Writing a token contract in Aztec.nr

In this tutorial we will go through writing an L2 native token contract for the Aztec Network, using the Aztec.nr contract libraries. It is recommended that you go through thethe introduction to contracts andsetup instructions section before this tutorial to gain some familiarity with writing Aztec smart contracts.

This tutorial is intended to help you get familiar with the Aztec.nr library, Aztec contract syntax and some of the underlying structure of the Aztec network.

In this tutorial you will learn how to:

- Write public functions that update public state
- Write private functions that update private state
- Implement access control on public and private functions
- Handle math operations safely
- Handle different private note types
- Pass data between private and public state

We are going to start with a blank project and fill in the token contract source code defined on Github here , and explain what is being added as we go.

Requirements

You will need to haveaztec-nargo installed in order to compile Aztec.nr contracts. See thesandbox reference for installation instructions.

You should also install theNoir Language Support extension for VS Code.

Check theDev Tools section of the awesome-noir repo for language support for additional editors (Vim, emacs, tree-sitter, etc).

Project setup

Create a new directory calledtoken_contract_tutorial

mkdir token_contract_tutorial inside that directory, create acontracts folder for the Aztec contracts.

cd token_contract_tutorial &&

mkdir contracts &&

cd contracts Create the following file structure

. └─── contracts ├─── Nargo.toml └─── src └─── main.nr Add the following content to Nargo.toml file:

[package] name

=

"token_contract" authors

=

[""] compiler_version

=

">=0.18.0" type

=

"contract"

[dependencies] aztec

=

{

git

"https://github.com/AztecProtocol/aztec-packages/" ,

tag

"aztec-packages-v0.28.1" ,

directory

"noir-projects/aztec-nr/aztec"

} authwit = {

git

"https://github.com/AztecProtocol/aztec-packages/" ,

tag

"aztec-packages-v0.28.1" ,

directory

"noir-projects/aztec-nr/authwit" } compressed_string

=

{ git = "https://github.com/AztecProtocol/aztec-packages/" ,

tag

"aztec-packages-v0.28.1" ,

directory

"noir-projects/aztec-nr/compressed-string" }

Contract Interface

contract Token

{

[aztec(private)]

fn

constructor ( )

{ }

[aztec(public)]

fn

set_admin (new_admin :

AztecAddress)

{ }

[aztec(public)]

fn

set_minter (minter :

AztecAddress , approve :

bool)

{ }

[aztec(public)]

fn

mint_public (to :

AztecAddress , amount :

Field)

->

Field

{ }

[aztec(public)]

fn

mint_private (amount :

Field , secret_hash :

Field)

->

Field

{ }

[aztec(public)]

fn

shield (from :

AztecAddress , amount :

Field , secret_hash :

Field , nonce :

Field)

->

Field

{ }

[aztec(public)]

fn

transfer_public (from :

AztecAddress , to :

AztecAddress , amount :

Field , nonce :

Field)

->

Field

{ }

[aztec(public)]

fn

burn_public (from :

AztecAddress , amount :

Field , nonce :

Field)

->

Field

{ }

// Private functions

[aztec(private)]

fn

redeem_shield (to :

AztecAddress , amount :

Field , secret :

Field)

->

Field

{ }

[aztec(private)]

fn

unshield (from :

AztecAddress , to :

AztecAddress , amount :

Field , nonce :

Field)

->

Field

{ }

[aztec(private)]

fn

transfer (from :

AztecAddress , to :

AztecAddress , amount :

Field , nonce :

Field)

->

Field

{ }

[aztec(private)]

fn

burn (from :

AztecAddress , amount :

Field , nonce :

Field)

->

Field

{ }

// Internal functions below

// Will be internal in the future

[aztec(public)]

fn

_initialize (new_admin :

AztecAddress)

{ }

[aztec(public)]

internal fn

_increase_public_balance (to :

AztecAddress , amount :

Field)

{ }

[aztec(public)]

internal fn

_reduce_total_supply (amount :

Field)

{ }

// Unconstrained functions (read only)

unconstrained fn

admin ( )

->

Field

{ }

unconstrained fn

is_minter (minter :

AztecAddress)

->

bool

{ }

unconstrained fn

total_supply ( )

->

Field

{ }

unconstrained fn

balance_of_private (owner :

AztecAddress)

->

Field

{ }

unconstrained fn

balance_of_public (owner :

AztecAddress)

->

Field

{ } } This specifies the interface of theToken contract. Go ahead and copy and paste this interface into yourmain.nr file.

