Deploy & Call Contracts with Typescript

In this step we will write a Typescript test to interact with the sandbox and call our contracts!

Test imports and setup

We need some helper files that can keep our code clean. Inside yoursrc/test directory:

cd fixtures touch utils.ts cd

..

&&

mkdir shared &&

cd shared touch cross_chain_test_harness.ts Inutils.ts , we need a delay function. Put this:

delay /* * *Sleep for a given number of milliseconds. * @param ms - the number of milliseconds to sleep for* export

function

delay (ms :

number) :

Promise < void

{ return

new

Promise < void

(resolve =>

setTimeout (resolve , ms)) ; }Source code: yarn-project/end-to-end/src/fixtures/utils.ts#L413-L421
Incross_chain_test_harness.ts , add:

cross_chain_test_harness import

{ AztecAddress , DebugLogger , EthAddress , ExtendedNote , Fr , Note , PXE , TxHash , TxStatus , Wallet , computeMessageSecretHash , deployL1Contract , sha256 , }

from

'@aztec/aztec.js' ; import

{ InboxAbi , OutboxAbi , PortalERC20Abi , PortalERC20Bytecode , TokenPortalAbi , TokenPortalBytecode , }

from

'@aztec/l1-artifacts' ; import

{ TokenContract }

from

'@aztec/noir-contracts.js/Token' ; import

{ TokenBridgeContract }

from

'@aztec/noir-contracts.js/TokenBridge' ;

import

{ Account , Chain , HttpTransport , PublicClient , WalletClient , getContract , getFunctionSelector }

from

'viem' ;

/* * Deploy L1 token and portal, initialize portal, deploy a non native l2 token contract, its L2 bridge contract and attach is to the portal. * @param wallet - the wallet instance * @param walletClient - A viem WalletClient. * @param publicClient - A viem PublicClient. * @param rollupRegistryAddress - address of rollup registry to pass to initialize the token portal * @param owner - owner of the L2 contract * @param underlyingERC20Address - address of the underlying ERC20 contract to use (if none supplied, it deploys one) * @returns l2 contract instance, bridge contract instance, token portal instance, token portal address and the underlying ERC20 instance / export

async

function

deployAndInitializeTokenAndBridgeContracts (wallet : Wallet , walletClient : WalletClient < HttpTransport , Chain , Account

, publicClient : PublicClient < HttpTransport , Chain

, rollupRegistryAddress : EthAddress , owner : AztecAddress , underlyingERC20Address ? : EthAddress ,) :

Promise < { /** **The L2 token contract instance. */ token : TokenContract ;** /* The L2 bridge contract instance./ *bridge : TokenBridgeContract ; / * The token portal contract address.* / tokenPortalAddress : EthAddress ; /* * *The token portal contract instance* / tokenPortal :

any ; /* * *The underlying ERC20 contract instance.* / underlyingERC20 :

any ; }

{ if

(! underlyingERC20Address)

{ underlyingERC20Address =

await

deployL1Contract (walletClient , publicClient , PortalERC20Abi , PortalERC20Bytecode) ; } const underlyingERC20 =

getContract ({ address : underlyingERC20Address . toString ( ) , abi : PortalERC20Abi , client : walletClient , }) ;

// deploy the token portal const tokenPortalAddress =

await

deployL1Contract (walletClient , publicClient , TokenPortalAbi , TokenPortalBytecode) ; const tokenPortal =

getContract ({ address : tokenPortalAddress . toString ( ) , abi : TokenPortalAbi , client : walletClient , }) ;

// deploy l2 token const token =

await TokenContract . deploy (wallet , owner ,

'TokenName' ,

'TokenSymbol' ,

18) . send ( ) . deployed ( ) ;

// deploy l2 token bridge and attach to the portal const bridge =

await TokenBridgeContract . deploy (wallet , token . address) . send ({ portalContract : tokenPortalAddress }) . deployed ( ) ;

if

((await token . methods . admin ( ) . view ( ))

!== owner . toBigInt ( ))

{ throw

new

Error (Token admin is not { owner }) ; }

if

(! (await bridge . methods . token ( ) . view ( )) . equals (token . address))

