Swapping Publicly

In this step we will create the flow for allowing a user to swap their tokens publicly on L1. It will have the functionality of letting anyone call this method on behalf of the user, assuming they have appropriate approvals. This means that an operator can pay gas fees on behalf of the user!

Inmain.nr paste this:

swap_public

[aztec(public)]

fn

swap_public (sender :

AztecAddress , input_asset_bridge :

AztecAddress , input_amount :

Field , output_asset_bridge :

AztecAddress , // params for using the transfer approval nonce_for_transfer_approval :

Field , // params for the swap uniswap_fee_tier :

Field , minimum_output_amount :

Field , // params for the depositing output_asset back to Aztec recipient :

AztecAddress , secret_hash_for_L1_to_l2_message :

Field , deadline_for_L1_to_l2_message :

Field , canceller_for_L1_to_L2_message :

EthAddress , caller_on_L1 :

EthAddress , // nonce for someone to call swap on sender's behalf nonce_for_swap_approval :

Field)

{ if

(! sender . eq (context . msg_sender ( )))

{ assert_current_call_valid_authwit_public (& mut context , sender) ; }

let input_asset =

TokenBridge :: at (input_asset_bridge) . token (& mut context) ;

// Transfer funds to this contract Token :: at (input_asset) . transfer_public (& mut context , sender , context . this_address ( ) , input_amount , nonce_for_transfer_approval) ;

// Approve bridge to burn this contract's funds and exit to L1 Uniswap Portal let _void = context . call_public_function (context . this_address ( ) , FunctionSelector :: from_signature ("_approve_bridge_and_exit_input_asset_to_L1((Field),(Field),Field)") , [input_asset . to_field ( ) , input_asset_bridge . to_field ( ) , input_amount]) ;

// Create swap message and send to Outbox for Uniswap Portal // this ensures the integrity of what the user originally intends to do on L1. let input_asset_bridge_portal_address =

get_portal_address (input_asset_bridge) ; let output_asset_bridge_portal_address =

get_portal_address (output_asset_bridge) ; // ensure portal exists - else funds might be lost assert (! input_asset_bridge_portal_address . is_zero ( ) ,

"L1 portal address of input_asset's bridge is 0") ; assert (! output_asset_bridge_portal_address . is_zero ( ) ,

"L1 portal address of output_asset's bridge is 0") ;

let content_hash =

compute_swap_public_content_hash (input_asset_bridge_portal_address , input_amount , uniswap_fee_tier , output_asset_bridge_portal_address , minimum_output_amount , recipient , secret_hash_for_L1_to_l2_message , deadline_for_L1_to_l2_message , canceller_for_L1_to_L2_message , caller_on_L1) ; context . message_portal (context . this_portal_address ( ) , content_hash) ; } Source code: noir-projects/noir-contracts/contracts/uniswap_contract/src/main.nr#L29-L98 This uses a util functioncompute_swap_public_content_hash() - let's add that.

Inutil.nr , add:

uniswap_public_content_hash use

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

EthAddress } ; use

dep :: aztec :: protocol_types :: hash :: sha256_to_field ;

// This method computes the L2 to L1 message content hash for the public // refer1-contracts/test/portals/UniswapPortal.sol on how L2 to L1 message is expected pub

fn

compute_swap_public_content_hash (input_asset_bridge_portal_address :

EthAddress , input_amount :

Field , uniswap_fee_tier :

Field , output_asset_bridge_portal_address :

EthAddress , minimum_output_amount :

Field , aztec_recipient :

AztecAddress , secret_hash_for_L1_to_l2_message :

Field , deadline_for_L1_to_l2_message :

Field , canceller_for_L1_to_L2_message :

EthAddress , caller_on_L1 :

EthAddress)

->

Field

{ let

mut hash_bytes :

[u8 ;

324]

=

[0 ;

324] ;

