Contract interactions

In this section, we'll write the logic in our app that will interact with the contract we have previously deployed. We'll be using the accounts already seeded in the Sandbox.

Showing user balance

Let's start by showing our user's private balance for the token across their accounts. To do this, we can leverage thebalance_of_private unconstrained view function of the token contract:

balance_of_private unconstrained fn

balance_of_private (owner :

AztecAddress)

->

pub

Field

{ storage . balances . balance_of (owner) . to_integer ( ) }Source code: noir-projects/noir-contracts/contracts/token_contract/src/main.nr#L355-L359 info Note that this function will only return a valid response for accounts registered in the Private eXecution Environment (PXE), since it requires access to theuser's private state . In other words, you cannot query the private balance of another user for the token contract. To do this, let's first initialize a newContract instance usingaztec.js that represents our deployed token contracts. Create a newsrc/contracts.mjs file with the imports for our artifacts and other dependencies:

// src/contracts.mjs import

{ Contract }

from

"@aztec/aztec.js" ; import

{ readFileSync }

from

"fs" ; import TokenContractJson from

"../contracts/token/target/token_contract-Token.json"

assert

{

type :

"json"

} ; And then add the following code for initializing theContract instances:

get-tokens export

async

function

getToken (client)

{ const addresses =

JSON . parse (readFileSync ('addresses.json')) ; return Contract . at (AztecAddress . fromString (addresses . token) ,

loadContractArtifact (TokenContractJson) , client) ; }Source code: yarn-project/end-to-end/src/sample-dapp/contracts.mjs#L6-L11 info You can use the typescript autogenerated interface instead of the genericContract class to get type-safe methods. We can now get the token instance in our main code insrc/index.mjs , and query the private balance for each of the user accounts. To query a function, without sending a transaction, use theview function of the method:

showPrivateBalances const accounts =

await pxe . getRegisteredAccounts ( ) ; const token =

await

getToken (pxe) ;

for

(const account of accounts)

{ const balance =

await token . methods . balance_of_private (account . address) . view ( ) ; console . log Balance of { account . address } : { balance }) ; } Source code: yarn-project/end-to-end/src/sample-dapp/index.mjs#L17-L26 Run this asnode src/index.mjs and you should now see the following output:

Balance of 0x0c8a6673d7676cc80aaebe7fa7504cf51daa90ba906861bfad70a58a98bf5a7d: 100 Balance of 0x226f8087792beff8d5009eb94e65d2a4a505b70baf4a9f28d33c8d620b0ba972: 0 Balance of 0x0e1f60e8566e2c6d32378bdcadb7c63696e853281be798c107266b8c3a88ea9b: 0

Transferring private tokens

Now that we can see the balance for each user, let's transfer tokens from one account to another. To do this, we will first need access to aWallet object. This wraps access to an PXE and also provides an interface to craft and sign transactions on behalf of one of the user accounts.

We can initialize a wallet using one of thegetAccount methods from theaccounts package, along with the corresponding signing and encryption keys:

import

{ getSchnorrAccount }

from

"@aztec/accounts/schnorr" ; const wallet =

await

getSchnorrAccount (client , ENCRYPTION_PRIVATE_KEY , SIGNING_PRIVATE_KEY) . getWallet ( ) ; For ease of use,accounts also ships with a helpergetInitialTestAccountsWallets method that returns a wallet for each of the pre-initialized accounts in the Sandbox, so you can send transactions as any of them.

import

{ getInitialTestAccountsWallets }

from

"@aztec/accounts/testing" ; We'll use one of these wallets to initialize theContract instance that represents our private token contract, so every transaction sent through it will be sent through that wallet.

transferPrivateFunds const

[owner , recipient]

=

await

getInitialTestAccountsWallets (pxe) ; const token =

await

getToken (owner) ;

const tx = token . methods . transfer (owner . getAddress ( ) , recipient . getAddress ( ) ,

1n ,

0) . send ( ) ; console . log (Sent transfer transaction { await tx . getTxHash ( ) }) ; await

showPrivateBalances (pxe) ;

console . log (Awaiting transaction to be mined) ; const receipt =

await tx . wait ( ) ; console . log (Transaction has been mined on block { receipt . blockNumber }) ; await

showPrivateBalances (pxe) ;Source code: yarn-project/end-to-end/src/sample-dapp/index.mjs#L60-L72 Let's go step-by-step on this snippet. We first get wallets for two of the Sandbox accounts, and name themowner andrecipient . Then, we initialize the private tokenContract instance using theowner wallet, meaning that any transactions sent through it will have theowner as sender.

