Create a batch session

Overview

This tutorial demonstrates how a dapp can create a number of simple sessions using viem and the Biconomy Smart Account with the@biconomy/account SDK. The provided code assumes you have a Biconomy Paymaster API key and a connected user. The following is appropriately viewed from the perspective of a dapp, looking to make txs on a users behalf.

You can get your Biconomy Paymaster API key from the dashboardhere .

Prerequisites

- Biconomy Paymaster API key
- A Bundler url if you don't want to use the testnet one (for Amoy you can usehttps://bundler.biconomy.io/api/v2/80002/nJPK7B3ru.dd7f7861-190d-41bd-af80-6877f74b8f44
-)
- A user with a connected signer (viem WalletClient or ethers.Wallet for example)

Step 1: Create the SmartAccountClient for the user

import

{ polygonAmoy as chain }

from

"viem/chains" ; import

{ PaymasterMode , createSmartAccountClient , createSession , Rule , Policy , }

from

"@biconomy/account" ;

const nftAddress =

"0x1758f42Af7026fBbB559Dc60EcE0De3ef81f665e" ; const token =

"0x747A4168DB14F57871fa8cda8B5455D8C2a8e90a" ; const amount =

parseUnits (".0001" ,

6) ; const withSponsorship =

{ paymasterServiceData :

{ mode : PaymasterMode . SPONSORED

} , } ;

// Create Biconomy Smart Account instance const usersSmartAccount =

await

createSmartAccountClient ({ signer : usersWalletClient ,

// assumes that a user has connected his walletClient (or an ethers Wallet) to your dapp biconomyPaymasterApiKey : config . biconomyPaymasterApiKey , bundlerUrl : config . bundlerUrl , }) ;

Step 2: Generate a store for your dapp's session keys

This function is used to create a new session key and store it in the sessionStorageClient. You can feed the sessionStorageClient into thecreateSessionKeyEOA(...args) as the third argument. If you do not provide a sessionStorageClient then one will get generated for you based on the environment. When localStorage is supported, it will return aSessionLocalStorage store, otherwise it will assume you are in a backend and useSessionFileStorage store. Seehere for detail regarding creating your own session storage client.

const

{ sessionKeyAddress , sessionStorageClient }

=

await

createSessionKeyEOA (usersSmartAccount , chain) ;

Step 3: Create the multiple policy leaves

Next we need to generate the criteria leafs that comprise the policy over which we can use to request permission (granted via the users signature). The policy is comprised of a list of leaves applied to a single contract method, along with an interval over which the session remains valid. Here we can use two different helpers for each of the two leaf types

info There is no need to choose to usecreateERC20SessionDatum overcreateABISessionDatum for your sessions. For the purposes of this demo we have used two different types to highlight how different session types might be used in the same batch, but the latter is always preferable over the former as it is a much more flexible module. const erc20SessionLeaf : CreateSessionDataParams =

createERC20SessionDatum ({ interval :

{ validUntil :

0 , validAfter :

0 , } , sessionKeyAddress , sessionKeyData :

encodeAbiParameters ([{ type :

"address"

} , { type :

"address"

} , { type :

"address"

} , { type :

"uint256"

} ,] , [sessionKeyAddress , token , recipient , amount]) , }) ;

const abiSessionLeaf : CreateSessionDataParams =

createABISessionDatum ({ interval :

{ validUntil :

0 , validAfter :

0 , } , sessionKeyAddress , contractAddress : nftAddress , functionSelector :

"safeMint(address)" , rules :

[{ offset :

0 , condition :

0 , referenceValue : usersSmartAccount , } ,] , valueLimit :

0n , }) ;

const policyLeaves : CreateSessionDataParams [ ]

=

[erc20SessionLeaf , abiSessionLeaf ,] ;

Step 4: Request Permission from the user with the policy leaves

The session keys are imbibed with the relevant permissions when the user signs over the policy. The session can then be accessed from the sessionStorageClient and later used, even after the usersSmartAccount signer has left the dapp.

const

{ wait , session }

=

await

createBatchSession (smartAccountFour , sessionKeyAddress , sessionStorageClient , policyLeaves , withSponsorship) ;

const

{ receipt :

{ transactionHash } , success , }

=

await

wait ( ) ; Send the transaction using the Biconomy Smart Account and get the transaction hash. The transaction will be built into a User Operation and then send to the Bundler.

That's it! You've successfully requested permissions from your user, and stored session keys which can later be used to make txs on their behalf. Previous Use a session Next Use a batch session