Public-Private Composability.

(A “private smart contract” architecture).

Mike Connor
Aztec
Aims: Private smart contracts

- Private states
- Private function execution
- Permissionless contract deployment
- Composability: calls between contracts
- Composability: calls between private & public functions
- Intuitive transaction semantics
What is a smart contract?

A collection of state variables, and functions which may edit them.

```solidity
contract ERC20 {
    mapping(address => uint256) balances;

    function transfer(address to, uint256 amount) {
        balances[msg.sender] -= amount;
        balances[to] += amount;
    }
}
```
What will an Aztec smart contract look like?
Aside: What’s a public state?

It’s just a value in a tree.

State Tree

balance[Alice]
Aside: What’s a private state?

It’s a commitment to a value in a tree. It has an owner.

h( owner_public_key, value, salt, other_stuff )
State - Account vs UTXO model

Public state → Account model

Private state → UTXO model
Private State change, within a Private Function

value: 10
owner: Alice,
salt: 0x1234,

Prove knowledge of Alice's secret key

nullifier = h(commitment, Alice_secret_key)

Perform operations on the private state's value

value: 15,
owner: Alice,
salt: 0x3456,

Prove it doesn't yet exist in the tree.

This has to happen on the user's device!
Private Functions
L2 functions are zk-SNARK circuits

- The Verification Key can be a unique ID for the circuit.
- $h(\text{Verification Key})$ succinctly represents the circuit.

*This will all be handled by Noir*
Contract tree

- contractTreeRoot
  - contractRoot
    - contractAddress
    - portalContractAddress
    - vkRoot
      - vk_0
      - vk_1
      - vk_2
      - vk_3
Modifying private state in Zexe

Each can be represented by $h(\text{verification_key})$

Instead...

$h(\text{contract_address}, h(\text{owner_pk}, \text{payload}, \text{birth Predicate}, \text{death Predicate}, \text{nonce}, \text{salt}))$

$h(\text{owner_pk}, \text{is_dummy}, \text{creator_pk}, \text{storage_slot}, \text{value}, \text{memo}, \text{nonce}, \text{salt})$
Hiding the function being executed with Zexe

Private circuit

\[\pi\]

Kernel circuit

\[\pi\]

\[\pi\]

No one learns which function was executed.
Functions calling functions

(...calling functions calling functions calling functions...)

Functions calling functions
Example

A decentralized exchange.

- Alice pings exchange contract to swap tokens A<>B
- Exchange contract pings contract A to transfer tokens to Exchange
- Exchange contract pings contract B to transfer tokens to Alice

Extremely basic transactions require nested function calls
Function calls

- contract_address
- 4 bytes of keccak hash of function signature
- arguments

- contract_address
- vk_index
- h(public inputs)
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);
    }
}

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();
    }
}

contract Contract3 {
    uint z;

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

    public function3_2() {
        z++;
    }
}
6 distinct proofs are generated (one for each function).

How do we prove they relate to one-another?
### Call stacks!

#### Contract 1

```solidity
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);
    }
}
```

### Import Contract 2

```solidity
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 2

```solidity
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();
    }
}
```

### Import Contract 3

```solidity
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();
    }
}
```

#### Contract 3

```solidity
contract Contract3 {
    private uint z;

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

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

---

<table>
<thead>
<tr>
<th>Functions which call other functions</th>
<th>private_call_stack</th>
<th>public_call_stack</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2_2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2_1</td>
<td></td>
</tr>
<tr>
<td>2_1</td>
<td>3_1</td>
<td>2_3</td>
</tr>
<tr>
<td>2_3</td>
<td></td>
<td>3_2</td>
</tr>
</tbody>
</table>

**Diagram:**

- **functions** (Contract 1 and Contract 2)
- **private_call_stack** (Contract 2)
- **public_call_stack** (Contract 2)
Kernel Recursion

1. **private kernel circuit**
   - 2_2
   - 2_1

2. **private kernel circuit**
   - 2_1
   - 3_1

3. **public kernel circuit**
   - 2_3
   - 3_2

4. **to rollup circuit**

Function stack:
- 1
- 2_2
- 2_1
- 2_3
- 3_1
- 3_2
Public Inputs ABIs,...

