Tutorial 1 — Send your first gasless transaction

In this tutorial, you will submit your first fully-gasless transaction from a smart account.

You will set up the necessary permissionless.js clients, create a user operation, ask Pimlico's verifying paymaster to sponsor it, and then submit it on-chain with Pimlico's bundler.

Steps

Get a Pimlico API key

To get started, please go to ourdashboard and generate a Pimlico API key.

Clone the Pimlico tutorial template repository

We have created aPimlico 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-1 cdpimlico-tutorial-1
```

Now, let's install the dependencies:

```
npminstall
```

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

```
npmstart
```

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

Create the public and paymaster clients, and generate a private key

The public client will be responsible for querying the blockchain, while the paymaster client will be responsible for interacting with Pimlico's verifying paymaster endpoint and requesting sponsorship.

Make sure to replaceYOUR_PIMLICO_API_KEY in the code below with your actual Pimlico API key.

Let's open upindex.ts , and add the following to the bottom:

```
constapiKey="YOUR_PIMLICO_API_KEY" constpaymasterUrl=https://api.pimlico.io/v2/sepolia/rpc?apikey={apiKey}

constprivateKey= (process.env.PRIVATE_KEYasHex)?? (()=>{ constpk=generatePrivateKey()
writeFileSync(".env",PRIVATE_KEY={pk}) returnpk })()

exportconstpublicClient=createPublicClient({ transport:http("https://rpc.ankr.com/eth_sepolia"), })

exportconstpaymasterClient=createPimlicoPaymasterClient({ transport:http(paymasterUrl),
entryPoint:ENTRYPOINT_ADDRESS_V07, })
```

Create theSmartAccount

instance

For the purposes of this guide, we will be usingSafe accounts. This account is an ERC-4337 wallet controlled by a single

EOA signer.

Tip Want to learn more about using Safe accounts? Take a look at oudedicated Safe guide To create the Safe account, we will use theprivateKeyToSafeSmartAccount utility function from permissionless.js. We need to specify the Safe version we are using as well as the global ERC-4337 EntryPoint address. For the signer, we will be using the previously generated private key.

Add the following to the bottom ofindex.ts :

```
constaccount=awaitprivateKeyToSimpleSmartAccount(publicClient, { privateKey,
entryPoint:ENTRYPOINT_ADDRESS_V07,// global entrypoint
factoryAddress:"0x91E60e0613810449d098b0b5Ec8b51A0FE8c8985", })

console.log(Smart account address: https://sepolia.etherscan.io/address{account.address})
```

Let's run this code withnpm start . You should see the smart account address printed to the console.

```
Smart account address: https://sepolia.etherscan.io/address/0x374b42bCFAcf85FDCaAB84774EA15ff36D42cdA7
```

If you visit the address on Etherscan, you might notice that no contract is actually deployed to this address yet. This is because smart account are counterfactual, meaning that they are only deployed on-chain the first time you send a transaction through the account.

Create the bundler and smart account clients and fetch the gas price

Now that we have aSmartAccount instance, we need to create aSmartAccountClient instance to make transact from it.SmartAccountClient is an almost drop-in replacement for a viemWalletClient , but it also includes some additional functionality for interacting with smart accounts.

We also extend theSmartAccountClient with bundler actions and Pimlico-specific bundler actions so we can fetch the appropriate gas price.

We also specify the optionalsponsorUserOperation middleware function, calling thesponsorUserOperation function from the paymaster client. This will make sure that the user operation is sponsored by Pimlico's verifying paymaster.

Finally, we will fetch the gas price from the bundler that we will use to submit the user operation in the next step.

Add the following to the bottom ofindex.ts :

```
constbundlerUrl=https://api.pimlico.io/v2/sepolia/rpc?apikey={apiKey}

constbundlerClient=createPimlicoBundlerClient({ transport:http(bundlerUrl), entryPoint:ENTRYPOINT_ADDRESS_V07, })

constsmartAccountClient=createSmartAccountClient({ account, entryPoint:ENTRYPOINT_ADDRESS_V07, chain: sepolia,
bundlerTransport:http(bundlerUrl), middleware: { gasPrice:async()=>{
return(awaitbundlerClient.getUserOperationGasPrice()).fast }, sponsorUserOperation:
paymasterClient.sponsorUserOperation, }, })
```

Let's run this code withnpm start . You should see the gas prices printed to the console.

```
Received gas prices: { slow: { maxFeePerGas: 26237682174n, maxPriorityFeePerGas: 26237682174n }, standard: {
maxFeePerGas: 27487095611n, maxPriorityFeePerGas: 27487095611n }, fast: { maxFeePerGas: 28736509048n,
maxPriorityFeePerGas: 28736509048n } }
```

Submit a transaction from the smart account

Finally, let's submit a transaction from the smart account. We will send a transaction to the0xd8da6bf26964af9d7eed9e03e53415d37aa96045 (vitalik.eth) address with0x1234 as examplecallData . We will also specify the gas price we want to use, which we fetched from the bundler in the previous step.

Underneath the hood, theSmartAccountClient will create aUserOperation instance, request Pimlico paymaster sponsorship, sign it with the smart account's private key, and then submit it to the bundler. The bundler will then query for receipts until it sees the user operation included on-chain.

Add the following to the bottom ofindex.ts :

```
consttxHash=awaitsmartAccountClient.sendTransaction({ to:"0xd8da6bf26964af9d7eed9e03e53415d37aa96045", value:0n, data:"0x1234", })

console.log(User operation included: https://sepolia.etherscan.io/tx{txHash})
```

Let's run this code again withnpm start . You should see the transaction hash bundling the user operation on-chain printed to the console.

```
User operation included: https://sepolia.etherscan.io/tx/0x7a2b61b4b7b6e9e66c459e3c9c24c7a292fc6c740533ce35dbf58710960cc0e5
```

You can now view the transaction on the Sepolia testnet explorer. By sending this user operation, you have:

- Deployed the counterfactual smart account contract
- Had this newly-deployed smart account verify the private key's signature
- Made Pimlico's verifying paymaster sponsor the user operation's gas fees
- Executed a simple transaction tovitalik.eth
- 's address

All in a couple lines of code.

Congratulations, you are now a pioneer of Account Abstraction!

Pleaseget in touch if you have any questions or if you'd like to share what you're building!

Combined code

If you want to see the complete code that combines all of the previous steps, we uploaded it to aseparate repository . If you're looking to run it, remember to replace the API key with your own!