Message signatures

Using the Protocol Kit, this guide explains how to generate and sign messages from a Safe account, including plain string messages and EIP-712 JSON messages.

Before starting, check this guide'ssetup .

Create the message

Messages can be plain strings or valid EIP-712 typed data structures.

// An example of a string message const

STRING_MESSAGE

=

"I'm the owner of this Safe account"

// An example of a typed data message const

TYPED_MESSAGE

= { types : { EIP712Domain : [{ name :

'name' , type :

'string' } , { name :

'version' , type :

'string' } , { name :

'chainId' , type :

'uint256' } , { name :

'verifyingContract' , type :

'address' }] , Person : [{ name :

'name' , type :

'string' } , { name :

'wallets' , type :

'address[]' }] , Mail : [{ name :

'from' , type :

'Person' } , { name :

'to' , type :

'Person[]' } , { name :

'contents' , type :

'string' }] } , domain : { name :

'Ether Mail' , version :

'1' , chainId :

Number (chainId) , verifyingContract :

'0xCcCCccccCCCCcCCCCCCcCcCccCcCCCcCccccccC' } , primaryType :

'Mail' , message : { from : { name :

'Cow' , wallets : ['0xCD2a3d9F938E13CD947Ec05AbC7FE734Df8DD826' ,

'0xDeaDbeefdEAdbeefdEadbEEFdeadbeEFdEaDbeeF'] } , to : [{ name :

'Bob' , wallets : ['0xbBbBBBBbbBBBbbbBbbBbbbbBBbBbbbbBbBbbBBbB' , '0xB0BdaBea57B0BDABeA57b0bdABEA57b0BDabEa57' , '0xB0B0b0b0b0b0B000000000000000000000000000'] }] , contents :

'Hello, Bob!' } }

ThecreateMessage method in the Protocol Kit allows for creating new messages and returns an instance of theEthSafeMessage class. Here, we are passingTYPED_MESSAGE , butSTRING_MESSAGE could also be passed.

let safeMessage =

protocolKit .createMessage (TYPED_MESSAGE)

The returnedsafeMessage object contains the message data (safeMessage.data) and a map of owner-signature pairs (safeMessage.signatures). The structure is similar to theEthSafeTransaction class but applied for messages instead of transactions.

We uselet to initialize thesafeMessage variable because we will add the signatures later.

class

EthSafeMessage

implements

SafeMessage { data :

EIP712TypedData

|

string signatures :

Map < string ,

SafeSignature

=

new

Map () ... // Other props and methods }

Sign the message

Once thesafeMessage object is created, we need to collect the signatures from the signers who will sign it.

Following oursetup , we will sign a message withsafe3_4 , the main Safe account in this guide. To do that, we first need to sign the same message with its owners:owner1 ,owner2 ,safe1_1 , andsafe2_3 .

ECDSA signatures

This applies toowner1 andowner2 accounts, as both are EOAs.

ThesignMessage method takes thesafeMessage together with aSigningMethod and adds the new signature to thesignMessage.signatures map. Depending on the type of message, theSigningMethod can take these values:

- SigningMethod.ETH_SIGN
- SigningMethod.ETH_SIGN_TYPED_DATA_V4

// Connect the EthAdapter from owner1 protocolKit =

await

protocolKit .connect ({ ethAdapter : ethAdapter1 })

// Sign the safeMessage with owner1 // After this, the safeMessage contains the signature from owner1 safeMessage =

await

protocolKit .signMessage (safeMessage , SigningMethod . ETH_SIGN_TYPED_DATA_V4)

// Connect the EthAdapter from owner2 protocolKit =

await

protocolKit .connect ({ ethAdapter : ethAdapter2 })

// Sign the safeMessage with owner2 // After this, the safeMessage contains the signature from owner1 and owner2 safeMessage =

await

protocolKit .signMessage (safeMessage , SigningMethod . ETH_SIGN_TYPED_DATA_V4)

Smart contract signatures

When signing with a Safe account, theSigningMethod will take the valueSigningMethod.SAFE_SIGNATURE .

