Transfer Tokens

In this tutorial, you will use Chainlink CCIP to transfer tokens from a smart contract to an account on a different blockchain. 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 Sepolia. 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 to fund your contract with any ERC20 tokens as long as they appear in the list of tokens in MetaMask.

Tutorial

In this tutorial, you will transferCCIP-BnM tokens from a contract on Ethereum Sepolia to an account on Polygon Mumbai. First, you will payCCIP fees in LINK , then you will payCCIP fees in native gas . The destination account can be anEOA (Externally Owned Account) or a smart contract. Moreover, the example shows how to transfer CCIP-BnM tokens, but you can re-use the same example to transfer other tokens as long as they are supported for yourlane .

// 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{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 contract for transferring tokens across chains.contractTokenTransferorisOwnerIsCreator{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.errorDestinationChainNotAllowlisted(uint64destinationChainSelector);// Used when the destination chain has not been allowlisted by the contract owner.errorInvalidReceiverAddress();// Used when the receiver address is 0.// Event emitted when the tokens are transferred to an account on another chain.eventTokensTransferred(bytes32indexedmessageId,// The unique ID of the message.uint64indexeddestinationChainSelector,// The chain selector of the destination chain.addressreceiver,// The address of the receiver on the destination chain.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.);// Mapping to keep track of allowlisted destination chains.mapping(uint64=>bool)publicallowlistedChains;IRouterClientprivates_router;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){s_router=IRouterClient(_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.modifieronlyAllowlistedChain(uint64_destinationChainSelector){if(!allowlistedChains[_destinationChainSelector])revertDestinationChainNotAllowlisted(_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 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{allowlistedChains[_destinationChainSelector]=allowed;}/// @notice Transfer tokens to receiver on the destination chain./// @notice pay in LINK./// @notice the token must be in the list of supported tokens./// @notice This function can only be called by the owner./// @dev Assumes your contract has sufficient LINK tokens to pay for the fees./// @param _destinationChainSelector The identifier (aka selector) for the destination blockchain./// @param _receiver The address of the recipient on the destination blockchain./// @param _token token address./// @param _amount token amount./// @return messageId The ID of the message that was sent.functiontransferTokensPayLINK(uint64_destinationChainSelector,address_receiver,address_token,uint256_amount)externalonlyOwneronlyAllowlistedChain(_destinationChainSelector)validateRece{// 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,_token,_amount,address(s_linkToken));// Get the fee required to send the messageuint256fees=s_router.getFee(_destinationChainSelector,evm2AnyMessage);if(fees>s_linkToken.balanceOf(address(this)))revertNotEnoughBalance(s_linkToken.balanceOf(address(this)),fees);// approve the Router to transfer LINK tokens on contract's behalf. It will spend the fees in LINKs_linkToken.approve(address(s_router),fees);// approve the Router to spend tokens on contract's behalf. It will spend the amount of the given tokenIERC20(_token).approve(address(s_router),_amount);// Send the message through the router and store the returned message IDmessageId=s_router.ccipSend(_destinationChainSelector,evm2AnyMessage);// Emit an event with message detailsemitTokensTransferred(messageId,_destinationChainSelector,_receiver,_token,_amount,address(s_linkToken),fees);// Return the message IDreturnmessageId;}/// @notice Transfer tokens to receiver on the destination chain./// @notice Pay in native gas such as ETH on Ethereum or MATIC on Polgon./// @notice the token must be in the list of supported tokens./// @notice This function can only be called by the owner./// @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 _token token address./// @param _amount token amount./// @return messageId The ID of the message that was sent.functiontransferTokensPayNative(uint64_destinationChainSelector,address_receiver,address_token,uint256_amount)externalonlyOwneronlyAllowlistedChain(_destinationChainSelector)validateRec{// 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,_token,_amount,address(0));// Get the fee required to send the messageuint256fees=s_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(s_router),_amount);// Send the message through the router and store the returned message IDmessageId=s_router.ccipSend{value:fees}(_destinationChainSelector,evm2AnyMessage);// Emit an event with message detailsemitTokensTransferred(messageId,_destinationChainSelector,_receiver,_token,_amount,address(0),fees);// Return the message IDreturnmessageId;}/// @notice Construct a CCIP message./// @dev This function will create an EVM2AnyMessage struct with all the necessary information for tokens transfer./// @param _receiver The address of the receiver./// @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,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:"",// No datatokenAmounts:tokenAmounts,// The amount and type of token being transferredextraArgs:Client._argsToBytes(// Additional arguments, setting gas limit to 0 as we are not sending any dataClient.EVMExtraArgsV1({gasLimit:0})),// 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 transferred 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 transferred.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 and fund your sender contract onEthereum Sepolia:
4. Open MetaMask and select the networkEthereum Sepolia.
5. In Remix IDE, clickDeploy & 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 transfer tokens toPolygon Mumbai:
10. In Remix IDE, underDeploy & Run Transactions, open the list of functions for your smart contract deployed onEthereum Sepolia.
11. Call theallowlistDestinationChainfunction with12532609583862916517as the destination chain selector, andtrueas allowed. Each chain selector is found on thesupported networks page .