Before we through the interface and implement each function, let's review the functions to get a sense of what the contract does.

Constructor interface

There is aconstructor function that will be executed once, when the contract is deployed, similar to the constructor function in Solidity. This is marked private, so the function logic will not be transparent. To execute public function logic in the constructor, this function will call_initialize (marked internal, more detail below).

Public functions

These are functions that have transparent logic, will execute in a publicly verifiable context and can update public storage.

- set_admin
- enables the admin to be updated
- set_minter
- enables accounts to be added / removed from the approved minter list
- mint_public
- enables tokens to be minted to the public balance of an account
- mint_private
- enables tokens to be minted to the private balance of an account (with some caveats we will dig into)
- shield
- enables tokens to be moved from a public balance to a private balance, not necessarily the same account (step 1 of a 2 step process)
- transfer_public
- enables users to transfer tokens from one account's public balance to another account's public balance
- burn_public
- enables users to burn tokens

Private functions

These are functions that have private logic and will be executed on user devices to maintain privacy. The only data that is submitted to the network is a proof of correct execution, new datacommitments andnullifiers , so users will not reveal which contract they are interacting with or which function they are executing. The only information that will be revealed publicly is that someone executed a private transaction on Aztec.

- redeem_shield
- enables accounts to claim tokens that have been made private viamint_private
- orshield
- by providing the secret
- unshield
- enables an account to send tokens from their private balance to any other account's public balance
- transfer
- enables an account to send tokens from their private balance to another account's private balance
- burn
- enables tokens to be burned privately

Internal functions

Internal functions are functions that can only be called by the contract itself. These can be used when the contract needs to call one of it's public functions from one of it's private functions.

- _initialize
- is a way to call a public function from theconstructor
- (which is a private function)
- _increase_public_balance
- increases the public balance of an account whenunshield
- is called
- _reduce_total_supply
- reduces the total supply of tokens when a token is privately burned

To clarify, let's review some details of the Aztec transaction lifecycle, particularly how a transaction "moves through" these contexts.

Execution contexts

Transactions are initiated in the private context, then move to the L2 public context, then to the Ethereum L1 context.

Step 1. Private Execution

Users provide inputs and execute locally on a their device for privacy reasons. Outputs of the private execution are commitment and nullifier updates, a proof of correct execution and any return data to pass to the public execution context.

Step 2. Public Execution

This happens remotely by the sequencer, which takes inputs from the private execution and runs the public code in the network virtual machine, similar to any other public blockchain.

Step 3. Ethereum execution

Aztec transactions can pass data to Ethereum contracts through the rollup via the outbox. The data can consumed by Ethereum contracts at a later time, but this is not part of the transaction flow for an Aztec transaction. The technical details of this are beyond the scope of this tutorial, but we will cover them in an upcoming piece.

Unconstrained functions

Unconstrained functions can be thought of as view functions from Solidity--they only return information from the contract storage or compute and return data without modifying contract storage.

Contract dependencies

Before we can implement the functions, we need set up the contract storage, and before we do that we need to import the appropriate dependencies.

