Writing your first smart contract

In this guide, we will create our first Aztec.nr smart contract. We will build a simple private counter. This contract will get you started with the basic setup and syntax of Aztec.nr, but doesn't showcase the awesome stuff Aztec is capable of.

If you already have some experience with Noir and want to build a cooler contract that utilizes both private and public state, you might want to check out thetoken contract tutorial instead .

Prerequisites

- You have followed thequickstart
- Running Aztec Sandbox

Set up a project

Create a new directory calledaztec-private-counter

mkdir aztec-private-counter then create acontracts folder inside where our Aztec.nr contract will live:

cd aztec-private-counter mkdir contracts Inside contracts create the following file structure:

. |-aztec-private-counter | |-contracts | | |--counter | | | |--src | | | | |--main.nr | | | |--Nargo.toml The filemain.nr will soon turn into our smart contract!

Add the following content toNargo.toml :

[package] name

=

"counter" type

=

"contract" authors

=

[""] compiler_version

=

">=0.18.0"

[dependencies] aztec

=

{

git

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

tag

"aztec-packages-v0.28.1" ,

directory

"noir-projects/aztec-nr/aztec"

} value_note

=

{

git

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

tag

"aztec-packages-v0.28.1" ,

directory

"noir-projects/aztec-nr/value-note" } easy_private_state

=

{

git

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

tag

"aztec-packages-v0.28.1" ,

directory

"noir-projects/aztec-nr/easy-private-state" }

Define the functions

Go tomain.nr and start with this contract initialization:

contract Counter

{ } This defines a contract calledCounter .

Imports

We need to define some imports.

Write this within your contract at the top

imports use

dep :: aztec :: prelude :: { AztecAddress ,

Map } ; use

dep :: aztec :: context :: Context ; use

dep :: value_note :: { balance_utils ,

value_note :: { ValueNote ,

VALUE_NOTE_LEN } } ; use

dep :: easy_private_state :: EasyPrivateUint ;Source code: noir-projects/noir-contracts/contracts/counter_contract/src/main.nr#L2-L7 context::{PrivateContext, Context}

Context gives us access to the environment information such asmsg.sender . We are also importingPrivateContext to access necessary information for our private functions. We'll be using it in the next step.

map::Map

Map is a private state variable that functions like a dictionary, relating Fields to other state variables.

value_note

Notes are fundamental to how Aztec manages privacy. A note is a privacy-preserving representation of an amount of tokens associated with an address, while encrypting the amount and owner. In this contract, we are using thevalue_note library. This is a type of note interface for storing a single Field, eg a balance - or, in our case, a counter.

We are also usingbalance_utils from this import, a useful library that allows us to utilize value notes as if they are simple balances.

EasyPrivateUint

This allows us to store our counter in a way that acts as an integer, abstracting the note logic.

Declare storage

Add this below the imports. It declares the storage variables for our contract. We are going to store a mapping of values for eachAztecAddress .

storage_struct struct

Storage

{ counters :

Map < AztecAddress ,

EasyPrivateUint

, } Source code: noir-projects/noir-contracts/contracts/counter_contract/src/main.nr#L9-L13

Keep the counter private

Now we've got a mechanism for storing our private state, we can start using it to ensure the privacy of balances.

Let's create a constructor method to run on deployment that assigns an initial supply of tokens to a specified owner. This function is calledinitialize , but

behaves like a constructor. It is the#[aztec(initializer)] decorator that specifies that this function behaves like a constructor. Write this:

constructor

[aztec(private)]

[aztec(initializer)]

// We can name our initializer anything we want as long as it's marked as aztec(initializer) fn

initialize (headstart :

u64 , owner :

AztecAddress)

{ let counters = storage . counters ; counters . at (owner) . add (headstart , owner) ;}Source code: noir-projects/noir-contracts/contracts/counter_contract/src/main.nr#L15-L23 This function accesses the counts from storage. Then it assigns the passed initial counter to theowner 's counter privately usingat().add() .

We have annotated this and other functions with#[aztec(private)] which are ABI macros so the compiler understands it will handle private inputs. Learn more about functions and annotationshere .

Incrementing our counter

Now let's implement theincrement function we defined in the first step.

increment

[aztec(private)]

fn

increment (owner :

AztecAddress)

{ let counters = storage . counters ; counters . at (owner) . add (1 , owner) ;}Source code: noir-projects/noir-contracts/contracts/counter_contract/src/main.nr#L25-L31 Theincrement function works very similarly to theconstructor , but instead directly adds 1 to the counter rather than passing in an initial count parameter.

Prevent double spending

Because our counters are private, the network can't directly verify if a note was spent or not, which could lead to double-spending. To solve this, we use a nullifier - a unique identifier generated from each spent note and its owner. Although this isn't really an issue in this simple smart contract, Aztec injects a special function calledcompute_note_hash_and_nullifier to determine these values for any given note produced by this contract.

