Hello Token

This tutorial contains asolidity contract that can be deployed onto many EVM chains to form a fully functioning cross-chain application with the ability for users to request, from one contract, that tokens are sent to an address on a different chain.

Here is an example of across-chain borrow lending application that uses the topics covered in this tutorial!

Summary

Included in thisrepository is:

- Example Solidity Code
- Example Forge local testing setup
- Testnet Deploy Scripts
- Example Testnet testing setup
-

Environment Setup

- Node 16.14.1 or later, npm 8.5.0 or laterhttps://docs.npmjs.com/downloading-and-installing-node-js-and-npm
- forge 0.2.0 or later:https://book.getfoundry.sh/getting-started/installation
-

Testing Locally

Clone down the repo, cd into it, then build and run unit tests:

```
Copy gitclonehttps://github.com/wormhole-foundation/hello-token.git cdhello-token npmrunbuild forgetest
```

Expected output is

```
Copy Running1testfortest/HelloToken.t.sol:HelloTokenTest [PASS]testCrossChainDeposit()(gas:1338038)
Testresult:ok.1passed;0failed;finishedin5.64s
```

Deploying to Testnet

You will need a wallet with at least 0.05 Testnet AVAX and 0.01 Testnet CELO.

- Obtain testnet AVAX here
- Obtain testnet CELO here
-

```
Copy EVM_PRIVATE_KEY=your_wallet_private_keynpmrundeploy
```

Testing on Testnet

You will need a wallet with at least 0.02 Testnet AVAXObtain testnet AVAX here

You must have also deployed contracts onto testnet (as described in the above section).

To test sending and receiving a message on testnet, execute the test as such:

```
Copy EVM_PRIVATE_KEY=your_wallet_private_keynpmruntest
```

Getting Started

Let's write aHelloToken contract that lets users send an arbitrary amount of an IERC20 token to an address of their choosing on another chain.

Valid Tokens

Before getting started, it is important to note that we use Wormhole'sTokenBridge to transfer tokens between chains!

So, in order to send a token using the method in this example, the token must be attested onto the Token Bridge contract that lives on our desired target blockchain.

In the test above, when you runnpm run deploy , a mock token contract was both deployed and attested onto the target chain's Token Bridge contract.

If you wish to attest a token yourself for the TokenBridge, you may use theattestWorkflow function.

To check if a token already is attested onto a TokenBridge, call thewrappedAsset(uint16 tokenChainId, bytes32 tokenAddress) function on the TokenBridge - this will return, if attested, the address of the wrapped token on this blockchain corresponding to the given token (from the source blockchain), and the 0 address if the input token hasn't been attested yet.

How attestWorkflow works The 'attestWorkflow' function does the following:

1. On the Source side: Calls the TokenBridgeattestToken
2. function with the token we're trying to send.
3. This creates a payload containing the token details so that it may be created on the receiving side
4. Off chain:Fetch the VAA
5. using the Wormhole Chain ID, Emitter address (TokenBridge address) and sequence number from theLogMessage
6. event.
7. This is the VAA that contains the token details with signatures from the Guardians
8. On the Receiving side: Calls the TokenBridgecreateWrapped
9. function with the VAA from the previous step
10. This allows the TokenBridge to create a wrapped version of the token we're sending so that it may mint the tokens to the receiver.
11.

Once this is done, the TokenBridge on the receiving side can successfully mint the token sent.

Wormhole Solidity SDK

To ease development, we'll make use of theWormhole Solidity SDK .

Include this SDK in your own cross-chain application by running:

```

Copy forgeinstallwormhole-foundation/wormhole-solidity-sdk

```

and import it in your contract:

```

Copy import"wormhole-solidity-sdk/WormholeRelayerSDK.sol";

```

This SDK provides helpers that make cross-chain development with Wormhole easier, and specifically provides us with the TokenSender and TokenReceiver abstract classes with useful functionality for sending and receiving tokens using TokenBridge

Implement Sending Function

Lets start by writing a function to send some amount of a token to a specific recipient on a target chain.

```

Copy functionsendCrossChainDeposit( uint16targetChain,// A wormhole chain id addresstargetHelloToken,// address of HelloToken contract on targetChain addressrecipient, uint256amount, addresstoken )publicpayable;

```

The body of this function will send the token as well as a payload to the HelloToken contract on the target chain. For our application, the payload will contain the intended recipient of the token, so that the target chain HelloToken contract can send the token to the intended recipient.

