Public, Private, and Unconstrained Functions

This page explains the three types of functions that exist on Aztec - public, private, and unconstrained. For a deeper dive into how these functions work under the hood, check out theInner Workings page.

Public

Functions

A public function is executed by the sequencer and has access to a state model that is very similar to that of the EVM and Ethereum. Even though they work in an EVM-like model for public transactions, they are able to write data into private storage that can be consumed later by a private function.

note All data inserted into private storage from a public function will be publicly viewable (not private). To create a public function you can annotate it with the#[aztec(public)] attribute. This will make thepublic context available within the function's execution scope.

set_minter

[aztec(public)]

fn

set_minter (minter :

AztecAddress , approve :

bool)

{ assert (storage . admin . read ( ) . eq (context . msg_sender ( )) ,

"caller is not admin") ; storage . minters . at (minter) . write (approve) ;Source code: noir-projects/noir-contracts/contracts/token_contract/src/main.nr#L125-L135

Private

Functions

A private function operates on private information, and is executed by the user. Annotate the function with the# [aztec(private)] attribute to tell the compiler it's a private function. This will make theprivate context available within the function's execution scope.

redeem_shield

[aztec(private)]

fn

redeem_shield (to :

AztecAddress , amount :

Field , secret :

Field)

{ let pending_shields = storage . pending_shields ; let secret_hash =

compute_secret_hash (secret) ; // Get 1 note (set_limit(1)) which has amount stored in field with index 0 (select(0, amount)) and secret_hash // stored in field with index 1 (select(1, secret_hash)). let options =

NoteGetterOptions :: new ( ) . select (0 , amount ,

Option :: none ( )) . select (1 , secret_hash ,

Option :: none ( )) . set_limit (1) ; let notes = pending_shields . get_notes (options) ; let note = notes [0] . unwrap_unchecked ( ) ; // Remove the note from the pending shields set pending_shields . remove (note) ;

// Add the token note to user's balances set storage . balances . add (to ,

U128 :: from_integer (amount)) ; }Source code: noir-projects/noir-contracts/contracts/token_contract/src/main.nr#L245-L261

unconstrained

functions

Unconstrained functions are an underlying part of Noir - a deeper explanation can be foundhere . In short, they are functions which are not directly constrained and therefore should be seen as un-trusted. That they are un-trusted means that the developer must make sure to constrain their return values when used. Note: Calling an unconstrained function from a private function means that you are injecting unconstrained values.

Beyond using them inside your other functions, they are convenient for providing an interface that reads storage, applies logic and returns values to a UI or test. Below is a snippet from exposing thebalance_of_private function from a token implementation, which allows a user to easily read their balance, similar to thebalanceOf function in the ERC20 standard.

balance_of_private unconstrained fn

balance_of_private (owner :

AztecAddress)

->

pub

Field

{ storage . balances . balance_of (owner) . to_integer ( ) }Source code: noir-projects/noir-contracts/contracts/token_contract/src/main.nr#L355-L359 info Note, that unconstrained functions can have access to both public and private data when executed on the user's device. This is possible since it is not actually part of the circuits that are executed in contract execution. Edit this page

Previous Function Context Next Visibility