Transfer Tokens with Data

In this tutorial, you will use Chainlink CCIP to transfer tokens and arbitrary data between smart contracts on different blockchains. First, you will pay for the CCIP fees on the source blockchain using LINK. Then, you will use the same contract to pay CCIP fees in native gas tokens. For example, you would use ETH on Ethereum or MATIC on Polygon.

Node Operator Rewards

CCIP rewards the oracle node and Risk Management node operators in LINK.

Transferring tokens

This tutorial uses the term "transferring tokens" even though the tokens are not technically transferred. Instead, they are locked or burned on the source chain and then unlocked or minted on the destination chain. Read theToken Pools section to understand the various mechanisms that are used to transfer value across chains.

Before you begin

1. You should understand how to write, compile, deploy, and fund a smart contract. If you need to brush up on the basics, read thistutorial , which will guide you through using theSolidity programming language , interacting with theMetaMask wallet and working within theRemix Development Environment .
2. Your account must have some ETH and LINK tokens onEthereum Sepoliaand MATIC tokens onPolygon Mumbai. Learn how toAcquire testnet LINK .
3. Check theSupported Networks page to confirm that the tokens you will transfer are supported for your lane. In this example, you will transfer tokens fromEthereum SepoliatoPolygon Mumbaiso check the list of supported tokenshere .
4. Learn how toacquire CCIP test tokens . Following this guide, you should have CCIP-BnM tokens, and CCIP-BnM should appear in the list of your tokens in MetaMask.
5. Learn how tofund your contract . This guide shows how to fund your contract in LINK, but you can use the same guide for funding your contract with any ERC20 tokens as long as they appear in the list of tokens in MetaMask.
6. Follow the previous tutorial:Transfer tokens .