Copy required files We will be going over the code inmain.nrhere . If you are following along and want to compilemain.nr yourself, you need to add the other files in the directory as they contain imports that are used inmain.nr . Just below the contract definition, add the following imports:

imports mod

types ;

// Minimal token implementation that supports AuthWit accounts. // The auth message follows a similar pattern to the cross-chain message and includes a designated caller. // The designated caller is ALWAYS used here, and not based on a flag as cross-chain. // message hash = H([caller, contract, selector, ...args]) // To be read as caller calls function at contract defined by selector with args // Including a nonce in the message hash ensures that the message can only be used once.

contract Token

{ // Libs

use

dep :: compressed_string :: FieldCompressedString ;

use

dep :: aztec :: prelude :: { NoteGetterOptions ,

NoteHeader ,

Map ,

PublicMutable ,

SharedImmutable ,

PrivateSet , FunctionSelector ,

AztecAddress } ; use

dep :: aztec :: hash :: compute_secret_hash ;

use

dep :: authwit :: { auth :: { assert_current_call_valid_authwit , assert_current_call_valid_authwit_public } } ;

use

crate :: types :: { transparent_note :: TransparentNote ,

token_note :: { TokenNote ,

TOKEN_NOTE_LEN } ,

balances_map :: BalancesMap } ;Source code: noir-projects/noir-contracts/contracts/token_contract/src/main.nr#L2-L28 We are importing the Option type, items from thevalue_note library to help manage private value storage, note utilities, context (for managing private and public execution contexts),state_vars for helping manage state,types for data manipulation andoracle for help passing data from the private to public execution context. We also import theauth library to handle token authorizations fromAccount Contracts . Check out the Account Contract with AuthWitnesshere .

For more detail on execution contexts, seeContract Communication .

Types files

We are also importing types from atypes.nr file, which imports types from thetypes folder. You can view therhere .

The main thing to note from this types folder is theTransparentNote definition. This defines how the contract moves value from the public domain into the private domain. It is similar to thevalue_note that we imported, but with some modifications namely, instead of a definedowner , it allows anyone that can produce the pre-image to the storedsecret_hash to spend the note.

Note on private state

Private state in Aztec is allUTXOs under the hood. Handling UTXOs is largely abstracted away from developers, but there are some unique things for developers to be aware of when creating and managing private state in an Aztec contract. SeeState Variables to learn more about public and private state in Aztec.

Contract Storage

Now that we have dependencies imported into our contract we can define the storage for the contract.

Below the dependencies, paste the following Storage struct:

storage_struct struct

Storage

{ admin :

PublicMutable < AztecAddress

, minters :

Map < AztecAddress ,

PublicMutable < bool

, balances :

BalancesMap < TokenNote

, total_supply :

PublicMutable < U128

, pending_shields :

PrivateSet < TransparentNote

, public_balances :

Map < AztecAddress ,

PublicMutable < U128

, symbol :

SharedImmutable < FieldCompressedString

, name :

SharedImmutable < FieldCompressedString

, decimals :

SharedImmutable < u8

, } Source code: noir-projects/noir-contracts/contracts/token_contract/src/main.nr#L30-L52 Reading through the storage variables:

- admin
- an Aztec address stored in public state.
- minters
- is a mapping of Aztec addresses in public state. This will store whether an account is an approved minter on the contract.
- balances
- is a mapping of private balances. Private balances are stored in aPrivateSet
- ofValueNote
- s. The balance is the sum of all of an account'sValueNote
- s.
- total_supply
- is an unsigned integer (max 128 bit value) stored in public state and represents the total number of tokens minted.
- pending_shields
- is aPrivateSet
- ofTransparentNote
- s stored in private state. What is stored publicly is a set of commitments toTransparentNote
- s.
- public_balances
- is a mapping of Aztec addresses in public state and represents the publicly viewable balances of accounts.

You can read more about ithere .

Functions

Copy and paste the body of each function into the appropriate place in your project if you are following along.