Note: TokenBridge only supports sending IERC20 tokens, and specifically only up to 8 decimals of a token. So, if your IERC20 token has 18 decimals, and you sendamount of a token, you will receiveamount rounded down to the nearest multiple of 10^10.

To send the token and payload to the HelloToken contract, we make use of thesendTokenWithPayloadToEvm helper from the Wormhole Solidity SDK.

For a successful transfer, several things need to happen:

- The user (or contract) who callssendCrossChainDeposit
- shouldapprove
- theHelloToken
- contract to useamount
- of the user's tokens. See how that is done in the forge test[here]
- We must transferamount
- of the token from the user to theHelloToken
- source contractIERC20(token).transferFrom(msg.sender, address(this), amount);
- We must encode the recipient address into a payloadbytes memory payload = abi.encode(recipient);
- We must ensure the correct amount ofmsg.value
- was passed in to send the token and payload.
-
    - The cost to send a token is provided by the value returned bywormhole.messageFee()
-
    - Currently this is 0 butmay
-
    - change in the future, so don't assume it will always be 0.
-
    - The cost to request a relay depends on the gas amount and receiver value you will need.(deliveryCost,) = wormholeRelayer.quoteEVMDeliveryPrice(targetChain, 0, GAS_LIMIT);
- *
-

```
Copy functionsendCrossChainDeposit( uint16targetChain, addresstargetHelloToken, addressrecipient, uint256amount, addresstoken )publicpayable{ uint256cost=quoteCrossChainDeposit(targetChain); require(msg.value==cost, "msg.value != quoteCrossChainDeposit(targetChain)");

IERC20(token).transferFrom(msg.sender,address(this),amount);

bytesmemorypayload=abi.encode(recipient); sendTokenWithPayloadToEvm( targetChain, targetHelloToken,// address (on targetChain) to send token and payload payload, 0,// receiver value GAS_LIMIT, token,// address of IERC20 token contract amount ); }

functionquoteCrossChainDeposit(uint16targetChain) publicviewreturns(uint256cost) { // Cost of delivering token and payload to targetChain uint256deliveryCost; (deliveryCost,)= wormholeRelayer.quoteEVMDeliveryPrice(targetChain,0,GAS_LIMIT);

// Total cost: delivery cost + // cost of publishing the 'sending token' wormhole message cost=deliveryCost+wormhole.messageFee(); }
```

Implement Receiving Function

Now, we'll implement theTokenReceiver abstract class - which is also included in the Wormhole Solidity SDK

```
Copy structTokenReceived{ bytes32tokenHomeAddress; uint16tokenHomeChain; addresstokenAddress; uint256amount; uint256amountNormalized; }

functionreceivePayloadAndTokens( bytesmemorypayload, TokenReceived[]memoryreceivedTokens, bytes32sourceAddress, uint16sourceChain, bytes32deliveryHash )internalvirtual{}
```

After we callsendTokenWithPayloadToEvm on the source chain, the message goes through the standard Wormhole message lifecycle. Once a VAA is available, the delivery provider will callreceivePayloadAndTokens on the target chain and target address specified, with the appropriate inputs.

The argumentspayload ,sourceAddress ,sourceChain , anddeliveryHash are all the same as on the normalreceiveWormholeMessages endpoint.

Let's delve into the fields that are provided to us in theTokenReceived struct:

- tokenHomeAddress
- The same as thetoken
- field in the call tosendTokenWithPayloadToEvm
- , as that is the original address of the token unless the original token sent is a wormhole-wrapped token. In the case a wrapped token is sent, this will be the address of the original version of the token (on it’s native chain) inwormhole address format
-
    - i.e. left-padded with 12 zeros
- tokenHomeChain
- The chain (in wormhole chain ID format) corresponding to the home address above - this will be the source chain, unless if the original token sent is a wormhole-wrapped asset, in which case it will be the chain of the unwrapped version of the token.
- tokenAddress
- This is the address of the IERC20 token on this chain (the target chain) that has been transferred to this contract. If tokenHomeChain == this chain, this will be the same as tokenHomeAddress; otherwise, it will be the wormhole-wrapped version of the token sent.
- amount
- This is the amount of the token that has been sent to you - the units being the same as the original token. Note that since TokenBridge only sends with 8 decimals of precision, if your token had 18 decimals, this will be the ‘amount’ you sent, rounded down to the nearest multiple of 10^10.
- amountNormalized
- This is the amount of token divided by (1 if decimals ≤ 8, else 10^(decimals - 8))
-

