Storage Slots

In Aztec, private data and public data are stored in two trees; a public data tree and a note hashes tree.

These trees have in common that they store state forall accounts on the Aztec network directly as leaves. This is different from Ethereum, where a state trie contains smaller tries that hold the individual accounts' storage.

It also means that we need to be careful about how we allocate storage to ensure that they don't collide! We say that storage should besiloed to its contract. The exact way of siloing differs a little for public and private storage. Which we will see in the following sections.

Public State Slots

As mentioned inState Model , Aztec public state behaves similarly to public state on Ethereum from the point of view of the developer. Behind the scenes however, the storage is managed differently. As mentioned, public state has just one large sparse tree in Aztec - so we silo slots of public data by hashing it together with its contract address.

The mental model is that we have a key-value store, where the siloed slot is the key, and the value is the data stored in that slot. You can think of thereal_storage_slot identifying its position in the tree, and thelogical_storage_slot identifying the position in the contract storage.

real_storage_slot

H (contract_address , logical_storage_slot) The siloing is performed by theKernel circuits .

For structs and arrays, we are logically using a similar storage slot computation to ethereum, e.g., as a struct with 3 fields would be stored in 3 consecutive slots. However, because the "actual" storage slot is computed as a hash of the contract address and the logical storage slot, the actual storage slot is not consecutive.

Private State Slots - Slots aren't real

Private storage is a different beast. As you might remember fromHybrid State Model , private state is stored in encrypted logs and the corresponding private state commitments in append-only tree where each leaf is a commitment. Being append-only, means that leaves are never updated or deleted; instead a nullifier is emitted to signify that some note is no longer valid. A major reason we used this tree, is that lookups at a specific storage slot would leak information in the context of private state. If you could look up a specific address balance just by looking at the storage slot, even if encrypted you would be able to see it changing! That is not good privacy.

Following this, the storage slot as we know it doesn't really exist. The leaves of the note hashes tree are just commitments to content (think of it as a hash of its content).

Nevertheless, the concept of a storage slot is very useful when writing applications, since it allows us to reason about distinct and disjoint pieces of data. For example we can say that the balance of an account is stored in a specific slot and that the balance of another account is stored in another slot with the total supply stored in some third slot. By making sure that these slots are disjoint, we can be sure that the balances are not mixed up and that someone cannot use the total supply as their balance.

But how?

If we include the storage slot, as part of the note whose commitment is stored in the note hashes tree, we canlogically link all the notes that make up the storage slot. For the case of a balance, we can say that the balance is the sum of all the notes that have the same storage slot - in the same way that your physicalbalance might be the sum of all the notes in your wallet.

Similarly to how we siloed the public storage slots, we can silo our private storage by hashing the logical storage slot together with the note content.

note_hash

H (logical_storage_slot , note_content_hash) ; This siloing (there will be more) is done in the application circuit, since it is not necessary for security of the network (but only the application).

info The private variable wrappersPrivateSet andPrivateMutable in Aztec.nr include thelogical_storage_slot in the commitments they compute, to make it easier for developers to write contracts without having to think about how to correctly handle storage slots. When reading the values for these notes, the application circuit can then constrain the values to only read notes with a specific logical storage slot.

To ensure that one contract cannot insert storage that other contracts would believe is theirs, we do a second siloing by hashing thecommitment with the contract address.

siloed_note_hash

H (contract_address , note_hash) ; By doing this address-siloing at the kernel circuit weforce the inserted commitments to include and not lie about thecontract_address .

info To ensure that nullifiers don't collide across contracts we also force this contract siloing at the kernel level. For an example of this seedeveloper documentation storage slots . Edit this page

Previous Indexed Merkle Tree Next Accounts