Paymaster ERC20

This tutorial represents the API integration flow where ERC20 tokens in smart wallet are used to sponsor the transactions. paymaster will cover the gas fees and equivalent ERC20 tokens will be deducted from user's smart account.

Pre-requisites:

- Biconomy bundler url (refer to the[docs](docs)
- to get the same)
- Partial user operation, where* sender is the smart account Address
-
    - nonce can be calculated using the smart account contract methods
-
    - initCode will be 0x, if the account is already deployed or can be fetched using contract methods
-
    - paymasterAndData will be 0x initially.
-
    - callData is the abi encoded form of transaction
-
    - It also requires putting a semi-valid/ dummy signature (e.g. a signature in the right length).

```
type

UserOperation

=

{ sender :

string ; nonce :

string ; initCode :

string ; callData :

string ; callGasLimit :

string ; verificationGasLimit :

string ; preVerificationGas :

string ; maxFeePerGas :

string ; maxPriorityFeePerGas :

string ; paymasterAndData :

string ; signature :

string ; }

let partialUserOp : Partial < UserOperation

=

{ sender :

'0x4dF23B78543F5c2F9CBCDF09956288B3e97bb9a4' , nonce :

'0x08' , initCode :

"0x" , paymasterAndData :

"0x" , callData :

"0x0000189a000000000000000000000000322af0da66d00be980c7aa006377fcaaeee3bdfd00000000000000000000000000000000000000000000000000002386f26fc1000000000000000000000000000000
, signature :

"0x0000000000000000000000000000000000000000000000000000000000000040000000000000000000000000000001c5b32F37F5beA87BDD5374eB2aC54eA8e000000000000000000000000000
, }
```

1. Calculate Gas estimations:

Based on the response, update the userOp gas values. Pass the dummy values to gas limits, which will be updated based on the paymaster call.

```
async

function

getGasFeeValues

( userOp : Partial < UserOperation

        )

:

Promise < UserOperation

{ const url = "https://bundler.biconomy.io/api/v2/80001/nJPK7B3ru.dd7f7861-190d-41bd-af80-6877f74b8f44"

const

{ data }

=

await axios . post ( url ,

{ jsonrpc :

"2.0" , method :

"biconomy_getGasFeeValues" , params :

[ ] , id : Date . now ( ) } ) ; const

{ maxPriorityFeePerGas , maxFeePerGas }

= data . result ; return

{

... userOp , maxPriorityFeePerGas , maxFeePerGas , callGasLimit :
```

"5000000" , verificationGasLimit :

"5000000" , preVerificationGas :

"5000000"

}

as UserOperation ; }

2. Token approval

In order to sponsor transactions using ERC20 tokens in smart account, first get the FeeQuotes for the tokens. Append the token approval transaction to the userOp callData. You can either go for the limited or max approval for token depending on the use case.

- Limited approval:
- In this case, approval will be batched for the amount that is required to cover the transaction fees. Callpm_getFeeQuoteOrData
- API to get the max fee quotes for all the tokens. Select the token in which user prefers to pay for the transaction fees. Create the callData for token approval and update the userOp callData.

async

function

getFeeQuotesOrData

(userOp : UserOperation)

{

const url = "paymaster url"

const requestData =

{ jsonrpc :

'2.0' , method :

'pm_getFeeQuoteOrData' , id : Date . now ( ) , params :

[userOp , { mode :

'ERC20' , tokenInfo :

{ tokenList :

["" ,

""] , preferredToken :

"" , } , expiryDuration :

300 , calculateGasLimits :

true , } ,] , } ; try { const response =

await axios . post (url , requestData) ; const feeQuotesResponse :

Array < any

= response . data . result . feeQuotes const selectedFeeQuote = feeQuotesResponse [i] ; // select the preferred token const maxGasFee = selectedFeeQuote . maxGasFee ; const selectedToken = selectedFeeQuote . tokenAddress ; // update the call Data for userOperation

}

catch

(error)

{ console . error (error) ; return error ; }

} * Max approval: * In this case, max approval callData will be batched to the userOp. One time max approval will make future userOps cheaper, given you will not need to batch the approval every time you send a transaction. In case of max approvals, You can also see the the supported tokens listhere * to reduce API call.

Checkoutthis for approval callData creation andthis for callData update.

3. Get paymaster data:

Get the paymaster url from the dashboard.

async

function

getPaymasterAndData

(userOp : UserOperation)

{ const

PAYMASTER_URL

"paymaster url" const requestData =

{ jsonrpc :

'2.0' , method :

'pm_sponsorUserOperation' , id : Date . now ( ) , params :

[userOp , { mode :

'ERC20' , tokenInfo :

{ preferredToken :

"address1" , tokenList :

["address1" ,

"address2"] , } , expiryDuration :

300 , calculateGasLimits :

true , } ,] , } ;

```
const
{ data }
=
await axios . post ( PAYMASTER_URL , requestData ) ; const
{ paymasterAndData , preVerificationGas , verificationGasLimit , callGasLimit }
= data . result ; return
{
... userOp , paymasterAndData , preVerificationGas : preVerificationGas . toString ( ) , verificationGasLimit : verificationGasLimit . toString ( ) , callGasLimit : callGasLimit . toString ( )
} ;
}
```