Constructor

This function sets the creator of the contract (passed asmsg_sender from the constructor) as the admin and makes them a minter, and sets name, symbol, and decimals.

constructor

[aztec(public)]

[aztec(initializer)]

fn

constructor (admin :

AztecAddress , name :

str < 31

, symbol :

str < 31

, decimals :

u8)

{ assert (! admin . is_zero ( ) ,

"invalid admin") ; storage . admin . write (admin) ; storage . minters . at (admin) . write (true) ; storage . name . initialize (FieldCompressedString :: from_string (name)) ; storage . symbol . initialize (FieldCompressedString :: from_string (symbol)) ; storage . decimals . initialize (decimals) ; } Source code: noir-projects/noir-contracts/contracts/token_contract/src/main.nr#L54-L67

Public function implementations

Public functions are declared with the#[aztec(public)] macro above the function name like so:

set_admin

[aztec(public)]

fn

set_admin (new_admin :

AztecAddress)

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

"caller is not admin") ; storage . admin . write (new_admin) ; }Source code: noir-projects/noir-contracts/contracts/token_contract/src/main.nr#L69-L77 As described in the execution contexts section above , public function logic and transaction information is transparent to the world. Public functions update public state, but can be used to prepare data to be used in a private context, as we will go over below (e.g. see the shield function).

Storage is referenced asstorage.variable .

set_admin

After storage is initialized, the contract checks that themsg_sender is theadmin . If not, the transaction will fail. If it is, thenew_admin is saved as theadmin .

set_admin

[aztec(public)]

fn

set_admin (new_admin :

AztecAddress)

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

"caller is not admin") ; storage . admin . write (new_admin) ; }Source code: noir-projects/noir-contracts/contracts/token_contract/src/main.nr#L69-L77

set_minter

This function allows theadmin to add or a remove aminter from the publicminters mapping. It checks thatmsg_sender is theadmin and finally adds theminter to theminters mapping.

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

mint_public

This function allows an account approved in the publicminters mapping to create new public tokens owned by the providedto address.

First, storage is initialized. Then the function checks that themsg_sender is approved to mint in theminters mapping. If it is, a newU128 value is created of theamount provided. The function reads the recipients public balance and then adds the amount to mint, saving the output asnew_balance , then reads to total supply and adds the amount to mint, saving the output assupply .new_balance andsupply are then written to storage.

The function returns 1 to indicate successful execution.

mint_public

[aztec(public)]

fn

mint_public (to :

AztecAddress , amount :

Field)

{ assert (storage . minters . at (context . msg_sender ( )) . read ( ) ,

"caller is not minter") ; let amount =

U128 :: from_integer (amount) ; let new_balance = storage . public_balances . at (to) . read ( ) . add (amount) ; let supply = storage . total_supply . read ( ) . add (amount) ;

storage . public_balances . at (to) . write (new_balance) ; storage . total_supply . write (supply) ; } Source code: noir-projects/noir-contracts/contracts/token_contract/src/main.nr#L137-L150

mint_private

This public function allows an account approved in the publicminters mapping to create new private tokens that can be claimed by anyone that has the pre-image to thesecret_hash .

First, public storage is initialized. Then it checks that themsg_sender is an approved minter. Then a newTransparentNote is created with the specifiedamount andsecret_hash . You can read the details of theTransparentNote in thetypes.nr filehere . Theamount is added to the existing publictotal_supply and the storage value is updated. Then the newTransparentNote is added to thepending_shields using theinsert_from_public function, which is accessible on thePrivateSet type. Then it's ready to be claimed by anyone with thesecret_hash pre-image using theredeem_shield function. It returns1 to indicate successful execution.

mint_private

[aztec(public)]

fn

mint_private (amount :

Field , secret_hash :

Field)

{ assert (storage . minters . at (context . msg_sender ( )) . read ( ) ,

"caller is not minter") ; let pending_shields = storage . pending_shields ; let

mut note =

TransparentNote :: new (amount , secret_hash) ; let supply = storage . total_supply . read ( ) . add (U128 :: from_integer (

amount)) ;

storage . total_supply . write (supply) ; pending_shields . insert_from_public (& mut note) ; }Source code: noir-projects/noir-contracts/contracts/token_contract/src/main.nr#L152-L165

shield

This public function enables an account to stage tokens from it'spublic_balance to be claimed as a privateTransparentNote by any account that has the pre-image to thesecret_hash .

