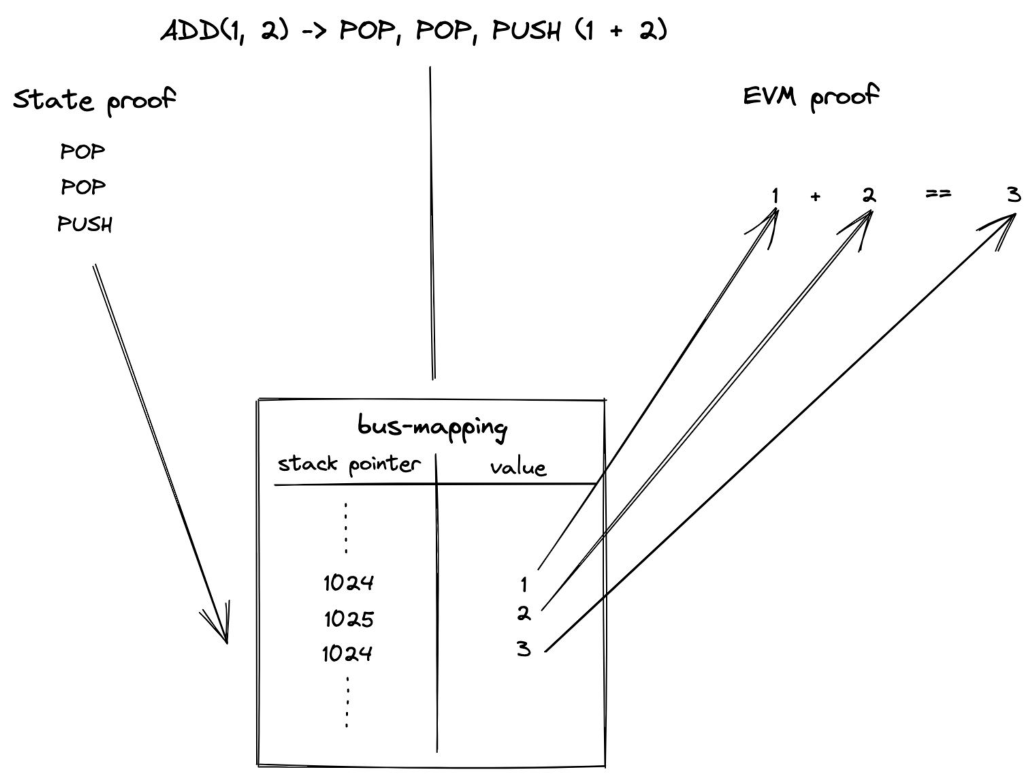
What zkEVM wants to prove?

- State proof proves the opcode behaviour (R/W)
- “bus-mapping” is generated during state proof
- In traditional computer Bus is used to transfer data from memory and storage to CPU
- “bus-mapping” is shared to EVM proof

What zkEVM wants to prove?

- “bus-mapping” is consisted of lookup tables of R/W access records
- State proof does not prove about the execution result
- In EVM proof, prover will lookup the W access of execution result in “bus-mapping”

What zkEVM wants to prove?



PSE

zkEVM backend - Halo2

- zkEVM uses Halo2 as proving system
- Halo2 uses PLONKish arithmetization
- Halo2 also supports lookup argument

zkEVM backend - Halo2

$\boldsymbol{A}$	$\boldsymbol{B}$	$\boldsymbol{C}$	$\boldsymbol{S}$	$\boldsymbol{P}$	$\boldsymbol{Z}$
$A(\omega^0)$	$B(\omega^0)$	$C(\omega^0)$	$S(\omega^0)$	$P(\omega^0)$	$Z(\omega^0)$
$A(\omega^1)$	$B(\omega^1)$	$C(\omega^1)$	$S(\omega^1)$	$P(\omega^1)$	$Z(\omega^1)$
$A(\omega^2)$	$B(\omega^2)$	$C(\omega^2)$	$S(\omega^2)$	$P(\omega^2)$	$Z(\omega^2)$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$A(\omega^{n-3})$	$B(\omega^{n-3})$	$C(\omega^{n-3})$	$S(\omega^{n-3})$	$P(\omega^{n-3})$	$Z(\omega^{n-3})$
$A(\omega^{n-2})$	$B(\omega^{n-2})$	$C(\omega^{n-2})$	$S(\omega^{n-2})$	$P(\omega^{n-2})$	$Z(\omega^{n-2})$
$A(\omega^{n-1})$	$B(\omega^{n-1})$	$C(\omega^{n-1})$	$S(\omega^{n-1})$	$P(\omega^{n-1})$	$Z(\omega^{n-1})$

Q & A