Privacy-Preserving Smart Contract Architectures

Guest Lecturer: Zac Williamson, AZTEC

Zero Knowledge Proofs

Instructors: Dan Boneh, Shafi Goldwasser, Dawn Song, Justin Thaler, Yupeng Zhang





What’s the goal?

- From 1st principles, derivate a blockchain architecture which has…
  - Programmable smart contracts with **private state** as a first-class primitive
  - Transactions are end-to-end encrypted
  - No trusted 3rd parties or hardware, only math!
  - Preserve traditional smart contract semantics
    - contracts can “call” other contracts
    - accessible to non-cryptographers

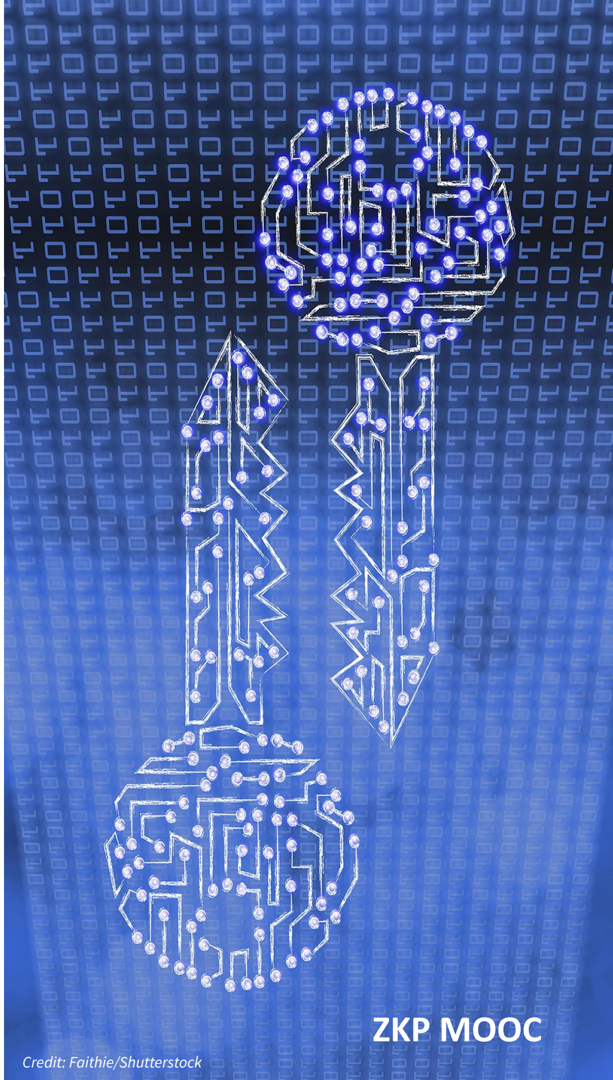
Prior work and influences

- (2015) Zerocoin paper, ZCash
- (2018) ZEXE
- (2020) Mina protocol
- …and over 40 years of zk research!

“Choose your SNARK/STARK”

- We need…
  - fast Prover w. minimal resources
  - fast *arbitrary-depth* recursive proof composition
    - => small proof sizes
- Sumcheck IOP + KZG commitment scheme fits the bill (e.g. Hyperplonk, Honk (TBD))
  - Recursion via Halo2-style curve cycles

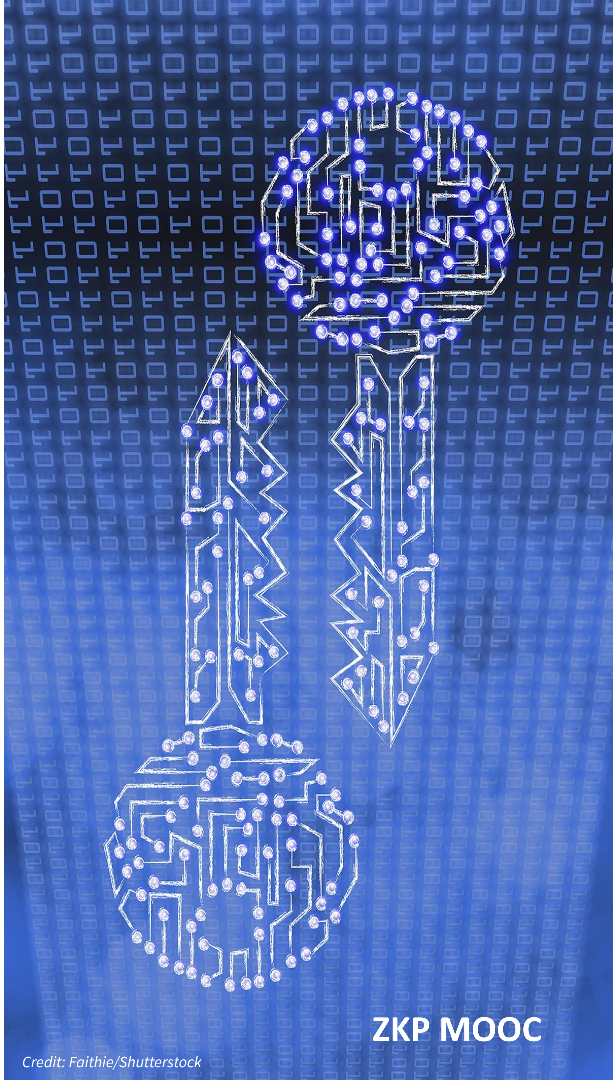
What is a blockchain?



Credit: Faithie/Shutterstock

<u>What is a ***private*** state machine?</u>

<u>What even ***is*** a state machine?</u>



Private state (⅓)

- State must be **encrypted**
  - Owner od decryption key “owns” the state
- State tree == Merkle tree of encrypted state, but…
- Modifying/deleting entry leaks information!
- => Merkle trees must be *append only*
- ***…how do we update state once it’s created?***

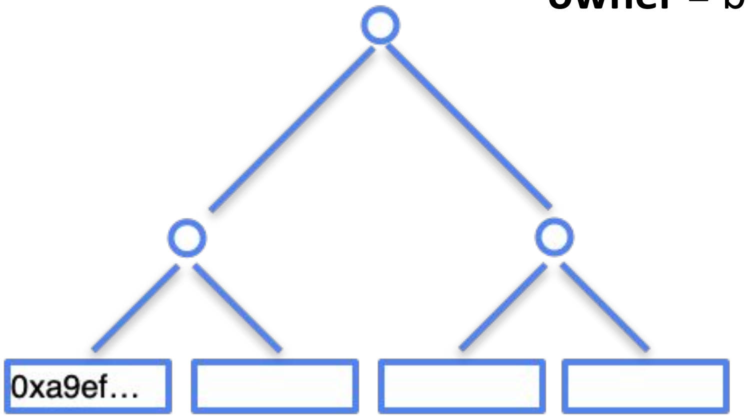
Private state (2/3)

- State is *deleted* via Nullifiers and a *nullifier* set
- Nullifier = encryption of encrypted state!
  - Cannot link nullifier to state w/o decryption key
- State is deleted by adding nullifier to nullifier set
- State is *live* iff nullifier does not exist in nullifier set

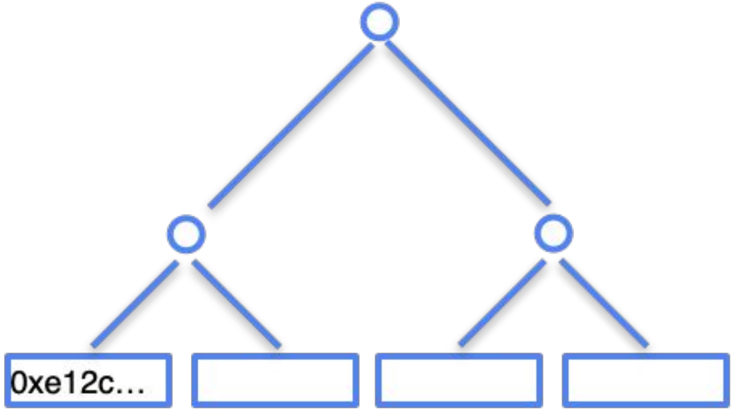
Private state has an inherent UTXO structure