1/1 Safe account

This applies to thesafe1_1 account, another owner ofsafe3_4 .

We need to connect the Protocol Kit tosafe1_1 and theowner3 account (the only owner ofsafe1_1) and sign the message.

// Create a new message object let messageSafe1_1 =

await

createMessage (TYPED_MESSAGE)

// Connect the EthAdapter from owner3 and the address of safe1_1 protocolKit =

await

protocolKit .connect ({ ethAdapter : ethAdapter3 , safeAddress : safe1_1 })

// Sign the messageSafe1_1 with owner3 // After this, the messageSafe1_1 contains the signature from owner3 messageSafe1_1 =

await

signMessage (messageSafe1_1 , SigningMethod . SAFE_SIGNATURE , safe3_4 // Parent Safe address)

// Build the contract signature of safe1_1 const

signatureSafe1_1

=

await

buildContractSignature (Array .from (messageSafe1_1 . signatures .values ()) , safe1_1)

// Add the signatureSafe1_1 to safeMessage // After this, the safeMessage contains the signature from owner1, owner2 and safe1_1 safeMessage .addSignature (signatureSafe1_1)

When signing with a child Safe account, we need to specify the parent Safe address to generate the signature based on the version of the contract.

2/3 Safe account

This applies to thesafe2_3 account, another owner ofsafe3_4 .

We need to connect the Protocol Kit tosafe2_3 and theowner4 andowner5 accounts (owners ofsafe2_3) and sign the message.

// Create a new message object let messageSafe2_3 =

await

createMessage (TYPED_MESSAGE)

// Connect the EthAdapter from owner4 and the address of safe2_3 protocolKit =

await

protocolKit .connect ({ ethAdapter : ethAdapter4 , safeAddress : safe2_3 })

// Sign the messageSafe2_3 with owner4 // After this, the messageSafe2_3 contains the signature from owner4 messageSafe2_3 =

await

protocolKit .signMessage (messageSafe2_3 , SigningMethod . SAFE_SIGNATURE , safe3_4 // Parent Safe address)

// Connect the EthAdapter from owner5 protocolKit =

await

protocolKit .connect ({ ethAdapter : ethAdapter5 })

// Sign the messageSafe2_3 with owner5 // After this, the messageSafe2_3 contains the signature from owner5 messageSafe2_3 =

await

protocolKit .signMessage (messageSafe2_3 , SigningMethod . SAFE_SIGNATURE , safe3_4 // Parent Safe address)

// Build the contract signature of safe2_3 const

signatureSafe2_3

=

await

buildContractSignature (Array .from (messageSafe2_3 . signatures .values ()) , safe2_3)

// Add the signatureSafe2_3 to safeMessage // After this, the safeMessage contains the signature from owner1, owner2, safe1_1 and safe2_3 safeMessage .addSignature (signatureSafe2_3)

After following all the steps above, thesafeMessage now contains all the signatures from the owners of the Safe.

Publish the signed message

As messages aren't stored in the blockchain, we must make them public and available to others by storing them elsewhere.

Safe messages can be stored on-chain and off-chain:

- Off-chain
- : Messages are stored in the Safe Transaction Service. This is the default option and doesn't require any on-chain interaction.
- On-chain
- : Messages arestored(opens in a new tab)
- in the Safe contract.

Safe supports signingEIP-191(opens in a new tab) messages andEIP-712(opens in a new tab) typed data messages all together with off-chainEIP-1271(opens in a new tab) validation for signatures.

Off-chain messages

To use off-chain messages, we need to use the functionality from this guide and call the Safe Transaction Service API to store the messages and signatures.

We mentioned the utility of storing messages in the contract. Off-chain messages have the same purpose, but they're stored in the Safe Transaction Service. It stores the messages and signatures in a database. It's a centralized service, but it's open-source and can be deployed by anyone.

The Safe Transaction Service is used bySafe{Wallet} to store messages and signatures by default.