// 10 fields of 32 bytes each + 4 bytes fn selector

let input_token_portal_bytes = input_asset_bridge_portal_address . to_field ( ) . to_be_bytes (32) ; let in_amount_bytes = input_amount . to_be_bytes (32) ; let uniswap_fee_tier_bytes = uniswap_fee_tier . to_be_bytes (32) ; let output_token_portal_bytes = output_asset_bridge_portal_address . to_field ( ) . to_be_bytes (32) ; let amount_out_min_bytes = minimum_output_amount . to_be_bytes (32) ; let aztec_recipient_bytes = aztec_recipient . to_field ( ) . to_be_bytes (32) ; let secret_hash_for_L1_to_l2_message_bytes = secret_hash_for_L1_to_l2_message . to_be_bytes (32) ; let deadline_for_L1_to_l2_message_bytes = deadline_for_L1_to_l2_message . to_be_bytes (32) ; let canceller_bytes = canceller_for_L1_to_L2_message . to_field ( ) . to_be_bytes (32) ; let caller_on_L1_bytes =

caller_on_L1 . to_field ( ) . to_be_bytes (32) ;

// function selector: 0xf3068cac
keccak256("swap_public(address,uint256,uint24,address,uint256,bytes32,bytes32,uint32,address,address)") hash_bytes [0]

=

0xf3 ; hash_bytes [1]

=

0x06 ; hash_bytes [2]

=

0x8c ; hash_bytes [3]

=

0xac ;

for i in

0 .. 32

{ hash_bytes [i +

4]

= input_token_portal_bytes [i] ; hash_bytes [i +

36]

= in_amount_bytes [i] ; hash_bytes [i +

68]

= uniswap_fee_tier_bytes [i] ; hash_bytes [i +

100]

= output_token_portal_bytes [i] ; hash_bytes [i +

132]

= amount_out_min_bytes [i] ; hash_bytes [i +

164]

= aztec_recipient_bytes [i] ; hash_bytes [i +

196]

= secret_hash_for_L1_to_l2_message_bytes [i] ; hash_bytes [i +

228]

= deadline_for_L1_to_l2_message_bytes [i] ; hash_bytes [i +

260]

= canceller_bytes [i] ; hash_bytes [i +

292]

= caller_on_L1_bytes [i] ; }

let content_hash =

sha256_to_field (hash_bytes) ; content_hash }Source code: noir-projects/noir-contracts/contracts/uniswap_contract/src/util.nr#L1-L54 What's happening here?

1. We check thatmsg.sender()

2. has appropriate approval to call this on behalf of the sender by constructing an authwit message and checking iffrom
3. has given the approval (read more about authwihere
4.).
5. We fetch the underlying aztec token that needs to be swapped.
6. We transfer the user's funds to the Uniswap contract. Like with Ethereum, the user must have provided approval to the Uniswap contract to do so. The user must provide the nonce they used in the approval for transfer, so that Uniswap can send it to the token contract, to prove it has appropriate approval.
7. Funds are added to the Uniswap contract.
8. Uniswap must exit the input tokens to L1. For this it has to approve the bridge to burn its tokens on its behalf and then actually exit the funds. We call theexit_to_l1_public() method on the token bridge
9. . We use the public flow for exiting since we are operating on public state.
10. It is not enough for us to simply emit a message to withdraw the funds. We also need to emit a message to display our swap intention. If we do not do this, there is nothing stopping a third party from calling the Uniswap portal with their own parameters and consuming our message.

So the Uniswap portal (on L1) needs to know:

- The token portals for the input and output token (to withdraw the input token to L1 and later deposit the output token to L2)
- The amount of input tokens they want to swap
- The Uniswap fee tier they want to use
- The minimum output amount they can accept (for slippage protection)

The Uniswap portal must first withdraw the input tokens, then check that the swap message exists in the outbox, execute the swap, and then call the output token to deposit the swapped tokens to L2. So the Uniswap portal must also be pass any parameters needed to complete the deposit of swapped tokens to L2. From the tutorial on building token bridges we know these are:

- The address on L2 which must receive the output tokens (remember this is public flow)
- The secret hash for consume the L1 to L2 message. Since this is the public flow the preimage doesn't need to be a secret
- The deadline to consume the l1 to l2 message (this is so funds aren't stuck in the processing state forever and the message can be cancelled. Else the swapped tokens would be stuck forever)
- The address that can cancel the message (and receive the swapped tokens)
- We include these params in the L2 → L1swap_public message content
- too. Under the hood, the protocol adds the sender (the Uniswap l2 contract) and the recipient (the Uniswap portal contract on L1).

In the next step we will write the code to execute this swap on L1Edit this page

Previous L2 Contract Setup Next Solidity Code to Execute Swap on L1