Use a batch session

Overview

This tutorial demonstrates how to use a multi session with viem and the Biconomy Smart Account with the@biconomy/account SDK. The provided code assumes you have a Biconomy Paymaster API key, and a valid session setup during theprevious step . 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 session granted by a usersee the previous step
- , with the sessionKey for the session

Step 1

import

{ polygonAmoy as chain }

from

"viem/chains" ; import

{ PaymasterMode , createSessionSmartAccountClient , createSession , Rule , Policy , DEFAULT_ERC20_MODULE , DEFAULT_ABI_SVM_MODULE , }

from

"@biconomy/account" ;

const nftAddress =

"0x1758f42Af7026fBbB559Dc60EcE0De3ef81f665e" ; const token =

"0x747A4168DB14F57871fa8cda8B5455D8C2a8e90a" ; const amount =

parseUnits (".0001" ,

6) ; const withSponsorship =

{ paymasterServiceData :

{ mode : PaymasterMode . SPONSORED

} , } ;

// session data retreived from the previous step. const session =

{ sessionStorageClient , sessionID , } ;

Step 2: Create the sessionSmartAccountClient

ThecreateSessionSmartAccountClient helper here is very similar to thecreateSmartAccountClient , in that it returns aBiconomySmartAccountV2 , but there are subtle differences in the configuration. createSessionSmartAccountClient expectsno signer and amandatory smartAccountAddress in the config (so that a dapp can emulate the users smartAccount without the original signer).

const emulatedUsersSmartAccount =

await

createSessionSmartAccountClient ({ accountAddress : sessionStorageClient . smartAccountAddress ,

// Dapp can set the account address on behalf of the user bundlerUrl , paymasterUrl , chainId , } , session , true

// if in batch session mode) ;

Step 3: Prepare the txs

Next we can use the users smartAccount in execute txs. Txs generated for the emulatedSmartAccount must be relevant to the policies over which the user has previously signed.

const transferTx : Transaction =

{ to : token , data :

encodeFunctionData ({ abi :

parseAbi (["function transfer(address _to, uint256 _value)"]) , functionName :

"transfer" , args :

[recipient , amount] , }) , } ; const nftMintTx : Transaction =

{ to : nftAddress , data :

encodeFunctionData ({ abi :

parseAbi (["function safeMint(address _to)"]) , functionName :

"safeMint" , args :

[sessionStorageClient . smartAccountAddress] , }) , } ;

Step 4: Execute txs on the users behalf

When using multiple session and sending a tx the batchSessionParams must be sent with each tx. BatchSessionParamsare described here . The order of thesessionValidationTypes array must correspond with the modules for the txs being sent. A utility functiongetBatchSessionTxParams is provided

const txs =

[transferTx , nftMintTx] ; const batchSessionParams =

await

getBatchSessionTxParams (["ERC20" ,

"ABI"] , [transferTx , nftMintTx] ,

// Order must match ^above fields session , chain) ;

const

{ wait }

=

await emulatedUsersSmartAccount . sendTransaction ([transferTx , nftMintTx] , { ... batchSessionParams , ... withSponsorship , }) ;

const

{ success }

=

await

wait ( ) ; Send the transaction using the users smartAccount 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 emulated a users smartAccount to execute multiple transactions using multiple sessions on their behalf. Previous Create a batch session Next Create SessionStorageClient