Setting the scene

Aztec connect has a very neat way of allowing someone to create notes for an anonymous

person: partial commitments.

Suppose the full commitment is of the form:

commitment = storage_slot * A + value * B + owner * C + salt * D + nonce * E

A,B,C,D,E

are generator points for some elliptic curve. This commitment is a Pedersen commitment, and it’s additively homomorphic, meaning we can add data to the hash at some later time.

Keeping with the Aztec Connect example for a moment, suppose everyone’s token balances were declared in a Noir++ contract as a state variable of the form:

// owner => asset_id => balance utxos mapping(address => mapping(uint => UTXOSet)) balances;

The storage_slot

for balances[alice_address][asset_id]

would be something like:

storage_slot = h(h(1, alice_address), asset_id)

The 1

is for example’s sake; the ‘base’ storage slot of the variable, based on the ordering of declaration of the variables in the contract’s scope, similar to how Solidity does it.

Now,

- Alice wants to remain anonymous.
- We don’t know what the storage_slot

will be, because we don’t know what the asset_id

will be which is returned from the bridge contract.

- Furthermore, we don’t want to reveal Alice’s address; we want her to remain anonymous.

This is an interesting point to pause. Alice’s address is actually contained in two

places in my mental model, in this example: the storage_slot

AND the owner

of the commitment.

So I propose we derive storage slots

with homomorphic commitments, so that Alice can embed alice_address

in the partial commitment, and then later Bob can complete

the storage slot with the asset_id

.

So using Pedersen hashes (ish) for h()

:

storage_slot = h(h(1, alice_address), asset_id)

is modified slightly to:

storage_slot = 1 * X + alice_address * Y + asset_id * Z

Again, X,Y,Z

are generator points.

If the storage slot is now a point, we can modify the commitment we wrote at the start, to remove A

:

commitment = storage_slot + value * B + owner * C + salt * D + nonce * E

With this, our storage slot can be “completed”.

Continuing:

- Alice doesn’t know what the value

will be until later either.

- So, Alice creates a partial commitment:partial_commitment = 1 * X + alice_address * Y + owner=alice_address * C + salt * D + nonce * E
- We can store this partial_commitment

as a public

state.

- Some time later, when we know the asset_id

and value

:

- Bob comes along and ‘completes’ Alice’s commitment:complete_commitment = partial_commitment + asset_id * Z + value * B

  = 1 * X + alice_address * Y + asset_id * Z + value * B + owner * C + salt * D + nonce * E

  = storage_slot + value * B + owner * C + salt * D + nonce * E

Summary

- Above we describe a little bit of Aztec Connect functionality, but in the context of Aztec3.
- Expressing every storage slot as an elliptic curve point, gives flexibility for storage slots to be ‘completed’ at some time in the future.

Syntax

So what would Noir++ syntax look like, to give us control

over which components of a storage_slot and/or a UTXO’s preimage we can insert some time later (to achieve partial commitment functionality).

My laptop’s about to die, but I’m open to suggestions for syntax…