Private state (3/3)

data = “we hold these truths to be self-evident”
owner = bfnklyn.eth



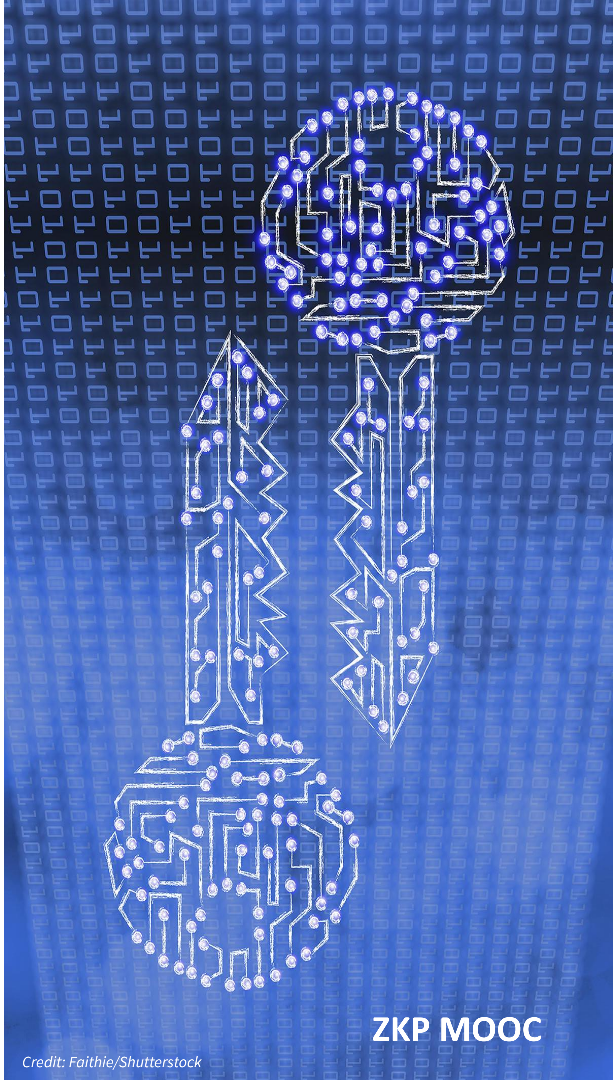
UTXO = Enc(data, owner, owner.sk)



Nullifier = Enc(UTXO, owner.sk)

Q: Is private UTXO state sufficient?

Can we re-create existing blockchain apps?



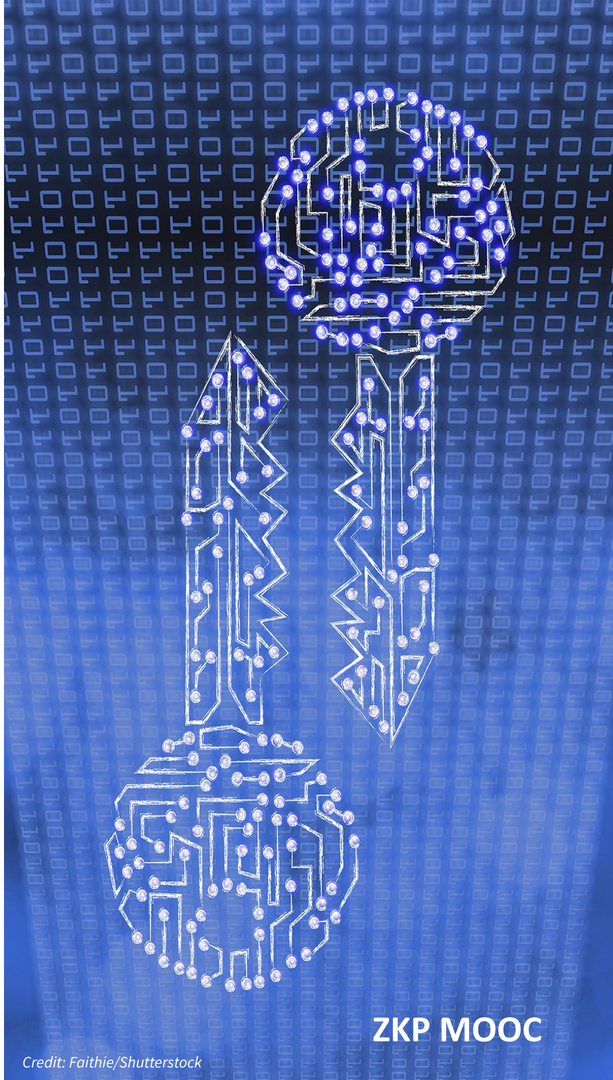
No! We have **PROBLEMS…**

- **Race conditions**
  - 1 UTXO cannot be modified twice in 1 block
- **Ownership requirement**
  - Cannot perform deterministic state updates w/o decryption keye
  - e.g. forced collateral liquidations
- **Need UTXO private state *and* account-model public state**

The road to a private + public state machine

- **Private state transitions**
  - require user-generated proofs of correctness
- Public state transitions
  - ordered + executed **sequentially** by 3rd party e.g.
    - Miner (Eth 1)
    - Validator (Eth 2)
    - Sequencer (L2)

Creating a Smart Contract with private + public state



Time-ordering of state transitions

- (user submits proof of private state transitions
- User tx consists of:
    - proof of private state transition algorithm
    - instruction to execute public state transition algorithm
- Private state transitions happen ***before*** public state transitions
    - How do we present semantics that express this?

Smart contracts for private blockchains

- Contract composed of **public** functions and **private** functions
- **Private functions**
  - Can update UTXO tree
  - Can update nullifier set
  - Can read from *historical* public state
  - Can *unilaterally* call public functions (no return params)

Contract composed of private and public **functions**

Private functions

- Can update UTXO tree
- Can update nullifier set
- Can read from *historical* public state
- Can *unilaterally* call public functions (no return params)

Public functions

- Can update UTXO tree
- Can update nullifier set
- Can read/write public state

Protocol representation of smart contracts

- Functions defined by ZK SNARK *verification keys*
- “Contract” defined by set of function verification keys

- Public inputs of ZK SNARK circuit conforms to a uniform **ABI**

Smart contract ABI example

Public input range	Purpose
0-9	Function argument parameters
10	UTXO tree state root
11	Nullifier tree state root
12	Public tree state root
13	msg.sender (encrypted)
10-19	UTXO leaves to add
20-29	Nullifier leaves to add
30-39	Event parameters

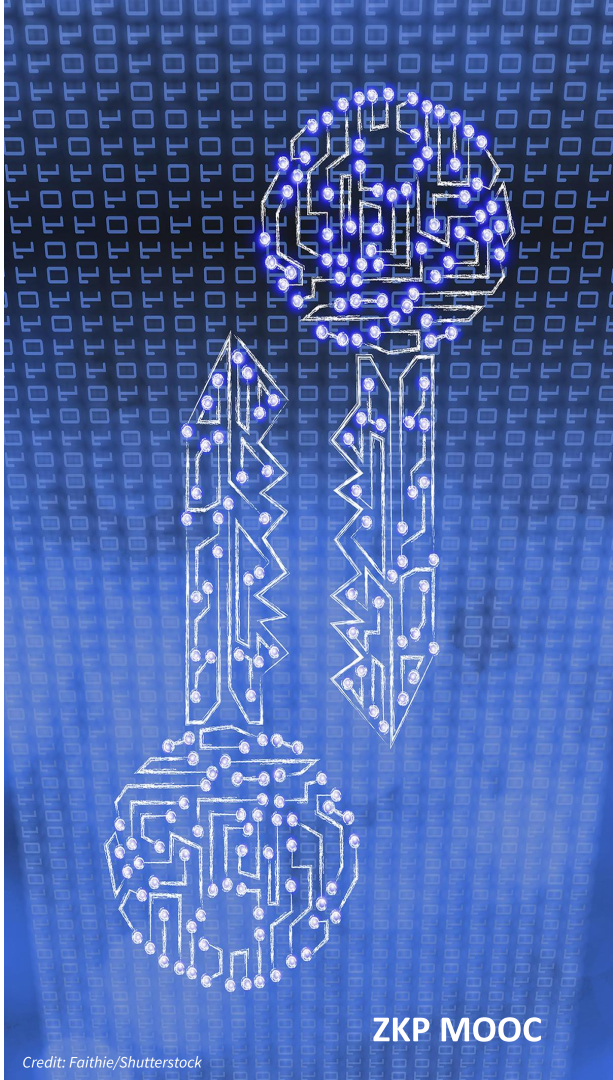
Executing private functions

- Private functions must be executed **client-side** to avoid leaking information
- Require proof of correctness of *sequence* of private function calls
- …what if a private function calls a function from a *different* contract?

We need **call semantics**!

The Private Kernel Circuit

or how I learned to stop worrying and love recursion



What is a “kernel” in general software terms?

