Send parallel user ops using 2D nonce

Overview

This tutorial demonstrates how to send parallel user operations using 2D nonces.

What is a 2D Nonce? A 2D nonce is used in smart accounts to manage transaction ordering and parallel execution. Unlike traditional EOAs (Externally Owned Accounts) where nonces are strictly sequential (1, 2, 3, ...), smart accounts use a two-dimensional nonce composed of a "nonce key" and a "nonce value." This allows for more complex transaction dependencies and parallel processing.

As long as one user operation is not dependent on another, it can be sent in parallel.

For example, having a look at this user action flow:

- Swap 100 USDC to DAI (tx 1)
- Swap 100 DAI to USDT (tx 2)
- Swap 1 WETH to USDT (tx 3)

We can see that tx 1 and 2 are dependent, these two cannot be executed in parallel but tx 3 can, as it is not dependent on the others

Prerequisites

- Node.js installed on your machine
- 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
-)
- An rpc url (for Amoy can usehttps://rpc-amoy.polygon.technology/
-)
- An address to send the transaction to (replace0xaddress
-)

Step 1: Generate the config and Create Biconomy Smart Account

- viem
- ethers

import

{ createWalletClient }

from

"viem" ; import

{ privateKeyToAccount }

from

"viem/accounts" ; import

{ polygonAmoy }

from

"viem/chains" ; import

{ createSmartAccountClient , UserOpResponse }

from

"@biconomy/account" ;

// Your configuration with private key and Biconomy API key const config =

{ privateKey :

"your-private-key" , bundlerUrl :

"" ,

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

// Generate EOA from private key using ethers.js const account =

privateKeyToAccount ("0x"

+ config . privateKey) ; const signer =

createWalletClient ({ account , chain : polygonAmoy , transport :

http ( ) , }) ;

// Create Biconomy Smart Account instance const smartWallet =

await

createSmartAccountClient ({ signer , bundlerUrl : config . bundlerUrl , }) ;

const saAddress =

await smartWallet . getAccountAddress ( ) ; console . log ("SA Address" , saAddress) ; import

{ ethers }

from

"ethers" ; import

{ createSmartAccountClient }

from

"@biconomy/account" ;

// Your configuration with private key and Biconomy API key const config =

{ privateKey :

"your-private-key" , bundlerUrl :

"" ,

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

"rpc-url" , } ;

// Generate EOA from private key using ethers.js let provider =

new

ethers . providers . JsonRpcProvider (config . rpcUrl) ; let signer =

new

ethers . Wallet (config . privateKey , provider) ;

// Create Biconomy Smart Account instance const smartWallet =

await

createSmartAccountClient ({ signer , bundlerUrl : config . bundlerUrl , }) ;

const saAddress =

await smartWallet . getAccountAddress ( ) ; console . log ("SA Address" , saAddress) ; Get your signer from either ethers.js or viem and create a Biconomy Smart Account instance.

Step 2: Generate Transaction Data

const toAddress =

"0xaddress" ;

// Replace with the recipient's address const transactionData =

"0x123" ;

// Replace with the actual transaction data

// Build the transaction const tx =

{ to : toAddress , data : transactionData , } ; Specify the recipient's address and transaction data to build the simple transaction.

Step 3: Send the parallel user operations

// Send the transaction and get the transaction hash const numberOfUserOpsToSend =

3 ; const txPromises :

Promise < UserOpResponse

[ ]

=

[ ] ; for

(let key =

1 ; key <= numberOfUserOpsToSend ; key ++)

{ // wrap the tx in a user op and send it // we don't want to await for the user op to be executed because we want to send all users ops at the same time const txPromise = smartAccount . sendTransaction ([tx] ,

{ nonceOptions :

{ nonceKey : key }) ; // save the promise response for later txPromises . push (txPromise) ; }

// later we can resolve the promises to get user operation receipts const userOpResponses : UserOpResponse [ ]

=

await

Promise . all (txPromises) ; userOpResponses . forEach (async

(userOpResponse)

=>

{ const

{ wait }

= userOpResponse ; const receipt =

await

wait ( ) ; console . log ("Receipt: " , receipt) ; }) caution * make sure your nonceKey is bigger than 0, a 0 nonceKey value will not use 2D nonces * make sure your transactions are not dependent one on another * make sure you do not await for each user operation to be executed, otherwise it will defeat the purpose of 2D nonces, causing sequential execution That's it! You've successfully sent 3 parallel user ops using ethers.js/viem and the Biconomy Smart Account. Feel free to customize this example based on your specific use case. Previous Pay gas in ERC20 tokens Next Sessions