D4D4D4;--ch-t-background: #1E1E1E;--ch-t-lighter-inlineBackground: #1e1e1ee6;--ch-t-editor-background: #1E1E1E;--ch-t-editor-foreground: #D4D4D4;--ch-t-editor-rangeHighlightBackground: #ffffff0b;--ch-t-editor-infoForeground: #3794FF;--ch-t-editor-selectionBackground: #264F78;--ch-t-focusBorder: #007FD4;--ch-t-tab-activeBackground: #1E1E1E;--ch-t-tab-activeForeground: #ffffff;--ch-t-tab-inactiveBackground: #2D2D2D;--ch-t-tab-inactiveForeground: #ffffff80;--ch-t-tab-border: #252526;--ch-t-tab-activeBorder: #1E1E1E;--ch-t-editorGroup-border: #444444;--ch-t-editorGroupHeader-tabsBackground: #252526;--ch-t-editorLineNumber-foreground: #858585;--ch-t-input-background: #3C3C3C;--ch-t-input-foreground: #D4D4D4;--ch-t-icon-foreground: #C5C5C5;--ch-t-sideBar-background: #252526;--ch-t-sideBar-foreground: #D4D4D4;--ch-t-sideBar-border: #252526;--ch-t-list-activeSelectionBackground: #094771;--ch-t-list-activeSelectionForeground: #fffffe;--ch-t-list-hoverBackground: #2A2D2E; }

Permissionless.js Quickstart Guide

In this guide, you will learn how to sponsor the deployment of an ERC-4337 Safe account and its user operations usingPimlico(opens in a new tab) infrastructure and thepermissionless(opens in a new tab) library.

This guide focuses high-level on deploying and using Safe accounts configured with theSafe4337Module enabled. For a more detailed guide that shows what happens under the hood, check thePermissionless detailed guide .

Pimlico is one of the most popular ERC-4337 account abstraction infrastructure platforms, which provides a suite of tools and services to help build, deploy, and manage smart accounts on EVM-compatible chains.

permissionless is a TypeScript library focused on building with the ERC-4337 stack, including smart accounts, bundlers, paymasters, and user operations. Some of its core principles are providing a great developer experience and avoiding vendor lock-in by supporting different providers and ERC-4337 smart accounts, including Safe.

â¹ï¸ If you are already building with the Safe{Core} SDK, you may want to follow theSafe4337Pack guide instead of integrating thepermissionless library directly into your application.

Prerequisites

- Node.js and npm(opens in a new tab)
- .
- APimlico account(opens in a new tab)
- and an API key.

Install dependencies

Installviem(opens in a new tab) andpermissionless(opens in a new tab) dependencies by running the following command:

_10 pnpm install viem permissionless

Steps

Imports

These are all the imports required in the script we are building for this guide, which includespermissionless andviem packages.

_10 import 'dotenv/config' _10 import { ENTRYPOINT_ADDRESS_V06, createSmartAccountClient } from 'permissionless' _10 import { signerToSafeSmartAccount } from 'permissionless/accounts' _10 import { _10 createPimlicoBundlerClient, _10 createPimlicoPaymasterClient, _10 } from 'permissionless/clients/pimlico' _10 import { createPublicClient, http, Hex, encodeFunctionData, parseEther } from 'viem' _10 import { privateKeyToAccount } from 'viem/accounts' _10 import { gnosis } from 'viem/chains'

Setup

These are the constants that will be used in this guide. We have selected Gnosis as the chain, but other supported networks can be used. We also need the Pimlico API key and a private key we will use for the owner of the Safe.

_10 // Network _10 const chain = gnosis _10 const chainName = 'gnosis' _10 const SPONSORSHIP_POLICY_ID =''_10 _10 // Keys _10 const PIMLICO_API_KEY = process.env.PIMLICO_API_KEY _10 const PRIVATE_KEY = process.env.PRIVATE_KEY as Hex

Create the signer

First, we need a signer instance that will be the owner of the Safe account once it is deployed.

_10 const signer = privateKeyToAccount(PRIVATE_KEY as Hash)

Initialize the clients

We need to create a few client instances to query the blockchain network and operate with Pimlico infrastructure.

