Building on Zac's proposal, which is here (definitely needs to be read before continuing).

This post just suggests some little modifications, as well as explaining that we might need some additional syntax.

Edit: actually, I'll split different syntax suggestions into different posts.

1.

This isn't the main suggestion of this post, but it's a slight syntax tweak which I adopt in section 2.

Private state variables can be represented in one of two forms:

1. Singleton mapping(address => Object)
2. Unbounded Array mapping(address => UArray(Object))

Suggestion:

Rename the types to UTXO

and UTXOSet

, respectively.

- "UTXO

" aligns nicely with the insert()

& remove()

methods of a UTXO

variable.

- "Set" in UTXOSet

conveys that a collection of UTXOs has no ordering; hence the need for the sort & filter functions.

2.

For a commitment to a private state within the utxo tree, its preimage will be something like:

- h(contract_address, h(storage_slot, value, owner, memo, nonce, salt))

I think the syntax will need to allow the dev to specify

who the owner

is when get

ting, insert

ing and remove

-ing a utxo, to avoid ambiguities. I give 3 examples below where the owner

would not be clear, without having extra syntax to explicitly declare who the owner

should be.

I've chosen this syntax to resolve this problem in each example: { owner:__ }

Compare these three contract excerpts.

Example 1

:

Users transferring tokens, similar to aztec.

With this example alone, one could think the owner

of a note could always be inferred to be equal to the address

value of the mapping's mapping key. But in examples 2 & 3, we'll see that sometimes the owner and mapping key won't be equal, sometimes the mapping key won't be an address

type, and sometimes the UTXOSet won't even be contained in a mapping.

mapping(address => UTXOSet) balances;

// Not shown: a deposit function.

transfer(uint amount, address to) { // We can't always assume that the owner of the notes will be // equal to the 'key' of the mapping (msg.sender on this line); // but the dev might want them to be distinct (see next e.g.) UTXO[2] notes = balances[msg.sender].get(2, sort, filter, { owner: msg.sender });

uint input_amount = notes[0] + notes[1];
require(input_amount >= amount);

notes[0].remove();
notes[1].remove();

balances[msg.sender].insert(input_amount - amount, { owner: msg.sender });

balances[to].insert(amount, { owner: to });

}

Example 2

: an example contract where people can increase other peoples' balances (via deposits or transfers), but only via a central party - let's call them a bank. I.e. all notes are owned by the bank, and can only be nullified by the bank, but anyone can add new notes (via a function which isn't shown here).

mapping(address => UTXOSet) balances; address bank_address;

// Not shown: a deposit function.

transfer(uint amount, address from, address to) { require(msg.sender == bank_address); // We need to convey who the *owner* of the note may be, // which in this example I want to be *distinct* from the mapping key. // I want the owner to be bank_address, // but the previous syntax didn't allow me to express that. UTXO[2] notes = balances[from].get(2, sort, filter, { owner: bank_address });

uint input_amount = notes[0] + notes[1];
require(input_amount >= amount);

notes[0].remove();
notes[1].remove();

balances[from].insert(input_amount - amount, { owner: bank_address });

balances[to].insert(amount, { owner: bank_address });

}

Example 3

. Sometimes a UTXOSet might not be stored in a mapping (from which one could argue the owner. Here's an example contract where a charity is the owner of all deposits (donations), so there's only one balance

which doesn't need a mapping to be tracked. This is a withdraw

function, to make the example make sense:

UTXOSet balance; address charity_address.

// Not shown: a deposit function.

withdraw(uint amount, address to) { require(msg.sender == charity_address); // We need to convey that the owner will always be the charity_address. UTXO[2] notes = balance.get(2, sort, filter, { owner : charity_address });

uint input_amount = notes[0] + notes[1];
require(input_amount >= amount);

```
notes[0].remove();
notes[1].remove();

balance.insert(input_amount - amount, { owner : charity_address });

// Not shown: also expose the withdrawal amount via a public input, or something.

}
```