I asked a question on Slack the other day, but I think there's more to squeeze out of this discussion, so here's some background, followed by some new points/questions…

Original Q

The question I asked:

"

UTXO question, maybe for Zac:_

If a storage slot is used in multiple not-yet-nullified UTXOs (i.e. the 'unbounded array-based utxo representation'), the only way my brain can currently think to interpret that is:

- the 'actual' value of the storage slot can be deduced by treating the value

within each UTXO as a summand

, and summing

all of these summands.

To broaden my brain: are there any other example interpretations of an 'unbounded array-based utxo representation' of a storage slot?

"

Background

Zac wrote this nice proposal for UTXOs and Private State Semantics. Definitely read it before continuing.

Responses so far to the Q

From Zac:

The mental model in my head is that, for the purposes of storage slots, we treat UTXOs very much like Solidity mappings.

If a contract has a private storage variable that represents UTXOs (e.g. private UnboundedArrayvalues

, that variable is given a defined slot number (e.g. values.slot = 4

if its the 5th storage variable in the contract).

However each actual UTXO object has a unique slot number, which is the hash of the slot number and a given UTXO commitment. e.g. slot = Hash(values.slot, singleValueNote.commitment

.

This is similar to how mappings are treated in Solidity. e.g. imagine we have mapping(address => uint) values

and the contract writes a value values[msg.sender] = 10

, under the hood, the storage slot will be defined as keccak(msg.sender, values.slot)

and the evm code will do the equivalent of sstore(keccak(msg.sender, values.slot), 10)

.

In the EVM model, the mapping 'key' is hashed with the variable slot to get the object slot.

In the UTXO model, the UTXO commitment is the equivalent of the mapping 'key'.

To clarify a bit more - the mental model I have treats UTXOs as distinct objects with unique slot numbers - derived from a variable slot and the UTXO commitment.

At the protool level there is no concept of a 'value' represented across multiple UTXOs (e.g. automatic summing) - you need to code that up yourself as a contract writer if you want it.

(this touches on the doc I wrote about private state semantics - https://hackmd.io/np0KGkUWRbqQf7n7EWt9xg. The idea is that treat UTXOs as UTXOs and don't attempt to create protocol-level (or language-level) abstractions that pretend a set of UTXOs are a single variable. Instead we provide enough flexibility in the Noir++ language to allow devs to code up ways of

effectively manipulating UTXOs so they can create subroutines that perform UTXO summing and similar actions).

(TLDR: we do what Solidity does and we treat UTXO collections like mappings)

Continuing the discussion

Zac's mental model makes sense to me - I just have some questions.

Q1:

Consider a mapping to collections of UTXOs:

private mapping(address => UnboundedArray) collections;

I'll call each entry in the mapping a "collection of UTXOs".

As per Zac's comments, and as per Solidity, collections[mike_address]

would live at storage slot keccak(mike_address, collections.slot)

. It makes sense to ascribe a storage slot to a "collection of UTXOs".

But I'm not sure about the need to ascribe a "storage slot" to each

UTXO within a collection of UTXOs.

My mental model is that each UTXO in a collection of UTXOs should be interchangeable, or 'fungible' in some abstract way. It feels like the information contained in the value

field of a UTXO's preimage should be the only way to distinguish between the UTXOs at a particular 'proper' storage slot (rather than ascribing each UTXO in a collection a unique "storage slot").

I'd hypothesise that if someone needs each UTXO to have a unique 'storage slot', then they should probably be using an array

, struct

or a mapping

to achieve their goals.

Q2:

Defining the 'storage slot' of a particular UTXO commitment to be slot = hash(collections[mike_address].slot, commitment)

seems cyclic.

The storage "key" is being derived from the "value" being stored, which is unconventional, and feels problematic.

Also, usually states are accessed based on meaningful state variable names. But looking-up a state w.r.t. a 'commitment' isn't 'meaningful' in the same way. I'm not sure I see the need to differentiate between each UTXO in a particular collection, in terms of ascribing 'storage slots' to each of them.

Q3:

I'd like to see some concrete examples of other interpretations of a 'collection of UTXOs', to justify the extra complexity of allowing devs full control over UTXOs.

The only example interpretation I've seen of UTXOs is to represent a single value, when the UTXOs are summed. E.g. bitcoin, zcash, aztec.

Here's my attempt at trying to come up with examples to support the notion that an Unbounded Array of UTXOs can be used more generally than just 'summing values':

Spoiler alert… I wrote a framework below, and then actually came up with some examples! You can see the moment it clicked!

Hypotheses

- The only time a dev would need a private state variable to be 'split' across multiple not-yet-nullified UTXOs (a.k.a. a "collection of UTXOs"), is if they would like someone other than the owner

to insert a new UTXO into that collection, in a way that's parallelisable and doesn't cause collisions.

- A "collection of UTXOs" should only be used to 'combine' a (possibly infinite) collection of identical 'types'.
- Any two UTXOs in a "collection of UTXOs" should be 'combinable' into a single UTXO of the same type. E.g. nullify two and create one to represent the 'combination' of them both. If a collection of 'things' doesn't meet this property, then the 'things' must be distinct in some way, and therefore I'd suggest each 'thing' should inhabit its own 'proper' storage slot, and therefore a "collection of UTXOs" isn't a suitable type for those 'things'.
- The 'proper' storage slot of a variable which points-to such a "collection of UTXOs" would always represent the 'statistic' or 'combination' of that collection.
- I'd suggest for many cases, we wouldn't need to use a "collection of UTXOs": a different container, such as a struct

, array

, or mapping

could be used instead. We could either entirely compress such a container into a single UTXO, or split each element of such a container across multiple single UTXOs where each UTXO is located at a distinct, 'proper' storage slot. The latter is different from a "collection of UTXOs". * Even if we want someone other than the owner

to be able to insert a thing into a set, a mapping could often be used instead - e.g. if each element in the mapping is mapped-to from a user's address.

