Communicating Between OP Mainnet and Ethereum in Solidity

This tutorial explains how to write Solidity contracts on OP Mainnet and Ethereum that can talk to each other. Here you'll use a contract on OP Mainnet that can set a "greeting" variable on a contract on Ethereum, and vice-versa. This is a simple example, but the same technique can be used to send any kind of message between the two chains.

You won't actually be deploying any smart contracts as part of this tutorial. Instead, you'll reuse existing contracts that have already been deployed to OP Mainnet and Ethereum. Later in the tutorial you'll learn exactly how these contracts work so you can follow the same pattern to deploy your own contracts.

Just looking to bridge tokens between OP Mainnet and Ethereum? Check out the tutorial onBridging ERC-20 Tokens to OP Mainnet With the Optimism SDK .

Message Passing Basics

OP Mainnet uses a smart contract called theCrossDomainMessenger to pass messages between OP Mainnet and Ethereum. Both chains have a version of this contract (theL1CrossDomainMessenger and theL2CrossDomainMessenger). Messages sent from Ethereum to OP Mainnet are automatically relayed behind the scenes. Messages sent from OP Mainnet to Ethereum must be explicitly relayed with a second transaction on Ethereum. Read more about message passing in the guide toSending Data Between L1 and L2 .

Dependencies

- node(opens in a new tab)
- pnpm(opens in a new tab)

Get ETH on Sepolia and OP Sepolia

This tutorial explains how to send messages from Sepolia to OP Sepolia. You will need to get some ETH on both of these testnets.

You can usethis faucet(opens in a new tab) to get ETH on Sepolia. You can use theSuperchain Faucet(opens in a new tab) to get ETH on OP Sepolia.

Review the Contracts

s You're about to use two contracts that have already been deployed to Sepolia and OP Sepolia, theGreeter contracts. You can review the source code for the L1Greeter contracthere on Etherscan(opens in a new tab) . You can review the source code for the L2Greeter contracthere on Etherscan(opens in a new tab) . Both contracts have exactly the same source code.

Feel free to review the source code for these two contracts now if you'd like. This tutorial will explain how these contracts work in detail later on in theHow It Works section below.

Interact With the L1 Greeter

You're first going to use the L1Greeter contract to set the greeting on the L2Greeter contract. You'll send a transaction directly to the L1Greeter contract which will ask theL1CrossDomainMessenger to send a message to the L2Greeter contract. After just a few minutes, you'll see the corresponding greeting set on the L2Greeter contract.

Connect to Etherscan

Sending a message to the L2Greeter contract via the L1Greeter contract requires that you call thesendGreeting function. For simplicity, you'll interact with the contract directly on Etherscan. Open up theL1Greeter contract on Sepolia Etherscan(opens in a new tab) and click the "Connect to Web3" button.

Send your greeting

Put a greeting into the field next to the "sendGreeting" function and click the "Write" button. You can use any greeting you'd like.

Wait a few minutes

It will take a few minutes for your message to reach L2. Feel free to take a quick break while you wait.

You can use the Optimism SDK to programmatically check the status of any message between L1 and L2. Later on in this tutorial you'll learn how to use the Optimism SDK and thewaitForMessageStatus function to wait for various message statuses. This same function can be used to wait for a message to be relayed from L1 to L2.

Check the L2 Greeter

After a few minutes, you should see the greeting on the L2Greeter contract change to the greeting you set. Open up theL2Greeter contract on OP Sepolia Etherscan(opens in a new tab)and click the "Read Contract" button. Paste your address into the field next to the "greeting" function and click the "Query" button. You should see the message you sent from L1.

Don't see your message yet? You might need to wait a little longer. L2 transactions triggered on L1 are typically processed within one minute but can occasionally be slightly delayed.

Interact With the L2 Greeter

Now you're going to use the L2Greeter contract to set the greeting on the L1Greeter contract. You'll send a transaction directly to the L2Greeter contract which will ask theL2CrossDomainMessenger to send a message to the L1Greeter contract. Unlike the previous step, you'll need to relay the message from L2 to L1 yourself! You'll do this by sending two transactions on Sepolia, one proving transaction and one relaying transaction.

