Notes in Aztec.nr

Most prominent blockchain networks don't have privacy at the protocol level. Aztec contracts can define public and private functions, that can read/write public and private state.

To be clear, "private" here is referring to the opposite of how things work on a public blockchain network, not the conventional syntax for visibility within code.

For private state we need encryption and techniques to hide information about state changes. For private functions, we need local execution and proof of correct execution.

Some context

- Public functions and storage work much like other blockchains in terms of having dedicated storage slots and being publicly visible
- Private functions are executed locally with proofs generated for sound execution, and commitments to private variable updates are stored using append-only trees
- "Note" types are part of Aztec.nr, a framework that facilitates use of Aztec's different storage trees to achieve things such as private variables

This page will focus on how private variables are implemented with Notes and storage trees.

Side-note about execution

Under the hood, the Aztec protocol handles some important details around public and private function calls. Calls between them are asynchronous due to different execution contexts (local execution vs. node execution). A detailed explanation of the transaction lifecycle can be foundhere .

Private state variables in Aztec

State variables in an Aztec contract are defined inside a struct specifically namedStorage , and must satisfy theNote Interface and contain aNote header .

The Note header struct contains the contract address which the value is effectively siloed to, a nonce to ensure unique Note hashes, and a storage "slot" (or ID) to associate multiple notes.

A couple of things to unpack here:

Storage "slot"

Storage slots are more literal for public storage, a place where a value is stored. For private storage, a storage slot is logical (morehere).

Silos

The address of the contract is included in a Note's data to ensure that different contracts don't arrive at the same hash with an identical variable. This is handled in the protocol's execution.

Note types

There is more than one Note type, such as theSet type is used for private variables. There are alsoSingleton andImmutableSingleton types.

Furthermore, notes can be completely custom types, storing any value or set of values that are desired by an application.

Initialization

Private state variables are stored locally when the contract is created. Depending on the application, values may be privately shared by the creator with others via encrypted logs onchain. A hash of a note is stored in the append-only note hash tree so as to prove existence of the current state of the note in a privacy preserving way.

Note Hash Tree

By virtue of being append only, notes are not edited. If two transactions amend a private value, multiple notes will be inserted into the tree. The header will contain the same logical storage slot.

Reading Notes

info * Only those with appropriate keys/information will be able to successfully read private values that they have permission to * Notes can be read outside of a transaction or "off-chain" with no changes to data structures on-chain When a note is read in a transaction, a subsequent read from another transaction of the same note would reveal a link between the two. So to preserve privacy, notes that are read in a transaction are said to be "consumed" (defined below), and new note(s) are then created with a unique hash (includes transaction identifier).

With typeSet , a private variable's value is interpreted as the sum of values of notes with the same logical storage slot.

Consuming, deleting, or otherwise "nullifying" a note is NOT done by deleting the Note hash, this would leak information, but rather by creating a nullifier deterministically linked to the value. This nullifier is inserted into another storage tree, aptly named the nullifier tree.

When interpreting a value, the local private execution checks that its notes (of the corresponding storage slot/ID) have not been nullified.

Updating

note Only those with appropriate keys/information will be able to successfully nullify a value that they have permission to. To update a value, its previous note hash(es) are nullified. The new note value is updated in the PXE, and the updated note hash inserted into the note hash tree.

Supplementary components

Some optional background resources on notes can be found here:

- High level network architecture
- , specifically the Private Execution Environment
- Transaction lifecycle (simple diagram)
- Public and Private state

Notes touch several core components of the protocol, but we will focus on a the essentials first.

Some code context

The way Aztec benefits from the Noir language is via three important components:

- Aztec-nr
-
    - a Noir framework enabling contracts on Aztec, written in Noir. Includes useful Note implementations
- noir contracts
-
    - example Aztec contracts
- noir-protocol-circuits
-
    - a crate containing essential circuits for the protocol (public circuits and private wrappers)

A lot of what we will look at will be inaztec-nr/aztec/src/note , specifically the lifecycle and note interface. Looking at the noir circuits in these components, you will see references to the distinction between public/private execution and state.

Lifecycle functions

Inside thelifecycle circuits we see the functions to create and destroy a note, implemented as insertions of note hashes and nullifiers respectively. This is helpful for regular private variables.

We also see a function to create a note hash from the public context, a way of creating a private variable from a public call (run in the sequencer). This could be used in application contracts to give private digital assets to users.

Note Interface functions

To see anote_interface implementation, we will look at a simpleValueNote .

The interface is required to work within an Aztec contract's storage, and a ValueNote is a specific type of note to hold a number (as aField).

Computing hashes and nullifiers

A few key functions in the note interface are around computing the note hash and nullifier, with logic to get/use secret keys from the private context.

In the ValueNote implementation you'll notice that it uses thepedersen_hash function. This is currently required by the protocol, but may be updated to another hashing function, like poseidon.

As a convenience, the outernote/utils.nr contains implementations of functions that may be needed in Aztec contracts, for example computing note hashes.

Serialization and deserialization

Serialization/deserialization of content is used to convert between the Note's variables and a generic array of Field elements. The Field type is understood and used by lower level crypographic libraries. This is analogous to the encoding/decoding between variables and bytes in solidity.

For example in ValueNote, theserialize_content function simply returns: the value, owner address (as a field) and the note randomness; as an array of Field elements.

Value as a sum of Notes

We recall that multiple notes are associated with a "slot" (or ID), and so the value of a numerical note (like ValueNote) is the sum of each note's value. The helper function inbalance_utils implements this logic taking aSet ofValueNotes .

A couple of things worth clarifying:

- ASet
- takes a Generic type, specified here asValueNote
- , but can be anyNote
- type (for all notes in the set)
- ASet
- of notes also specifiesthe
- slot of all Notes that it holds

Example - Notes in action

The Aztec.nr framework includes examples of high-level stateseasy_private_uint for use in contracts.

The struct (EasyPrivateUint) contains a Context, Set of ValueNotes, and storage_slot (used when setting the Set).

Notice how theadd function shows the simplicity of appending a new note to all existing ones. On the other hand,sub (subtraction), needs to first add up all existing values (consuming them in the process), and then insert a single new value of the difference between the sum and parameter.

Apply

Try theToken tutorial to see what notes can achieve. In this section you will also find other tutorials using notes in different ways.

Further reading

- Storage Trees
- Proof of prior notes
-
    - public/private reading of public/private proof of state (public or private)

If you're curious about any of the following related topics, search the documentation for...

- Private and public contexts
- Encryption keys and events
- Oracle's role in using notes
- Value Serialization/Deserialization

References

- "Stable" state variable
- Code: Aztec-Patterns Edit this page

Previous How to define contract storage Next Storage slots