Writing a private voting smart contract in Aztec.nr

In this tutorial we will go through writing a very simple private voting smart contract in Aztec.nr. You will learn about private functions, public functions, composability between them, state management and creatively using nullifiers to prevent people from voting twice!

We will build this:

- The contract will be initialized with an admin, stored publicly
- A voter can vote privately, which will call a public function and update the votes publicly
- The admin can end the voting period, which is a public boolean

To keep things simple, we won't create ballots or allow for delegate voting.

Prerequisites

- You have followed thequickstart
- to installaztec-nargo
- ,aztec-cli
- andaztec-sandbox
- .
- Running Aztec Sandbox

Set up a project

First, create a new contract project withaztec-nargo .

aztec-nargo new --contract private_voting Your file structure should look something like this:

. | | |--private_voting | | | |--src | | | | |--main.nr | | | |--Nargo.toml The filemain.nr will soon turn into our smart contract!

We will need the Aztec library to create this contract. Add the following content toNargo.toml :

[package] name

=

"private_voting" 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"

}

Initiate the contract and define imports

Go tomain.nr and delete the sample code. Replace it with this contract initialization:

contract Voting

{

} This defines a contract calledVoter . Everything will sit inside this block.

Inside this, paste these imports:

imports use

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

FunctionSelector ,

NoteHeader ,

NoteInterface ,

NoteGetterOptions ,

PrivateContext , Map ,

PublicMutable } ; use

dep :: aztec :: context :: Context ;Source code: noir-projects/noir-contracts/contracts/easy_private_voting_contract/src/main.nr#L2-L8 We are using various utils within the Aztec library:

- context
  - exposes things such as the contract address, msg_sender, etc
- context.request_nullifier_secret_key
  - get your secret key to help us create a randomized nullifier
- FunctionSelector::from_signature
  - compute a function selector from signature so we can call functions from other functions
- state_vars::{Map, PublicMutable}
  - we will use a Map to store the votes (key = voteId, value = number of votes), and PublicMutable to hold our public values that we mentioned earlier
- types::type_serialization::{..}
  - various serialization methods for defining how to use these types
- types::address::{AztecAddress},
  - our admin will be held as an address
- constants::EMPTY_NULLIFIED_COMMITMENT,
  - this will come in useful when creating our nullifier

Set up storage

Under these imports, we need to set up our contract storage. Define the storage struct like so:

storage_struct struct

Storage

{ admin :

PublicMutable < AztecAddress

,

// admin can end vote tally :

Map < Field ,

PublicMutable < Field

,

// we will store candidate as key and number of votes as value voteEnded :

PublicMutable < bool

,

// voteEnded is boolean } Source code: noir-projects/noir-contracts/contracts/easy_private_voting_contract/src/main.nr#L9-L15 In this contract, we will store three vars:

1. admin, as an Aztec address held in public state
2. tally, as a map with key as the persona and value as the number (in Field) held in public state
3. voteEnded, as a boolean held in public state

Constructor

The next step is to initialize the contract with a constructor. The constructor will take an address as a parameter and set the admin.

constructor

[aztec(public)]

[aztec(initializer)]

// annotation to mark function as a constructor fn

constructor (admin :

AztecAddress)

{ storage . admin . write (admin) ; storage . voteEnded . write (false) ;} Source code: noir-projects/noir-contracts/contracts/easy_private_voting_contract/src/main.nr#L17-L24 This function takes the admin argument and writes it to the storage. We are also using this function to set thevoteEnded boolean as false in the same way.

Casting a vote privately

For the sake of simplicity, we will have three requirements:

1. Everyone with an Aztec account gets a vote
2. They can only vote once in this contract
3. Who they are is private, but their actual vote is not

To ensure someone only votes once, we will create a nullifier as part of the function call. If they try to vote again, the function will revert as it creates the same nullifier again, which can't be added to the nullifier tree (as that indicates a double spend).

Create a private function calledcast_vote :

cast_vote

[aztec(private)]

// annotation to mark function as private and expose private context fn

cast_vote (candidate :

Field)

{ let secret = context . request_nullifier_secret_key (context . msg_sender ( )) ;

// get secret key of caller of function let nullifier =

dep :: std :: hash :: pedersen_hash ([context . msg_sender ( ) . to_field ( ) , secret . low , secret . high]) ;

// compute nullifier with this secret key so others can't descrypt it context . push_new_nullifier (nullifier ,

0) ;

// push nullifier context . call_public_function (context . this_address ( ) , FunctionSelector :: from_signature ("add_to_tally_public(Field)") , [candidate]) ; } Source code: noir-projects/noir-contracts/contracts/easy_private_voting_contract/src/main.nr#L26-L38 In this function, we do not create a nullifier with the address directly. This would leak privacy as it would be easy to reverse-engineer. We must add some randomness or some form of secret, likenullifier secrets .