- A software layer between user code and the CPU & hardware
- Enforces code **execution rules** and chooses which app runs next on the CPU
- Manages **resource access** and allows cross-app communication

What is a “kernel” in a ZK SNARK?

- A circuit layer between user code (e.g. Noir “contract”) and the protocol execution layer (e.g. L2 rollup)
- Enforces code **deployment** and **execution rules**
- Manages **access to data** and **functions** from within a contract
- Maintains **privacy** of some information

Why do we need a Private Kernel Circuit? (⅓)

- **Privacy**
  - Authenticate user w/o revealing identity
  - Hide contract being called
- **Composability**
  - Functions should be able to call functions of **other contracts**
  - Every contract function is its own circuit && generates own proofs

Why do we need a Private Kernel Circuit? (⅔)

- One TX can contain **multiple proofs** (1 per function)
  - e.g. User calls A.foo(), A.foo() calls B.bar(0 etc
  - A.foo(), B.bar() each represented by a circuit + proof
  - Who combines them and how?

Why do we need a Private Kernel Circuit? (3/3)

- **Combining function proofs requires privacy**
  - **What if a.foo() -> b.bar() passes sensitive information?**

```
function B(some_secret) {
  // Use the secret and return a new one
  return some_secret + other_secret;
}

function A(some_secret) {
  // A calls B, passing in the secret
  new_secret = B(some_secret);
  // maybe call C...
}
```

Alice submits a TX calling "**A(12345)**", and "12345" is an important secret!

High-level recap of Private Kernel (½)

- A circuit that validates the correct execution of ONE private function call
- Circuit structure is **recursive**
- A ***sequence*** of private function calls can be executed via iteratively computing kernel circuit proofs

Can unwind recursion into 1 layer but will leak info

High-level recap of Private Kernel (2/2)

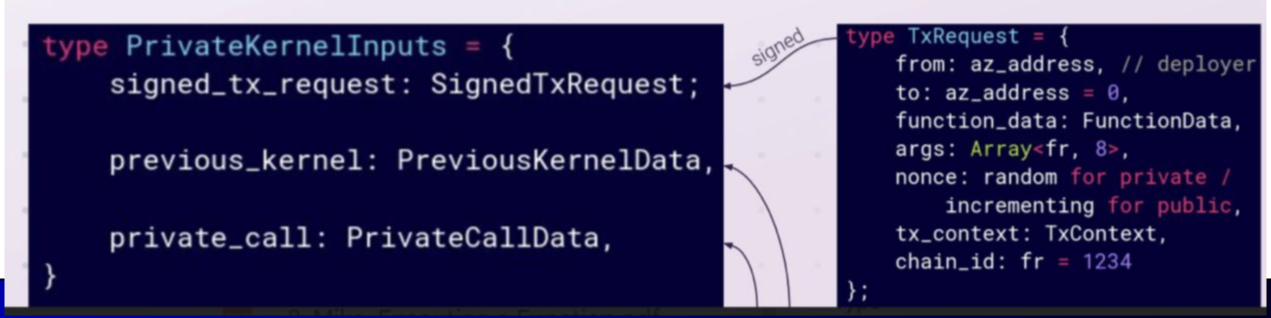
- User generates proof
- Preserves **privacy** of
  - user (tx.origin)
  - (nested) function args and return values
  - state reads
  - the function itself
- User submits a **single proof** for full execution of private function callstack

For each function call in the callstack..

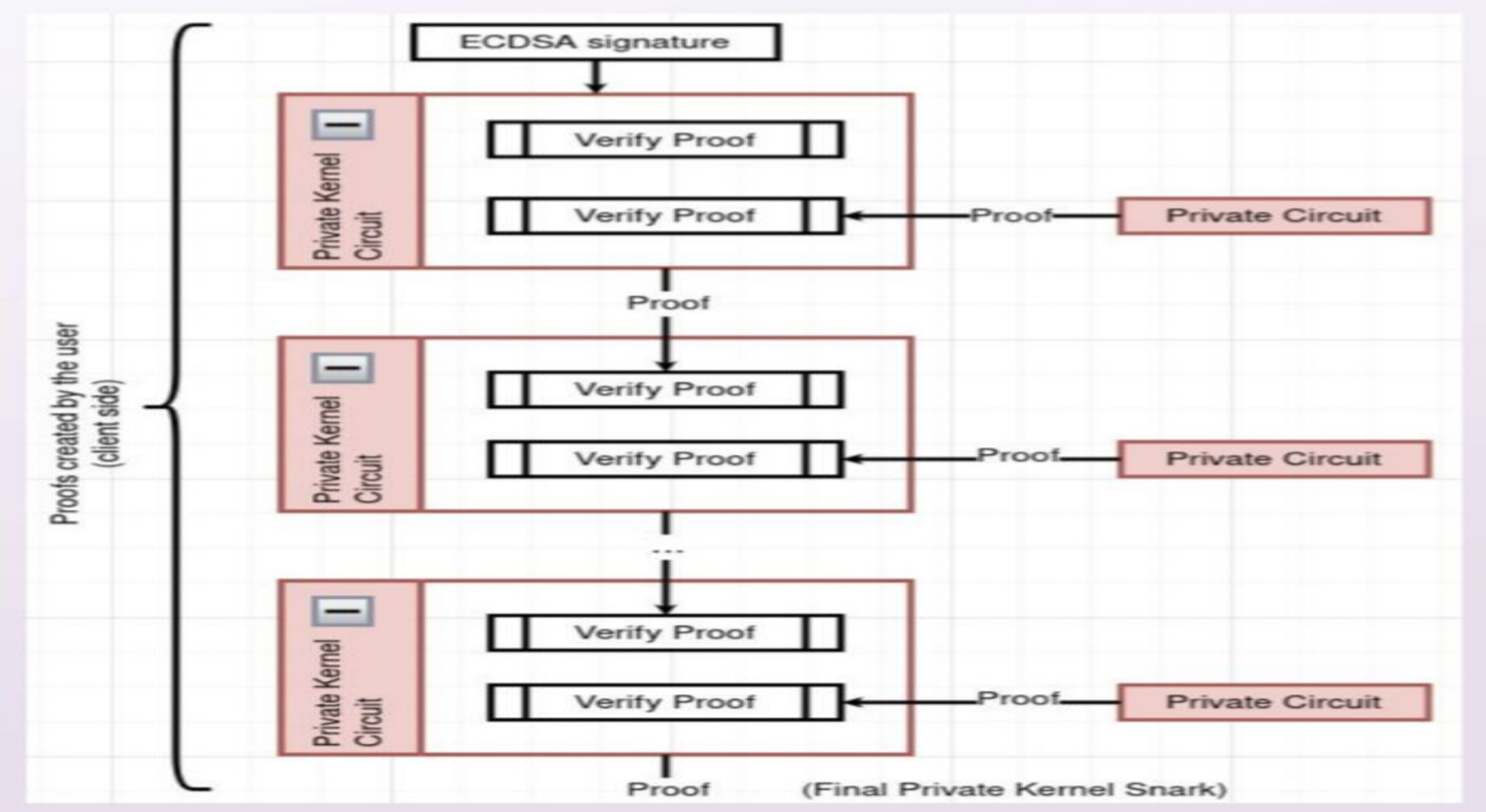
- Prove the following
  - signed TX request matches first call in callstack
  - function exists in function tree
  - contract exists in contract tree
  - commitments referenced by function are in data tree
- Collect new commitments, nullifiers, contracts
- Verify previous kernel proof
- Verify proof for current function being processed