Tutorial

In this tutorial, you will send astringtext and CCIP-BnM tokens between smart contracts onEthereum SepoliaandPolygon Mumbaiusing CCIP. First, you will payCCIP fees in LINK , then you will payCCIP fees in native gas .

```
// SPDX-License-Identifier: MITpragmasolidity0.8.19;import{IRouterClient}from"@chainlink/contracts-ccip/src/v0.8/ccip/interfaces/IRouterClient.sol";import{OwnerIsCreator}from"@chainlink/contracts-
ccip/src/v0.8/shared/access/OwnerIsCreator.sol";import{Client}from"@chainlink/contracts-ccip/src/v0.8/ccip/libraries/Client.sol";import{CCIPReceiver}from"@chainlink/contracts-
ccip/src/v0.8/ccip/applications/CCIPReceiver.sol";import{IERC20}from"@chainlink/contracts-ccip/src/v0.8/vendor/openzeppelin-
solidity/v4.8.3/contracts/token/ERC20/IERC20.sol";import{SafeERC20}from"@chainlink/contracts-ccip/src/v0.8/vendor/openzeppelin-solidity/v4.8.3/contracts/token/ERC20/utils/SafeERC20.sol";/ * THIS
IS AN EXAMPLE CONTRACT THAT USES HARDCODED VALUES FOR CLARITY. * THIS IS AN EXAMPLE CONTRACT THAT USES UN-AUDITED CODE. * DO NOT USE THIS CODE IN
PRODUCTION. *//// @title - A simple messenger contract for transferring/receiving tokens and data across
chains.contractProgrammableTokenTransfersisCCIPReceiver,OwnerIsCreator{usingSafeERC20forIERC20;// Custom errors to provide more descriptive revert
messages.errorNotEnoughBalance(uint256currentBalance,uint256calculatedFees);// Used to make sure contract has enough balance to cover the fees.errorNothingToWithdraw();// Used
when trying to withdraw Ether but there's nothing to withdraw.errorFailedToWithdrawEth(addressowner,addresstarget,uint256value);// Used when the withdrawal of Ether
fails.errorDestinationChainNotAllowed(uint64destinationChainSelector);// Used when the destination chain has not been allowlisted by the contract
owner.errorSourceChainNotAllowed(uint64sourceChainSelector);// Used when the source chain has not been allowlisted by the contract owner.errorSenderNotAllowed(addresssender);//
Used when the sender has not been allowlisted by the contract owner.errorInvalidReceiverAddress();// Used when the receiver address is 0.// Event emitted when a message is sent to
another chain.eventMessageSent(bytes32indexedmessageId,// The unique ID of the CCIP message.uint64indexeddestinationChainSelector,// The chain selector of the destination
chain.addressreceiver,// The address of the receiver on the destination chain.stringtext,// The text being sent.addresstoken,// The token address that was
transferred.uint256tokenAmount,// The token amount that was transferred.addressfeeToken,// the token address used to pay CCIP fees.uint256fees// The fees paid for sending the
message.);// Event emitted when a message is received from another chain.eventMessageReceived(bytes32indexedmessageId,// The unique ID of the CCIP
message.uint64indexedsourceChainSelector,// The chain selector of the source chain.addresssender,// The address of the sender from the source chain.stringtext,// The text that was
received.addresstoken,// The token address that was transferred.uint256tokenAmount// The token amount that was transferred.);bytes32privates_lastReceivedMessageId;// Store the last
received messageId.addressprivates_lastReceivedTokenAddress;// Store the last received token address.uint256privates_lastReceivedTokenAmount;// Store the last received
amount.stringprivates_lastReceivedText;// Store the last received text.// Mapping to keep track of allowlisted destination
chains.mapping(uint64=>bool)publicallowlistedDestinationChains;// Mapping to keep track of allowlisted source chains.mapping(uint64=>bool)publicallowlistedSourceChains;// Mapping
to keep track of allowlisted senders.mapping(address=>bool)publicallowlistedSenders;IERC20privates_linkToken;/// @notice Constructor initializes the contract with the router address.///
@param router The address of the router contract./// @param _link The address of the link contract.constructor(address_router,address_link)CCIPReceiver(_router)
{s_linkToken=IERC20(_link);}/// @dev Modifier that checks if the chain with the given destinationChainSelector is allowlisted./// @param _destinationChainSelector The selector of the
destination chain.modifieronlyAllowlistedDestinationChain(uint64_destinationChainSelector)
{if(!allowlistedDestinationChains[_destinationChainSelector])revertDestinationChainNotAllowed(_destinationChainSelector);_;}/// @dev Modifier that checks the receiver address is not 0.///
@param receiver The receiver address.modifiervalidateReceiver(address_receiver){if(_receiver==address(0))revertInvalidReceiverAddress();_;}/// @dev Modifier that checks if the chain
with the given sourceChainSelector is allowlisted and if the sender is allowlisted./// @param sourceChainSelector The selector of the destination chain./// @param _sender The address of
the sender.modifieronlyAllowlisted(uint64_sourceChainSelector,address_sender)
{if(!allowlistedSourceChains[_sourceChainSelector])revertSourceChainNotAllowed(_sourceChainSelector);if(!allowlistedSenders[_sender])revertSenderNotAllowed(_sender);_;}/// @dev
Updates the allowlist status of a destination chain for transactions./// @notice This function can only be called by the owner./// @param _destinationChainSelector The selector of the
destination chain to be updated./// @param allowed The allowlist status to be set for the destination
chain.functionallowlistDestinationChain(uint64_destinationChainSelector,boolallowed)externalonlyOwner{allowlistedDestinationChains[_destinationChainSelector]=allowed;}/// @dev
Updates the allowlist status of a source chain/// @notice This function can only be called by the owner./// @param _sourceChainSelector The selector of the source chain to be updated.///
@param allowed The allowlist status to be set for the source
chain.functionallowlistSourceChain(uint64_sourceChainSelector,boolallowed)externalonlyOwner{allowlistedSourceChains[_sourceChainSelector]=allowed;}/// @dev Updates the allowlist
status of a sender for transactions./// @notice This function can only be called by the owner./// @param _sender The address of the sender to be updated./// @param allowed The allowlist
status to be set for the sender.functionallowlistSender(address_sender,boolallowed)externalonlyOwner{allowlistedSenders[_sender]=allowed;}/// @notice Sends data and transfer tokens
