Creating Schnorr Accounts

Introduction

This section shows how to create schnorr account wallets on the Aztec Sandbox.

An in-depth explaining about accounts on aztec can be found here . But creating an account on the Sandbox does 2 things:

1. Deploys an account contract -- representing you -- allowing you to perform actions on the network (deploy contracts, call functions etc).
2. Adds your encryption keys to the Private eXecution Environment (PXE) allowing it to decrypt and manage your private state.

Pre-requisites

Have a running Sandbox and a repository that interacts with it as explained here .

Let's assume you have a filesrc/index.ts from the example used in the Sandbox page.

Create accounts on the sandbox

1. Import relevant modules:

imports import

{ getSchnorrAccount }

from

'@aztec/accounts/schnorr' ; import

{ getDeployedTestAccountsWallets }

from

'@aztec/accounts/testing' ; import

{ ExtendedNote , Fr , GrumpkinScalar , Note , PXE , computeMessageSecretHash , createDebugLogger , createPXEClient , waitForPXE , }

from

'@aztec/aztec.js' ; import

{ TokenContract }

from

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

import

{ format }

from

'util' ;

const

{

PXE_URL

=

'http://localhost:8080'

}

= process . env ;Source code: yarn-project/end-to-end/src/e2e_sandbox_example.test.ts#L1-L20 1. Code to create an account. You must run this inside of a function:

create_accounts ////////////// CREATE SOME ACCOUNTS WITH SCHNORR SIGNERS ////////////// // Creates new accounts using an account contract that verifies schnorr signatures // Returns once the deployment transactions have settled const

createSchnorrAccounts

=

async

(numAccounts :

number , pxe :

PXE)

=>

{ const accountManagers =

Array (numAccounts) . fill (0) . map (( )

=> getSchnorrAccount (pxe , GrumpkinScalar . random ( ) ,

// encryption private key GrumpkinScalar . random ( ) ,

// signing private key) ,) ; return

await

Promise . all (accountManagers . map (async x =>

{ await x . waitSetup ({ }) ; return x ; }) ,) ; } ;

// Create 2 accounts and wallets to go with each logger (Creating accounts using schnorr signers...) ; const accounts =

await

createSchnorrAccounts (2 , pxe) ;

////////////// VERIFY THE ACCOUNTS WERE CREATED SUCCESSFULLY //////////////

const

[alice , bob]

=

(await

Promise . all (accounts . map (x => x . getCompleteAddress ( ))) . map (x => x . address) ;

// Verify that the accounts were deployed const registeredAccounts =

(await pxe . getRegisteredAccounts ( )) . map (x => x . address) ; for

(const

[account , name]

of

[[alice ,

'Alice'] , [bob ,

'Bob'] ,]

as

const)

{ if

(registeredAccounts . find (acc => acc . equals (account)))

{ logger (Created { name } 's account at { account . toShortString ( ) }) ; continue ; } logger (Failed to create account for { name } !) ; }Source code: yarn-project/end-to-end/src/e2e_sandbox_example.test.ts#L183-L225 1. Runningyarn start 2. should now output:

token Aztec Sandbox Info { version: 1, chainId: 31337, rollupAddress: EthAddress { buffer: }, client: 'pxe@0.1.0', compatibleNargoVersion: '0.11.1-aztec.0' } token Creating accounts using schnorr signers... +3ms token Created Alice's account at 0x1509b252...0027 +10s token Created Bob's account at 0x031862e8...e7a3 +0ms That might seem like a lot to digest but it can be broken down into the following steps:

1. We create 2Account
2. objects in Typescript. This object heavily abstracts away the mechanics of configuring and deploying an account contract and setting up a 'wallet' for signing transactions. If you aren't interested in building new types of account contracts or wallets then you don't need to be too concerned with it. In this example we have constructed account contracts and corresponding wallets that sign/verify transactions using schnorr signatures.
3. We wait for the deployment of the 2 account contracts to complete.
4. We retrieve the expected account addresses from theAccount
5. objects and ensure that they are present in the set of account addresses registered on the Sandbox.

Note, we use thegetRegisteredAccounts API to verify that the addresses computed as part of the account contract deployment have been successfully added to the Sandbox.

If you were looking at your terminal that is running the Sandbox you should have seen a lot of activity. This is because the Sandbox will have simulated the deployment of both contracts, executed the private kernel circuit for each account deployment and later on submitted the 2 transactions to the pool. The sequencer will have picked them up and inserted them into an L2 block and executed the recursive rollup circuits before publishing the L2 block on L1 (in our case Anvil). Once this has completed, the L2 block is retrieved and pulled down to the PXE so that any new account state can be decrypted.

Next Steps

Check out our section onWriting your own Account Contract leveraging our account abstraction Edit this page

Previous How to run more than one PXE in the sandbox Next CLI Commands