{ throw

new

Error (Bridge token is not { token . address }) ; }

// make the bridge a minter on the token: await token . methods . set_minter (bridge . address ,

true) . send ( ) . wait ( ) ; if

((await token . methods . is_minter (bridge . address) . view ( ))

===

1n)

{ throw

new

Error (Bridge is not a minter) ; }

// initialize portal await tokenPortal . write . initialize ([rollupRegistryAddress . toString ( ) , underlyingERC20Address . toString ( ) , bridge . address . toString ( )] , { }

as

any ,) ;

return

{ token , bridge , tokenPortalAddress , tokenPortal , underlyingERC20 } ; }

/* * *A Class for testing cross chain interactions, contains common interactions * shared between cross chain tests.* / export

class

CrossChainTestHarness

{ static

async

new (pxeService :

PXE , publicClient : PublicClient < HttpTransport , Chain

, walletClient :

any , wallet : Wallet , logger : DebugLogger , underlyingERC20Address ? : EthAddress ,) :

Promise < CrossChainTestHarness

{ const ethAccount = EthAddress . fromString ((await walletClient . getAddresses ( )) [0]) ; const owner = wallet . getCompleteAddress ( ) ; const l1ContractAddresses =

(await pxeService . getNodeInfo ( )) . l1ContractAddresses ;

const inbox =

getContract ({ address : l1ContractAddresses . inboxAddress . toString ( ) , abi : InboxAbi , client : walletClient , }) ;

const outbox =

getContract ({ address : l1ContractAddresses . outboxAddress . toString ( ) , abi : OutboxAbi , client : walletClient , }) ;

// Deploy and initialize all required contracts logger ('Deploying and initializing token, portal and its bridge...') ; const

{ token , bridge , tokenPortalAddress , tokenPortal , underlyingERC20 }

= await

deployAndInitializeTokenAndBridgeContracts (wallet , walletClient , publicClient , l1ContractAddresses . registryAddress , owner . address , underlyingERC20Address ,) ; logger ('Deployed and initialized token, portal and its bridge.') ;

return

new

CrossChainTestHarness (pxeService , logger , token , bridge , ethAccount , tokenPortalAddress , tokenPortal , underlyingERC20 , inbox , outbox , publicClient , walletClient , owner . address ,) ; }

constructor (/* *Private eXecution Environment (PXE).* / public pxeService :

PXE , /* *Logger.* / public logger : DebugLogger ,

/ **L2 Token contract. */ public l2Token : TokenContract ,** /L2 Token bridge contract. */ public l2Bridge : TokenBridgeContract ,

/* *Eth account to interact with.*/ public ethAccount : EthAddress ,

/ **Portal address. */ public tokenPortalAddress : EthAddress ,** /Token portal instance. */ public tokenPortal :

any , /* *Underlying token for portal tests.* / public underlyingERC20 :

any , /* *Message Bridge Inbox.* / public inbox :

any , /* *Message Bridge Outbox.* / public outbox :

any , /* *Viem Public client instance.* / public publicClient : PublicClient < HttpTransport , Chain

, /* *Viem Wallet Client instance.* / public walletClient :

any ,

/* *Aztec address to use in tests.*/ public ownerAddress : AztecAddress ,)

{ }

generateClaimSecret ( ) :

[Fr , Fr]

{ this . logger ("Generating a claim secret using pedersen's hash function") ; const secret = Fr . random ( ) ; const secretHash =

computeMessageSecretHash (secret) ; this . logger ('Generated claim secret: '

+ secretHash . toString ( )) ; return

[secret , secretHash] ; }

async

mintTokensOnL1 (amount : bigint)

{ this . logger ('Minting tokens on L1') ; const txHash =

await

this . underlyingERC20 . write . mint ([this . ethAccount . toString ( ) , amount] ,

{ }

as

any) ; await

this . publicClient . waitForTransactionReceipt ({ hash : txHash }) ; expect (await

this . underlyingERC20 . read . balanceOf ([this . ethAccount . toString ( )])) . toBe (amount) ; }

async

getL1BalanceOf (address : EthAddress)

{ return

await

this . underlyingERC20 . read . balanceOf ([address . toString ( )]) ; }

async

sendTokensToPortalPublic (bridgeAmount : bigint , secretHash : Fr)

