Passport

Passport is an MPC-based programmable, distributed, and non-custodial key management system, that allows users to generate wallets, scoped to their application, either via user Passkeys or any developer defined authentication method. Our signer allows you to sign messages and transactions with a Passport Network account.

Dependencies

You will need the following dependencies to create a Smart Account this way:

yarn add @biconomy/account viem@2.9.4 @0xpass/passport @0xpass/webauthn-signer @0xpass/passport-viem @0xpass/key-signer

Imports

import

{ Passport ,

TESTNET_RSA_PUBLIC_KEY

}

from

"@0xpass/passport" ; import

{ WebauthnSigner }

from

"@0xpass/webauthn-signer" ; import

{ KeySigner }

from

"@0xpass/key-signer" ; import

{ createPassportClient }

from

"@0xpass/passport-viem" ; import

{ http }

from

"viem" ; import

{ sepolia }

from

"viem/chains" ; import

{ createSmartAccountClient }

from

"@biconomy/account" ;

Passport Configuration

To use Passport, you'll first need to make sure you have configured a scope for your application. For this you can follow the guides below:

- Refer to thePassport documentation
- for instructions on setting up your application with Passport.

- For a primer on setting up your scope you can checkhere
- .

Passkey Signer

Passport offers a Passkey signer, which follows the webauthn standard, which is a user owned authentication method that allows users to sign messages and transactions with their Passkey.

// The rpId and rpName are the same as the ones you set up in your passport application scope. They follow the webauthn standard, of the following values // rpId: the domain of where the passkey is generated // rpName: human readable name for the domain // You can read more on this here https://docs.0xpass.io/authentication/configuring-your-scope#scope-configuration const webauthnSigner =

new

WebAuthnSigner ({ rpId :

"rpId" , rpName :

"rpName" , }) ;

const passport =

new

Passport ({ scope_id :

"scope_id" ,

// <-- Replace this with your scope id you get after configuring your scope by following the Passport documentation signer : webauthnSigner , enclave_public_key :

TESTNET_RSA_PUBLIC_KEY , }) ;

const fallbackProvider =

http ("https://rpc.ankr.com/eth_sepolia") ;

await passport . setupEncryption ( ) ; // Only need to call this once to register the user passkey to your passport application scope await passport . register ({ username :

"test" , userDisplayName :

"test" , }) ;

const

[authenticatedHeader]

=

await passport . authenticate ({ username :

"test" , userDisplayName :

"test" , }) ;

const signer =

await

createPassportClient (authenticatedHeader , fallbackProvider , sepolia , "https://tiramisu.0xpass.io") ;

Developer Owned Authentication Signer

Passport also offers Developer Owned Authentication (DOA) signers, which allows developers to authenticate users with their own preferred authentication methods. This can be a social login, a hardware wallet, or any other authentication method that the developer chooses to implement. To setup a DOA signer, you'll need to setup a private key, which you can do by following the instructionhere

const keySigner =

new

KeySigner (process . env . PRIVATE_KEY ,

true) ;

const passport =

new

Passport ({ scope_id :

"scope_id"

// <-- Replace this with your scope id you get after configuring your scope by following the Passport documentation, signer : keySigner , enclave_public_key :

TESTNET_RSA_PUBLIC_KEY , }) ;

const fallbackProvider =

http ("https://rpc.ankr.com/eth_sepolia") ;

await passport . setupEncryption ( ) ; // Only need to call this once to register the user identifier you authenticate with to your passport application scope await passport . delegatedRegisterAccount ({ username :

"test"

}) ;

passport . setUserData ({ username :

"test"

}) ;

const signer =

await

createPassportClient (await passport . getDelegatedAuthenticatedHeaders ( ) , fallbackProvider , sepolia , "https://tiramisu.0xpass.io") ;

Create the Biconomy Smart Account

const

connect

=

async

( )

=>

{ try

{ const smartAccount =

await

createSmartAccountClient ({ signer : signer as LightSigner ,

// Your signer can be either a WebAuthnSigner or a KeySigner bundlerUrl :

"" ,

// <-- Read about this at https://docs.biconomy.io/dashboard#bundler-url biconomyPaymasterApiKey :

"" ,

// <-- Read about at https://docs.biconomy.io/dashboard/paymaster rpcUrl :

"" ,

// <-- read about this at https://docs.biconomy.io/Account/methods#createsmartaccountclient }) ;

const address =

await smartAccount . getAccountAddress ( ) ; }

catch

(error)

{ console . error (error) ; } } Previous Web3Auth Next Enable Fiat On Ramp