Connect to Etherscan

Just like before, sending a message to the L1Greeter contract via the L2Greeter contract requires that you call thesendGreeting function. Open up theL2Greeter contract on OP Sepolia Etherscan(opens in a new tab)and click the "Connect to Web3" button.

Send your greeting

Put a greeting into the field next to the "sendGreeting" function and click the "Write" button. You can use any greeting you'd like.

Copy the transaction hash from the transaction you just sent. You'll need this for the next few steps. Feel free to keep this tab open so you can easily copy the transaction hash later.

Create a demo project folder

You're going to use the Optimism SDK to prove and relay your message to L1. Since the Optimism SDK is a Node.js(opens in a new tab) library, you'll need to create a Node.js project to use it.

mkdir

op-sample-project cd

op-sample-project

Initialize the Project

Set up the project as a basic Node.js project withpnpm or your favorite package manager.

pnpm

init

Install the Optimism SDK

Install the Optimism SDK withpnpm or your favorite package manager.

pnpm

add

@eth-optimism/sdk

Install ethers.js

Installethers withpnpm or your favorite package manager.

pnpm

add

ethers@^5

Add your private key to your environment

You need a private key in order to sign transactions. Set your private key as an environment variable with theexport command. Make sure this is the private key for the address you used to send the transaction to the L2Greeter contract.

export TUTORIAL_PRIVATE_KEY = 0 x...

Add your transaction hash to your environment

You'll also need the hash of the transaction you sent to the L2Greeter contract. Set this as an environment variable with theexport command.

export TUTORIAL_TRANSACTION_HASH = 0 x...

Start a Node REPL

Now you'll use the Node.js REPL to run a few commands. Start the Node.js REPL with thenode command.

node

Import the Optimism SDK

const

optimism

=

require ("@eth-optimism/sdk")

Import ethers.js

const

ethers

=

require ("ethers")

Load your private key

const

privateKey

=

process . env . TUTORIAL_PRIVATE_KEY

Load your transaction hash

const

transactionHash

=

process . env . TUTORIAL_TRANSACTION_HASH

Create the RPC providers and wallets

const

l1Provider

=

new

ethers . providers .StaticJsonRpcProvider ("https://rpc.ankr.com/eth_sepolia") const

l2Provider

=

new

ethers . providers .StaticJsonRpcProvider ("https://sepolia.optimism.io") const

l1Wallet

=

new

ethers .Wallet (privateKey , l1Provider) const

l2Wallet

=

new

ethers .Wallet (privateKey , l2Provider)

Create a CrossChainMessenger instance

The Optimism SDK exports aCrossChainMessenger class that makes it easy to prove and relay cross-chain messages.

Create an instance of theCrossChainMessenger class:

const

messenger

=

new

optimism .CrossChainMessenger ({ l1ChainId :

11155111 ,

// 11155111 for Sepolia, 1 for Ethereum l2ChainId :

11155420 ,

// 11155420 for OP Sepolia, 10 for OP Mainnet l1SignerOrProvider : l1Wallet , l2SignerOrProvider : l2Wallet , })

Wait until the message is ready to prove

The second step to send messages from L2 to L1 is to prove that the message was sent on L2. You first need to wait until the message is ready to prove.

await

messenger .waitForMessageStatus (transactionHash ,

optimism . MessageStatus . READY_TO_PROVE) This step can take a few minutes. Feel free to take a quick break while you wait.

Prove the message on L1

Once the message is ready to be proven, you'll send an L1 transaction to prove that the message was sent on L2.

await

messenger .proveMessage (transactionHash)

Wait until the message is ready for relay

The final step to sending messages from L2 to L1 is to relay the messages on L1. This can only happen after the fault proof period has elapsed. On OP Sepolia, this is only a few seconds. On OP Mainnet, this takes 7 days.

await

messenger .waitForMessageStatus (transactionHash ,

optimism . MessageStatus . READY_FOR_RELAY)

Relay the message on L1

Once the withdrawal is ready to be relayed you can finally complete the message sending process.

await

messenger .finalizeMessage (transactionHash)

Wait until the message is relayed

Now you simply wait until the message is relayed.

await

messenger .waitForMessageStatus (transactionHash ,

optimism . MessageStatus . RELAYED)

Check the L1 Greeter

Now that you've relayed the message, you should see the greeting on the L1Greeter contract change to the greeting you set. Open up theL1Greeter contract on Sepolia Etherscan(opens in a new tab)and click the "Read Contract" button. Paste your address into the field next to the "greeting" function and click the "Query" button. You should see the message you sent from L2.