{ const txHash1 =

await

this . underlyingERC20 . write . approve ([this . tokenPortalAddress . toString ( ) , bridgeAmount] , { }

as

any ,) ; await

this . publicClient . waitForTransactionReceipt ({ hash : txHash1 }) ;

// Deposit tokens to the TokenPortal const deadline =

2

**

32

-

1 ;

// max uint32

this . logger ('Sending messages to L1 portal to be consumed publicly') ; const args =

[this . ownerAddress . toString ( ) , bridgeAmount , this . ethAccount . toString ( ) , deadline , secretHash . toString ( ) ,]

as

const ; const

{ result : entryKeyHex }

=

await

this . tokenPortal . simulate . depositToAztecPublic (args ,

{ account :

this . ethAccount . toString ( ) , }

as

any) ; const txHash2 =

await

this . tokenPortal . write . depositToAztecPublic (args ,

{ }

as

any) ; await

this . publicClient . waitForTransactionReceipt ({ hash : txHash2 }) ;

return Fr . fromString (entryKeyHex) ; }

async

sendTokensToPortalPrivate (secretHashForRedeemingMintedNotes : Fr , bridgeAmount : bigint , secretHashForL2MessageConsumption : Fr ,)

{ const txHash1 =

await

this . underlyingERC20 . write . approve ([this . tokenPortalAddress . toString ( ) , bridgeAmount] , { }

as

any ,) ; await

this . publicClient . waitForTransactionReceipt ({ hash : txHash1 }) ; // Deposit tokens to the TokenPortal const deadline =

2

**

32

-

1 ;

// max uint32

this . logger ('Sending messages to L1 portal to be consumed privately') ; const args =

[secretHashForRedeemingMintedNotes . toString ( ) , bridgeAmount , this . ethAccount . toString ( ) , deadline , secretHashForL2MessageConsumption . toString ( ) ,]

as

const ; const

{ result : entryKeyHex }

=

await

this . tokenPortal . simulate . depositToAztecPrivate (args ,

{ account :

this . ethAccount . toString ( ) , }

as

any) ; const txHash2 =

await

this . tokenPortal . write . depositToAztecPrivate (args ,

{ }

as

any) ; await

this . publicClient . waitForTransactionReceipt ({ hash : txHash2 }) ;

return Fr . fromString (entryKeyHex) ; }

async

mintTokensPublicOnL2 (amount : bigint)

{ this . logger ('Minting tokens on L2 publicly') ; const tx =

this . l2Token . methods . mint_public (this . ownerAddress , amount) . send ( ) ; const receipt =

await tx . wait ( ) ; expect (receipt . status) . toBe (TxStatus . MINED) ; }

async

mintTokensPrivateOnL2 (amount : bigint , secretHash : Fr)

{ const tx =

this . l2Token . methods . mint_private (amount , secretHash) . send ( ) ; const receipt =

await tx . wait ( ) ; expect (receipt . status) . toBe (TxStatus . MINED) ; await

this . addPendingShieldNoteToPXE (amount , secretHash , receipt . txHash) ; }

async

performL2Transfer (transferAmount : bigint , receiverAddress : AztecAddress)

{ // send a transfer tx to force through rollup with the message included const transferTx =

this . l2Token . methods . transfer_public (this . ownerAddress , receiverAddress , transferAmount ,

0) . send ( ) ; const receipt =

await transferTx . wait ( ) ; expect (receipt . status) . toBe (TxStatus . MINED) ; }

async

consumeMessageOnAztecAndMintSecretly (secretHashForRedeemingMintedNotes : Fr , bridgeAmount : bigint , secretForL2MessageConsumption : Fr ,)

{ this . logger ('Consuming messages on L2 secretively') ; // Call the mint tokens function on the Aztec.nr contract const consumptionTx =

this . l2Bridge . methods . claim_private (secretHashForRedeemingMintedNotes , bridgeAmount ,

this . ethAccount , secretForL2MessageConsumption) . send ( ) ; const consumptionReceipt =

await consumptionTx . wait ( ) ; expect (consumptionReceipt . status) . toBe (TxStatus . MINED) ;

await

this . addPendingShieldNoteToPXE (bridgeAmount , secretHashForRedeemingMintedNotes , consumptionReceipt . txHash) ; }

async

consumeMessageOnAztecAndMintPublicly (bridgeAmount : bigint , secret : Fr)