to receiver on the destination chain./// @notice Pay for fees in LINK./// @dev Assumes your contract has sufficient LINK to pay for CCIP fees./// @param _destinationChainSelector The
identifier (aka selector) for the destination blockchain./// @param _receiver The address of the recipient on the destination blockchain./// @param _text The string data to be sent.///
@param _token token address./// @param _amount token amount./// @return messageId The ID of the CCIP message that was
sent.functionsendMessagePayLINK(uint64_destinationChainSelector,address_receiver,stringcalldata_text,address_token,uint256_amount)externalonlyOwneronlyAllowlistedDestinationCh
{// Create an EVM2AnyMessage struct in memory with necessary information for sending a cross-chain message// address(linkToken) means fees are paid in
LINKClient.EVM2AnyMessagememoryevm2AnyMessage=_buildCCIPMessage(_receiver,_text,_token,_amount,address(s_linkToken));// Initialize a router client instance to interact with
cross-chain routerIRouterClient router=IRouterClient(this.getRouter());// Get the fee required to send the CCIP
messageuint256fees=router.getFee(_destinationChainSelector,evm2AnyMessage);if(fees>s_linkToken.balanceOf(address(this)))revertNotEnoughBalance(s_linkToken.balanceOf(address(t
approve the Router to transfer LINK tokens on contract's behalf. It will spend the fees in LINKs_linkToken.approve(address(router),fees);// approve the Router to spend tokens on
contract's behalf. It will spend the amount of the given tokenIERC20(_token).approve(address(router),_amount);// Send the message through the router and store the returned message
IDmessageId=router.ccipSend(_destinationChainSelector,evm2AnyMessage);// Emit an event with message
detailsemitMessageSent(messageId,_destinationChainSelector,_receiver,_text,_token,_amount,address(s_linkToken),fees);// Return the message IDreturnmessageId;}/// @notice Sends
data and transfer tokens to receiver on the destination chain./// @notice Pay for fees in native gas./// @dev Assumes your contract has sufficient native gas like ETH on Ethereum or
MATIC on Polygon./// @param _destinationChainSelector The identifier (aka selector) for the destination blockchain./// @param _receiver The address of the recipient on the destination
blockchain./// @param _text The string data to be sent./// @param _token token address./// @param _amount token amount./// @return messageId The ID of the CCIP message that was
sent.functionsendMessagePayNative(uint64_destinationChainSelector,address_receiver,stringcalldata_text,address_token,uint256_amount)externalonlyOwneronlyAllowlistedDestinationC
{// Create an EVM2AnyMessage struct in memory with necessary information for sending a cross-chain message// address(0) means fees are paid in native
gasClient.EVM2AnyMessagememoryevm2AnyMessage=_buildCCIPMessage(_receiver,_text,_token,_amount,address(0));// Initialize a router client instance to interact with cross-chain
routerIRouterClient router=IRouterClient(this.getRouter());// Get the fee required to send the CCIP
messageuint256fees=router.getFee(_destinationChainSelector,evm2AnyMessage);if(fees>address(this).balance)revertNotEnoughBalance(address(this).balance,fees);// approve the
Router to spend tokens on contract's behalf. It will spend the amount of the given tokenIERC20(_token).approve(address(router),_amount);// Send the message through the router and
store the returned message IDmessageId=router.ccipSend{value:fees}(_destinationChainSelector,evm2AnyMessage);// Emit an event with message
detailsemitMessageSent(messageId,_destinationChainSelector,_receiver,_text,_token,_amount,address(0),fees);// Return the message IDreturnmessageId;}/ * @notice Returns the details of
the last CCIP received message. * @dev This function retrieves the ID, text, token address, and token amount of the last received CCIP message. * @return messageId The ID of the last received
CCIP message. * @return text The text of the last received CCIP message. * @return tokenAddress The address of the token in the last CCIP received message. * @return tokenAmount The amount
of the token in the last CCIP received message. */functiongetLastReceivedMessageDetails()publicviewreturns(bytes32messageId,stringmemorytext,addresstokenAddress,uint256tokenAmount)
{return(s_lastReceivedMessageId,s_lastReceivedText,s_lastReceivedTokenAddress,s_lastReceivedTokenAmount);}/// handle a received
messagefunction_ccipReceive(Client.Any2EVMMessagememoryany2EvmMessage)internaloverrideonlyAllowlisted(any2EvmMessage.sourceChainSelector,abi.decode(any2EvmMessage.sender,
(address)))// Make sure source chain and sender are allowlisted{s_lastReceivedMessageId=any2EvmMessage.messageId;// fetch the
messageIds_lastReceivedText=abi.decode(any2EvmMessage.data,(string));// abi-decoding of the sent text// Expect one token to be transferred at once, but you can transfer several
tokens.s_lastReceivedTokenAddress=any2EvmMessage.destTokenAmounts[0].token;s_lastReceivedTokenAmount=any2EvmMessage.destTokenAmounts[0].amount;emitMessageReceived(any2EvmM
fetch the source chain identifier (aka selector)abi.decode(any2EvmMessage.sender,(address)),// abi-decoding of the sender address,abi.decode(any2EvmMessage.data,
(string)),any2EvmMessage.destTokenAmounts[0].token,any2EvmMessage.destTokenAmounts[0].amount);}/// @notice Construct a CCIP message./// @dev This function will create an
EVM2AnyMessage struct with all the necessary information for programmable tokens transfer./// @param _receiver The address of the receiver./// @param _text The string data to be sent./// @param
_token The token to be transferred./// @param _amount The amount of the token to be transferred./// @param _feeTokenAddress The address of the token used for fees. Set address(0) for native
gas./// @return Client.EVM2AnyMessage Returns an EVM2AnyMessage struct which contains information for sending a CCIP
message.function_buildCCIPMessage(address_receiver,stringcalldata_text,address_token,uint256_amount,address_feeTokenAddress)privatepurereturns(Client.EVM2AnyMessagememory){// Set the
token amountsClient.EVMTokenAmount[]memorytokenAmounts=newClient.EVMTokenAmount;tokenAmounts[0]=Client.EVMTokenAmount({token:_token,amount:_amount});// Create an
```

