Specialized Relayer

?

Wormhole is compatible with manyecosystems and integration is straight forward.

On Chain

In order to send and receive messages between chains, someon chain components are important to understand.

Sending a message

To send a message, regardless of the environment or chain, the core contract is invoked with a message argument from anemitter .

Thisemitter may be your contract or an existing application such as theToken Bridge , orNFT Bridge .

EVM Solana Using theIWormhole interface (source), we can publish a message directly to thecore contract .

```

Copy // ...

IWormhole wormhole=IWormhole(wormholeAddr);

// Get the fee for publishing a message uint256wormholeFee=wormhole.messageFee();

// ... fee/value checks

// create the HelloWorldMessage struct HelloWorldMessagememoryparsedMessage=HelloWorldMessage({ payloadID:uint8(1), message:helloWorldMessage });

// encode the HelloWorldMessage struct into bytes bytesmemoryencodedMessage=encodeMessage(parsedMessage);

// Send the HelloWorld message by calling publishMessage on the // Wormhole core contract and paying the Wormhole protocol fee. messageSequence=wormhole.publishMessage{value:wormholeFee}( 0,// batchID encodedMessage, wormholeFinality() );

```

More details inExample Source Using thewormhole_anchor_sdk::wormhole module and given the wormhole program account, we can pass a message directly to the core contract.

```

Copy // ...

letfee=ctx.accounts.wormhole_bridge.fee(); // ... fee check/send

letconfig=&ctx.accounts.config; letpayload:Vec=HelloWorldMessage::Hello{ message }.try_to_vec()?;

// Invoke wormhole::post_message. wormhole::post_message( CpiContext::new_with_signer( ctx.accounts.wormhole_program.to_account_info(), wormhole::PostMessage{ // ... set fields }, &[ // ... set seeds ], ), config.batch_id, payload, config.finality.into(), )?;

// ...

```

More details inExample Source Once the message is emitted from the core contract, theGuardian Network will observe the message and sign the digest of an Attestation (VAA). We'll discuss this in more depth in theOff Chain section below.

By default, VAAs aremulticast . This means there is no defaulttarget chain for a given message. It's up to the application developer to decide on the format of the message and its treatment on receipt.

Receiving a message

The way a message may be received depends on the environment.

EVM Solana On EVM chains, the message passed is the raw VAA encoded as binary.

It hasnot been verified by the core contract and should be treated as untrusted input untilparseAndVerifyVM has been called.

```

Copy functionreceiveMessage(bytesmemoryencodedMessage)public{ // call the Wormhole core contract to parse and verify the encodedMessage ( IWormhole.VMmemorywormholeMessage, boolvalid, stringmemoryreason )=wormhole().parseAndVerifyVM(encodedMessage);

// ... safety checks

// decode the message payload into the HelloWorldMessage struct HelloWorldMessagememoryparsedMessage=decodeMessage( wormholeMessage.payload );

// ... do something cool }

```

More details inExample Source On Solana, the VAA is first posted and verified by the core contract, after which it can be read by the receiving contract and action taken.

```

Copy pubfnreceive_message(ctx:Context, vaa_hash:[u8;32])->Result<()> { letposted_message=&ctx.accounts.posted;

ifletHelloWorldMessage::Hello{ message }=posted_message.data() { // ... check message // ... do something cool // Done Ok(()) }else{ Err(HelloWorldError::InvalidMessage.into()) } }

```

More details inExample Source In addition to environment specific checks that should be performed, a contract should take care to check otherfields in the body such as:

- Emitter
- : Is this coming from an emitter address and chain id I expect? Typically contracts will provide a method to register a new emitter and check the incoming message against the set of emitters it trusts.
- Sequence
- : Is this the sequence number I expect? How should I handle out of order deliveries?
- Consistency Level
- : For the chain this message came from, is theconsistency level
- enough to guarantee the transaction wont be reverted after I take some action?
-

Outside of body of the VAA, but also relevant, is the digest of the VAA which can be used for replay protection by checking if the digest has already been seen.

Since the payload itself is application specific, there may be other elements to check to ensure safety.

Off Chain

In order to shuttle messages between chains, someoff chain processes are involved. TheGuardians observe the events from the core contract and sign aVAA .