Inputs to the Private Kernel

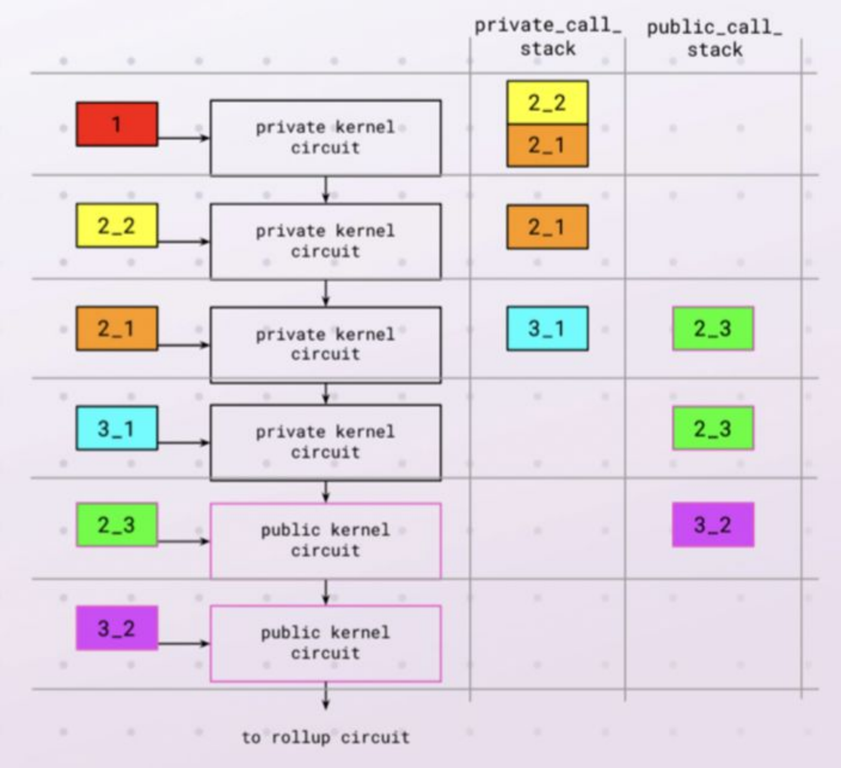
- SignedTxRequest
  - Original request from user to call 1st function in the stack
- PreviousKernelData
  - Kernel is recursive! Accumulated data from previous iterations
- PrivateCallData
  - Data relevant to function call being processed



Kernel recursion



Kernel recursion through callstack

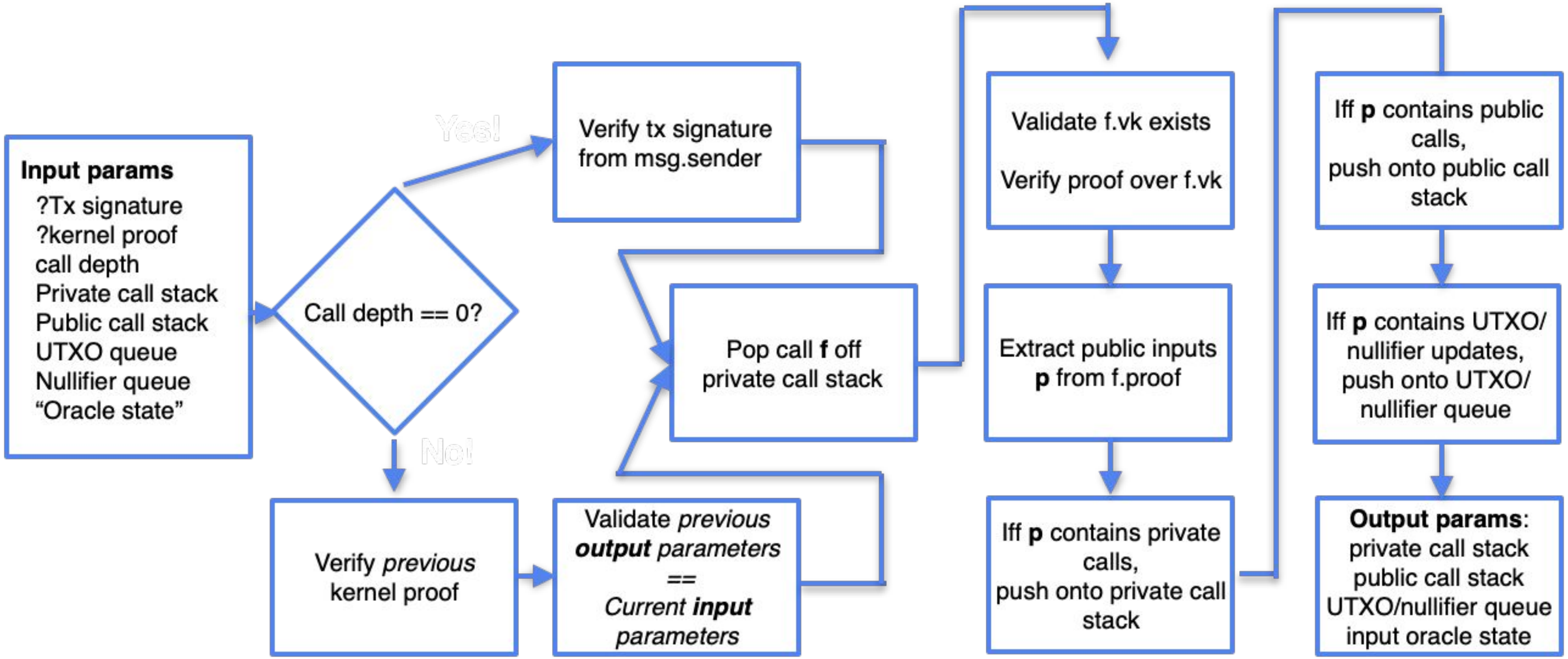



```
import Contract2;

contract Contract1 {

  private uint x;

  function1(uint a, uint b, uint c) {
    d = Contract2.function2_1(a, b);
    x += d;
    Contract2.function2_2(c, x);
  }
}
```

```
contract Contract3 {

  private uint z;

  function3_1(uint a, uint b) {
    return a * b;
  }

  public function3_2() {
    z++;
  }
}
```

```
import Contract3;

contract Contract2 {

  private uint y_1;
  uint y_2;

  function2_1(uint a, uint b) {
    d = Contract3.function3_1(a, b);
    y_1 += d;
    function2_3(a);
    return d;
  }

  function2_2(uint c, uint x) {
    return c * c;
  }

  public function2_3(uint a) {
    y_2 += a;
    Contract3.function3_2();
  }
}
```

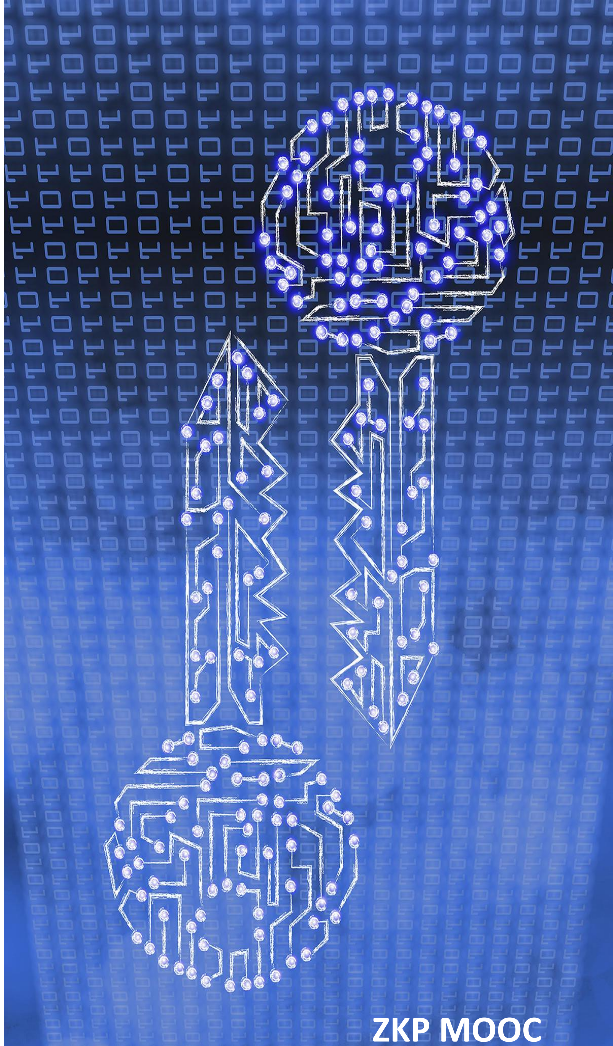
Private kernel circuit architecture



Kernel Circuit **does not**:

- Execute function circuits themselves
  - Done prior to the kernel
- Perform tree insertions
  - Commitments, nullifiers etc…
  - This is done in a “rollup” circuit by Sequencer/Prover
- Merge multiple separate TXs
  - Sequencer/prover aggregates TXs in a “rollup” circuit

The Public Kernel Circuit: public function execution



Credit: Faithie/Shutterstock

ZKP MOOC

State of a tx in the public mempool

- ZK Proof of private kernel
  - private callstack must be **empty**
  - public callstack contains **functions to be executed**
- Public function execution must be validated via a **public kernel circuit**
- Public kernel proofs generated via Sequencer/Prover

Computing proofs of public functions

- Public function proofs computed by 3rd party sequencer/prover
- Function proofs wrapped in a **public kernel circuit**
- One significant complication:
- **Sequencer must be fairly compensated for the work they perform**