EVM2AnyMessage struct in memory with necessary information for sending a cross-chain messagereturnClient.EVM2AnyMessage({receiver:abi.encode(_receiver),// ABI-encoded receiver addressdata:abi.encode(_text),// ABI-encoded stringtokenAmounts:tokenAmounts,// The amount and type of token being transferredextraArgs:Client._argsToBytes(// Additional arguments, setting gas limitClient.EVMExtraArgsV1({gasLimit:200_000})),// Set the feeToken to a feeTokenAddress, indicating specific asset will be used for feesfeeToken:_feeTokenAddress});}/// @notice Fallback function to allow the contract to receive Ether./// @dev This function has no function body, making it a default function for receiving Ether./// It is automatically called when Ether is sent to the contract without any data.receive()externalpayable{}/// @notice Allows the contract owner to withdraw the entire balance of Ether from the contract./// @dev This function reverts if there are no funds to withdraw or if the transfer fails./// It should only be callable by the owner of the contract./// @param _beneficiary The address to which the Ether should be sent.functionwithdraw(address_beneficiary)publiconlyOwner{// Retrieve the balance of this contractuint256amount=address(this).balance;// Revert if there is nothing to withdrawif(amount==0)revertNothingToWithdraw();// Attempt to send the funds, capturing the success status and discarding any return data(boolsent,)=_beneficiary.call{value:amount}("");// Revert if the send failed, with information about the attempted transferif(!sent)revertFailedToWithdrawEth(msg.sender,_beneficiary,amount);}/// @notice Allows the owner of the contract to withdraw all tokens of a specific ERC20 token./// @dev This function reverts with a 'NothingToWithdraw' error if there are no tokens to withdraw./// @param _beneficiary The address to which the tokens will be sent./// @param _token The contract address of the ERC20 token to be withdrawn.functionwithdrawToken(address_beneficiary,address_token)publiconlyOwner{// Retrieve the balance of this contractuint256amount=IERC20(_token).balanceOf(address(this));// Revert if there is nothing to withdrawif(amount==0)revertNothingToWithdraw();IERC20(_token).safeTransfer(_beneficiary,amount);}} Open in Remix What is Remix?

