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; }

Safe accounts with the Safe4337Module

In this guide, you will learn how to create and execute multiple Safe transactions grouped in a batch from a Safe account that is not yet deployed and where the executor may or may not have funds to pay for the transaction fees. This can be achieved by supporting theERC-4337 execution flow, which is supported by theSafe4337Module and exposed via the Relay Kit from the Safe{Core} SDK.

Pimlico(opens in a new tab) is used in this guide as the service provider, but any other provider compatible with the ERC-4337 can be used.

Read theSafe4337Module documentation to understand its benefits and flows better.

Prerequisites

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

Install dependencies

_10 yarn add ethers @safe-global/protocol-kit @safe-global/relay-kit

Steps

Imports

Here are all the necessary imports for the script we implement in this guide.

_10 import { ethers } from 'ethers' _10 import { EthersAdapter } from '@safe-global/protocol-kit' _10 import { Safe4337Pack } from '@safe-global/relay-kit'

Create a signer

Firstly, we need to get and instantiate a signer using theEthersAdapter from the Protocol Kit, which will be the owner of a Safe account after it's deployed.

In this example, we use a private key, but any method to get a signer(opens in a new tab) can be used.

_10 const PRIVATE_KEY = '0x...' _10 _10 const provider = new ethers.JsonRpcProvider('https://rpc.ankr.com/eth_sepolia') _10 const signer = new ethers.Wallet(PRIVATE_KEY, provider) _10 const ethersAdapter = new EthersAdapter({ _10 ethers, _10 signerOrProvider: signer _10 })

Initialize theSafe4337Pack

TheSafe4337Pack class is exported from the Relay Kit and implements the ERC-4337 to create, sign, and submit Safe user operations.

To instantiate this class, the staticinit() method allows connecting existing Safe accounts (as long as they have theSafe4337Module enabled) or setting a custom configuration to deploy a new Safe account at the time where the first Safe transaction is submitted.

New Safe account Existing Safe account When deploying a new Safe account, we need to pass the configuration of the Safe in theoptions property. In this case, we are configuring a Safe account that will have our signer as the only owner.

_10 const safe4337Pack = await Safe4337Pack.init({ _10 ethersAdapter, _10 rpcUrl: 'https://rpc.ankr.com/eth_sepolia', _10 bundlerUrl: https://api.pimlico.io/v1/sepolia/rpc?apikey={PIMLICO_API_KEY}, _10 options: { _10 owners: [await signer.getAddress()], _10 threshold: 1 _10 }, _10 // ... _10 }) By default, the transaction fees will be paid in the native token and extracted from the Safe account, so there must be enough funds in the Safe address. However, depending on how we want the transactions to be paid, there are two other ways we can initialize theSafe4337Pack .

With an ERC-20 Paymaster Sponsored by an ERC-20 Paymaster A paymaster will execute the transactions and get reimbursed from the Safe account, which must have enough funds in the Safe address in advance.

Payment of transaction fees is made using an ERC-20 token specified with thepaymasterTokenAddress property. If an ERC-20 token is used, the Safe must approve that token to the paymaster. If no balance is approved, it can be specified using theamountToApprove property.

_10 const safe4337Pack = await Safe4337Pack.init({ _10 // ... _10 paymasterOptions: { _10 paymasterAddress: '0x...' _10 paymasterTokenAddress: '0x...', _10 amountToApprove // Optional _10 } _10 })

Create a user operation

To create a Safe user operation, use thecreateTransaction() method, which takes the array of transactions to execute and returns aSafeOperation object.

_10 // Define the transactions to execute _10 const transaction1 = { to, data, value } _10 const transaction2 = { to, data, value } _10 _10 // Build the transaction array _10 const transactions = [transaction1, transaction2] _10 _10 // Create the SafeOperation with all the transactions _10 const safeOperation = await safe4337Pack.createTransaction({ transactions }) ThesafeOperation object has thedata andsignatures properties, which contain all the information about the transaction batch and the signatures of the Safe owners, respectively.

Sign the user operation

Before sending the user operation to the bundler, it's required to sign thesafeOperation object with the connected signer. ThesignSafeOperation() method, which receives aSafeOperation object, generates a signature that will be checked when theSafe4337Module validates the user operation.

_10 const signedSafeOperation = await safe4337Pack.signSafeOperation(safeOperation)

Submit the user operation

Once thesafeOperation object is signed, we can call theexecuteTransaction() method to submit the user operation to the bundler.

_10 const userOperationHash = await safe4337Pack.executeTransaction({ _10 executable: signedSafeOperation _10 }) This method returns the hash of the user operation. With it, we can monitor the transaction status using a block explorer or the bundler's API.

Check the transaction status

To check the transaction status, we can use thegetTransactionReceipt() method, which returns the transaction receipt after it's executed.

_10 let userOperationReceipt = null _10 _10 while (!userOperationReceipt) { _10 // Wait 2 seconds before checking the status again _10 await new Promise((resolve) => setTimeout(resolve, 2000)) _10 userOperationReceipt = await safe4337Pack.getUserOperationReceipt(_10 userOperationHash _10) _10 } In addition, we can use thegetUserOperationByHash() method with the returned hash to retrieve the user operation object we sent to the bundler.

_10 const userOperationPayload = await safe4337Pack.getUserOperationByHash(_10 userOperationHash _10)

Recap and further reading

After following this guide, we are able to deploy new Safe accounts and create, sign, and execute Safe transactions in a batch without the executor needing to have funds to pay for the transaction fees.

Learn more about the ERC-4337 standard and theSafe4337Module contract following these links:

- ERC-4337 website(opens in a new tab)
- EIP-4337 on Ethereum EIPs(opens in a new tab)
- Safe4337Module on GitHub(opens in a new tab)

Relay Kit Gelato Relay

Was this page helpful?

Report issue