First, storage is initialized. Then it checks whether the calling contract (context.msg_sender) matches the account that the funds will be debited from.

Authorizing token spends

If themsg_sender isNOT the same as the account to debit from, the function checks that the account has authorized themsg_sender contract to debit tokens on its behalf. This check is done by computing the function selector that needs to be authorized (in this case, theshield function), computing the hash of the message that the account contract has approved. This is a hash of the contract that is approved to spend (context.msg_sender), the token contract that can be spent from (context.this_address()), theselector , the account to spend from (from.address), theamount , thesecret_hash and anonce to prevent multiple spends. This hash is passed toassert_valid_public_message_for to ensure that the Account Contract has approved tokens to be spent on it's behalf.

If themsg_sender is the same as the account to debit tokens from, the authorization check is bypassed and the function proceeds to update the account'spublic_balance and adds a newTransparentNote to thepending_shields .

It returns1 to indicate successful execution.

shield

[aztec(public)]

fn

shield (from :

AztecAddress , amount :

Field , secret_hash :

Field , nonce :

Field)

{ if

(! from . eq (context . msg_sender ( )))

{ // The redeem is only spendable once, so we need to ensure that you cannot insert multiple shields from the same message. assert_current_call_valid_authwit_public (& mut context , from) ; }

else

{ assert (nonce ==

0 ,

"invalid nonce") ; }

let amount =

U128 :: from_integer (amount) ; let from_balance = storage . public_balances . at (from) . read ( ) . sub (amount) ;

let pending_shields = storage . pending_shields ; let

mut note =

TransparentNote :: new (amount . to_integer ( ) , secret_hash) ;

storage . public_balances . at (from) . write (from_balance) ; pending_shields . insert_from_public (& mut note) ; }
Source code: noir-projects/noir-contracts/contracts/token_contract/src/main.nr#L186-L205

transfer_public

This public function enables public transfers between Aztec accounts. The sender's public balance will be debited the specifiedamount and the recipient's public balances will be credited with that amount.

After storage is initialized, theauthorization flow specified above is checked. Then the sender and recipient's balances are updated and saved to storage.

transfer_public

[aztec(public)]

fn

transfer_public (from :

AztecAddress , to :

AztecAddress , amount :

Field , nonce :

Field)

{ if

(! from . eq (context . msg_sender ( )))

{ assert_current_call_valid_authwit_public (& mut context , from) ; }

else

{ assert (nonce ==

0 ,

"invalid nonce") ; }

let amount =

U128 :: from_integer (amount) ; let from_balance = storage . public_balances . at (from) . read ( ) . sub (amount) ; storage . public_balances . at (from) . write (from_balance) ;

let to_balance = storage . public_balances . at (to) . read ( ) . add (amount) ; storage . public_balances . at (to) . write (to_balance) ; } Source code: noir-projects/noir-contracts/contracts/token_contract/src/main.nr#L207-L223

burn_public

This public function enables public burning (destroying) of tokens from the sender's public balance.

After storage is initialized, theauthorization flow specified above is checked. Then the sender's public balance and thetotal_supply are updated and saved to storage.

burn_public

[aztec(public)]

fn

burn_public (from :

AztecAddress , amount :

Field , nonce :

Field)

{ if

(! from . eq (context . msg_sender ( )))

{ assert_current_call_valid_authwit_public (& mut context , from) ; }

else

{ assert (nonce ==

0 ,

"invalid nonce") ; }

let amount =

U128 :: from_integer (amount) ; let from_balance = storage . public_balances . at (from) . read ( ) . sub (amount) ; storage . public_balances . at (from) . write (from_balance) ;

let new_supply = storage . total_supply . read ( ) . sub (amount) ; storage . total_supply . write (new_supply) Source code: noir-projects/noir-contracts/contracts/token_contract/src/main.nr#L225-L243

Private function implementations

Private functions are declared with the#[aztec(private)] macro above the function name like so:

[aztec(private)]

fn

redeem_shield (As described in theexecution contexts section above , private function logic and transaction information is hidden from the world and is executed on user devices. Private functions update private state, but can pass data to the public execution context (e.g. see theunshield function).

