Minting tokens on Aztec

In this step we will start writing our Aztec.nr bridge smart contract and write a function to consume the message from the token portal to mint funds on Aztec

Initial contract setup

In ourtoken-bridge Aztec project inaztec-contracts , undersrc there is an examplemain.nr file. Paste this to define imports and initialize the constructor:

token_bridge_imports mod

token_interface ;

// Minimal implementation of the token bridge that can move funds between L1 <> L2. // The bridge has a corresponding Portal contract on L1 that it is attached to // And corresponds to a Token on L2 that uses the AuthWit accounts pattern. // Bridge has to be set as a minter on the token before it can be used

contract TokenBridge

{ use

dep :: aztec :: prelude :: { FunctionSelector ,

AztecAddress ,

EthAddress ,

PublicMutable } ;

use

dep :: aztec :: { context :: Context ,

hash :: compute_secret_hash } ;

use

dep :: token_portal_content_hash_lib :: { get_mint_public_content_hash , get_mint_private_content_hash , get_withdraw_content_hash } ;

use

crate :: token_interface :: Token ;Source code: noir-projects/noir-contracts/contracts/token_bridge_contract/src/main.nr#L1-L17 token_bridge_storage_and_constructor // Storage structure, containing all storage, and specifying what slots they use. struct

Storage

{ token :

PublicMutable < AztecAddress

, }

// Constructs the contract.

[aztec(private)]

[aztec(initializer)]

fn

constructor (token :

AztecAddress)

{ let selector =

```
FunctionSelector :: from_signature ( "_initialize((Field))" ) ; context . call_public_function ( context . this_address ( ) , selector
,

[ token . to_field ( ) ] ) ; }
```
Source code: noir-projects/noir-contracts/contracts/token_bridge_contract/src/main.nr#L19-L32
This imports Aztec-related dependencies and our helper filetoken_interface.nr . (The code above will give errors right now - this is because we haven't implemented util and token_interface yet.)

Intoken_interface.nr , add this:

```
token_bridge_token_interface use

dep :: aztec :: prelude :: { FunctionSelector ,

AztecAddress ,

EthAddress ,

PrivateContext } ; use

dep :: aztec :: context :: { PublicContext ,

Context } ;

struct

Token

{ address :

AztecAddress , }

impl

Token

{ pub

fn

at ( address :

AztecAddress )

->

Self

{ Self

{ address } }

pub

fn

mint_public ( self :

Self , context :

& mut

PublicContext , to :

AztecAddress , amount :

Field )

{ let _return_values = context . call_public_function ( self . address , FunctionSelector :: from_signature (
"mint_public((Field),Field)" ) , [ to . to_field ( ) , amount ] ) ; }

pub

fn
```

burn_public (self :

Self , context :

& mut

PublicContext , from :

AztecAddress , amount :

Field , nonce :

Field)

{ let _return_values = context . call_public_function (self . address , FunctionSelector :: from_signature ("burn_public((Field),Field,Field)") , [from . to_field ( ) , amount , nonce]) ; }

pub

fn

mint_private (self :

Self , context :

& mut

PublicContext , amount :

Field , secret_hash :

Field)

{ let _return_values = context . call_public_function (self . address , FunctionSelector :: from_signature ("mint_private(Field,Field)") , [amount , secret_hash]) ; }

pub

fn

burn (self :

Self , context :

& mut

PrivateContext , from :

AztecAddress , amount :

Field , nonce :

Field)

{ let _return_values = context . call_private_function (self . address , FunctionSelector :: from_signature ("burn((Field),Field,Field)") , [from . to_field ( ) , amount , nonce]) ; } } Source code: noir-projects/noir-contracts/contracts/token_bridge_contract/src/token_interface.nr#L1-L56

Consume the L1 message

In the previous step, we have moved our funds to the portal and created a L1->L2 message. Upon building the next rollup, the sequencer asks the inbox for any incoming messages and adds them to Aztec's L1->L2 message tree, so an application on L2 can prove that the message exists and consumes it.

Inmain.nr , now paste thisclaim_public function:

claim_public // Consumes a L1->L2 message and calls the token contract to mint the appropriate amount publicly

[aztec(public)]

fn

claim_public (to :

AztecAddress , amount :

Field , canceller :

EthAddress , secret :

Field)

{ let content_hash =

get_mint_public_content_hash (to , amount , canceller) ;

// Consume message and emit nullifier context . consume_l1_to_l2_message (content_hash , secret , context . this_portal_address ( )) ;

// Mint tokens Token :: at (storage . token . read ( )) . mint_public (& mut context , to , amount) }Source code: noir-projects/noir-contracts/contracts/token_bridge_contract/src/main.nr#L34-L46 Theclaim_public function enables anyone to consume the message on the user's behalf and mint tokens for them on L2. This is fine as the minting of tokens is done publicly anyway.