Since all we intend to do is send the received token to the recipient, our fields of interest arepayload (containing recipient),receivedTokens[0].tokenAddress (token we received), andreceivedTokens[0].amount (amount of token we received and that we must send)

We can complete the implementation as follows:

```

Copy functionreceivePayloadAndTokens( bytesmemorypayload, TokenReceived[]memoryreceivedTokens, bytes32,// sourceAddress uint16, bytes32// deliveryHash )internaloverrideonlyWormholeRelayer{ require(receivedTokens.length==1,"Expected 1 token transfers");

addressrecipient=abi.decode(payload,(address));

IERC20(receivedTokens[0].tokenAddress).transfer(recipient,receivedTokens[0].amount); }

```

Note: In this case, we don't need to prevent duplicate deliveries using the delivery hash, because TokenBridge already provides a form of duplicate prevention when redeeming sent tokens

And voila! We have acomplete working example of a cross-chain application that uses TokenBridge to send and receive tokens!

Trycloning and running HelloToken to see this example work for yourself!

How do these Solidity Helpers Work?

Let’s walk through the details ofsendTokenWithPayloadToEvm andreceivePayloadAndTokens to see how they make use of the IWormholeRelayer interface and IWormholeReceiver interface to send and receive tokens.

Sending a Token

To send a token, we make use of the EVM TokenBridge contract, specifically thetransferTokensWithPayload method (implementation)

Note: We leave thepayload field here blank because we are using thepayload field on the IWormholeRelayer interface instead

```
```

Copy / ** @notice Send ERC20 token through portal. * * @dev This type of transfer is called a "contract-controlled transfer". * There are three differences from a regular token transfer: * 1) Additional arbitrary payload can be attached to the message * 2) Only the recipient (typically a contract) can redeem the transaction * 3) The sender's address (msg.sender) is also included in the transaction payload * * With these three additional components, xDapps can implement cross-chain * composable interactions.* / functiontransferTokensWithPayload(addresstoken, uint256amount, uint16recipientChain, bytes32recipient, uint32nonce, bytesmemorypayload)publicpayablenonReentrantreturns(uint64sequence)

```

TokenBridge implements this function by publishing a wormhole message to the blockchain logs that indicates thatamount of thetoken was sent (with the intended address beingrecipient onrecipientChain ). TokenBridge then returns the sequence number of this published wormhole message.

ThetransferTokens function in the Wormhole Solidity SDK makes use of this TokenBridge endpoint by

- approving the TokenBridge to spendamount
- of our ERC20token
- callingtransferTokensWithPayload
- with the appropriate inputs
- returning aVaaKey
- struct containing information about the published wormhole message for the token transfer
- 

```

Copy functiontransferTokens(addresstoken, uint256amount, uint16targetChain, addresstargetAddress)internalreturns(VaaKeymemory) { IERC20(token).approve(address(tokenBridge),amount);

uint64sequence=tokenBridge.transferTokensWithPayload {value:wormhole.messageFee()}(token,amount,targetChain,toWormholeFormat(targetAddress),0,bytes(""));

returnVaaKey({ emitterAddress:toWormholeFormat(address(tokenBridge)), chainId:wormhole.chainId(), sequence:sequence }); }

```

Now, it is our task to get the signed VAA corresponding to this published token bridge wormhole message to be delivered to our target chain HelloToken contract. To do this, we make use of thesendVaasToEvm endpoint in the IWormholeRelayer interface.

```

Copy functionsendVaasToEvm(uint16targetChain, addresstargetAddress, bytesmemorypayload, uint256receiverValue, uint256gasLimit, VaaKey[]memoryvaaKeys)externalpayablereturns(uint64sequence);

```

This allows us to specify existing wormhole message(s) and get the signed VAA(s) corresponding to those messages delivered to the targetAddress (in theadditionalVaas field ofreceiveWormholeMessages ).