Storage is referenced asstorage.variable .

redeem_shield

This private function enables an account to move tokens from aTransparentNote in thepending_shields mapping to any Aztec account as aValueNote in privatebalances .

Going through the function logic, first thesecret_hash is generated from the given secret. This ensures that only the entity possessing the secret can use it to redeem the note. Following this, aTransparentNote is retrieved from the set, using the provided amount and secret. The note is subsequently removed from the set, allowing it to be redeemed only once. The recipient's private balance is then increased using theincrement helper function from thevalue_note library .

The function returns1 to indicate successful execution.

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

unshield

This private function enables un-shielding of privateValueNote s stored inbalances to any Aztec account'spublic_balance .

After initializing storage, the function checks that themsg_sender is authorized to spend tokens. Seethe Authorizing token spends section above for more detail--the only difference being thatassert_valid_message_for is modified to work specifically in the private context. After the authorization check, the sender's private balance is decreased using thedecrement helper function for thevalue_note library. Then it stages a public function call on this contract (_increase_public_balance) to be executed in thepublic execution phase of transaction execution._increase_public_balance is marked as aninternal function, so can only be called by this token contract.

The function returns1 to indicate successful execution.

unshield

[aztec(private)]

fn

unshield (from :

AztecAddress , to :

AztecAddress , amount :

Field , nonce :

Field)

{ if

(! from . eq (context . msg_sender ( )))

{ assert_current_call_valid_authwit (& mut context , from) ; }

else

{ assert (nonce ==

0 ,

"invalid nonce") ; }

storage . balances . sub (from ,

U128 :: from_integer (amount)) ;

let selector =

FunctionSelector :: from_signature ("_increase_public_balance((Field),Field)") ; let _void = context . call_public_function (context . this_address ( ) , selector ,

[to . to_field ( ) , amount]) ; }Source code: noir-projects/noir-contracts/contracts/token_contract/src/main.nr#L263-L277

transfer

This private function enables private token transfers between Aztec accounts.

After initializing storage, the function checks that themsg_sender is authorized to spend tokens. See the Authorizing token spends section above for more detail--the only difference being thatassert_valid_message_for is modified to work specifically in the private context. After authorization, the function gets the current balances for the sender and recipient and decrements and increments them, respectively, using thevalue_note helper functions.

transfer

[aztec(private)]

fn

transfer (from :

AztecAddress , to :

AztecAddress , amount :

Field , nonce :

Field)

{ if

(! from . eq (context . msg_sender ( )))

{ assert_current_call_valid_authwit (& mut context , from) ; }

else

{ assert (nonce ==

0 ,

"invalid nonce") ; }

let amount =

U128 :: from_integer (amount) ; storage . balances . sub (from , amount) ; storage . balances . add (to , amount) ; }
Source code: noir-projects/noir-contracts/contracts/token_contract/src/main.nr#L279-L296

burn

This private function enables accounts to privately burn (destroy) tokens.

After initializing storage, the function checks that themsg_sender is authorized to spend tokens. Then it gets the sender's current balance and decrements it. Finally it stages a public function call to_reduce_total_supply .

burn

[aztec(private)]

fn

burn (from :

AztecAddress , amount :

Field , nonce :

Field)

{ if

(! from . eq (context . msg_sender ( )))

{ assert_current_call_valid_authwit (& mut context , from) ; }

else

{ assert (nonce ==

0 ,

"invalid nonce") ; }

storage . balances . sub (from ,

U128 :: from_integer (amount)) ;

let selector =

FunctionSelector :: from_signature ("_reduce_total_supply(Field)") ; let _void = context . call_public_function (context . this_address ( ) , selector ,

[amount]) ; }Source code: noir-projects/noir-contracts/contracts/token_contract/src/main.nr#L298-L312

Internal function implementations

Internal functions are functions that can only be called by this contract. The following 3 functions are public functions that are called from theprivate execution context . Marking these asinternal ensures that only the desired private functions in this contract are able to call them. Private functions defer execution to public functions because private functions cannot update public state directly.