Deploy your contracts

To use this contract:

1. Open the contract in Remix.
2. Compile your contract.

3. Deploy, fund your sender contract onEthereum Sepoliaand enable sending messages toPolygon Mumbai:

4. Open MetaMask and select the networkEthereum Sepolia.

5. In Remix IDE, click onDeploy & Run Transactionsand selectInjected Provider - MetaMaskfrom the environment list. Remix will then interact with your MetaMask wallet to communicate withEthereum Sepolia.
6. Fill in your blockchain's router and LINK contract addresses. The router address can be found on thesupported networks page and the LINK contract address on theLINK token contracts page. ForEthereum Sepolia, the router address is0x0BF3dE8c5D3e8A2B34D2BEeB17ABfCeBaf363A59and the LINK contract address is0x779877A7B0D9E8603169DdbD7836e478b4624789.
7. Click thetransactbutton. After you confirm the transaction, the contract address appears on theDeployed Contractslist. Note your contract address.
8. Open MetaMask and fund your contract with CCIP-BnM tokens. You can transfer0.002CCIP-BnMto your contract.
9. Enable your contract to send CCIP messages toPolygon Mumbai:1. In Remix IDE, underDeploy & Run Transactions, open the list of transactions of your smart contract deployed onEthereum Sepolia.
10. Call theallowlistDestinationChain, setting the destination chain selector to12532609583862916517and settingallowedtotrue. Each chain selector is found on thesupported networks page.

11. Deploy your receiver contract onPolygon Mumbaiand enable receiving messages from your sender contract:

12. Open MetaMask and select the networkPolygon Mumbai.

13. In Remix IDE, underDeploy & Run Transactions, make sure the environment is stillInjected Provider - MetaMask.
14. Fill in your blockchain's router and LINK contract addresses. The router address can be found on thesupported networks page and the LINK contract address on theLINK token contracts page. ForPolygon Mumbai, the router address is0x1035CabC275068e0F4b745A29CEDf38E13aF41b1and the LINK contract address is0x326C977E6efc84E512bB9C30f76E30c160eD06FB.
15. Click thetransactbutton. After you confirm the transaction, the contract address appears on theDeployed Contractslist. Note your contract address.
16. Enable your contract to receive CCIP messages fromEthereum Sepolia:1. In Remix IDE, underDeploy & Run Transactions, open the list of transactions of your smart contract deployed onPolygon Mumbai.
17. Call theallowlistSourceChainwith16015286601757825753as the source chain selector, andtrueas allowed. Each chain selector is found on thesupported networks page.
18. Enable your contract to receive CCIP messages from the contract that you deployed onEthereum Sepolia:1. In Remix IDE, underDeploy & Run Transactions, open the list of transactions of your smart contract deployed onPolygon Mumbai.
19. Call theallowlistSenderwith the contract address of the contract that you deployed onEthereum Sepolia, andtrueas allowed.

At this point, you have onesendercontract onEthereum Sepoliaand onereceivercontract onPolygon Mumbai. As security measures, you enabled the sender contract to send CCIP messages toPolygon Mumbaiand the receiver contract to receive CCIP messages from the sender andEthereum Sepolia.