```

Copy functionsendTokenWithPayloadToEvm(uint16targetChain, addresstargetAddress, bytesmemorypayload, uint256receiverValue, uint256gasLimit, addresstoken, uint256amount)internalreturns(uint64) { VaaKey[]memoryvaaKeys=newVaaKey; vaaKeys[0]=transferTokens(token,amount,targetChain,targetAddress);

(uint256cost,)= wormholeRelayer.quoteEVMDeliveryPrice(targetChain,receiverValue,gasLimit);

returnwormholeRelayer.sendVaasToEvm{value:cost}(targetChain,targetAddress,payload,receiverValue,gasLimit,vaaKeys); }

```

Note: If you wish to send multiple different tokens along with the payload, thesendTokenWithPayloadToEvm helper as currently implemented will not help (as it sends only one token). However, you can still calltransferToken twice and request delivery of both of those TokenBridge wormhole messages by providing twoVaaKey structs in thevaaKeys array. See an example of HelloToken with more than one tokenhere .

Receiving a Token

We know that oursendVaasToEvm call will causereceiveWormholeMessages ontargetAddress to be called with
```

- The payload as the encodedrecipient
- address
- TheadditionalVaas
- field being an array of length 1, with the first element being the signed VAA corresponding to our token bridge transfer
-

Crucially, we don't have the transferred tokens yet! There are a few things that we need to do before gaining access to these tokens.

1. We parse the signed VAA, and check that
2.
    - The emitterAddress of the VAA is a valid token bridge - i.e. the message was published by one of the TokenBridge contracts
3.
    - The transfer was sent to this address
4.
    -
5. note: this step isn't strictly necessary because the call tocompleteTransferWithPayload
6. would fail if these were not true**
7. We calltokenBridge.completeTransferWithPayload
8. , passing the VAA - this completes the transfer of the tokens and causes us to receive the (potentially wormhole-wrapped) transferred token
9. We return aTokenReceived
10. struct containing useful information about the transfer
11. We callreceivePayloadAndTokens
12. with the appropriate inputs
13.

```

Copy functionreceiveWormholeMessages( bytesmemorypayload, bytes[]memoryadditionalVaas, bytes32sourceAddress, uint16sourceChain, bytes32deliveryHash )externalpayable{ TokenReceived[]memoryreceivedTokens= newTokenReceived;

for(uint256i=0; i<additionalVaas.length;++i) { IWormhole.VMmemoryparsed=wormhole.parseVM(additionalVaas[i]); require( parsed.emitterAddress==tokenBridge.bridgeContracts(parsed.emitterChainId),"Not a Token Bridge VAA" ); ITokenBridge.TransferWithPayloadmemorytransfer=tokenBridge.parseTransferWithPayload(parsed.payload); require( transfer.to==toWormholeFormat(address(this))&&transfer.toChain==wormhole.chainId(), "Token was not sent to this address" );

tokenBridge.completeTransferWithPayload(additionalVaas[i]);

addressthisChainTokenAddress=getTokenAddressOnThisChain(transfer.tokenChain,transfer.tokenAddress); uint8decimals=getDecimals(thisChainTokenAddress); uint256denormalizedAmount=transfer.amount; if(decimals>8) denormalizedAmount=*uint256(10)**(decimals-8);

receivedTokens[i]=TokenReceived({ tokenHomeAddress:transfer.tokenAddress, tokenHomeChain:transfer.tokenChain, tokenAddress:thisChainTokenAddress, amount:denormalizedAmount, amountNormalized:transfer.amount }); }

// call into overriden method
receivePayloadAndTokens(payload,receivedTokens,sourceAddress,sourceChain,deliveryHash); }

```

See the full implementation of the Wormhole Relayer SDK helpershere

Also, see a version of HelloToken implemented without any Wormhole Relayer SDK helpershere

as well as a version of HelloToken where native currency is depositedhere

Wormhole integration complete?

Let us know so we can list your project in our ecosystem directory and introduce you to our global, multichain community!

Reach out now!

Last updated1 month ago