To do this, we make anoracle call to fetch the caller's secret key, hash it to create a nullifier, and push the nullifier to Aztec. Thesecret.high andsecret.low values here refer to how we divide a largeGrumpkin scalar value into its higher and lower parts. This allows for faster cryptographic computations so our hash can still be secure but is calculated faster.

After pushing the nullifier, we update thetally to reflect this vote. As we know from before, a private function cannot update public state directly, so we are calling a public function.

Create this new public function like this:

add_to_tally_public

[aztec(public)]

[aztec(internal)]

fn

add_to_tally_public (candidate :

Field)

{ assert (storage . voteEnded . read ( )

==

false ,

"Vote has ended") ;

// assert that vote has not ended let new_tally = storage . tally . at (candidate) . read ( )

+

1 ; storage . tally . at (candidate) . write (new_tally) ;$ource code: noir-projects/noir-contracts/contracts/easy_private_voting_contract/src/main.nr#L40-L48 The first thing we do here is assert that the vote has not ended.

assert() takes two arguments: the assertion, in this case thatstorage.voteEnded is not false, and the error thrown if the assertion fails.

The code after the assertion will only run if the assertion is true. In this snippet, we read the current vote tally at the voteId, add 1 to it, and write this new number to the voteId. TheField element allows us to use+ to add to an integer.

Getting the number of votes

We will create a function that anyone can call that will return the number of votes at a given vote Id. Paste this in your contract:

get_vote unconstrained fn

get_vote (candidate :

Field)

->

pub

Field

{ storage . tally . at (candidate) . read ( ) }Source code: noir-projects/noir-contracts/contracts/easy_private_voting_contract/src/main.nr#L57-L61 We set it asunconstrained and do not annotate it because it is only reading from state. You can read more about unconstrained functionshere .

Allowing an admin to end a voting period

To ensure that only an admin can end a voting period, we can use anotherassert() statement.

Paste this function in your contract:

end_vote

[aztec(public)]

fn

end_vote ( )

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

"Only admin can end votes") ;

// assert that caller is admin storage . voteEnded . write (true) ; }Source code: noir-projects/noir-contracts/contracts/easy_private_voting_contract/src/main.nr#L50-L56 Here, we are asserting that themsg_sender() is equal to the admin stored in public state. We have to create anAztecAddress type from themsg_sender() in order to do a direct comparison.

Compiling and deploying

The easiest way to compile the contract is withaztec-nargo . Run the following command in the directory with your Nargo.toml file:

aztec-nargo compile This will create a new directory calledtarget and a JSON artifact inside it. To optionally create a typescript interface, run:

aztec-cli codegen target -o src/artifacts --ts Once it is compiled you candeploy it to the sandbox. Ensure yoursandbox is running and run this in the same dir as before:

aztec-cli deploy ./target/private_voting-Voting.json --args ADMIN_ADDRESS The constructor takes an address as an argument to set the admin, so you can use an address that is deployed with the sandbox - check the sandbox terminal or runaztec-cli get-accounts .

You should see a success message with the contract address. Now we can start calling functions!

Cast a vote like this:

aztec-cli send cast_vote --contract-artifact ./target/private_voting-Voting.json --contract-address CONTRACT_ADDRESS --args 1 --private-key PRIVATE_KEY You can get the contract address from the sandbox terminal or the message printed when you deployed the contract. You can also get a private key from the sandbox terminal, or generate one withaztec-cli generate-private-key .

This should return amined success message.

You can now try running this command again to ensure our nullifier works.

Get the number of votes like this:

aztec-cli call get_vote --contract-artifact ./target/private_voting-Voting.json --contract-address CONTRACT_ADDRESS --args 1 This should return1n .

You can follow this pattern to testend_vote() and access control of other functions. Find more information about calling functions from the CLIhere .

Next steps

Now you have learned the foundations of Aztec smart contracts, you can start to play around with some more advanced features. Some ideas:

- Add some more features into this contract, like the admin can distribute votes, people can delegate their votes, or voteIds can have more data like names, descriptions, etc
- Create a frontend for this contract using Aztec.js
- .
- Go to the next tutorial
- and learn how to write a token contract Edit this page

Previous Writing a token contract in Aztec.nr Next Dapp Development Tutorial