What’s happening here?

1. We first recompute the L1->L2 message content by callingget_mint_public_content_hash()
2. . Note that the method does exactly the same as what the TokenPortal contract does indepositToAztecPublic()
3. to create the content hash.
4. We then attempt to consume the L1->L2 message by passing themsg_key
5. , the the content hash, and the "secret". Since we are depositing to Aztec publicly, this secret is public, anyone can know this and is usually 0.* context.consume_l1_to_l2_message()
6.
   - takes in the content_hash and secret to recreate the original message. The L1 to L2 message consists of:* Sender - who on L1 sent the message + chain ID of L1. The context variable knows the portal address on L1 and adds that
7.
   -
     - Recipient - i.e. this aztec contract address which is consuming the message + the current version of the aztec rollup.
8.
   -
     - The content - which is reconstructed in theget_mint_public_content_hash()
9.
   - Note that thecontent_hash
10.
    - requiresto
11.
    - ,amount
12.
    - andcanceller
13.
    - . If a malicious user tries to mint tokens to their address by changing the to address, the content hash will be different to what the token portal had calculated on L1 and themsg_Key
14.
    - will also be different, thus preventing the L1->L2 message from being consumed. This is why we add these parameters into the content.
15. Then we callToken::at(storage.token.read()).mint_public()
16. to mint the tokens to the to address.

Private flow

Now we will create a function to mint the amount privately. Paste this into yourmain.nr

claim_private // Consumes a L1->L2 message and calls the token contract to mint the appropriate amount in private assets // User needs to call token.redeem_shield() to get the private assets

[aztec(private)]

fn

claim_private (secret_hash_for_redeeming_minted_notes :

Field ,

// secret hash used to redeem minted notes at a later time. This enables anyone to call this function and mint tokens to a user on their behalf amount :

Field , canceller :

EthAddress , secret_for_L1_to_L2_message_consumption :

Field

// secret used to consume the L1 to L2 message)

{ // Consume L1 to L2 message and emit nullifier let content_hash =

get_mint_private_content_hash (secret_hash_for_redeeming_minted_notes , amount , canceller) ; context . consume_l1_to_l2_message (content_hash , secret_for_L1_to_L2_message_consumption , context . this_portal_address ( )) ;

// Mint tokens on L2 // mint_private on token is public. So we call an internal public function // which then calls the public method on the token contract. // Since the secret_hash is passed, no secret is leaked. context . call_public_function (context . this_address ( ) , FunctionSelector :: from_signature ("_call_mint_on_token(Field,Field)") , [amount , secret_hash_for_redeeming_minted_notes]) ; } Source code: noir-projects/noir-contracts/contracts/token_bridge_contract/src/main.nr#L66-L94 call_mint_on_token // This is a public call as we need to read from public storage. // Also, note that user hashes their secret in private and only sends the hash in public // meaning only user can redeem_shield at a later time with their secret.

[aztec(public)]

[aztec(internal)]

fn

_call_mint_on_token (amount :

Field , secret_hash :

Field)

{ Token :: at (storage . token . read ( )) . mint_private (& mut context , amount , secret_hash) ; } Source code: noir-projects/noir-contracts/contracts/token_bridge_contract/src/main.nr#L147-L156 Theget_mint_private_content_hash function is imported from thetoken_portal_content_hash_lib .

If the content hashes were constructed similarly formint_private andmint_publicly , then content intended for private execution could have been consumed by calling theclaim_public method. By making these two content hashes distinct, we prevent this scenario.

While we mint the tokens on L2, westill don't actually mint them to a certain address . Instead we continue to pass thesecret_hash_for_redeeming_minted_notes like we did on L1. This means that a user could reveal their secret for L2 message consumption for anyone to mint tokens on L2 but they can redeem these notes at a later time.This enables a paradigm where an app can manage user's secrets for L2 message consumption on their behalf .The app or any external party can also mint tokens on the user's behalf should they be comfortable with leaking the secret for L2 Message consumption. This doesn't leak any new information to the app because their smart contract on L1 knew that a user wanted to move some amount of tokens to L2. The app still doesn't know which address on L2 the user wants these notes to be in, but they can mint tokens nevertheless on their behalf.

To mint tokens privately,claim_private calls an internal function_call_mint_on_token() which then callstoken.mint_private() which is a public method since it operates on public storage. Note that mint_private (on the token contract) is public because it too reads from public storage. Since thesecret_hash_for_redeeming_minted_notes is passed publicly (and not the secret), nothing that should be leaked is, and the only the person that knows the secret can actually redeem their notes at a later time by callingToken.redeem_shield(secret, amount) .

In the next step we will see how we can cancel a messageEdit this page

Previous Depositing Tokens to AztecNext Cancelling Deposits