Protocol Kit

TheProtocol Kit(opens in a new tab) enables developers to interact with theSafe contracts(opens in a new tab) using a TypeScript interface. This Kit can be used to create new Safe accounts, update the configuration of existing Safes, propose and execute transactions, among other features.

.mui-style-tn3bmg{box-sizing:border-box;margin:0;-webkit-flex-direction:row;-ms-flex-direction:row;flex-direction:row;margin-top:24px;}

@safe-global/protocol-kit

Quickstart

In this quickstart guide, you will create a 2 of 3 multi-sig Safe and propose and execute a transaction to send some ETH out of this Safe.

For a more detailed guide, including how to integrate withweb3.js and more Safe transaction configuration options, seeGuide: Integrating the Protocol Kit and API Kit(opens in a new tab) andProtocol Kit Reference(opens in a new tab) .

Prerequisites

1. Node.js and npm(opens in a new tab)
2. Three externally-owned accounts with Testnet ETH in at least one account

Install dependencies

First, we need to install some dependencies fromsafe-core-sdk and theethers library.

To interact with Ethereum and other EVM blockchains in Node, we can either use: web3.js or ethers.js. In this tutorial, we will use the ethers.js library. To useweb3js , seeInstantiate an EthAdapter section in Guide: Integrating the Safe Core SDK(opens in a new tab) .

The Protocol Kit is compatible only withethers.js v6 . Make sure you specify this version when installing the SDK.

You can store your environment variables such as private keys in a.env file. To read easily from.env files, use thedotenv library.

yarn

add

ethers

@safe-global/protocol-kit \ @safe-global/api-kit \ @safe-global/safe-core-sdk-types \ dotenv

Create the.env file:

touch

.env

Put your signing account private keys into the.env file you just created.

export OWNER_1_PRIVATE_KEY = ' export OWNER_2_PRIVATE_KEY = ' export OWNER_3_PRIVATE_KEY = '

Create anindex.ts file that you will use to run the following code snippets.

touch

index.ts

Tip: Usets-node(opens in a new tab) to run a Typescript file in Node.js.

npx

ts-node

examples/protocol-kit/index.ts

Initialize Signers, Providers, and EthAdapter

The signers trigger transactions to the Ethereum blockchain or off-chain transactions. The provider connects to the Ethereum blockchain.

You can get a public RPC URL fromChainlist(opens in a new tab) , however, public RPC URLs can be unreliable so you can also try a dedicated provider like Infura or Alchemy.

For this tutorial, we will be creating a Safe on the Sepolia Testnet.

import { ethers } from

'ethers' import {Â EthersAdapter } from

'@safe-global/protocol-kit' import dotenv from

'dotenv'

dotenv .config ()

// https://chainlist.org/?search=sepolia&testnets=true const

RPC_URL

'https://eth-sepolia.public.blastapi.io' const

provider

=

new

ethers .JsonRpcProvider (RPC_URL)

// Initialize signers const

owner1Signer

=

new

ethers .Wallet (process . env . OWNER_1_PRIVATE_KEY ! , provider) const

owner2Signer

=

new

ethers .Wallet (process . env . OWNER_2_PRIVATE_KEY ! , provider) const

owner3Signer

=

new

ethers .Wallet (process . env . OWNER_3_PRIVATE_KEY ! , provider)

const

ethAdapterOwner1

=

new

EthersAdapter ({ ethers , signerOrProvider : owner1Signer })

Initialize the API Kit

TheAPI Kit(opens in a new tab) consumes theSafe Transaction Service API(opens in a new tab) . To use this library, create a new instance of theSafeApiKit class, imported from@safe-global/api-kit . In chains where Safe provides a Transaction

Service, it's enough to specify thechainId. You can specify your own service using the optionaltxServiceUrl parameter.

You will be using Sepolia for this tutorial, however, you can also getservice URLs for different networks .

import SafeApiKit from

'@safe-global/api-kit'

const

apiKit

=

new

SafeApiKit ({ chainId :

1 n })

// or using a custom service const

apiKit

=

new

SafeApiKit ({ chainId :

1 n ,

// set the correct chainId txServiceUrl :

'https://url-to-your-custom-service' })

Initialize the Protocol Kit

Sepolia is a supported network so you don't need to specify the contract addresses, however, to see how to create a safe on a local or unsupported network, seeInstantiate an EthAdapter(opens in a new tab) .

Safe Factory is used to create Safes. While Safe class represents an instance of a specific Safe account.

import { SafeFactory } from

'@safe-global/protocol-kit'

const

safeFactory

=

await

SafeFactory .create ({ ethAdapter : ethAdapterOwner1 })

Deploy a Safe

Calling thedeploySafe method will deploy the desired Safe and return a Protocol Kit initialized instance ready to be used. Check theAPI Reference(opens in a new tab) for more details on additional configuration parameters and callbacks.

import { SafeAccountConfig } from

'@safe-global/protocol-kit'

const

safeAccountConfig :

SafeAccountConfig

= { owners : [await

owner1Signer .getAddress () , await

owner2Signer .getAddress () , await

owner3Signer .getAddress ()] , threshold :