{ this . logger ('Consuming messages on L2 Publicly') ; // Call the mint tokens function on the Aztec.nr contract const tx =

this . l2Bridge . methods . claim_public (this . ownerAddress , bridgeAmount ,

this . ethAccount , secret) . send ( ) ; const receipt =

await tx . wait ( ) ; expect (receipt . status) . toBe (TxStatus . MINED) ; }

async

withdrawPrivateFromAztecToL1 (withdrawAmount : bigint , nonce : Fr = Fr . ZERO)

{ const withdrawTx =

this . l2Bridge . methods . exit_to_l1_private (this . l2Token . address ,

this . ethAccount , withdrawAmount , EthAddress . ZERO , nonce) . send ( ) ; const withdrawReceipt =

await withdrawTx . wait ( ) ; expect (withdrawReceipt . status) . toBe (TxStatus . MINED) ; }

async

withdrawPublicFromAztecToL1 (withdrawAmount : bigint , nonce : Fr = Fr . ZERO)

{ const withdrawTx =

this . l2Bridge . methods . exit_to_l1_public (this . ethAccount , withdrawAmount , EthAddress . ZERO , nonce) . send ( ) ; const withdrawReceipt =

await withdrawTx . wait ( ) ; expect (withdrawReceipt . status) . toBe (TxStatus . MINED) ; }

async

getL2PrivateBalanceOf (owner : AztecAddress)

{ return

await

this . l2Token . methods . balance_of_private (owner) . view ({ from : owner }) ; }

async

expectPrivateBalanceOnL2 (owner : AztecAddress , expectedBalance : bigint)

{ const balance =

await

this . getL2PrivateBalanceOf (owner) ; this . logger (Account { owner } balance: { balance }) ; expect (balance) . toBe (expectedBalance) ; }

async

getL2PublicBalanceOf (owner : AztecAddress)

{ return

await

this . l2Token . methods . balance_of_public (owner) . view ( ) ; }

async

expectPublicBalanceOnL2 (owner : AztecAddress , expectedBalance : bigint)

{ const balance =

await

this . getL2PublicBalanceOf (owner) ; expect (balance) . toBe (expectedBalance) ; }

async

checkEntryIsNotInOutbox (withdrawAmount : bigint , callerOnL1 : EthAddress = EthAddress . ZERO) :

Promise < Fr

{ this . logger ('Ensure that the entry is not in outbox yet') ;

const content = Fr . fromBufferReduce (sha256 (Buffer . concat ([Buffer . from (getFunctionSelector ('withdraw(address,uint256,address)') . substring (2) ,

'hex') , this . ethAccount . toBuffer32 ( ) , new

Fr (withdrawAmount) . toBuffer ( ) , callerOnL1 . toBuffer32 ( ) ,]) ,) ,) ; const entryKey = Fr . fromBufferReduce (sha256 (Buffer . concat ([this . l2Bridge . address . toBuffer ( ) , new

Fr (1) . toBuffer ( ) ,

// aztec version this . tokenPortalAddress . toBuffer32 ( )

?? Buffer . alloc (32 ,

0) , new

Fr (this . publicClient . chain . id) . toBuffer ( ) ,

// chain id content . toBuffer ( ) ,]) ,) ,) ; expect (await

this . outbox . read . contains ([entryKey . toString ( )])) . toBeFalsy ( ) ;

return entryKey ; }

async

withdrawFundsFromBridgeOnL1 (withdrawAmount : bigint , entryKey : Fr)

{ this . logger ('Send L1 tx to consume entry and withdraw funds') ; // Call function on L1 contract to consume the message const

{ request : withdrawRequest , result : withdrawEntryKey }

=

await

this . tokenPortal . simulate . withdraw ([this . ethAccount . toString ( ) , withdrawAmount , false ,]) ;

expect (withdrawEntryKey) . toBe (entryKey . toString ( )) ; expect (await

this . outbox . read . contains ([withdrawEntryKey])) . toBeTruthy ( ) ;

await

this . walletClient . writeContract (withdrawRequest) ; return withdrawEntryKey ; }

async

shieldFundsOnL2 (shieldAmount : bigint , secretHash : Fr)