Note: Another security measure enforces that only the router can call the_ccipReceivefunction. Read theexplanation section for more details.

Transfer and Receive tokens and data and pay in LINK

You will transfer0.001 CCIP-BnMand a text. The CCIP fees for using CCIP will be paid in LINK. Read thisexplanation for a detailed description of the code example.

1. Open MetaMask and connect toEthereum Sepolia. Fund your contract with LINK tokens. You can transfer0.1LINKto your contract. In this example, LINK is used to pay the CCIP fees.

2. Send a string data with tokens fromEthereum Sepolia:

3. Open MetaMask and select the networkEthereum Sepolia.

4. In Remix IDE, underDeploy & Run Transactions, open the list of transactions of your smart contract deployed onEthereum Sepolia.
5. Fill in the arguments of thesendMessagePayLINKfunction:

ArgumentValue and Description_destinationChainSelector12532609583862916517CCIP Chain identifier of the destination blockchain (Polygon Mumbaiin this example). You can find each chain selector on thesupported networks page._receiverYour receiver contract address atPolygon Mumbai.The destination contract address._textHello World!Anystring_token0xFd57b4ddBf88a4e07fF4e34C487b99af2Fe82a05TheCCIP-BnMcontract address at the source chain (Ethereum Sepoliain this example). You can find all the addresses for each supported blockchain on thesupported networks page._amount1000000000000000The token amount (0.001 CCIP-BnM). 4. Click ontransactand confirm the transaction on MetaMask. 5. After the transaction is successful, record the transaction hash. Here is anexample of a transaction onEthereum Sepolia.

note

During gas price spikes, your transaction might fail, requiring more than0.1 LINKto proceed. If your transaction fails, fund your contract with moreLINKtokens and try again. 3. Open theCCIP explorer and search your cross-chain transaction using the transaction hash. 4. The CCIP transaction is completed once the status is marked as "Success". In this example, the CCIP message ID is0x1d49f223a33de970e98d05124223860b4bb0ca23cd93971aa3697bd203eb37f3. 5. Check the receiver contract on the destination chain:

1. Open MetaMask and select the networkPolygon Mumbai.
2. In Remix IDE, underDeploy & Run Transactions, open the list of transactions of your smart contract deployed onPolygon Mumbai.
3. Call thegetLastReceivedMessageDetailsfunction.
4. Notice the received messageId is0x1d49f223a33de970e98d05124223860b4bb0ca23cd93971aa3697bd203eb37f3, the received text isHello World!, the token address is0xf1E3A5842EeEF51F2967b3F05D45DD4f4205FF40(CCIP-BnM token address onPolygon Mumbai) and the token amount is 1000000000000000 (0.001 CCIP-BnM).

Note: These example contracts are designed to work bi-directionally. As an exercise, you can use them to transfer tokens with data fromEthereum SepoliatoPolygon Mumbaiand fromPolygon Mumbaiback toEthereum Sepolia.

Transfer and Receive tokens and data and pay in native

You will transfer0.001 CCIP-BnMand a text. The CCIP fees for using CCIP will be paid in Sepolia's native ETH. Read thisexplanation for a detailed description of the code example.

1. Open MetaMask and connect toEthereum Sepolia. Fund your contract with ETH tokens. You can transfer0.01ETHto your contract. The native gas tokens are used to pay the CCIP fees.