How It Works

Congratulations! You've successfully sent a message from L1 to L2 and from L2 to L1. This section will explain how theGreeter contracts work so you can follow the same pattern to deploy your own contracts. Luckily, bothGreeter contracts are exactly the same so it's easy to see how everything comes together.

The Messenger Variable

TheGreeter contract has aMESSENGER variable that keeps track of theCrossDomainMessenger contract on the current chain. Check out theContract Addresses page to see the addresses of theCrossDomainMessenger contracts on whichever network you'll be using.

ICrossDomainMessenger public

immutable MESSENGER;

The Other Greeter Variable

TheGreeter contract also has anOTHER_GREETER variable that keeps track of theGreeter contract on the other chain. On L1, this variable is set to the address of the L2Greeter contract, and vice-versa.

Greeter public

immutable OTHER_GREETER;

The Greetings Mapping

TheGreeter contract keeps track of the different greetings that users have sent inside agreetings mapping. By using a mapping, this contract can keep track of greetings from different users at the same time.

mapping (address

=>

string) public greetings;

The Constructor

TheGreeter has a simple constructor that sets theMESSENGER andOTHER_GREETER variables.

constructor (ICrossDomainMessenger

_messenger , Greeter

_otherGreeter) { MESSENGER = _messenger; OTHER_GREETER = _otherGreeter; }

The Send Greeting Function

ThesendGreeting function is the most important function in theGreeter contract. This is what you called earlier to send messages in both directions. All this function is doing is using thesendMessage function found within theCrossChainMessenger contract to send a message to theGreeter contract on the other chain.

Here, the first parameter is the address of the recipient of the message (theGreeter contract on the other chain). The second parameter is the ABI-encoded function call to thesetMessage function. The final parameter is the gas limit that gets used when the message is relayed on the other side.

function

sendGreeting (string

memory

_greeting) public { MESSENGER. sendMessage (address (OTHER_GREETER) , abi. encodeCall (this .setGreeting , (msg.sender , _greeting)) , 200000); }

The Set Greeting Function

ThesetMessage function is the function that actually sets the greeting. This function is called by theCrossDomainMessenger contract on the other chain. It checks explicitly that the function can only be called by theCrossDomainMessenger contract. It also checks that theCrossChainMessenger is saying that the message came from theGreeter contract on the other chain. Finally, it sets the greeting in thegreetings mapping.

function

setGreeting (address

_sender ,

string

memory

_greeting) public { require (msg.sender ==

address (MESSENGER) , "Greeter: Direct sender must be the CrossDomainMessenger");

require (MESSENGER. xDomainMessageSender () ==

address (OTHER_GREETER) , "Greeter: Remote sender must be the other Greeter contract");

greetings[_sender]

_greeting; } The tworequire statements in this function are important! Without them, anyone could call this function and set the greeting to whatever they want. You can follow a similar pattern in your own smart contracts.

Conclusion

You just learned how you can write Solidity contracts on Sepolia and OP Sepolia that can talk to each other. You can follow the same pattern to write contracts that can talk to each other on Ethereum and OP Mainnet.

This sort of cross-chain communication is useful for a variety of reasons. For example, theStandard Bridge contracts use this same system to bridge ETH and ERC-20 tokens between Ethereum and OP Mainnet.

One cool way to take advantage of cross-chain communication is to do most of your heavy lifting on OP Mainnet and then send a message to Ethereum only when you have important results to share. This way you can take advantage of the low gas costs on OP Mainnet while still being able to use Ethereum when you need it.

Deploying Your First Contract on OP MainnetBridging ETH With the Optimism SDK