- Note: an array seemingly doesn't fulfil the requirement to be able to add to a collection 'in parallel and without collisions'. But a public function could determine the ordering of pushing partial-commitments of private values to an array, by 'completing the commitments' with a storage slot.

- Even if we want someone other than the owner

to be able to insert a thing into a set, a mapping could often be used instead - e.g. if each element in the mapping is mapped-to from a user's address.

- Note: an array seemingly doesn't fulfil the requirement to be able to add to a collection 'in parallel and without collisions'. But a public function could determine the ordering of pushing partial-commitments of private values to an array, by 'completing the commitments' with a storage slot.

So what examples can I come up with, other than summing values, for which an unbounded array of utxos would be useful…

:

I list some below [and suggest in square brackets why an unbounded collection of UTXOs might not be necessary].

- A sum of numeric types in a collection, to represent a single summed value.
- A count of items in a collection
- [A 'count of items' doesn't fulfil the requirement that two items can be combined into a single UTXO, since then the count would reduce from 2 to 1].
- [A 'count of items' doesn't fulfil the requirement that two items can be combined into a single UTXO, since then the count would reduce from 2 to 1].
- The weighted average of values in a collection.
- [Maybe this fits into a 'collection of UTXOs' model if each value itself represents a 'weighted average' (since then two UTXOs could be combined into one, and the resulting UTXO would still represent a 'weighted average'), but that's quite abstract, and I can't think of a real-world use for this, and you could probably perform this calculation on single UTXOs which each inhabit a distinct storage slot of a mapping, for example].
- [Maybe this fits into a 'collection of UTXOs' model if each value itself represents a 'weighted average' (since then two UTXOs could be combined into one, and the resulting UTXO would still represent a 'weighted average'), but that's quite abstract, and I can't think of a real-world use for this, and you could probably perform this calculation on single UTXOs which each inhabit a distinct storage slot of a mapping, for example].
- The median / standard deviation / variance of values in a collection
- [as above]
- [as above]

- A collection of strings, interpreted as a single string or sentence.
- Possibly…
- A geometric or topological property of items in a collection, e.g. area or volume of a group of objects.
- Hmm… this has led me to something more general…
- Hmm… this has led me to something more general…
- A collection of private structs of the same type, where the struct describes something fungible.
- E.g. an inventory of apples. Apples are pretty interchangeable. No one cares about keeping track of each.
- Bob adds an item to Alice's collection: { type: apple, quantity: 300, weight: 1 kg, average_weight: 3.33 g }
- Charlie adds an item to Alice's collection: { type: apple, quantity: 100, weight: 0.2 kg, average_weight: 2 g }
- Now… these can be combined into a single UTXO of the same type, based on custom

logic which is not

simply summation: { type: apple, quantity: 400, weight: 1.2 kg, average_weight: 3 g }

- Oooh, interesting!!!

- Bob adds an item to Alice's collection: { type: apple, quantity: 300, weight: 1 kg, average_weight: 3.33 g }
- Charlie adds an item to Alice's collection: { type: apple, quantity: 100, weight: 0.2 kg, average_weight: 2 g }
- Now… these can be combined into a single UTXO of the same type, based on custom

logic which is not

simply summation: { type: apple, quantity: 400, weight: 1.2 kg, average_weight: 3 g }

- Oooh, interesting!!!
- E.g. an inventory of apples. Apples are pretty interchangeable. No one cares about keeping track of each.
- Bob adds an item to Alice's collection: { type: apple, quantity: 300, weight: 1 kg, average_weight: 3.33 g }
- Charlie adds an item to Alice's collection: { type: apple, quantity: 100, weight: 0.2 kg, average_weight: 2 g }
- Now… these can be combined into a single UTXO of the same type, based on custom

logic which is not

simply summation: { type: apple, quantity: 400, weight: 1.2 kg, average_weight: 3 g }

- Oooh, interesting!!!
- Bob adds an item to Alice's collection: { type: apple, quantity: 300, weight: 1 kg, average_weight: 3.33 g }
- Charlie adds an item to Alice's collection: { type: apple, quantity: 100, weight: 0.2 kg, average_weight: 2 g }
- Now… these can be combined into a single UTXO of the same type, based on custom

logic which is not

simply summation: { type: apple, quantity: 400, weight: 1.2 kg, average_weight: 3 g }

- Oooh, interesting!!!
- Summing a collection of complex numbers
- Summing a collection of polynomials
- Multiplying a collection of numbers, such as annualised compound interest rates.
- Combining a collection of sets of pixels to construct an image. (E.g. contributing to privately to some (likely very messy) artwork or collage)!
- A collection of functions, being combined into a single function?

- A collection of snarks, being rolled-up into a single snark?

Q4: Any other nice examples?

Q5: Is there a nice generalised description for what a collection of UTXOs should be used for?

Q6: Can any of these examples be done without

a collection of UTXOs? Or rather, which of these examples can only be done

with a collection of UTXOs?