

Tutorial 3 — Submit a user operation with an ERC-20 Paymaster

This is a low-level tutorial that walks you through the steps of constructing a user operation from scratch. If you would like to leverage permissionless.js's high-level functions, take a look at [tutorial 1](#) . If you would like to learn more about leveraging the ERC-20 paymaster with the high-level API, take a look at the [relevant how-to guide](#) .

You can visit our [ERC-20 Paymaster overview page](#) to learn more about the design and architecture of our ERC-20 Paymaster, and the deployed smart contract addresses.

This tutorial uses the v0.6 of the EntryPoint, instead of the newest v0.7 like the other tutorials. This is because the ERC-20 Paymaster is not yet compatible with v0.7. We are working on a new version of the ERC-20 Paymaster that will be compatible with v0.7.

In this tutorial, you will deploy an ERC-4337 smart contract wallet on Mumbai, and submit a User Operation that pays for its gas fees with USDC using Pimlico's ERC-20 Paymaster.

Steps

Get a Pimlico API key

The ERC-20 paymaster itself is completely permissionless, there is no requirement to use an API key for it . We are using an API key in this tutorial to get access to the bundler and verifying paymaster to make the tutorial flow easier. To get started, please go to our [dashboard](#) and generate a Pimlico API key.

Clone the Pimlico tutorial template repository

We have created a [Pimlico tutorial template repository](#) that you can use to get started. It comes set up with Typescript, viem, and permissionless.js.

```
...  
  
gitclone https://github.com/pimlicolabs/tutorial-template.git pimlico-tutorial-3 cdpimlico-tutorial-3  
  
...
```

Now, let's install the dependencies:

```
...  
  
npminstall  
  
...
```

The main file we will be working with is index.ts . Let's run it to make sure everything is working:

```
...  
  
npmstart  
  
...
```

If everything has been set up correctly, you should see Hello world! printed to the console.

Define the constants we'll need

Let's open up index.ts .

First, let's define some of the constants we'll need to use to make the flow work.

Before going any further, let's replace privateKey with a new private key. You can use the randomly generated one below, or by calling generatePrivateKey() .

Do not use this private key in production. Private key: Address: Remember to replace the apiKey variable with your own Pimlico API key. This will allow you to use our bundler to submit your User Operation.

After replacing these two variables, add the following to the bottom of index.ts :

```
...  
  
const privateKey = "GENERATED_PRIVATE_KEY" // replace this with a private key you generate! const apiKey = "YOUR_PIMICO_API_KEY" // replace with your Pimlico API key  
  
const ENTRY_POINT_ADDRESS = "0x5FF137D4b0FDCD49DcA30c7CF57E578a026d2789"  
const SIMPLE_ACCOUNT_FACTORY_ADDRESS = "0x9406Cc6185a346906296840746125a0E44976454"  
  
const chain = "mumbai"  
  
if (apiKey === undefined) { throw new Error("Please replace the apiKey env variable with your Pimlico API key") }  
  
if (privateKey.match(/GENERATED_PRIVATE_KEY/)) { throw new Error("Please replace the privateKey variable with a newly generated private key. You can use generatePrivateKey() for this", ) }  
  
const signer = privateKeyToAccount(privateKey as Hash)  
  
const bundlerClient = createClient({ transport: http(https://api.pimlico.io/v1{chain}/rpc?apiKey={apiKey}), chain: polygonMumbai, })
```

```
.extend(bundlerActions(ENTRYPOINT_ADDRESS_V06)) .extend(pimlicoBundlerActions(ENTRYPOINT_ADDRESS_V06))

constpaymasterClient=createClient({ // ⚠ using v2 of the API ⚠ transport:http(https://api.pimlico.io/v2{chain}/rpc?apikey={apiKey}), chain: polygonMumbai,
}).extend(pimlicoPaymasterActions(ENTRYPOINT_ADDRESS_V06))

constpublicClient=createPublicClient({ transport:http("https://mumbai.rpc.thirdweb.com"), chain: polygonMumbai, })

...

```

Calculate the deterministic address of your smart wallet

Since we will be looking to fund our account with USDC (that we will use to sponsor UserOperations with), we need to know the address where our smart wallet will be deployed.