**Private Circuits**

custom_inputs: [],
custom_outputs: [],
emitted_data: [],
output_commitments: [],
input_nullifiers: [],
old_private_data_tree_root,
private_call_stack: [],
public_call_stack: [],
contract_deployment_call_stack: [],
l1_call_stack: [],
callback_stack: [{
  success_callback,
  failure_callback,
},
executed_as_callback: {
  l1_result_hash,
  l1_results_tree_leaf_index,
},
bools,

**Public Circuits**

custom_inputs: [],
custom_outputs: [],
emitted_data: [],
state_transitions: [],
state_reads: [],
old_private_data_tree_root,

public_call_stack: [],
contract_deployment_call_stack: [],
l1_call_stack: [],
callback_stack: [{
  success_callback,
  failure_callback,
}],
executed_callback: {
  l1_result_hash,
  l1_results_tree_leaf_index,
},
bools,
prover_address,
Private Kernel ABI

Private Inputs

signature,
start: {
  aggregated_proof,
  private_call_count,
  private_call_stack: [],
  public_call_stack: [],
  contract_deployment_call_stack: [],
  l1_call_stack: [],
  callback_stack: [],
  optionally_revealed_data: [],
  output_commitments: [],
  input_nullifiers: [],
},
previous_kernel: {
  proof,
  public_inputs,
  vk,
  vk_index,
  vk_path,
},
private_call: {
  function_signature,
  public_inputs,
  call_context,
  proof,
  vk,
  vk_index,
  vk_path,
  portal_contract_address,
  contract_leaf_index,
  contract_path,
  privately_executed_as_callback: [],
},
end: {
  aggregated_proof,
  private_call_count,
  private_call_stack: [],
  public_call_stack: [],
  contract_deployment_call_stack: [],
  l1_call_stack: [],
  callback_stack: [],
  optionally_revealed_data: [],
  output_commitments: [],
  input_nullifiers: [],
},
constants: {
  old_tree_roots: {
    private_data_tree,
    contract_tree,
    l1_results_tree,
    private_kernel_vk_tree,
  },
  is_constructor_recursion,
  is_callback_recursion,
  executed_as_callback: [],
},
globals: [],
bools

Public Inputs
Rollup topology

TX 1
- function1: private kernel circuit
- function2: private kernel circuit
- function3: private kernel circuit
- function4: private kernel circuit
- function5: public kernel circuit
- function6: public kernel circuit

TX 2
- function1: public kernel circuit
- function2: public kernel circuit
- function3: private kernel circuit

TX 3
- function1: private kernel circuit
- function2: private kernel circuit
- function3: private kernel circuit

TX 4
- function1: private kernel circuit
- function2: private kernel circuit
- function3: public kernel circuit

CONSTRUCTOR
- function1: private kernel circuit
- function2: private kernel circuit
- function3: public kernel circuit

contract deployment kernel circuit

ECDSA signature

BASE ROLLUP verify proof

MERGE ROLLUP verify proof

SQUISHER verify proof

ROLLUP SMART CONTRACT
What does the kernel circuit do?

It makes sure txs follow the rules.
- Verifies msg_sender’s signature
- Pops the next call (app proof) off the call stack
- Verifies the app proof
- Verifies the previous kernel proof
- Validates consistency between previous kernel’s accumulated end data, and this kernel’s start data
- Ensures the function (app proof) being verified belongs to the purported contract
- Ensures the contract exists
- For L1 calls, checks the purported portal contract’s address corresponds to this contract
- Pushes new function calls to call stacks
- Checks delegatecall / staticcall contexts
- ‘Silos’ new commitments & nullifiers
- Optionally reveals data to public L2 / L1:
  - Fees, Events, Calls, Deployment data
- Checks that executed callbacks refer to valid L1 Result leaves
- (And some other stuff)

We recurse through the call stacks until they’re empty.
Thank you!

Mike Connor
Aztec

@mike_connor
@aztecnetwork