Transfer tokens and pay in LINK

You will transfer0.001 CCIP-BnM. 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.Note: The LINK tokens are used to pay for CCIP fees.
2. Transfer CCIP-BnM 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 thetransferTokensPayLINKfunction:

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 account address atPolygon Mumbai.The destination account address. It could be a smart contract or an EOA._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 thetransactbutton and 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 than0.1 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". The data field is empty because you are only transferring tokens. 5. Check the receiver account on the destination chain:

1. Note the destination transaction hash from the CCIP explorer.0x65349bfd4016a3e1feb1bd91a0043315bcc0c356f466fcd463b7db096d33932ein this example.
2. Open the block explorer for your destination chain. ForPolygon Mumbai, openpolygonscan .
3. Search thetransaction hash .
4. Notice in theTokens Transferredsection that CCIP-BnM tokens have been transferred to your account (0.001 CCIP-BnM).

Transfer tokens and pay in native

You will transfer0.001 CCIP-BnM. 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 native gas tokens. You can transfer0.01ETHto your contract.Note: The native gas tokens are used to pay for CCIP fees.
2. Transfer CCIP-BnM 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 thetransferTokensPayNativefunction:

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 account address atPolygon Mumbai.The destination account address. It could be a smart contract or an EOA._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 .._amount10000000000000000The token amount (0.001 CCIP-BnM). 4. Click thetransactbutton and 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". The data field is empty because you only transfer tokens. 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 account on the destination chain:

1. Note the destination transaction hash from the CCIP explorer.0xac12b6e611571a736678b8c69c84f5c6ed20a6b9529b772dc14000389ab0dd8cin this example.
2. Open the block explorer for your destination chain. ForPolygon Mumbai, openpolygonscan .
3. Search thetransaction hash .
4. Notice in theTokens Transferredsection that CCIP-BnM tokens have been transferred to your account (0.001 CCIP-BnM).

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

yarnadd@chainlink/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 a supported token to an account on a destination chain. The contract code contains supporting comments clarifying the functions, events, and underlying logic. This section further explains initializing the contract and transferring tokens.

Initializing of the contract

When you deploy the contract, you define the router address and LINK contract address of the blockchain where you deploy the contract. The contract uses the router address to interact with the router to estimate the CCIP fees and the transmission of CCIP messages.

Transferring tokens and pay in LINK

ThetransferTokensPayLINKfunction 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 empty because you only transfer tokens.
4. ThetokenAmountsis an array, with each element comprising aEVMTokenAmountstruct that contains 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 thetransferTokensPayLINKfunction.
5. TheextraArgsspecifies thegasLimitfor relaying the message to the recipient contract on the destination blockchain. In this example, thegasLimitis set to0because the contract only transfers tokens and does not expect function calls on the destination blockchain.
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, thetransferTokensPayLINKfunction is protected by theonlyAllowlistedChainto ensure the contract owner has allowlisted a destination chain.

Transferring tokens and pay in native

ThetransferTokensPayNativefunction 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 empty because you only transfer tokens.
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 thetransferTokensPayNativefunction.
5. TheextraArgsspecifies thegasLimitfor relaying the message to the recipient contract on the destination blockchain. In this example, thegasLimitis set to0because the contract only transfers tokens and does not expect function calls on the destination blockchain.
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, thetransferTokensPayNativefunction is protected by theonlyAllowlistedChain, ensuring the contract owner has allowlisted a destination chain.