Cross-Chain Bridge

Overview

The OHM token is available cross-chain! Olympus utilizes LayerZero, a messaging protocol, to send and receive native OHM on supported chains. Whereas most bridge architectures create a wrapped representation of a token, Olympus takes a different approach: OHM is burned on one chain, and minted on another chain. The minted OHM is native to that chain which makes it composable with every dApp.

How to bridge

1. Navigate tohttps://app.olympusdao.finance
2. Click Bridge in the sidebar
3. Select the source chain and the destination chain
4. Enter amounts, approve and click Bridge. Note that neither Olympus nor LayerZero charge a fee for bridging. You only pay for gas and message passing fee.
5. You can view the transaction under the Transactions list.

Supported networks

Arbitrum; other chains soon.

Mechanism

When sending OHM from a source chain (e.g. mainnet), CrossChainBridge smart contract invokes MINTR module to burn OHM on the source chain and send a message payload over LayerZero Endpoint.

function

sendOhm (uint16 dstChainId_ ,

address to_ ,

uint256 amount_)

external

payable

{ if

(! bridgeActive)

revert

Bridge_Deactivated ( ) ; if

(ohm . balanceOf (msg . sender)

< amount_)

revert

Bridge_InsufficientAmount ( ) ;

bytes

memory payload = abi . encode (to_ , amount_) ;

MINTR . burnOhm (msg . sender , amount_) ;*sendMessage (dstChainId* , payload ,

payable (msg . sender) ,

address (0x0) ,

bytes ("") , msg . value) ;

emit

BridgeTransferred (msg . sender , amount_ , dstChainId_) ; } When the message is received by the destination chain (e.g. Arbitrum), CrossChainBridge smart contract mints native OHM on the source chain.

function

receiveMessage (uint16 srcChainId, bytes

memory , uint64 , bytes

memory payload_)

internal

{ (address to ,

uint256 amount)

= abi . decode (payload_ ,

(address ,

uint256)) ;

MINTR . increaseMintApproval (address (this) , amount) ; MINTR . mintOhm (to , amount) ;

emit

BridgeReceived (to , amount , srcChainId_) ; } Once bridged, users can use native OHM in any protocol that accepts OHM.

Security

Olympus CrossChainBridge smart contract was reviewed by the LayerZero integrations team and audited by OtterSec. The audit is available here.

To consider the pros and cons of Olympus' approach to bridging, it's worth understanding the difference between native tokens and non-native tokens. Native tokens (to a chain) are those tokens that are deployed by the smart contract; non-native tokens are those that are wrapped and managed by a third-party.

Pros

Under the non-native bridge design, source tokens are stored in bridge contracts and wrapped representations are minted. This makes the contract a single point-of-attack, and why bridges account for ~50% of all DeFi exploits.

Olympus' bridge native bridge design avoids this problem by deploying a MINTR contract to mint/burn OHM on each chain it's integrated with. This makes OHM a truly native asset on each chain it's deployed on.

Cons

Native bridge architecture is not without risk. LayerZero tech uses an Oracle to pass block headers, and a Relayer to pass proofs between chains. An Oracle/Relayer collusion could manipulate block header and proof data. This risk can be mitigated by having Olympus run its own Oracle/Relayer.

Contracts

Ethereum

Contract Address CrossChainBridge 0x45e563c39cddba8699a90078f42353a57509543a

Arbitrum

Contract Address CrossChainBridge 0x20B3834091f038Ce04D8686FAC99CA44A0FB285c Edit this page Previous Cooler Loans Next Intro * Overview * How to bridge * Supported networks * Mechanism * Security * * Pros * * Cons * Contracts * * Ethereum * * Arbitrum