User/Sequencer trust problem

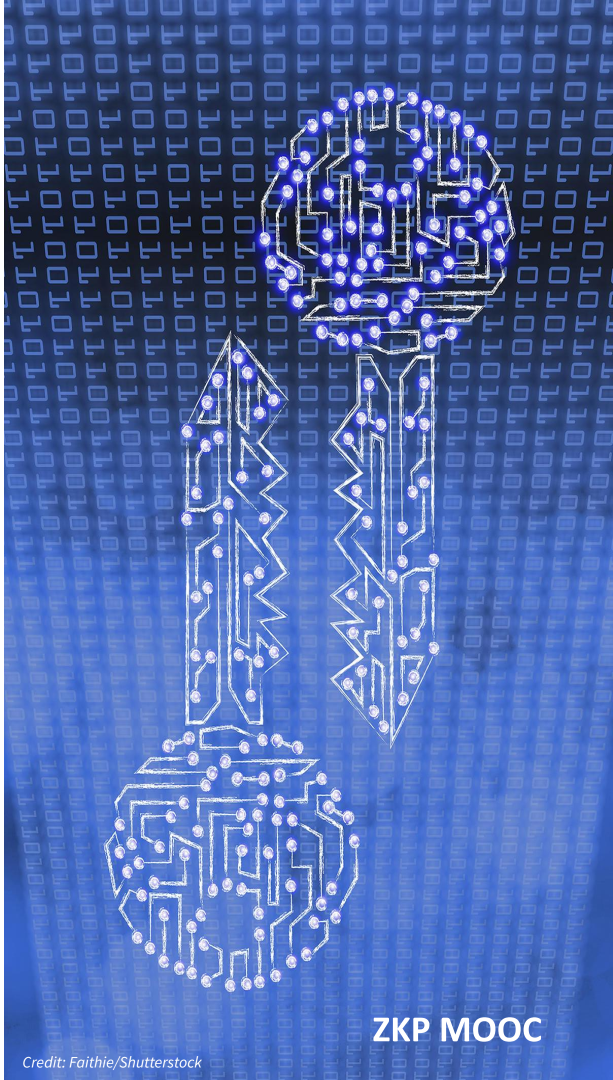
- A function proof can be invalid from 2 causes:
  - Choice of public inputs creates unsatisfiable constraints (i.e. transaction throws an error)
  - Witness assignment is deliberately invalid

- For public functions…
  - 1st failure case caused by tx sender
  - 2nd failure case caused by sequencer

Public functions require a VM!

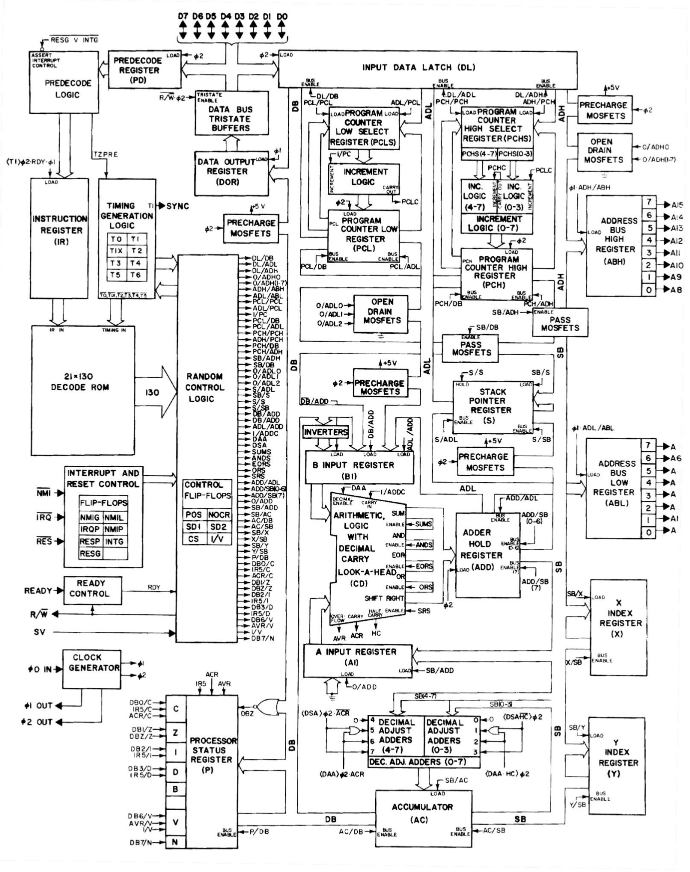
- A valid tx requires the VM proof to be valid
  - i.e. sequencer can’t grief a user
- A valid VM proof can return execution result as a public input
  - i.e. user cannot force sequencer to do unpaid work

How do Virtual Machines Work?

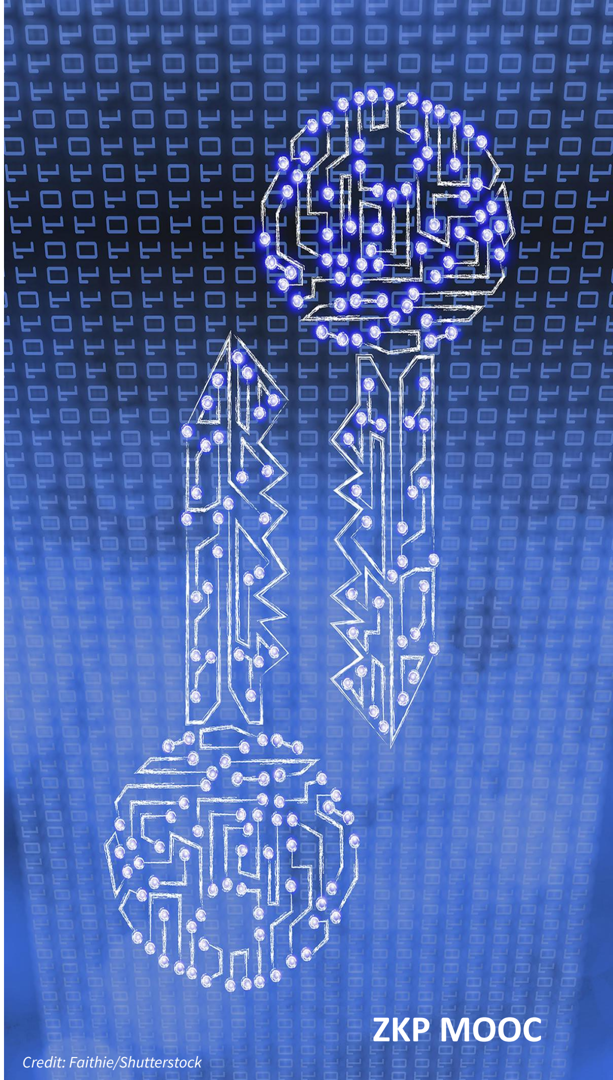


CPU Architectures: high-level

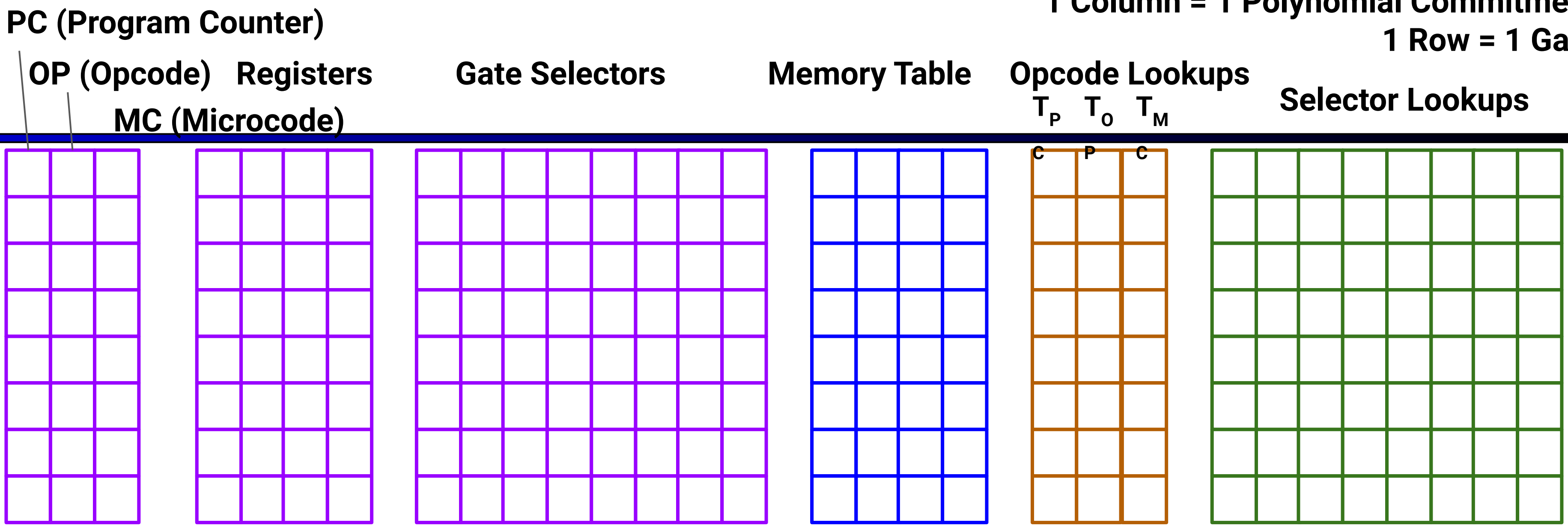
- Opcode: part of CPU instruction set: treated as atomic operation
- Microcode: Opcodes split into micro-opcodes. 1 clock cycle performs 1 microcode operation
- Registers store data being worked on
- RAM stores remaining data
- Arithmetic instructions executed by “Arithmetic Logic Unit”