{ this . logger ('Shielding funds on L2') ; const shieldTx =

this . l2Token . methods . shield (this . ownerAddress , shieldAmount , secretHash ,

0) . send ( ) ; const shieldReceipt =

await shieldTx . wait ( ) ; expect (shieldReceipt . status) . toBe (TxStatus . MINED) ;

await

this . addPendingShieldNoteToPXE (shieldAmount , secretHash , shieldReceipt . txHash) ; }

async

addPendingShieldNoteToPXE (shieldAmount : bigint , secretHash : Fr , txHash : TxHash)

{ this . logger ('Adding note to PXE') ; const storageSlot =

new

Fr (5) ; const noteTypeId =

new

Fr (84114971101151129711410111011678111116101n) ;

// TransparentNote const note =

new

Note ([new

Fr (shieldAmount) , secretHash]) ; const extendedNote =

new

ExtendedNote (note , this . ownerAddress , this . l2Token . address , storageSlot , noteTypeId , txHash ,) ; await

this . pxeService . addNote (extendedNote) ; }

async

redeemShieldPrivatelyOnL2 (shieldAmount : bigint , secret : Fr)

{ this . logger ('Spending note in private call') ; const privateTx =

this . l2Token . methods . redeem_shield (this . ownerAddress , shieldAmount , secret) . send ( ) ; const privateReceipt =

await privateTx . wait ( ) ; expect (privateReceipt . status) . toBe (TxStatus . MINED) ; }

async

unshieldTokensOnL2 (unshieldAmount : bigint , nonce = Fr . ZERO)

{ this . logger ('Unshielding tokens') ; const unshieldTx =

this . l2Token . methods . unshield (this . ownerAddress ,

this . ownerAddress , unshieldAmount , nonce) . send ( ) ; const unshieldReceipt =

await unshieldTx . wait ( ) ; expect (unshieldReceipt . status) . toBe (TxStatus . MINED) ; }Source code: yarn-project/end-to-end/src/shared/cross_chain_test_harness.ts#L1-L455 This

- gets your Solidity contract ABIs
- uses Aztec.js to deploy them to Ethereum
- uses Aztec.js to deploy the token and token bridge contract on L2, sets the bridge's portal address totokenPortalAddress
- and initializes all the contracts
- exposes easy to use helper methods to interact with our contracts.

Now let's write our tests.

We will write two tests:

1. Test the deposit and withdraw in the private flow
2. Do the same in the public flow

Opencross_chain_messaging.test.ts and paste the initial description of the test:

import

{ expect , jest }

from

'@jest/globals' import

{ AccountWallet , AztecAddress , DebugLogger , EthAddress , Fr , computeAuthWitMessageHash , createDebugLogger , createPXEClient , waitForSandbox }

from

'@aztec/aztec.js' ; import

{ getSandboxAccountsWallets }

from

'@aztec/accounts/testing' ; import

{ TokenContract }

from

'@aztec/noir-contracts.js/Token' ; import

{ TokenBridgeContract }

from

'@aztec/noir-contracts.js/TokenBridge' ;

import

{ CrossChainTestHarness }

from

'./shared/cross_chain_test_harness.js' ; import

{ delay }

from

'./fixtures/utils.js' ; import

{ mnemonicToAccount }

from

'viem/accounts' ; import

{ createPublicClient , createWalletClient , http }

from

'viem' ; import

{ foundry }

from

'viem/chains' ;

const

{

PXE_URL

=

'http://localhost:8080' ,

ETHEREUM_HOST

=

'http://localhost:8545'

}

= process . env ; const

MNEMONIC

=

'test test test test test test test test test test test junk' ; const hdAccount =

mnemonicToAccount (MNEMONIC) ;

describe ('e2e_cross_chain_messaging' ,

( )

=>

{ jest . setTimeout (90_000) ;

let logger : DebugLogger ; // include code: let user1Wallet : AccountWallet ; let user2Wallet : AccountWallet ; let ethAccount : EthAddress ; let ownerAddress : AztecAddress ;

let crossChainTestHarness : CrossChainTestHarness ; let l2Token : TokenContract ; let l2Bridge : TokenBridgeContract ; let

outbox :

any ;

beforeEach (async

( )

