Batching Multiple Transactions

In this guide, we will edit the functionality in the previous section to not only mint a gassless transaction but to showcase batching transactions by minting multiple transactions together.

Click to view code from previous section import

{ config }

from

"dotenv" ; import

{ IBundler , Bundler }

from

"@biconomy/bundler" ; import

{ BiconomySmartAccount , BiconomySmartAccountConfig , DEFAULT_ENTRYPOINT_ADDRESS , }

from

"@biconomy/account" ; import

{ Wallet , providers , ethers }

from

"ethers" ; import

{ ChainId }

from

"@biconomy/core-types" ; import

{ IPaymaster , BiconomyPaymaster , IHybridPaymaster , PaymasterMode , SponsorUserOperationDto , }

from

"@biconomy/paymaster" ;

config ( ) ;

const bundler : IBundler =

new

Bundler ({ bundlerUrl : "https://bundler.biconomy.io/api/v2/80001/nJPK7B3ru.dd7f7861-190d-41bd-af80-6877f74b8f44" , chainId : ChainId . POLYGON_MUMBAI , entryPointAddress :

DEFAULT_ENTRYPOINT_ADDRESS , }) ;

const paymaster : IPaymaster =

new

BiconomyPaymaster ({ paymasterUrl : "https://paymaster.biconomy.io/api/v1/80001/Tpk8nuCUd.70bd3a7f-a368-4e5a-af14-80c7f1fcda1a" , }) ;

const provider =

new

providers . JsonRpcProvider ("https://rpc.ankr.com/polygon_mumbai" ,) ; const wallet =

new

Wallet (process . env . PRIVATE_KEY

||

```
"" , provider ) ;

const biconomySmartAccountConfig : BiconomySmartAccountConfig =

{ signer : wallet , chainId : ChainId . POLYGON_MUMBAI , bundler : bundler , paymaster : paymaster , } ;

let smartAccount : BiconomySmartAccount ; let address :

string ;

async

function

createAccount ( )

{ console . log ( "creating address" ) ; let biconomySmartAccount =

new

BiconomySmartAccount ( biconomySmartAccountConfig , ) ; biconomySmartAccount =

await biconomySmartAccount . init ( ) ; address =

await biconomySmartAccount . getSmartAccountAddress ( ) ; smartAccount = biconomySmartAccount ; return
biconomySmartAccount ; }

async

function

mintNFT ( )

{ await

createAccount ( ) ; const nftInterface =

new

ethers . utils . Interface ( [ "function safeMint(address _to)" , ] ) ;

const data = nftInterface . encodeFunctionData ( "safeMint" ,

[ address ] ) ;

const nftAddress =

"0x1758f42Af7026fBbB559Dc60EcE0De3ef81f665e" ;

const transaction =

{ to : nftAddress , data : data , } ;

console . log ( "creating nft mint userop" ) ; let partialUserOp =

await smartAccount . buildUserOp ( [ transaction ] ) ;

const biconomyPaymaster = smartAccount . paymaster as IHybridPaymaster < SponsorUserOperationDto

      ;

let paymasterServiceData : SponsorUserOperationDto =

{ mode : PaymasterMode . SPONSORED , } ; console . log ( "getting paymaster and data" ) ; try

{ const paymasterAndDataResponse = await biconomyPaymaster . getPaymasterAndData ( partialUserOp ,
paymasterServiceData , ) ; partialUserOp . paymasterAndData = paymasterAndDataResponse . paymasterAndData ; }

catch

( e )

{ console . log ( "error received " , e ) ; } console . log ( "sending userop" ) ; try
```

{ const userOpResponse =

await smartAccount . sendUserOp (partialUserOp) ; const transactionDetails =

await userOpResponse . wait ( ) ; console . log (ransactionDetails: https://mumbai.polygonscan.com/tx/ { transactionDetails . receipt . transactionHash } ,) ; console . log (view minted nfts for smart account: https://testnets.opensea.io/ { address } ,) ; }

catch

(e)

{ console . log ("error received " , e) ; } }

mintNFT ( ) ; Our Focus for this edit will be on the following section of the mintNFT function:

const nftInterface =

new

ethers . utils . Interface (["function safeMint(address _to)" ,]) ;

const data = nftInterface . encodeFunctionData ("safeMint" ,

[address]) ;

const nftAddress =

"0x1758f42Af7026fBbB559Dc60EcE0De3ef81f665e" ;

const transaction =

{ to : nftAddress , data : data , } ;

console . log ("creating nft mint userop") ; let partialUserOp =

await smartAccount . buildUserOp ([transaction]) ; We start contructiong our transaction here and pass the transaction to an array within the Smart AccountsbuildUserOp method. The quickest edit we can do here is simply pass the transaction multiple times:

let partialUserOp =

await smartAccount . buildUserOp ([transaction , transaction]) ; Passing the transaction twice in the Array will allow you to mint multiple NFTs in one transaction. This is useful for use cases such as using NFT's as a ticketing system and needing to mint multiple tickets, or creating one click experiences in defi by passing multiple types of transactions into the array.

Try running the script yourself with this edit and you will notice multiple mints in the transaction details, allowing you to save gas in cases where users need to conduct multiple transactions.

In the next section we'll take a look at converting our script from being a gasless transaction to now instead paying for gas using an ERC20 token. Previous Creating Gasless Transactions Next Pay for Gas with ERC20 Tokens