How does a SNARK VM Work?



1 Column = 1 Polynomial Commitment
1 Row = 1 Gate

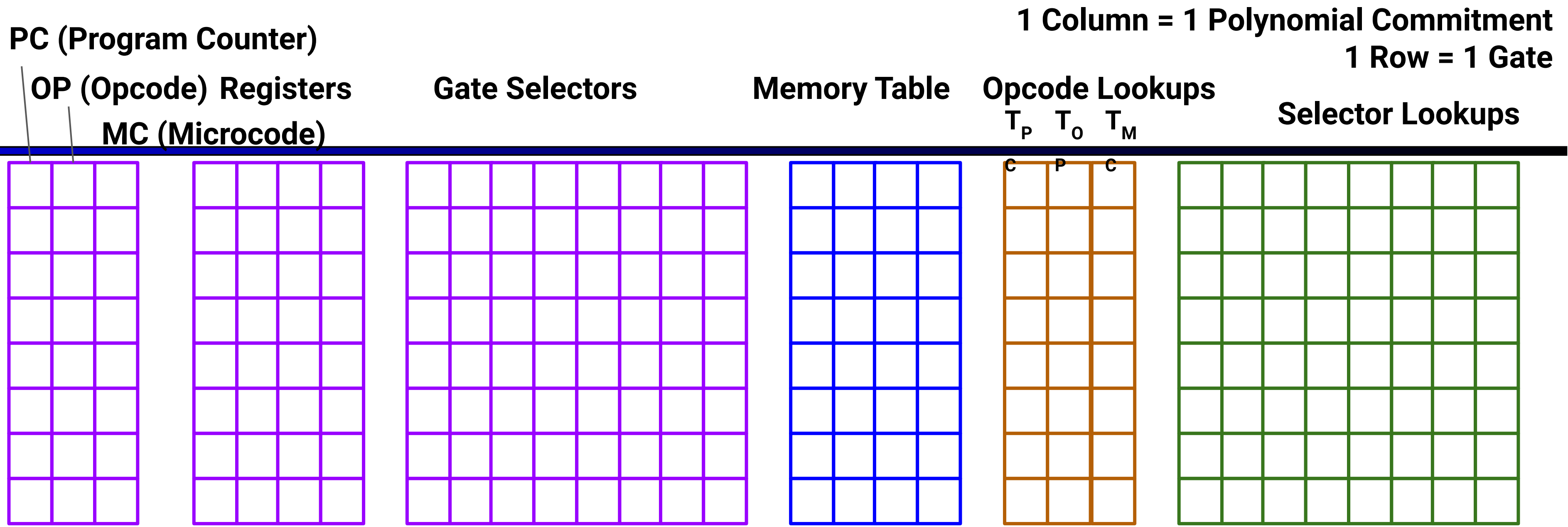


Runtime columns (committed to by Prover)

Program-specific lookup table (precomputed)

Runtime lookup table (committed to by Prover)

VM lookup table (precomputed)



OP, MC read from **Opcode Lookup** table (indexed by **PC**)

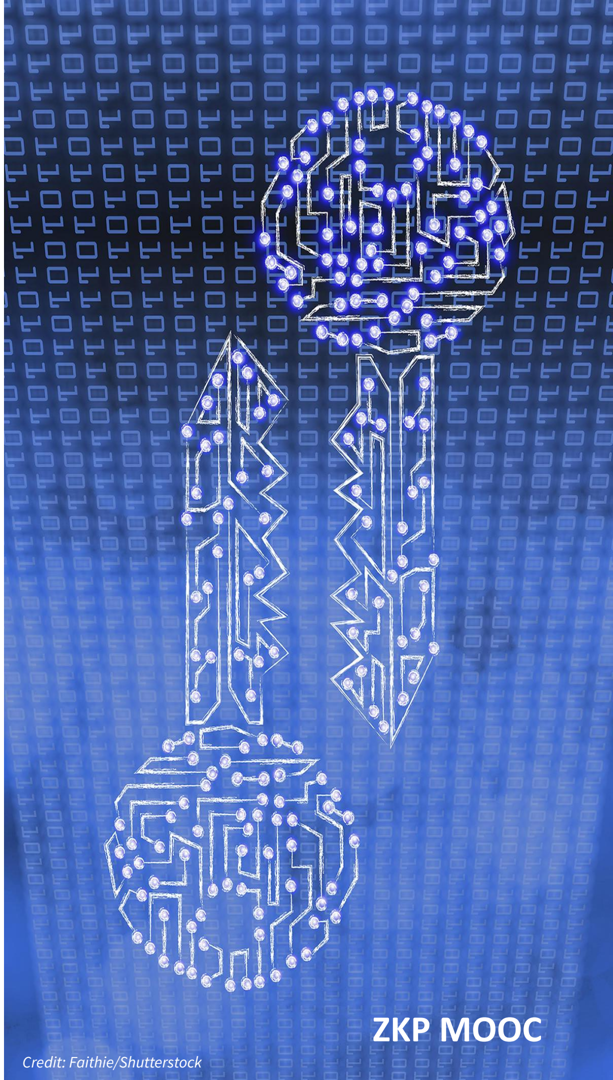
Gate Selectors read from **Selector Lookup** table (indexed by **OP, MC**)

Registers, PC values dependent on **Gate Selectors**

Example SNARK VM Opcodes

Opcode	Num Microcode Ops	Gate Expression	Technique
Add	1	$R1_{i+1} = R1_i + R2_i$	Custom gate
MOV [R1]	1	$R1_{i+1} = M[R1_i]$	Lookup
XOR R1 [R2]	1	$R1_{i+1} = R1 \wedge M[R2_i]$	Custom gate + Lookup
SHA256	3,000	$M[R1_i] = SHA256(M[R2_i])$	3,000 gates + lookups!
JUMPI X	1	$PC_{i+1} = (R1_i == 0)\ ?\ PC_i + 1 : X$	Custom gate

