How to use the Arhive Tree

The Aztec Protocol uses an append-only Merkle tree to store hashes of the headers of all previous blocks in the chain as its leaves. This is known as an archive tree. You can learn more about how it works in theconcepts section .

View the History lib referencehere .

History library

The history library allows you to prove any of the following at a given block height before the current height:

- Note inclusion
- Nullifier inclusion
- Note validity
- Existence of public value
- Contract inclusion

Using this library, you can check that specific notes or nullifiers were part of Aztec network state at specific blocks. This can be useful for things such as:

- Verifying a minimum timestamp from a private context
- Checking eligibility based on historical events (e.g. for an airdrop by proving that you owned a note)
- Verifying historic ownership / relinquishing of assets
- Proving existence of a value in public data tree at a given contract slot
- Proving that a contract was deployed in a given block with some parameters

In this guide you will learn how to

- Prove a note was included in a specified block
- Create a nullifier and prove it was not included in a specified block

For a more extensive reference, go tothe reference page .

1. Import thehistory

library fromaztec

aztec :: { use

dep :: aztec :: history :: { contract_inclusion :: { prove_contract_deployment_at , prove_contract_non_deployment_at , prove_contract_initialization_at , prove_contract_non_initialization_at } , note_inclusion :: { prove_note_inclusion , prove_note_inclusion_at } , note_validity :: { prove_note_validity , prove_note_validity_at } , nullifier_inclusion :: { prove_nullifier_inclusion , prove_nullifier_inclusion_at , prove_note_is_nullified , prove_note_is_nullified_at } , nullifier_non_inclusion :: { prove_note_not_nullified , prove_note_not_nullified_at } , public_value_inclusion :: { prove_public_value_inclusion , prove_public_value_inclusion_at } } ; } This imports all functions from thehistory library. You should only import the functions you will use in your contract.

2. Create a note to prove inclusion of

In general you will likely have the note you want to prove inclusion of. But if you are just experimenting you can create a note with a function like below:

create_note // Creates a value note owned byowner.

[aztec(private)]

fn

create_note (owner :

AztecAddress , value :

Field)

{ let owner_private_values = storage . private_values . at (owner) ; let mut note =

ValueNote :: new (value , owner) ; owner_private_values . insert (& mut note ,

true) ; } Source code: noir-projects/noir-contracts/contracts/inclusion_proofs_contract/src/main.nr#L41-L49

3. Get the note from the PXE

Retrieve the note from the user's PXE.

get_note_from_pxe // 1) Get the note from PXE. let private_values = storage . private_values . at (owner) ; let

mut options =

NoteGetterOptions :: new ( ) . select (1 , owner . to_field ( ) ,

Option :: none ( )) . set_limit (1) ; if

(nullified)

{ options = options . set_status (NoteStatus . ACTIVE_OR_NULLIFIED) ; } let notes = private_values . get_notes (options) ; let maybe_note = notes [0] ; Source code: noir-projects/noir-contracts/contracts/inclusion_proofs_contract/src/main.nr#L58-L67 In this example, the user's notes are stored in a map calledprivate_values . We retrieve this map, then select 1 note from it with the value of1 .

4. Prove that a note was included in a specified block

To prove that a note existed in a specified block, callprove_note_inclusion_at as shown in this example:

prove_note_inclusion prove_note_inclusion_at (maybe_note . unwrap_unchecked ( ) , block_number , context) Source code: noir-projects/noir-contracts/contracts/inclusion_proofs_contract/src/main.nr#L71-L73 This function takes in 3 arguments:

1. The note (maybe_note.unwrap_unchecked()
2.). Here,unwrap_unchecked()
3. returns the inner value without assertingself.is_some()
4. The block number
5. Private context

This will only prove the note existed at the specific block number, not whether or not the note has been nullified. You can prove that a note existed and had not been nullified in a specified block by usingprove_note_validity_at which takes the same arguments:

prove_note_validity prove_note_validity_at (note , block_number ,

& mut context) ; Source code: noir-projects/noir-contracts/contracts/inclusion_proofs_contract/src/main.nr#L142-L144

5. Create a nullifier to prove inclusion of

You can easily nullify a note like so:

nullify_note

[aztec(private)]

fn

nullify_note (owner :

AztecAddress)

{ let private_values = storage . private_values . at (owner) ; let options =

NoteGetterOptions :: new ( ) . select (1 , owner . to_field ( ) ,

Option :: none ( )) . set_limit (1) ; let notes = private_values . get_notes (options) ; let note = notes [0] . unwrap ( ) ;

private_values . remove (note) ; }Source code: noir-projects/noir-contracts/contracts/inclusion_proofs_contract/src/main.nr#L150-L160 This function gets a note from the PXE like we did instep 3 and nullifies it withremove() .

You can then compute this nullifier withnote.compute_nullifier(&mut context) .

6. Prove that a nullifier was included in a specified block

Callprove_nullifier_inclusion_at like so:

prove_nullifier_inclusion prove_nullifier_inclusion_at (nullifier , block_number , context) ;Source code: noir-projects/noir-contracts/contracts/inclusion_proofs_contract/src/main.nr#L172-L174 This takes three arguments:

1. The nullifier
2. Block number
3. Private context

You can also prove that a nullifier was not included in a specified block by usingprove_nullifier_not_included_at which takes the same arguments.

Prove contract inclusion, public value inclusion, and use current state in lookups

To see what else you can do with thehistory library, check out thereference . Edit this page