Paying for gas with ERC20 Tokens

In this guide, we will edit the functionality in the gasless transaction section to switch from minting an NFT with no gas costs for the user, to instead minting the NFT and using USDC to pay for the gas costs.

For this tutorial you will need some test USDC tokens. You can useUniswap while connected to the Polygon Mumbai network in your EOA to swap test Matic for USDC POS tokens which are one of the supported tokens on test nets for our SDK. You can get additional test matic at thepolygon faucet .

Make sure to swap for Polygon Mumbai USDC test tokens at this address: 0xda5289fcaaf71d52a80a254da614a192b693e977 onUniswap .

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 , } ) ;

console . log ( { ep :

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 , smartAccountInfo :

{ name :

"BICONOMY" , version :

"2.0.0" , } , } ; 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 ( ) ;
```

Mint NFT starting point

In our mintNFT function (which you can view above) we are going to keep the initial set up of the contract connection but remove code specific to a gasless transaction. Here is a good starting point for the edit on this function:

```
async

function

mintNFT ( )

{ await

createAccount ( ) ; const nftInterface =

new

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

const data = nftInterface . encodeFunctionData ( "safeMint" ,

[ address ] ) ;

const nftAddress =

"0x1758f42Af7026fBbB559Dc60EcE0De3ef81f665e" ; 0xda5289fcaaf71d52a80a254da614a192b693e977 ;

const transaction =

{ to : nftAddress , data : data , } ;

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

await smartAccount . buildUserOp ( [ transaction ] ) ; }
```

Partial UserOP

We've got the connection to the contract as well as the partial userOp constructed. Our paymaster has two key features:

helping sponsor transactions for the user, and allowing the user to pay for gas in ERC20 tokens. Let's get ourselves set up to have users pay for gas in USDC. Add the following lines:

let finalUserOp = partialUserOp ;

const biconomyPaymaster = smartAccount . paymaster as IHybridPaymaster < SponsorUserOperationDto

; We will be manipulating our partialUserOp into finalUserOp before executing it.

ERC20 Fee Quotes

Now lets get fee quotes for our USDC token:

const feeQuotesResponse =

await biconomyPaymaster . getPaymasterFeeQuotesOrData (partialUserOp , { mode : PaymasterMode . ERC20 , tokenList :

["0xda5289fcaaf71d52a80a254da614a192b693e977"] , } ,) ; For the Paymaster Mode we are usingERC20 instead ofSPONSORED and passing a token list with the USDC token address on mumbai. Note that if you pass an empty array you will recieve an array of all supported ERC20 tokens.

const feeQuotes = feeQuotesResponse . feeQuotes as PaymasterFeeQuote [ ] ; const spender = feeQuotesResponse . tokenPaymasterAddress ||

"" ; const usdcFeeQuotes = feeQuotes [0] ; Now we get the response from the fee quotes and specify that the Paymaster will be the eventual spender for the transaction. The user will pay USDC and the Paymaster will pay in native tokens. Lastly since we specified the USDC address the only result in the array will be our USDC fee quote object.

Now let's update the userOp with our gathered information:

finalUserOp

await smartAccount . buildTokenPaymasterUserOp (partialUserOp ,

{ feeQuote : usdcFeeQuotes , spender : spender , maxApproval :

false , }) ;

let paymasterServiceData =

{ mode : PaymasterMode . ERC20 , feeTokenAddress : usdcFeeQuotes . tokenAddress , } ;

Paymaster Service Data response

Now similar to sponsored userOps we need to get the paymasterAndData field of the userOp:

try

{ const paymasterAndDataWithLimits = await biconomyPaymaster . getPaymasterAndData (finalUserOp , paymasterServiceData ,) ; finalUserOp . paymasterAndData = paymasterAndDataWithLimits . paymasterAndData ; }

catch

(e)

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

Execute UserOp

And finally we execute our userop!

try

{ const userOpResponse =

await smartAccount . sendUserOp (finalUserOp) ; const transactionDetails =

await userOpResponse . wait ( ) ; console . log (ransactionDetails: https://mumbai.polygonscan.com/tx/ { transactionDetails . logs [0] .

transactionHash } ,) ; console . log (view minted nfts for smart account: https://testnets.opensea.io/ { address } ,) ; }

catch

(e)

{ console . log ("error received " , e) ; } Click to view final code 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 , PaymasterFeeQuote , }

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 , }) ;

console . log ({ ep :

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 ( )
{ let biconomySmartAccount =
new
BiconomySmartAccount ( biconomySmartAccountConfig , ) ; biconomySmartAccount =
await biconomySmartAccount . init ( ) ; address =
await biconomySmartAccount . getSmartAccountAddress ( ) ; console . log ( address ) ; 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 ] ) ;
let finalUserOp = partialUserOp ;
const biconomyPaymaster = smartAccount . paymaster as IHybridPaymaster < SponsorUserOperationDto
       ;
const feeQuotesResponse =
await biconomyPaymaster . getPaymasterFeeQuotesOrData ( partialUserOp , { mode : PaymasterMode . ERC20 , tokenList
:
```

["0xda5289fcaaf71d52a80a254da614a192b693e977"] , } ,) ;

const feeQuotes = feeQuotesResponse . feeQuotes as PaymasterFeeQuote [ ] ; const spender = feeQuotesResponse . tokenPaymasterAddress ||

"" ; const usdcFeeQuotes = feeQuotes [0] ;

finalUserOp

await smartAccount . buildTokenPaymasterUserOp (partialUserOp ,

{ feeQuote : usdcFeeQuotes , spender : spender , maxApproval :

false , }) ;

let paymasterServiceData =

{ mode : PaymasterMode . ERC20 , feeTokenAddress : usdcFeeQuotes . tokenAddress , } ;

try

{ const paymasterAndDataWithLimits = await biconomyPaymaster . getPaymasterAndData (finalUserOp , paymasterServiceData ,) ; finalUserOp . paymasterAndData = paymasterAndDataWithLimits . paymasterAndData ; }

catch

(e)

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

try

{ const userOpResponse =

await smartAccount . sendUserOp (finalUserOp) ; const transactionDetails =

await userOpResponse . wait ( ) ; console . log (ransactionDetails: https://mumbai.polygonscan.com/tx/ { transactionDetails . logs [0] . transactionHash } ,) ; console . log (view minted nfts for smart account: https://testnets.opensea.io/ { address } ,) ; }

catch

(e)

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

mintNFT ( ) ; Congratulations you can now execute gasless transactions, batched transactions, and ERC20 gas payment transactions utilizing the Biconomy SDK! Previous Batching Multiple Transactions Next React + Vite Fullstack Tutorial