Next, we send a transfer transaction, moving 1 unit of balance to therecipient account address. This has no immediate effect, since the transaction first needs to be simulated locally and then submitted and mined. Only once this has finished we can query the balances again and see the effect of our transaction. We are using ashowPrivateBalances helper function here which has the code we wrote in the section above.

Run this new snippet and you should see the following:

Sent transfer transaction 16025a7c4f6c44611d7ac884a5c27037d85d9756a4924df6d97fb25f6e83a0c8

Balance of 0x0c8a6673d7676cc80aaebe7fa7504cf51daa90ba906861bfad70a58a98bf5a7d: 100 Balance of 0x226f8087792beff8d5009eb94e65d2a4a505b70baf4a9f28d33c8d620b0ba972: 0 Balance of 0x0e1f60e8566e2c6d32378bdcadb7c63696e853281be798c107266b8c3a88ea9b: 0

Awaiting transaction to be mined Transaction has been mined on block 4

Balance of 0x0c8a6673d7676cc80aaebe7fa7504cf51daa90ba906861bfad70a58a98bf5a7d: 99 Balance of 0x226f8087792beff8d5009eb94e65d2a4a505b70baf4a9f28d33c8d620b0ba972: 1 Balance of 0x0e1f60e8566e2c6d32378bdcadb7c63696e853281be798c107266b8c3a88ea9b: 0 info At the time of this writing, there are no events emitted when new private notes are received, so the only way to detect of a change in a user's private balance is via polling on every new block processed. This will change in a future release.

Working with public state

Whileprivate and public state are fundamentally different, the API for working with private and public functions and state fromaztec.js is equivalent. To query the balance in public tokens for our user accounts, we can just call thebalance_of_public view function in the contract:

showPublicBalances const accounts =

await pxe . getRegisteredAccounts ( ) ; const token =

await

getToken (pxe) ;

for

(const account of accounts)

{ const balance =

await token . methods . balance_of_public (account . address) . view ( ) ; console . log(Balance of { account . address } : { balance }) ; }Source code: yarn-project/end-to-end/src/sample-dapp/index.mjs#L76-L85 info Since this we are working with pubic balances, we can now query the balance for any address, not just those registered in our local PXE. We can also send funds to addresses for which we don't know theirpublic encryption key . Here, since the token contract does not mint any initial funds upon deployment, the balances for all of our user's accounts will be zero. But we can send a transaction to mint tokens, using very similar code to the one for sending private funds:

mintPublicFunds const

[owner]

=

await

getInitialTestAccountsWallets (pxe) ; const token =

await

```
getToken ( owner ) ;

const tx = token . methods . mint_public ( owner . getAddress ( ) ,

100n ) . send ( ) ; console . log (Sent mint transaction { await tx . getTxHash ( ) } ) ; await

showPublicBalances ( pxe ) ;

console . log ( Awaiting transaction to be mined ) ; const receipt =

await tx . wait ( ) ; console . log (Transaction has been mined on block { receipt . blockNumber } ) ; await

showPublicBalances ( pxe ) ;
```

Source code: yarn-project/end-to-end/src/sample-dapp/index.mjs#L89-L101 And get the expected results:

```
Sent mint transaction 041d5b4cc68bcb5c6cb45cd4c79f893d94f0df0792f66e6fddd7718c049fe925 Balance of 0x0c8a6673d7676cc80aaebe7fa7504cf51daa90ba906861bfad70a58a98bf5a7d: 0 Balance of 0x226f8087792beff8d5009eb94e65d2a4a505b70baf4a9f28d33c8d620b0ba972: 0 Balance of 0x0e1f60e8566e2c6d32378bdcadb7c63696e853281be798c107266b8c3a88ea9b: 0

Awaiting transaction to be mined Transaction has been mined on block 5

Balance of 0x0c8a6673d7676cc80aaebe7fa7504cf51daa90ba906861bfad70a58a98bf5a7d: 100 Balance of 0x226f8087792beff8d5009eb94e65d2a4a505b70baf4a9f28d33c8d620b0ba972: 0 Balance of 0x0e1f60e8566e2c6d32378bdcadb7c63696e853281be798c107266b8c3a88ea9b: 0
```

Next steps

In the next and final section, we'll set up automated tests for our application . Edit this page

Previous Contract Deployment Next Testing