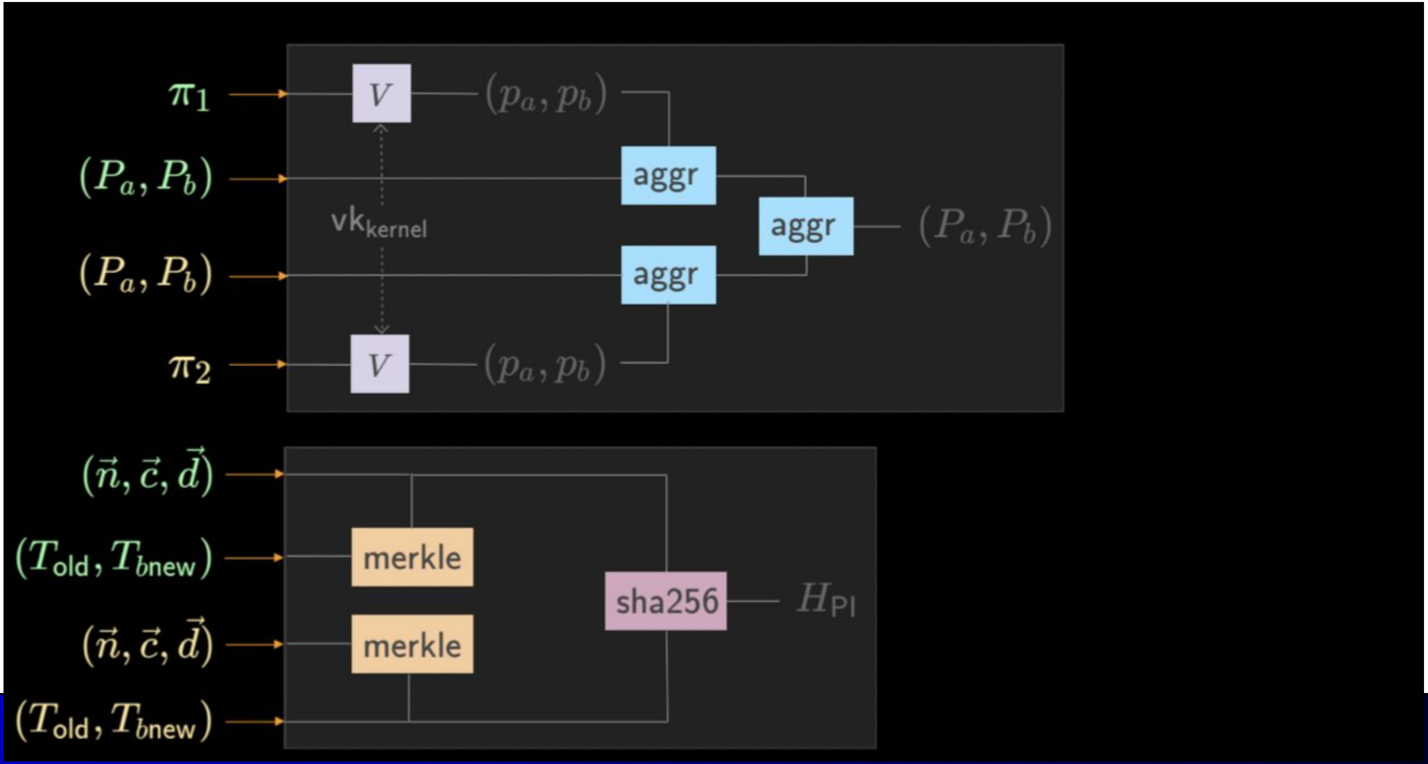
Rollup Circuit: Aggregating txs



Why do we need a rollup?

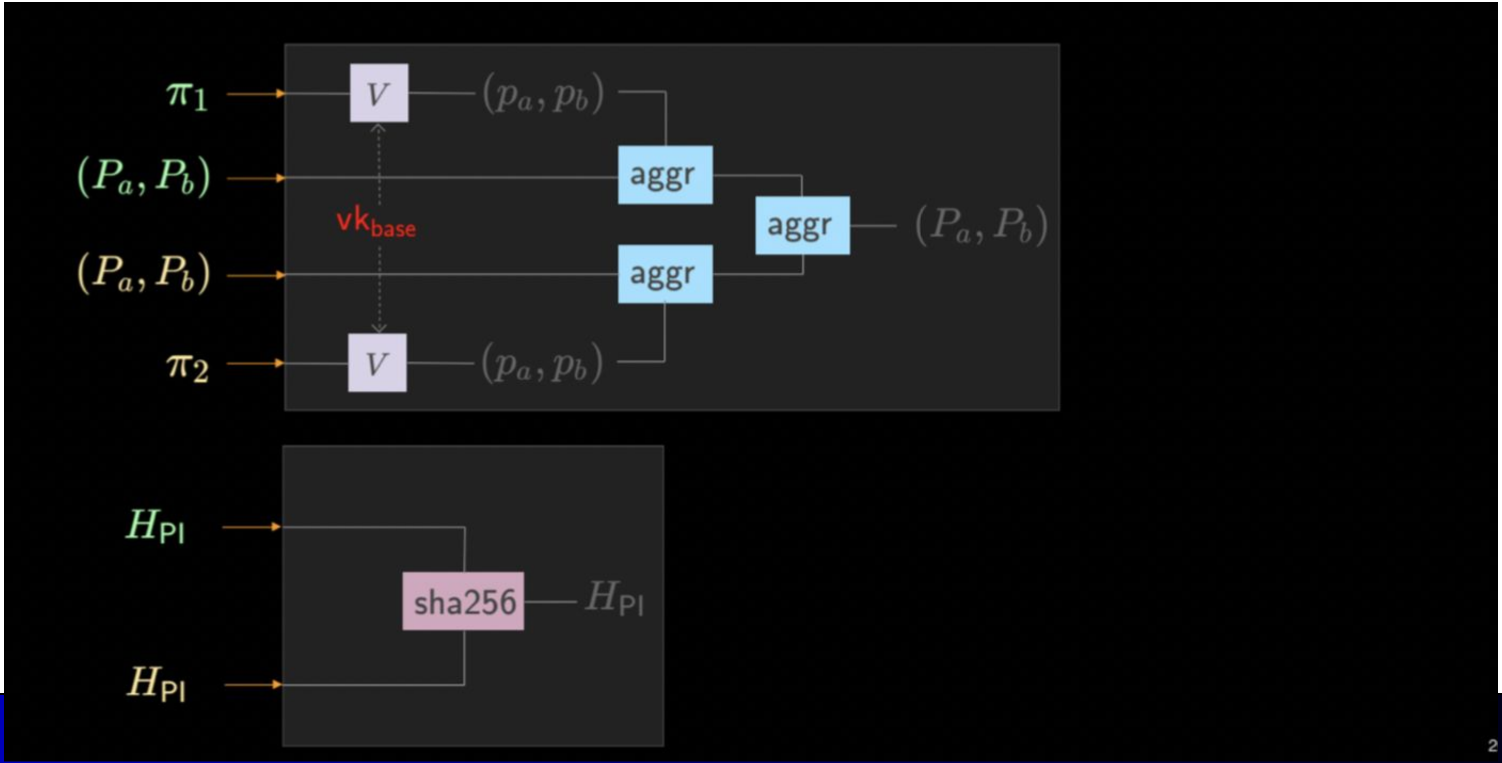
- Validation of a **block** of txns is expensive due to verifier costs!
- Ideal if consensus layer only needs to validate **proof of block correctness**