2. Send a string data with tokens fromEthereum Sepolia:

3. Open MetaMask and select the networkEthereum Sepolia.

4. In Remix IDE, underDeploy & Run Transactions, open the list of transactions of your smart contract deployed onEthereum Sepolia.
5. Fill in the arguments of thesendMessagePayNativefunction:

ArgumentValue and Description_destinationChainSelector12532609583862916517CCIP Chain identifier of the destination blockchain (Polygon Mumbaiin this example). You can find each chain selector on thesupported networks page._receiverYour receiver contract address atPolygon Mumbai.The destination contract address._textHello World!Anystring_token0xFd57b4ddBf88a4e07fF4e34C487b99af2Fe82a05TheCCIP-BnMcontract address at the source chain (Ethereum Sepoliain this example). You can find all the addresses for each supported blockchain on thesupported networks page._amount1000000000000000The token amount (0.001 CCIP-BnM). 4. Click ontransactand confirm the transaction on MetaMask. 5. Once the transaction is successful, note the transaction hash. Here is anexample of a transaction onEthereum Sepolia.

note

During gas price spikes, your transaction might fail, requiring more than0.01 ETHto proceed. If your transaction fails, fund your contract with moreETHand try again. 3. Open theCCIP explorer and search your cross-chain transaction using the transaction hash. 4. The CCIP transaction is completed once the status is marked as "Success". In this example, the CCIP message ID is0x83b396df0fd6fac599459bc68003c8721ee2c191f98ea8222cea19af62b9a3ef. Note that CCIP fees are denominated in LINK. Even if CCIP fees are paid using native gas tokens, node operators will be paid in LINK. 5. Check the receiver contract on the destination chain:

1. Open MetaMask and select the networkPolygon Mumbai.
2. In Remix IDE, underDeploy & Run Transactions, open the list of transactions of your smart contract deployed onPolygon Mumbai.
3. Call thegetLastReceivedMessageDetailsfunction.
4. Notice the received messageId is0x83b396df0fd6fac599459bc68003c8721ee2c191f98ea8222cea19af62b9a3ef, the received text isHello World!, the token address is0xf1E3A5842EeEF51F2967b3F05D45DD4f4205FF40(CCIP-BnM token address onPolygon Mumbai) and the token amount is 1000000000000000 (0.001 CCIP-BnM).

Note: These example contracts are designed to work bi-directionally. As an exercise, you can use them to transfer tokens with data fromEthereum SepoliatoPolygon Mumbaiand fromPolygon Mumbaiback toEthereum Sepolia.

Explanation

Integrate Chainlink CCIP into your project

npmyarnfoundryIf you useNPM , install the@chainlink/contracts-ccip NPM package and set it to the v1.4.0 release:

npminstall@chainlink/[email protected]If you useYarn , install the@chainlink/contracts-ccip NPM package and set it to the v1.4.0 release:

yarnadd@chainlink/[email protected]If you useFoundry , install the v1.4.0 release:

forgeinstallsmartcontractkit/ccip@b06a3c2eecb9892ec6f76a015624413fffa1a122

The smart contract featured in this tutorial is designed to interact with CCIP to transfer and receive tokens and data. The contract code contains supporting comments clarifying the functions, events, and underlying logic. Here we will further explain initializing the contract and sending data with tokens.

Initializing the contract

When deploying the contract, we define the router address and LINK contract address of the blockchain we deploy the contract on. Defining the router address is useful for the following:

- Sender part:
- Calls the router'sgetFeefunction to estimate the CCIP fees.
- Calls the router'sccipSendfunction to send CCIP messages.
- Receiver part:
- The contract inherits fromCCIPReceiver , which serves as a base contract for receiver contracts. This contract requires that child contracts implement the_ccipReceivefunction ._ccipReceiveis called by theccipReceivefunction , which ensures that only the router can deliver CCIP messages to the receiver contract.

Transferring tokens and data and pay in LINK

ThesendMessagePayLINKfunction undertakes six primary operations:

1. Call the_buildCCIPMessageprivate function to construct a CCIP-compatible message using theEVM2AnyMessagestruct :
2. The_receiveraddress is encoded in bytes to accommodate non-EVM destination blockchains with distinct address formats. The encoding is achieved throughabi.encode .
3. Thedatais encoded from astringtobytesusingabi.encode .
4. ThetokenAmountsis an array, with each element comprising anEVMTokenAmountstruct containing the token address and amount. The array contains one element where the_token(token address) and_amount(token amount) are passed by the user when calling thesendMessagePayLINKfunction.
5. TheextraArgsspecifies thegasLimitfor relaying the message to the recipient contract on the destination blockchain. In this example, thegasLimitis set to `200000.
6. The_feeTokenAddressdesignates the token address used for CCIP fees. Here,address(linkToken)signifies payment in LINK.