4. Sign userOperation

To sign the userOp, calculate the userOpHash and then sign it using the same signer, account was created. Follow this tutorial to learn about signing the userOp. Below is an example with ECDSA module .

```
async
function
signUserOp
( userOp : UserOperation )
{ const userOpHash =
getUserOpHash ( userOp ) ;
const moduleSig =
await signer . signMessage ( ethers . utils . arrayify ( userOpHash ) ) ; const signatureWithModuleAddress = ethers . utils . defaultAbiCoder . encode ( [ "bytes" ,
"address" ] , [ moduleSig ,
"0x0000001c5b32F37F5beA87BDD5374eB2aC54eA8e" ] , ) ; return
{
... userOp , signature : signatureWithModuleAddress } ; }
```

5. Send UserOperation:

eth_sendUserOperation sends a user operation to the given network.

```
async
function
sendUserOp ( userOp : UserOperation )
{ const url = "https://bundler.biconomy.io/api/v2/80001/nJPK7B3ru.dd7f7861-190d-41bd-af80-6877f74b8f44"
const requestData =
{ jsonrpc :
'2.0' , method :
'eth_sendUserOperation' , id : Date . now ( ) , params :
[ userOp , "0x5ff137d4b0fdcd49dca30c7cf57e578a026d2789" ] , } ;
const
{ data }
=
await axios . post ( url , requestData ) return data . result ; }
```

6. Fetch user operation receipt:

This API returns null until the transaction is mined, you will either need to poll or set a timeout. For reference checkout the sdk code . You can also get transaction hash from the response

```
async
function
getUserOpReceipt ( userOpHash :
string )
{ const url = "https://bundler.biconomy.io/api/v2/80001/nJPK7B3ru.dd7f7861-190d-41bd-af80-6877f74b8f44" const requestData =
{ jsonrpc :
'2.0' , method :
'eth_getUserOperationReceipt' , id : Date . now ( ) , params :
[ userOpHash ] , } ;
const
{ data }
=
await axios . post ( url , requestData ) ; return data . result ; }
```

If you are facing errors while integration, do checkout the common errors .

View Complete Code

```
import
{ ethers , utils }
from
"ethers" ; import axios ,
{ AxiosRequestConfig , AxiosResponse , AxiosError }
from
'axios' ; import
{
string ,
string
}
from
"ethers" ;
let provider =
new
ethers . providers . JsonRpcProvider ( "https://rpc.ankr.com/polygon_mumbai"
) ; let signer =
new
ethers . Wallet ( "private key" , provider ) ;
type
UserOperation
=
{ sender :
string ; nonce :
string ; initCode :
string ; callData :
string ; callGasLimit :
string ; verificationGasLimit :
string ; preVerificationGas :
string ; maxFeePerGas :
string ; maxPriorityFeePerGas :
string ; paymasterAndData :
string ; signature :
string ; }
async
function
getGasFeeValues
( userOp : Partial < UserOperation
    )
:
Promise < UserOperation
{ const url = "https://bundler.biconomy.io/api/v2/80001/nJPK7B3ru.dd7f7861-190d-41bd-af80-6877f74b8f44"
const
{ data }
=
await axios . post ( url ,
{ jsonrpc :
"2.0" , method :
"biconomy_getGasFeeValues" , params :
[ ] , id : Date . now ( ) } ) ; const
{ maxPriorityFeePerGas , maxFeePerGas }
= data . result ; return
{
... userOp , maxPriorityFeePerGas , maxFeePerGas , callGasLimit :
5000000 , verificationGasLimit :
5000000 , preVerificationGas :
5000000
}
as UserOperation ; }
async
function
getFeeQuotesOrData
( userOp : UserOperation )
{
```

const url = "paymaster url"

const requestData =

{ jsonrpc :

'2.0' , method :

'pm_getFeeQuoteOrData' , id : Date . now ( ) , params :

[userOp , { mode :

'ERC20' , tokenInfo :

{ tokenList :

["" ,

""] , preferredToken :

"" , } , expiryDuration :

300 , calculateGasLimits :

true , } ,] , } ; try

{ const response =

await axios . post (url , requestData) ; console . log ('Response:' , response . data) ; const feeQuotesResponse :

Array < any

= response . data . result . feeQuotes const selectedFeeQuote = feeQuotesResponse [5] ;

// select the preferred token const maxGasFee = selectedFeeQuote . maxGasFee ; const selectedToken = selectedFeeQuote . tokenAddress ; return userOp ; }

catch

(error)

{ console . error (error) ; return error ; } }

async

function

getPaymasterAndData

(userOp : UserOperation)

{ const

PAYMASTER_URL

"paymaster url" const requestData =

{ jsonrpc :

'2.0' , method :

'pm_sponsorUserOperation' , id : Date . now ( ) , params :