Firstly, we instantiate a standardpublicClient instance for regular Ethereum RPC calls. To do this, we must first define the corresponding RPC URL, depending on our network.

_10 const publicClient = createPublicClient({ _10 transport: http(https://rpc.ankr.com{chainName}) _10 }) Secondly, we instantiate thebundlerClient using the Pimlico APIv1 , which is dedicated to the Bundler methods. This API requires aPIMLICO_API_KEY that we can get from theirdashboard(opens in a new tab) .

_10 const bundlerClient = createPimlicoBundlerClient({ _10 transport: http(https://api.pimlico.io/v1{chainName}/rpc?apikey={PIMLICO_API_KEY}), _10 entryPoint: ENTRYPOINT_ADDRESS_V06 _10 }) Lastly, we instantiate thepaymasterClient using the Pimlico APIv2 , which is dedicated to the Paymaster methods and responsible for interacting with Pimlico's Verifying Paymaster endpoint and requesting sponsorship.

_10 const paymasterClient = createPimlicoPaymasterClient({ _10 transport: http(https://api.pimlico.io/v2{chainName}/rpc?apikey={PIMLICO_API_KEY}), _10 entryPoint: ENTRYPOINT_ADDRESS_V06 _10 })

Fetch the gas price

We fetch the current gas price values to add them later to our transaction.

_10 const gasPrices = await bundlerClient.getUserOperationGasPrice()

Create the Safe account

We can create a Safe account based on oursigner address, theentryPoint address, an optionalsaltNonce that will affect the resulting address of the deployed Safe, and thesafeVersion (that must be1.4.1 or higher).

_10 const safeAccount = await signerToSafeSmartAccount(publicClient, { _10 entryPoint: ENTRYPOINT_ADDRESS_V06, _10 signer: signer, _10 saltNonce: 0n, // Optional _10 safeVersion: '1.4.1', _10 address: '0x...' // Optional. Only for existing Safe accounts. _10 }) The optionaladdress parameter is only used when we already have a Safe account and want to initialize it. Deployments of new Safe accounts should remove this parameter.

Create the Safe account client

We need to create the smart account client with the following parameters:

_10 const safeAccountClient = createSmartAccountClient({ _10 account: safeAccount, _10 entryPoint: ENTRYPOINT_ADDRESS_V06, _10 chain: chain, _10 bundlerTransport: http(https://api.pimlico.io/v1{chainName}/rpc?apikey={PIMLICO_API_KEY}), _10 middleware: { _10 gasPrice: async () => (await bundlerClient.getUserOperationGasPrice()).fast, _10 sponsorUserOperation: paymasterClient.sponsorUserOperation _10 } _10 }) In case we want to sponsor the transactions of this Safe using a Pimlico sponsorship id, we need to replace thesponsorUserOperation passed in themiddleware property like this:

_10 sponsorUserOperation: ({ userOperation }) => { _10 return paymasterClient.sponsorUserOperation({ _10 userOperation, _10 sponsorshipPolicyId: SPONSORSHIP_POLICY_ID _10 }) _10 }

Submit a transaction

Finally, we call thesendTransaction method to submit our transaction.

_11 const txHash = await safeAccountClient.sendTransaction({ _11 to: safeAccount.address, _11 value: parseEther('0'), _11 data: encodeFunctionData({ _11 abi: '', _11 functionName: '', _11 args: [] _11 }), _11 maxFeePerGas: gasPrices.fast.maxFeePerGas, _11 maxPriorityFeePerGas: gasPrices.fast.maxPriorityFeePerGas _11 })

Recap and further reading

This guide covered how to sponsor the deployment of a new ERC-4337 Safe and its user operations with Pimlico infrastructure using a Paymaster.

Feel free to try out other ideas and possibilities, as there are many more regarding:

- The deployment and initial setup of ERC-4337 accounts.
- The entity responsible for paying the transaction fees.
- The tokens used to pay the transaction fees.

Explore our4337-gas-metering(opens in a new tab) repository on GitHub to see how most of these options work with Safe and notice the integrations with different providers like Alchemy, Gelato, and Pimlico.

ERC-4337 Safe SDK Permissionless.js Detailed