2 , // ... (Optional params) }

/ *This Safe is tied to owner 1 because the factory was initialized with an adapter that had owner 1 as the signer*/ const

protocolKitOwner1

=

await

safeFactory .deploySafe ({ safeAccountConfig })

const

safeAddress

=

await

protocolKitOwner1 .getAddress ()

console .log ('Your Safe has been deployed:') console .log (https://sepolia.etherscan.io/address/ { safeAddress }) console .log (https://app.safe.global/sep: { safeAddress })

Send ETH to the Safe

You will send some ETH to this Safe.

const

safeAddress

=

protocolKit .getAddress ()

const

safeAmount

=

ethers .parseUnits ('0.01' ,

'ether') .toHexString ()

const

transactionParameters

= { to : safeAddress , value : safeAmount }

const

tx

=

await

owner1Signer .sendTransaction (transactionParameters)

console .log ('Fundraising.') console .log (Deposit Transaction: https://sepolia.etherscan.io/tx/ { tx .hash })

Making a transaction from a Safe

The first signer will sign and propose a transaction to send 0.005 ETH out of the Safe. Then, the second signer will add their own proposal and execute the transaction since it meets the 2 of 3 thresholds.

At a high level, making a transaction from the Safe requires the following steps:

Overview

The high-level overview of a multi-sig transaction is PCE: Propose. Confirm. Execute.

1. First signer proposes a transaction
2.
    1. Create transaction: define the amount, destination, and any additional data
3.
    1. Perform an off-chain signature of the transaction before proposing
4.
    1. Submit the transaction and signature to the Safe Transaction Service
5. Second signer confirms the transaction
6.
    1. Get pending transactions from the Safe service
7.
    1. Perform an off-chain signature of the transaction
8.
    1. Submit the signature to the service
9. Anyone executes the transaction
10.
    1. In this example, the first signer executes the transaction
11.
    1. Anyone can get the pending transaction from the Safe service
12.
    1. Account executing the transaction pays the gas fee

Create a transaction

For more details on what to include in a transaction see Create a Transaction in the Safe Core SDK Guide(opens in a new tab) .

import { MetaTransactionData } from

'@safe-global/safe-core-sdk-types'

// Any address can be used. In this example you will use vitalik.eth const

destination

=

'0xd8dA6BF26964aF9D7eEd9e03E53415D37aA96045' const

amount

=

ethers .parseUnits ('0.005' ,

'ether') .toString ()

const

safeTransactionData :

MetaTransactionData

= { to : destination , data :

'0x' , value : amount } // Create a Safe transaction with the provided parameters const

safeTransaction

=

await

protocolKitOwner1 .createTransaction ({ transactions : [safeTransactionData] })

Propose the transaction

To propose a transaction to the Safe Transaction Service we need to call theproposeTransaction method from the API Kit instance.

For a full list and description of the properties thatproposeTransaction accepts, seePropose the transaction to the service(opens in a new tab) in the Safe Core SDK guide.

// Deterministic hash based on transaction parameters const

safeTxHash

=

await

protocolKitOwner1 .getTransactionHash (safeTransaction)

// Sign transaction to verify that the transaction is coming from owner 1 const

senderSignature

=

await

protocolKitOwner1 .signHash (safeTxHash)

await

apiKit .proposeTransaction ({ safeAddress , safeTransactionData :

safeTransaction .data , safeTxHash , senderAddress :

await

owner1Signer .getAddress () , senderSignature :

senderSignature .data , })

Get pending transactions

Recall that you created theapiKit inInitialize the API Kit .

const

pendingTransactions

=

await

apiKit .getPendingTransactions (safeAddress).results

Confirm the transaction: Second confirmation

When owner 2 is connected to the application, the Protocol Kit should be initialized again with the existing Safe address the address of the owner 2 instead of the owner 1.

// Assumes that the first pending transaction is the transaction you want to confirm const

transaction

= pendingTransactions[0] const

safeTxHash

=

transaction .safeTxHash

```
const
ethAdapterOwner2
=
new
EthersAdapter ({ ethers , signerOrProvider : owner2Signer })
const
protocolKitOwner2
=
await
Safe .create ({ ethAdapter : ethAdapterOwner2 , safeAddress })
const
signature
=
await
protocolKitOwner2 .signHash (safeTxHash) const
response
=
await
apiKit .confirmTransaction (safeTxHash ,
signature .data)
```

Execute the transaction

Anyone can execute the Safe transaction once it has the required number of signatures. In this example, owner 1 will execute the transaction and pay for the gas fees.

```
const
safeTransaction
=
await
apiKit .getTransaction (safeTxHash) const
executeTxResponse
=
await
protocolKit .executeTransaction (safeTransaction) const
receipt
=
await
executeTxResponse . transactionResponse ?.wait ()
console .log ( 'Transaction executed:' ) console .log (https://sepolia.etherscan.io/tx/ { receipt .transactionHash } )
```

Confirm that the transaction was executed

You know that the transaction was executed if the balance in your Safe changes.

const

afterBalance

=

await

protocolKit .getBalance ()

console .log (`The final balance of the Safe: { ethers .formatUnits (afterBalance ,

'ether') } ETH`)

node

index.js

Fundraising.

Initial

balance

of

Safe:

0.01

ETH Buying

a

car. The

final

balance

of

the

Safe:

0.005

ETH

Conclusion

In this quickstart, you learned how to create and deploy a Safe and to propose and then execute a transaction for the Safe.

SafeAuthPack Signatures

Was this page helpful?

Report issue