=>

{ logger =

createDebugLogger ('aztec:e2e_uniswap') ; const pxe =

createPXEClient (PXE_URL) ; await

waitForSandbox (pxe) ; const wallets =

await

getSandboxAccountsWallets (pxe) ;

const walletClient =

createWalletClient ({ account : hdAccount , chain : foundry , transport :

http (ETHEREUM_HOST) , }) ; const publicClient =

createPublicClient ({ chain : foundry , transport :

http (ETHEREUM_HOST) , }) ;

crossChainTestHarness

await CrossChainTestHarness . new (pxe , publicClient , walletClient , wallets [0] , logger ,) ;

l2Token

crossChainTestHarness . l2Token ; l2Bridge = crossChainTestHarness . l2Bridge ; ethAccount = crossChainTestHarness . ethAccount ; ownerAddress = crossChainTestHarness . ownerAddress ; outbox = crossChainTestHarness . outbox ; user1Wallet = wallets [0] ; user2Wallet = wallets [1] ; logger = logger ; logger ('Successfully deployed contracts and initialized portal') ; }) ; This fetches the wallets from the sandbox and deploys our cross chain harness on the sandbox!

Private flow test

e2e_private_cross_chain it ('Privately deposit funds from L1 -> L2 and withdraw back to L1' ,

async

( )

=>

{ // Generate a claim secret using pedersen const l1TokenBalance =

1000000n ; const bridgeAmount =

100n ;

const

[secretForL2MessageConsumption , secretHashForL2MessageConsumption]

= crossChainTestHarness . generateClaimSecret ( ) ; const

[secretForRedeemingMintedNotes , secretHashForRedeemingMintedNotes]

= crossChainTestHarness . generateClaimSecret ( ) ;

// 1. Mint tokens on L1 await crossChainTestHarness . mintTokensOnL1 (l1TokenBalance) ;

// 2. Deposit tokens to the TokenPortal const entryKeyInbox =

await crossChainTestHarness . sendTokensToPortalPrivate (secretHashForRedeemingMintedNotes , bridgeAmount , secretHashForL2MessageConsumption ,) ; expect (await crossChainTestHarness . getL1BalanceOf (ethAccount)) . toBe (l1TokenBalance - bridgeAmount) ; expect (await crossChainTestHarness . inbox . read . contains ([entryKeyInbox . toString ( )])) . toBeTruthy ( ) ;

// Wait for the archiver to process the message await

delay (5000) ;

/// waiting 5 seconds.

// Perform an unrelated transaction on L2 to progress the rollup. Here we mint public tokens. const unrelatedMintAmount =

99n ; await crossChainTestHarness . mintTokensPublicOnL2 (unrelatedMintAmount) ; await crossChainTestHarness . expectPublicBalanceOnL2 (ownerAddress , unrelatedMintAmount) ;

// 3. Consume L1-> L2 message and mint private tokens on L2 await crossChainTestHarness . consumeMessageOnAztecAndMintSecretly (secretHashForRedeemingMintedNotes , bridgeAmount , secretForL2MessageConsumption ,) ; // tokens were minted privately in a TransparentNote which the owner (person who knows the secret) must redeem: await crossChainTestHarness . redeemShieldPrivatelyOnL2 (bridgeAmount , secretForRedeemingMintedNotes) ; await crossChainTestHarness . expectPrivateBalanceOnL2 (ownerAddress , bridgeAmount) ;

// time to withdraw the funds again! logger ('Withdrawing funds from L2') ;

// 4. Give approval to bridge to burn owner's funds: const withdrawAmount =

9n ; const nonce = Fr . random ( ) ; const burnMessageHash =

computeAuthWitMessageHash (l2Bridge . address , l2Token . methods . burn (ownerAddress , withdrawAmount , nonce) . request ( ) ,) ; const witness =

await user1Wallet . createAuthWitness (burnMessageHash) ; await user1Wallet . addAuthWitness (witness) ;

// 5. Withdraw owner's funds from L2 to L1 const entryKey =

await crossChainTestHarness . checkEntryIsNotInOutbox (withdrawAmount) ; await crossChainTestHarness . withdrawPrivateFromAztecToL1 (withdrawAmount , nonce) ; await crossChainTestHarness . expectPrivateBalanceOnL2 (ownerAddress , bridgeAmount - withdrawAmount) ;