Do not hardcode extraArgs

To simplify this example,extraArgsare hardcoded in the contract. For production deployments, make sure thatextraArgsis mutable. This allows you to build it offchain and pass it in a call to a function or store it in a variable that you can update on-demand. This makesextraArgscompatible with future CCIP upgrades. 2. Computes the message fees by invoking the router'sgetFeefunction . 3. Ensures your contract balance in LINK is enough to cover the fees. 4. Grants the router contract permission to deduct the fees from the contract's LINK balance. 5. Grants the router contract permission to deduct the amount from the contract'sCCIP-BnMbalance. 6. Dispatches the CCIP message to the destination chain by executing the router'sccipSendfunction .

Note: As a security measure, thesendMessagePayLINKfunction is protected by theonlyAllowlistedDestinationChain, ensuring the contract owner has allowlisted a destination chain.

Transferring tokens and data and pay in native

ThesendMessagePayNativefunction undertakes five primary operations:

1. Call the_buildCCIPMessageprivate function to construct a CCIP-compatible message using theEVM2AnyMessagestruct :
2. The_receiveraddress is encoded in bytes to accommodate non-EVM destination blockchains with distinct address formats. The encoding is achieved throughabi.encode .
3. Thedatais encoded from astringtobytesusingabi.encode .
4. ThetokenAmountsis an array, with each element comprising anEVMTokenAmountstruct containing the token address and amount. The array contains one element where the_token(token address) and_amount(token amount) are passed by the user when calling thesendMessagePayNativefunction.
5. TheextraArgsspecifies thegasLimitfor relaying the message to the recipient contract on the destination blockchain. In this example, thegasLimitis set to `200000.
6. The_feeTokenAddressdesignates the token address used for CCIP fees. Here,address(0)signifies payment in native gas tokens (ETH).

Do not hardcode extraArgs

To simplify this example,extraArgsare hardcoded in the contract. For production deployments, make sure thatextraArgsis mutable. This allows you to build it offchain and pass it in a call to a function or store it in a variable that you can update on-demand. This makesextraArgscompatible with future CCIP upgrades. 2. Computes the message fees by invoking the router'sgetFeefunction . 3. Ensures your contract balance in native gas is enough to cover the fees. 4. Grants the router contract permission to deduct the amount from the contract'sCCIP-BnMbalance. 5. Dispatches the CCIP message to the destination chain by executing the router'sccipSendfunction .Note:msg.valueis set because you pay in native gas.

Note: As a security measure, thesendMessagePayNativefunction is protected by theonlyAllowlistedDestinationChain, ensuring the contract owner has allowlisted a destination chain.

Receiving messages

On the destination blockchain, the router invokes the_ccipReceivefunction which expects aAny2EVMMessagestruct that contains:

- The CCIPmessageId.
- ThesourceChainSelector.
- Thesenderaddress in bytes format. Given that the sender is known to be a contract deployed on an EVM-compatible blockchain, the address is decoded from bytes to an Ethereum address using theABI specifications .
- ThetokenAmountsis an array containing received tokens and their respective amounts. Given that only one token transfer is expected, the first element of the array is extracted.
- Thedata, which is also in bytes format. Given astringis expected, the data is decoded from bytes to a string using theABI specifications .

Note: Three important security measures are applied:

- _ccipReceiveis called by theccipReceivefunction , which ensures that only the router can deliver CCIP messages to the receiver contract. See theonlyRoutermodifier for more information.
- The modifieronlyAllowlistedensures that only a call from an allowlisted source chain and sender is accepted.