[{

... userOp , preVerificationGas : userOp . preVerificationGas . toString ( ) , verificationGasLimit : userOp . verificationGasLimit . toString ( ) , callGasLimit : userOp . callGasLimit . toString ( ) , maxFeePerGas : userOp . maxFeePerGas . toString ( ) , maxPriorityFeePerGas : userOp . maxPriorityFeePerGas . toString ( ) , paymasterAndData :

"0x"

} , { mode :

'ERC20' , tokenInfo :

{ preferredToken :

"address1" , tokenList :

["address1" ,

"address2"] , } , expiryDuration :

300 , calculateGasLimits :

true , } ,] , } ;

const

{ data }

=

await axios . post (PAYMASTER_URL , requestData) ; const

{ paymasterAndData , preVerificationGas , verificationGasLimit , callGasLimit }

= data . result ; return

{

... userOp , paymasterAndData , preVerificationGas : preVerificationGas . toString ( ) , verificationGasLimit : verificationGasLimit . toString ( ) , callGasLimit : callGasLimit . toString ( )

} ;

}

function

getUserOpHash (useOpMinusSignature : UserOperation)

{ const packedData = ethers . utils . defaultAbiCoder . encode (["address" , "uint256" , "bytes32" , "bytes32" , "uint256" , "uint256" , "uint256" , "uint256" , "uint256" , "bytes32" ,] , [useOpMinusSignature . sender , useOpMinusSignature . nonce , ethers . utils . keccak256 (useOpMinusSignature . initCode) , ethers . utils . keccak256 (useOpMinusSignature . callData) , useOpMinusSignature . callGasLimit , useOpMinusSignature . verificationGasLimit , useOpMinusSignature . preVerificationGas , useOpMinusSignature . maxFeePerGas , useOpMinusSignature . maxPriorityFeePerGas , ethers . utils . keccak256 (useOpMinusSignature . paymasterAndData) ,]) ;

const enc = ethers . utils . defaultAbiCoder . encode (["bytes32" ,

"address" ,

```
"uint256" ] , [ ethers . utils . keccak256 ( packedData ) ,
"0x5ff137d4b0fdcd49dca30c7cf57e578a026d2789" ,
80001 ] ) ;
const userOpHash = ethers . utils . keccak256 ( enc ) ; return userOpHash ; }
async
function
signUserOp
( userOp : UserOperation )
{ const userOpHash =
getUserOpHash ( userOp ) ;
const moduleSig =
await signer . signMessage ( ethers . utils . arrayify ( userOpHash ) ) ; const signatureWithModuleAddress = ethers . utils . defaultAbiCoder . encode ( [ "bytes" ,
"address" ] , [ moduleSig ,
"0x0000001c5b32F37F5beA87BDD5374eB2aC54eA8e" ] , ) ; userOp . signature = signatureWithModuleAddress return userOp ; }
async
function
sendUserOp ( userOp : UserOperation )
{ const url = "https://bundler.biconomy.io/api/v2/80001/nJPK7B3ru.dd7f7861-190d-41bd-af80-6877f74b8f44"
const requestData =
{ jsonrpc :
'2.0' , method :
'eth_sendUserOperation' , id : Date . now ( ) , params :
[ userOp , "0x5ff137d4b0fdcd49dca30c7cf57e578a026d2789" ] , } ;
const
{ data }
=
await axios . post ( url , requestData ) return data . result ; }
async
function
getUserOpReceipt ( userOpHash :
string )
{ const url = "https://bundler.biconomy.io/api/v2/80001/nJPK7B3ru.dd7f7861-190d-41bd-af80-6877f74b8f44" const requestData =
{ jsonrpc :
'2.0' , method :
'eth_getUserOperationReceipt' , id : Date . now ( ) , params :
[ userOpHash ] , } ;
const
{ data }
=
await axios . post ( url , requestData ) ; return data . result ; }
async
function
executePartialUserOp ( )
{
try
{ let partialUserOp =
{ sender :
'0x4dF23B78543F5c2F9CBCDF09956288B3e97bb9a4' , nonce :
'0x1D' , initCode :
"0x" , paymasterAndData :
"0x" , callData :
"0x0000189a000000000000000000000000322af0da66d00be980c7aa006377fcaaeee3bdfd000000000000000000000000000000000000000000000000002386f26fc100000000000000000000000000000
, signature :
"0x00000000000000000000000000000000000000000000000000000000000000400000000000000000000000000000001c5b32F37F5beA87BDD5374eB2aC54eA8e0000000000000000000000000000
, }
// Step 1 Gas estimation let userOp =
await
getGasFeeValues ( partialUserOp )
// Step 2 Get ERC20 fee quotes await
getFeeQuotesOrData ( userOp )
// Step 3 Get paymaster data userOp =
await

getPaymasterAndData ( userOp )

// Step 4 sign user op userOp =

await

signUserOp ( userOp )

// Step 5: send user operation const userOpHash =

await

sendUserOp ( userOp ) ;

// Step 6: Get UserOpReceipt const receipt =

await

getUserOpReceipt ( userOpHash ) ;

} catch

( error )

{ console . error ( error ) } }

executePartialUserOp ( ) ;
```

Previous Paymaster sponsored Next Signing UserOp