Add the following to the bottom of index.ts :

```
...

constinitCode=concat([ SIMPLE_ACCOUNT_FACTORY_ADDRESS, encodeFunctionData({ abi: [ { inputs: [ { name:"owner", type:"address"}, {
name:"salt", type:"uint256"}, ], name:"createAccount", outputs: [{ name:"ret", type:"address"}], stateMutability:"nonpayable", type:"function", }, ],
args: [signer.address,0n], }, )])

constsenderAddress=awaitgetSenderAddress(publicClient, { initCode, entryPoint:ENTRY_POINT_ADDRESS, }) console.log("Counterfactual
sender address:", senderAddress)

...

```

Let's run this code with `npm start` . You should see something like this:

```
...

Counterfactual sender address: 0xbAd38BdCf884ED92ab370f69C0CD0B7b8a1459A1

...

```

Get Testnet USDC on Mumbai

Let's get some USDC on the Mumbai testnet to the counterfactual address of the wallet we will be deploying. This will be used to pay for the gas fees of the User Operations we will be submitting.

The recommended way to do this is to use the [USDC faucet](#) , select POLYGON and enter the counterfactual sender address you generated in the previous step.

Alternatively, if the above step does not work, you can get some testnet MATIC. Then, if you're using Metamask [switch your chain to Mumbai](#) and then [visit this page to swap some of the testnet MATIC to testnet USDC on Uniswap](#) and then send that USDC to the counterfactual sender address.

Deploy a SimpleWallet

In our [first tutorial](#) we went through an example flow where we deployed [SimpleWallet](#) using Pimlico's verifying paymaster and bundler. We will do something similar in this guide, except that we will be mainly looking to leverage the ERC-20 Paymaster.

During deployment, let's send an approval transaction that will approve unlimited USDC tokens to the ERC-20 Paymaster. This saves us an extra step and allows us to already start using the ERC-20 Paymaster starting from the next User Operation.

Add the following to the bottom of index.ts :

```
...

constgenerateApproveCallData=(erc20TokenAddress:Address,paymasterAddress:Address)=>{ constapproveData=encodeFunctionData({ abi: [ {
inputs: [ { name:"_spender", type:"address"}, { name:"_value", type:"uint256"}, ], name:"approve", outputs: [{ name:"", type:"bool"}], payable:false,
stateMutability:"nonpayable", type:"function", }, ], args: [paymasterAddress,0xffffffffffffffffffffffffffffffffffffffff], })

// GENERATE THE CALLDATA TO APPROVE THE USDC constto=erc20TokenAddress constvalue=0n constdata=approveData

constcallData=encodeFunctionData({ abi: [ { inputs: [ { name:"dest", type:"address"}, { name:"value", type:"uint256"}, { name:"func", type:"bytes"}, ],
name:"execute", outputs: [], stateMutability:"nonpayable", type:"function", }, ], args: [to, value, data], })

returncallData }

constsubmitUserOperation=async(userOperation:UserOperation<"v0.6">)=>{ constuserOperationHash=awaitbundlerClient.sendUserOperation({
userOperation, }) console.log(UserOperation submitted. Hash: {userOperationHash})

console.log("Querying for receipts...") constreceipt=awaitbundlerClient.waitForUserOperationReceipt({ hash: userOperationHash, })
console.log(Receipt found!\nTransaction hash: {receipt.receipt.transactionHash} ) }

// You can get the paymaster addresses from https://docs.pimlico.io/reference/erc20-paymaster/contracts
consterc20PaymasterAddress="0x0000000000009B901DeC1aaB9389285965F49D387"
constusdcTokenAddress="0x9999f7Fea5938fD3b1E26A12c3f2fb024e194f97"// USDC on Polygon Mumbai

constsenderUsdcBalance=awaitpublicClient.readContract({ abi: [ { inputs: [{ name:"_owner", type:"address"}], name:"balanceOf", outputs: [{
name:"balance", type:"uint256"}], type:"function", stateMutability:"view", }, ], address: usdcTokenAddress, functionName:"balanceOf", args:
[senderAddress], })

if(senderUsdcBalance<1_000_000n) { thrownewError( insufficient USDC balance for counterfactual wallet address {senderAddress}: {

```


...

If you visit the address of the sender account on [Mumbai Scan](#) , you should also see that some of your USDC balance has been deducted!

That's it!

Congratulations, you have submitted your first User Operation using Pimlico's ERC-20 Paymaster! From now on, you will no longer need to keep native gas tokens like ETH and MATIC on your wallet to be able to transact.

Combined code

If you want to see the complete code that combines all of the previous steps, we uploaded it to [separate repository](#) . If you're looking to run it, remember to replace the API key with your own and to use the [USDC Faucet](#) to fill your counterfactual sender address with some USDC after you calculate it.