Base Rollup Circuit

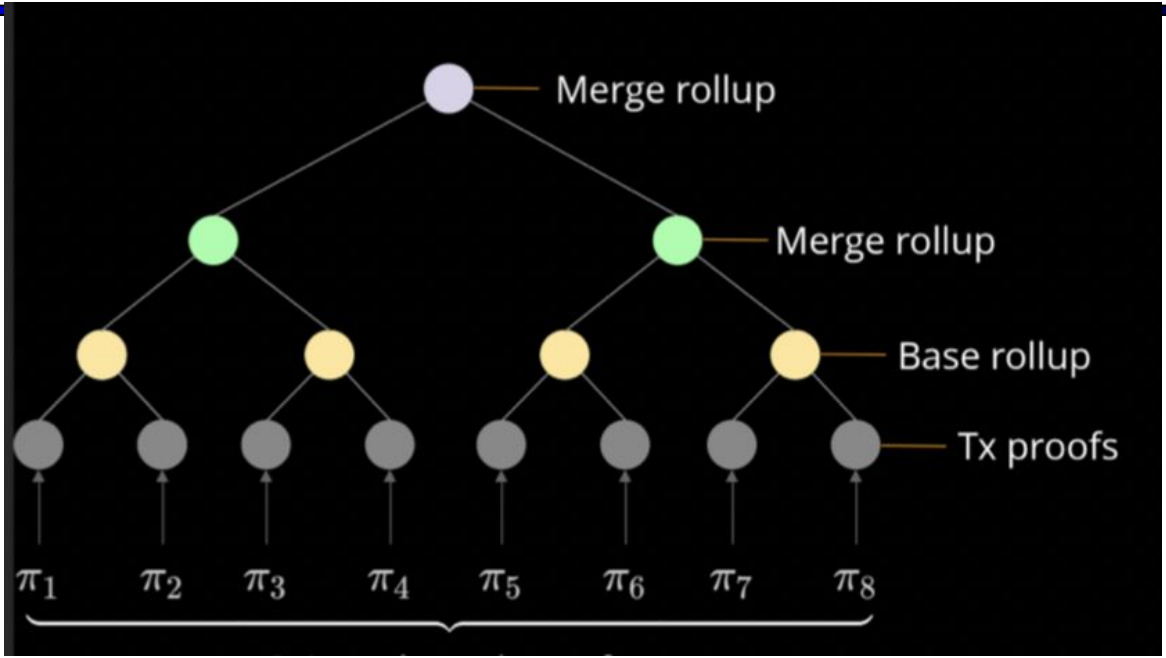


ZKP MOOC

Merge Rollup Circuit

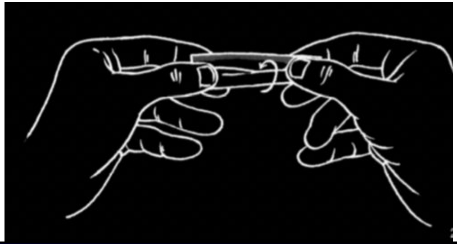


Rolling Up



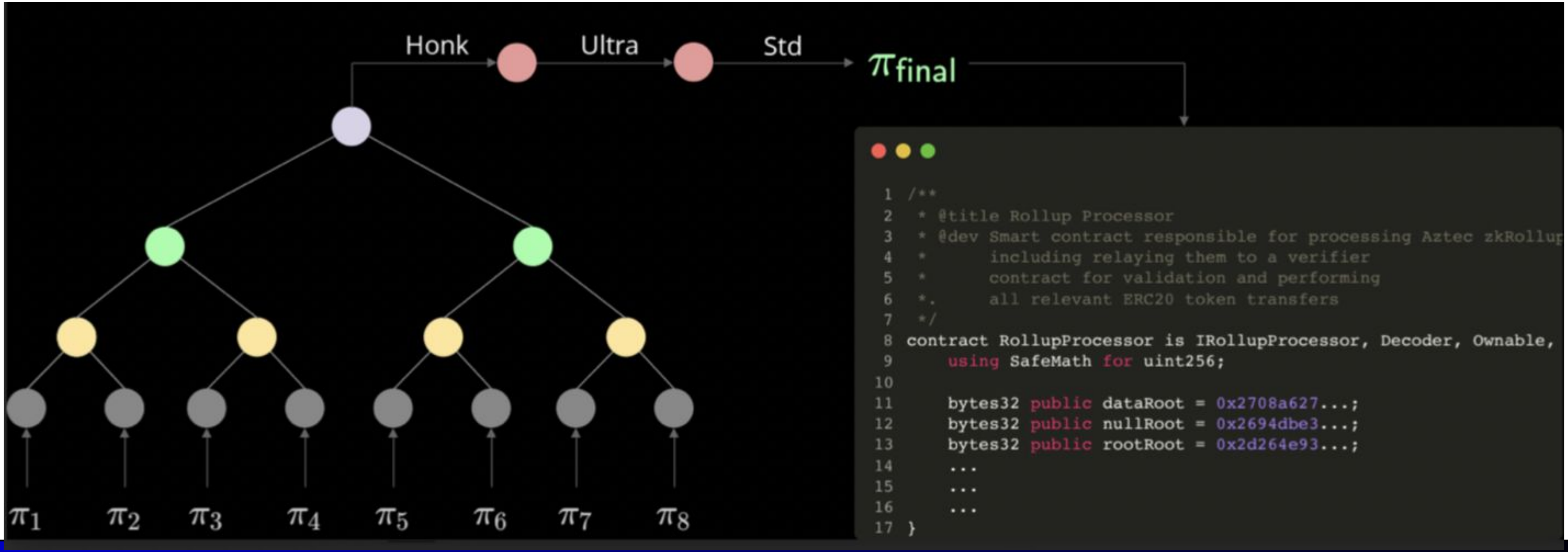
Public kernel proofs

- We roll 2 proofs/circuit
- Small circuit sizes = fast proofs
- Helps decentralization

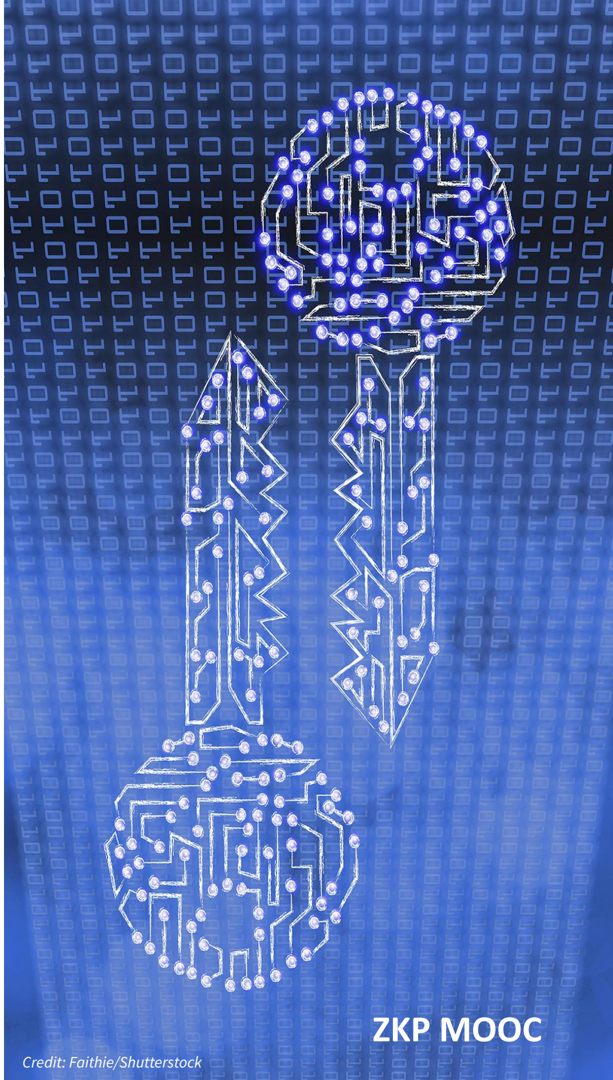


Root Rollup Circuit

- Recursively “devolve” proof systems to reduce vinyl verification cost



Putting it all together



Recap (1 / 2)

- 3 State trees (private state, public state, contract state)
- 1 Nullifier set (private state)
- Contracts defined via set of verification keys for private/public functions

Recap (2 / 2)

- Private kernel circuit validates private function execution
- Public kernel circuit validates public function execution + private kernel proof
- Rollup circuit validates public kernel proof + performs state updates
- Root rollup circuit validates rollup proof using SNARK protocol w. low verification costs

Many thanks to

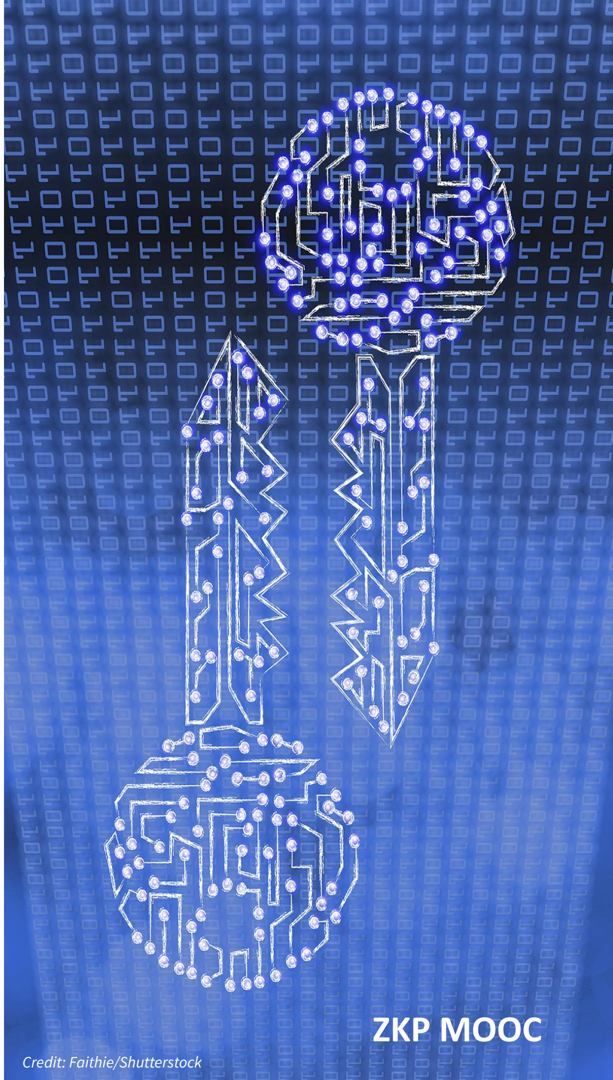
David Banks

Suyash Bagad

Michael Connor

Genevieve Birdsall

Joseph Andrews



Credit: Faithie/Shutterstock