After enough Guardians have signed the message (at least2/3 + 1 or 13 of 19 guardians), the VAA is available to be delivered to a target chain.

Once the VAA is available, aRelayer may deliver it in a properly formatted transaction to the target chain.

Specialized Relayer

A relayer is needed to deliver the VAA containing the message to the target chain. When the relayer is written specifically for a custom application, it's referred to as aSpecialized Relayer

A specialized relayer might be as simple as an in browser process that polls theAPI for the availability of a VAA after submitting a transaction and delivers it to the target chain. It might also be implemented with aSpy coupled with some daemon listening for VAAs from a relevantchainID andemitter then taking action when one is observed.

Simple Relayer

Regardless of the environment, in order to get the VAA we intend to relay, we need:

1. Theemitter
2. address
3. Thesequence
4. id of the message we're interested in

5. ThechainId
6. for the chain that emitted the message
7.

With these three components, we're able to uniquely identify a VAA and fetch it from theAPI .

Fetching the VAA

Using thegetSignedVAAWithRetry function provided in theSDK , we're able to poll the Guardian RPC until the signed VAA is ready.

```

Copy import{ getSignedVAAWithRetry, parseVAA, CHAIN_ID_SOLANA, CHAIN_ID_ETH, }from"@certusone/wormhole-sdk";

constRPC_HOSTS=[/ .../]

asyncfunctiongetVAA(emitter:string,sequence:string,chainId:number):Promise { // Wait for the VAA to be ready and fetch it from // the guardian network const{vaaBytes}=awaitgetSignedVAAWithRetry( RPC_HOSTS, chainId, emitter, sequence ) returnvaaBytes }

constvaaBytes=awaitgetVAA('0xdeadbeef',1,CHAIN_ID_ETH);

```

Once we have the VAA, the delivery method is chain dependent.

EVM Solana On EVM chains, the bytes for the VAA can be passed directly as an argument to an ABI method.

```

Copy // setup eth wallet constethProvider=newethers.providers.StaticJsonRpcProvider(ETH_HOST); constethWallet=newethers.Wallet(WALLET_PRIVATE_KEY,ethProvider);

// create client to interact with our target app constethHelloWorld=HelloWorld__factory.connect( '0xbeefdead', ethWallet );

// invoke the receiveMessage on the ETH contract // and wait for confirmation constreceipt=awaitethHelloWorld .receiveMessage(vaaBytes) .then((tx:ethers.ContractTransaction)=>tx.wait()) .catch((msg:any)=>{ console.error(msg); returnnull; });

Copy

import{CONTRACTS}from'@certusone/wormhole-sdk'

exportconstWORMHOLE_CONTRACTS=CONTRACTS[NETWORK]; exportconstCORE_BRIDGE_PID=newPublicKey(WORMHOLE_CONTRACTS.solana.core);

// First, post the VAA to the core bridge awaitpostVaaSolana( connection, wallet.signTransaction, CORE_BRIDGE_PID, wallet.key(), vaaBytes );

constprogram=createHelloWorldProgramInterface(connection,programId); constparsed=isBytes(wormholeMessage) ? parseVaa(wormholeMessage) :wormholeMessage;

constix=program.methods .receiveMessage([...parsed.hash]) .accounts({ payer:newPublicKey(payer), config:deriveConfigKey(programId), wormholeProgram:newPublicKey(wormholeProgramId), posted:derivePostedVaaKey(wormholeProgramId,parsed.hash), foreignEmitter:deriveForeignEmitterKey(programId,parsed.emitterChain), received:deriveReceivedKey( programId, parsed.emitterChain, parsed.sequence ), }) .instruction();

consttransaction=newTransaction().add(ix); const{blockhash}=awaitconnection.getLatestBlockhash(commitment); transaction.recentBlockhash=blockhash; transaction.feePayer=newPublicKey(payerAddress);

constsigned=awaitwallet.signTxn(transaction); consttxid=awaitconnection.sendRawTransaction(signed);

awaitconnection.confirmTransaction(txid);

``` See theSpecialized Relayer Tutorial for a detailed walkthrough.

Reference

Read more about the architecture and core componentshere

Last updated1 month ago

On this page * On Chain * Sending a message * Receiving a message * Off Chain * Specialized Relayer * Reference

Was this helpful? Edit on GitHub