// Check balance before and after exit. expect (await crossChainTestHarness . getL1BalanceOf (ethAccount)) . toBe (l1TokenBalance - bridgeAmount) ; await crossChainTestHarness . withdrawFundsFromBridgeOnL1 (withdrawAmount , entryKey) ; expect (await crossChainTestHarness . getL1BalanceOf (ethAccount)) . toBe (l1TokenBalance - bridgeAmount + withdrawAmount) ;

expect (await outbox . read . contains ([entryKey . toString ( )])) . toBeFalsy ( ) ; } ,

120_000) ;Source code: yarn-project/end-to-end/src/e2e_cross_chain_messaging.test.ts#L60-L128

Public flow test

e2e_public_cross_chain it ('Publicly deposit funds from L1 -> L2 and withdraw back to L1' ,

async

( )

=>

{ // Generate a claim secret using pedersen const l1TokenBalance =

1000000n ; const bridgeAmount =

100n ;

const

[secret , secretHash]

= crossChainTestHarness . generateClaimSecret ( ) ;

// 1. Mint tokens on L1 await crossChainTestHarness . mintTokensOnL1 (l1TokenBalance) ;

// 2. Deposit tokens to the TokenPortal await crossChainTestHarness . sendTokensToPortalPublic (bridgeAmount , secretHash) ; expect (await crossChainTestHarness . getL1BalanceOf (ethAccount)) . toBe (l1TokenBalance - bridgeAmount) ;

// Wait for the archiver to process the message await

sleep (5000) ;

// waiting 5 seconds.

// Perform an unrelated transaction on L2 to progress the rollup. Here we mint public tokens. const unrelatedMintAmount =

99n ; await crossChainTestHarness . mintTokensPublicOnL2 (unrelatedMintAmount) ; await crossChainTestHarness . expectPublicBalanceOnL2 (ownerAddress , unrelatedMintAmount) ; const balanceBefore = unrelatedMintAmount ;

// 3. Consume L1 -> L2 message and mint public tokens on L2 await crossChainTestHarness . consumeMessageOnAztecAndMintPublicly (bridgeAmount , secret) ; await crossChainTestHarness . expectPublicBalanceOnL2 (ownerAddress , balanceBefore + bridgeAmount) ; const afterBalance = balanceBefore + bridgeAmount ;

// time to withdraw the funds again! logger ('Withdrawing funds from L2') ;

// 4. Give approval to bridge to burn owner's funds: const withdrawAmount =

9n ; const nonce = Fr . random ( ) ; const burnMessageHash =

computeAuthWitMessageHash (l2Bridge . address , l2Token . methods . burn_public (ownerAddress , withdrawAmount , nonce) . request ( ) ,) ; await user1Wallet . setPublicAuth (burnMessageHash ,

true) . send ( ) . wait ( ) ;

// 5. Withdraw owner's funds from L2 to L1 const entryKey =

await crossChainTestHarness . checkEntryIsNotInOutbox (withdrawAmount) ; await crossChainTestHarness . withdrawPublicFromAztecToL1 (withdrawAmount , nonce) ; await crossChainTestHarness . expectPublicBalanceOnL2 (ownerAddress , afterBalance - withdrawAmount) ;

// Check balance before and after exit. expect (await crossChainTestHarness . getL1BalanceOf (ethAccount)) . toBe (l1TokenBalance - bridgeAmount) ; await crossChainTestHarness . withdrawFundsFromBridgeOnL1 (withdrawAmount , entryKey) ; expect (await crossChainTestHarness . getL1BalanceOf (ethAccount)) . toBe (l1TokenBalance - bridgeAmount + withdrawAmount) ;

expect (await outbox . read . contains ([entryKey . toString ( )])) . toBeFalsy ( ) ; } ,

120_000) ;Source code: yarn-project/end-to-end/src/e2e_public_cross_chain_messaging.test.ts#L79-L132

Running the test

cd packages/src DEBUG = 'aztec:e2e_uniswap'

yarn

test

Error handling

Note - you might have a jest error at the end of each test saying "expected 1-2 arguments but got 3". In case case simply remove the "120_000" at the end of each test. We have already set the timeout at the top so this shouldn't be a problem.
Edit this page

Previous Withdrawing to L1 Next Build an Aztec Connect-style Uniswap