Getting a counter

The last thing we need to implement is the function in order to retrieve a counter. In thegetCounter we defined in the first step, write this:

get_counter unconstrained fn

get_counter (owner :

AztecAddress)

->

pub

Field

{ let counters = storage . counters ; balance_utils :: get_balance (counters . at (owner) . set)}Source code: noir-projects/noir-contracts/contracts/counter_contract/src/main.nr#L33-L38 This function isunconstrained which allows us to fetch data from storage without a transaction. We retrieve a reference to theowner 'scounter from thecounters Map. Theget_balance function then operates on the owner's counter. This yields a private counter that only the private key owner can decrypt.

Test with the CLI

Now we've written a simple Aztec.nr smart contract, it's time to ensure everything works by testing with the CLI.

Compile the smart contract

In./contracts/counter/ directory, run this:

aztec-nargo compile This will compile the smart contract and create atarget folder with a.json artifact inside.

After compiling, you can generate a typescript class. In the same directory, run this:

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

Deploy

You can use the previously generated artifact to deploy the smart contract. Our constructor takes two arguments -initial_counter andowner so let's make sure to pass those in.

initial_counter can be any uint. In this guide we'll pick 100, but you can pick anything.

For theowner you can get the account addresses in your sandbox by running:

aztec-cli get-accounts This will return something like this:

➜ counter aztec-cli get-accounts Accounts found:

Address: 0x2fd4503a9b855a852272945df53d7173297c1469cceda31048b85118364b09a3 Public Key: 0x27c20118733174347b8082f578a7d8fb84b3ad38be293715eee8119ee5cd8a6d0d6b7d8124b37359663e75bcd2756f544a93b821a06f8e33fba68cc8029794d9 Partial Address: 0x11ee4cb5330545b3e82ace531526bc1995501a5596a85f90e5e60f4c1ad204dc

Address: 0x054ae9af363c6388cc6242c6eb0ed8a5860c15290744c81ecd5109434f9bb8b1 Public Key: 0x08145e8e8d46f51cda8d4c9cad81920236366abeafb8d387002bad879a3e87a81570b04ac829e4c007141d856d5a36d3b9c464e0f3c1c99cdbadaa6bb93f3257 Partial Address: 0x23ae266d9f8905bc4ef42e1435560ac78f3b5b55eb99b85398eb7011cd38fd8c

Address: 0x0d919c38d75484f8dd410cebaf0e17ccd196901d554d88f81b7e079375a4335d Public Key: 0x13e6151ea8e7386a5e7c4c5221047bf73d0b1b7a2ad14d22b7f73e57c1fa00c614bc6da69da1b581b09ee6cdc195e5d58ae4dce01b63bbb744e58f03855a94dd Partial Address: 0x2cf8f09aef15e219bf782049a3183a8adfd1fa254bf62bea050dc9a28fc979a7 Use one of theseaddress es as theowner . You can either copy it or export.

To deploy the counter contract,ensure the sandbox is running and run this in the root of your Noir project:

aztec-cli deploy contracts/counter/target/counter-Counter.json --args 100 0x0a0ab6320e2981cc543fedb9ad0f524c0a750397ca3372508d14af5b3c3c7cf0 --private-key 0x2153536ff6628eee01cf4024889ff977a18d9fa61d0e414422f7681cf085c281 You can also test the functions by applying what you learned in thequickstart .

Congratulations, you have now written, compiled, and deployed your first Aztec.nr smart contract!

Deploying your contract via the CLI will not register the deployed contract with thePXE . To do so, useaztec-cli add-contract .

aztec-cli add-contract --contract-artifact contracts/counter/target/counter-Counter.json --contract-address < contract-address

> note You can also deploy contracts using Aztec.js. Seethe next page for details.

Install Noir LSP (recommended)

Install theNoir Language Support extension to get syntax highlighting, syntax error detection and go-to definitions for your Aztec contracts.

Once the extension is installed, check your nargo binary by hovering overNargo in the status bar on the bottom right of the application window. Click to choose the path toaztec-nargo (or regularnargo , if you have that installed).

You can print the path of youraztec-nargo executable by running:

which aztec-nargo To specify a custom nargo executable, go to the VSCode settings and search for "noir", or click extension settings on thenoir-lang LSP plugin. Update theNoir: Nargo Path field to point to your desiredaztec-nargo executable.

What's next?

Now you can explore.

Interested in learning more about how Aztec works under the hood?

Understand the high level architecture on theCore Components page . You can also explore Aztec'shybrid state model andthe lifecycle of a transaction .

Want to write more contracts?

Follow the series of tutorials, starting with the private voting contracthere .

Ready to dive into Aztec and Ethereum cross-chain communication?

Read thePortals page and learn how to practically implement portals in thetoken bridge tutorial . Edit this page

Previous Quickstart Next An introduction to Aztec.js