Propose the message

To store a new message, we need to call theaddMessage from the API Kit, passing the Safe address, an object with the

message, and a signature from one owner.

const

signerAddress

= (await

ethAdapter1 .getSignerAddress ()) ||

'0x'

// Get the signature from owner1 const

signatureOwner1

=

safeMessage .getSignature (signerAddress) as

EthSafeSignature

// Instantiate the API Kit // Use the chainId where you have the Safe account deployed const

apiKit

=

new

SafeApiKit ({ chainId })

// Propose the message apiKit .addMessage (safe3_4 , { message :

TYPED_MESSAGE ,

// or STRING_MESSAGE signature :

buildSignatureBytes ([signatureOwner1]) })

The message is now publicly available in the Safe Transaction Service with the signature of the owner who submitted it.

Confirm the message

To add the signatures from the remaining owners, we need to call theaddMessageSignature , passing thesafeMessageHash and a signature from the owner.

// Get the safeMessageHash const

safeMessageHash

=

await

protocolKit .getSafeMessageHash (hashSafeMessage (TYPED_MESSAGE) // or STRING_MESSAGE)

// Get the signature from owner2 const

signerAddress

= (await

ethAdapter2 .getSignerAddress ()) ||

'0x' const

signatureOwner2

=

safeMessage .getSignature (signerAddress) as

EthSafeSignature

// Add signature from owner2 await

apiKit .addMessageSignature (safeMessageHash , buildSignatureBytes ([signatureOwner2]))

// Add signature from the owner safe1_1 await

apiKit .addMessageSignature (safeMessageHash , buildSignatureBytes ([signatureSafe1_1]))

// Add signature from the owner safe2_3 await

apiKit .addMessageSignature (safeMessageHash , buildSignatureBytes ([signatureSafe2_3]))

At this point, the message stored in the Safe Transaction Service contains all the required signatures from the owners of the Safe.

ThegetMessage method returns the status of a message.

const

confirmedMessage

=

await

apiKit .getMessage (safeMessageHash)

Safe{Wallet}(opens in a new tab) exposes to its users the list of off-chain messages signed by a Safe account.

https://app.safe.global/transactions/messages?safe=:

On-chain messages

Storing messages on-chain is less efficient than doing it off-chain because it requires executing a transaction to store the message hash in the contract, resulting in additional gas costs. To do this on-chain, we use theSignMessageLib contract.

// Get the contract with the correct version const

signMessageLibContract

=

await

ethAdapter1 .getSignMessageLibContract ({ safeVersion :

'1.4.1' })

We need to calculate themessageHash , encode the call to thesignMessage function in theSignMessageLib contract and create the transaction that will store the message hash in that contract.

const

messageHash

=

hashSafeMessage (MESSAGE) const

txData

=

signMessageLibContract .encode ('signMessage' , [messageHash])

const

safeTransactionData :

SafeTransactionDataPartial

= { to :

signMessageLibContract .address , value :

'0' , data : txData , operation :

OperationType .DelegateCall , }

const

signMessageTx

=

await

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

Once the transaction object is instantiated, the owners must sign and execute it.

// Collect the signatures using the signTransaction method

// Execute the transaction to store the messageHash await

protocolKit .executeTransaction (signMessageTx)

Once the transaction is executed, the message hash will be stored in the contract.

Validate the signature

On-chain

When a message is stored on-chain, theisValidSignature method in the Protocol Kit needs to be called with the parametersmessageHash and0x . The method will check the stored hashes in the Safe contract to validate the signature.

import { hashSafeMessage } from

'@safe-global/protocol-kit'

const

messageHash

=

hashSafeMessage (MESSAGE)

const

isValid

=

await

protocolKit .isValidSignature (messageHash ,

'0x')

Off-chain

When a message is stored off-chain, theisValidSignature method in the Protocol Kit must be called with themessageHash and theencodedSignatures parameters. The method will check theisValidSignature function defined in theCompatibilityFallbackHandler contract(opens in a new tab) to validate the signature.

const

encodedSignatures

=

safeMessage .encodedSignatures ()

const

isValid

=

await

protocolKit .isValidSignature (messageHash , encodedSignatures)

Transactions Reference

Was this page helpful?

Report issue