Interchain Account Interface

Developers can use theInterchain Account interface to create and control an account on a remote chain from their local chain.

Unlike general message passing, which requires recipients to implement a specific interface, Interchain Accounts (ICAs) allow developers to interact withany remote contract.

Overview

Interchain Accounts allow you to make a remote call fromChain A toChain B using the router (InterchainAccountRouter). Here's how it works:

- We useCREATE2
- to compute the deterministicOwnableMulticall
- contract address for you, which serves as a proxy for your cross-chain calls. You can explore thishere
- .
- You can encode your call which includes the to address, call data, and themsg.value
- for each call, batched together in an array.
- You send the encoded call to theChain A
- router which gets relayed to theChain B
- router.
- After decoding the calls, theChain B
- router checks if the computed address is already deployed or not. If not, we deploy theOwnableMulticall
- contract.
- The router then performs a multicall on the ICA address, which in turn makes the desired arbitrary call onChain B
- .

The Interchain Account interface assigns every(uint32 origin, address owner, address remoteRouter, address remoteISM) tuple a unique ICA address. The sender owns that ICA on the destination chain, and can direct it to make arbitrary function calls via theInterchainAccountRouter.callRemote() endpoint.

For core chains supported by Hyperlane, you are able to use the defaults that are set by the owner of the router contract. See the#overrides section to see how to make calls toany chain.

Interface

```
// SPDX-License-Identifier: MIT OR Apache-2.0 pragma

solidity

   = 0.6.11 ;

import

{ CallLib }

from

"../contracts/libs/Call.sol" ;

interface

IInterchainAccountRouter

{ function

callRemote ( uint32 _destinationDomain , CallLib . Call [ ]

calldata calls )

external

returns

( bytes32 ) ;

function

getRemoteInterchainAccount ( uint32 _destination ,
```

address _owner) external view returns

(address) ; } tip * UseInterchainAccountRouter * out of the box - ICA routers have already been deployed to core chains. Please refer toaddresses * . Try using thecallRemote * method to do a call via your wallet's interchain account.

Example Usage

Encoding

To use thecallRemote function, first prepare an array ofCall structs.Call.data can be easily encoded with theabi.encodeCall function.

struct

Call

{ bytes32 to ;

// supporting non EVM targets uint256 value ; bytes data ; }

interface

IUniswapV3Pool

{ function

swap (address recipient , bool zeroForOne , int256 amountSpecified , uint160 sqrtPriceLimitX96 , bytes

calldata data)

external

returns

(int256 amount0 ,

int256 amount1) ; }

IUniswapV3Pool pool

IUniswapV3Pool (. . .) ; Call swapCall =

Call ({ to : TypeCasts . addressToBytes32 (address (pool)) , data : abi . encodeCall (pool . swap ,

(. . .)) ; value :

0 , }) ; uint32 ethereumDomain =

1 ; IInterchainAccountRouter (0xabc . . .) . callRemote (ethereumDomain ,

[swapCall]) ;

Typescript Usage

We also have Typescript tooling to easily deploy ICA accounts and callcallRemote on the origin chain:

const localChain =

'ethereum' ; const signer =

< YOUR_SIGNER

> ; const localRouter : InterchainAccountRouter = InterchainAccountRouter__factory . connect (< ICA_ROUTER_ADDRESS
>
> , signer) ; const recipientAddress =

< EXAMPLE_ADDRESS

> ;

```
// use your own address here const recipientF =

new

TestRecipient__factory . connect ( recipientAddress , signer ) ;

// use your own contract here const fooMessage =

"Test" ; const data = recipient . interface . encodeFunctionData ( "fooBar" ,

[ 1 , fooMessage ] ) ;

const call =

{ to : recipientAddress , data , value : BigNumber . from ( "0" ) , } ; const quote =

await local [ "quoteGasPayment(uint32)" ] ( multiProvider . getDomainId ( remoteChain ) ) ;

const config : AccountConfig =

{ origin : localChain , owner : signer . address , localRouter : localRouter . address , } ; await localRouter . callRemote (
localChain , remoteChain ,

[ call ] , config ) ;
```

Determine addresses

It may be useful to know the remote address of your ICA before sending a message. For example, you may want to first fund the address with tokens. ThegetRemoteInterchainAccount function can be used to get the address of an ICA given the destination chain and owner address.

An example is included below of a contract precomputing its own Interchain Account address.

```
address myInterchainAccount =
```

IInterchainAccountRouter (. . .) . getRemoteInterchainAccount (destination , address (this)) ; If you are using#overrides to specify remote chains, pass those overrides when computing the remote ICA address.

```
address myRemoteIca =

IInterchainAccountRouter ( . . . ) . getRemoteInterchainAccount ( address ( this ) , remoteRouterOverride ,
remoteIsmOverride ) ;
```

Overrides

Interchain Accounts allow developers to override the default chains and security models configured in theInterchainAccountRouter .

These are useful for:

- Calling an ICA on chains not configured inInterchainAccountRouter
- .
- Using different ISM than the defaults configured in theInterchainAccountRouter
- Adjusting the gas limit for IGP payments or setting other parameters.

Interface

ThecallRemoteWithOverrides function looks similar to thecallRemote function, but takes three additional arguments.

First, developers can override_router , the address of theInterchainAccountRouter on the remote chain. This allows developers to control an ICA on remote chains that have not been configured on the localInterchainAccountRouter .

Second, developers can override_ism , the address of the remote interchain security module (ISM) used to secure their ICA. This ISM will be used to verify the interchain messages passed between the local and remoteInterchainAccountRouters . This allows developers to use a custom security model that best suits their needs.

Third, developers can override_hookMetadata , theStandardHookMetadata metadata passed to the message hooks for each ICA call (for example, overriding the gas limit for the IGP payment).

/ * @notice Dispatches a sequence of remote calls to be made by an owner's * interchain account on the destination domain * @dev Recommend using CallLib.build to format the interchain calls * @param _destination The remote domain of*

*the chain to make calls on * @param _router The remote router address * @param _ism The remote ISM address * @param _calls The sequence of calls to make * @param _hookMetadata The hook metadata to override with for the hook set by the owner * @return The Hyperlane message ID* / function

callRemoteWithOverrides (uint32 _destination , bytes32 _router , bytes32 _ism , CallLib . Call [ ]

calldata _calls , bytes

memory _hookMetadata)

public

payable

returns

(bytes32)

function

getRemoteInterchainAccount (address _owner , address _router , address _ism)

public

view

returns

(address) Edit this page Previous Warp Routes: Example Usage Next Deploying a Bridge UI for Hyperlane Warp Routes