How to create and use a Safe account with multiple signers

Safe is the most battle-tested Ethereum smart account provider. With their recent release of their ERC-4337 module, it is now possible to plug in Safe accounts to ERC-4337 bundlers and paymasters. This guide will walk you through how to create and use a Safe account with permissionless.js.

Steps

Import the required packages

```
import{ createSmartAccountClient }from"permissionless" import{ createPublicClient, getContract, http, parseEther }from"viem" import{ sepolia }from"viem/chains"
```

Create the clients

First we must create the public, (optionally) pimlico paymaster clients that will be used to interact with the Safe account.

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

exportconstpaymasterClient=createPimlicoClient({ transport:http("https://api.pimlico.io/v2/sepolia/rpc?apikey=API_KEY"), entryPoint: { address: entryPoint07Address, version:"0.7", }, })
```

Get the owner addresses

The Safe account will need to have a signer to sign user operations. In permissionless.js, the default Safe account validates ECDSA signatures.Any permissionless.js-compatible signer can be used for the Safe account.

For example, to create a signer based on a private key:

```
import{ privateKeyToAccount, toAccount }from"viem/accounts" import{ createPimlicoClient }from"permissionless/clients/pimlico" import{ entryPoint07Address }from"viem/account-abstraction" import{ toSafeSmartAccount }from"permissionless/accounts"

constownerOne="0xPUBLIC-ADDRESS-ONE" constownerTwo="0xPUBLIC-ADDRESS-TWO" constownerThree="0xPUBLIC-ADDRESS-THREE"
```

Create the Safe account

For a full list of options for creating a Safe account, take a look at the reference documentation page for toSafeSmartAccount . With a signer, you can create a Safe account as such:

```
constowners=[toAccount(ownerOne),toAccount(ownerTwo),toAccount(ownerThree)]

constsafeAccount=awaittoSafeSmartAccount({ client: publicClient, entryPoint: { address: entryPoint07Address, version:"0.7", }, owners, saltNonce:0n,// optional version:"1.4.1", })
```

You can also create a Safe account with 7579 module, read more about it here .

Create the smart account client

The smart account client is a permissionless.js client that is meant to serve as an almost drop-in replacement for viem'swalletClient .

```
constsmartAccountClient=createSmartAccountClient({ account: safeAccount, chain: sepolia, paymaster: paymasterClient, bundlerTransport:http("https://api.pimlico.io/v2/sepolia/rpc?apikey=API_KEY"), userOperation: { estimateFeesPerGas:async()=>(awaitpaymasterClient.getUserOperationGasPrice()).fast, }, })
```

Prepare a user operation

Since we may not have access to all the signers at once, we should prepare a user operation and then submit it later after all the signers have signed.

```
constunSignedUserOperation=awaitsmartAccountClient.prepareUserOperation({ calls: [ { to:"0xd8da6bf26964af9d7eed9e03e53415d37aa96045", value:parseEther("0.1"), }, ], })
```

Collect signatures

You can use theSafeSmartAccount.signUserOperation method to collect signatures from the signers.

```
import{ SafeSmartAccount }from"permissionless/accounts/safe"

constownerOneAccount=privateKeyToAccount("0xPRIVATE-KEY-ONE")// this can any LocalAccount | EIP1193Provider | WalletClient

letpartialSignatures=awaitSafeSmartAccount.signUserOperation({ version:"1.4.1", entryPoint: { address: entryPoint07Address, version:"0.7", }, chainId: sepolia.id, owners: owners.map((owner)=>toAccount(owner.address)), account: ownerOneAccount,// the owner that will sign the user operation ...unSignedUserOperation, })

constownerTwoAccount=privateKeyToAccount("0xPRIVATE-KEY-TWO")// this can any LocalAccount | EIP1193Provider | WalletClient partialSignatures=awaitSafeSmartAccount.signUserOperation({ version:"1.4.1", entryPoint: { address: entryPoint07Address, version:"0.7", }, chainId: sepolia.id, owners: owners.map((owner)=>toAccount(owner.address)), account: ownerTwoAccount,// the owner that will sign the user operation signatures: partialSignatures, ...unSignedUserOperation, })

constownerThreeAccount=privateKeyToAccount("0xPRIVATE-KEY-THREE")// this can any LocalAccount | EIP1193Provider | WalletClient constfinalSignature=awaitSafeSmartAccount.signUserOperation({ version:"1.4.1", entryPoint: { address: entryPoint07Address, version:"0.7", }, chainId: sepolia.id, owners: owners.map((owner)=>toAccount(owner.address)), account: ownerThreeAccount,// the owner that will sign the user operation signatures: partialSignatures, ...unSignedUserOperation, })
```

Submit the user operation

Once you have the final signature, you can submit the user operation.

```
constuserOpHash=awaitsmartAccountClient.sendUserOperation({ ...unSignedUserOperation, signature: finalSignature, })

constreceipt=awaitsmartAccountClient.waitForUserOperationReceipt({ hash: userOpHash, })
```

Understanding the errors

If you're getting an error that starts withGS , it probably means that something went off with the Safe account. Checkout the Safe error codeshere .