_increase_public_balance

This function is called fromunshield . The account's private balance is decremented inshield and the public balance is increased in this function.

increase_public_balance

[aztec(public)]

[aztec(internal)]

fn

_increase_public_balance (to :

AztecAddress , amount :

Field)

{ let new_balance = storage . public_balances . at (to) . read ( ) . add (U128 :: from_integer (amount)) ; storage . public_balances . at (to) . write (new_balance) ; } Source code: noir-projects/noir-contracts/contracts/token_contract/src/main.nr#L316-L323

_reduce_total_supply

This function is called fromburn . The account's private balance is decremented inburn and the publictotal_supply is reduced in this function.

reduce_total_supply

[aztec(public)]

[aztec(internal)]

fn

_reduce_total_supply (amount :

Field)

{ // Only to be called from burn. let new_supply = storage . total_supply . read ( ) . sub (U128 :: from_integer (amount)) ; storage . total_supply . write (new_supply) ; } Source code: noir-projects/noir-contracts/contracts/token_contract/src/main.nr#L325-L333

Unconstrained function implementations

Unconstrained functions are similar toview functions in Solidity in that they only return information from the contract storage or compute and return data without modifying contract storage.

admin

A getter function for reading the publicadmin value.

admin unconstrained fn

admin ( )

->

pub

Field

{ storage . admin . read ( ) . to_field ( ) } Source code: noir-projects/noir-contracts/contracts/token_contract/src/main.nr#L337-L341

is_minter

A getter function for checking the value of associated with aminter in the publicminters mapping.

is_minter unconstrained fn

is_minter (minter :

AztecAddress)

->

pub

bool

{ storage . minters . at (minter) . read ( ) } Source code: noir-projects/noir-contracts/contracts/token_contract/src/main.nr#L343-L347

total_supply

A getter function for checking the tokentotal_supply .

total_supply unconstrained fn

total_supply ( )

->

pub

Field

{ storage . total_supply . read ( ) . to_integer ( ) }Source code: noir-projects/noir-contracts/contracts/token_contract/src/main.nr#L349-L353

balance_of_private

A getter function for checking the private balance of the provided Aztec account. Note that thePrivate Execution Environment (PXE) must have access to theowner s decryption keys in order to decrypt their notes.

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

balance_of_public

A getter function for checking the public balance of the provided Aztec account.

balance_of_public unconstrained fn

balance_of_public (owner :

AztecAddress)

->

pub

Field

{ storage . public_balances . at (owner) . read ( ) . to_integer ( ) }Source code: noir-projects/noir-contracts/contracts/token_contract/src/main.nr#L361-L365

Compiling

Now that the contract is complete, you can compile it withaztec-nargo . See theSandbox reference page for instructions on setting it up.

Run the following command in the directory where yourNargo.toml file is located:

aztec-nargo compile Once your contract is compiled, optionally generate a typescript interface with the following command:

aztec-cli codegen target -o src/artifacts --ts

Next Steps

Testing

Review the end to end tests for reference:

https://github.com/AztecProtocol/aztec-packages/blob/aztec-packages-v0.28.1/yarn-project/end-to-end/src/e2e_token_contract.test.ts

Token Bridge Contract

Thetoken bridge tutorial is a great follow up to this one.

It builds on the Token contract described here and goes into more detail about Aztec contract composability and Ethereum (L1) and Aztec (L2) cross-chain messaging. Edit this page

Previous Tutorials Next Writing a private voting smart contract in Aztec.nr