Security Stack Configuration

When configuring your OApp'sSecurity Stack , you configure it on a per-chain pathway basis. It's essential to ensure that the same DVN is available on both the source and destination chains.

For example, if a DVN is only on Chain A, but not Chain B, you'd need to choose an alternative DVN that's supported on both chains for that pathway.Check DVN deployments here .

Remember, the objective is to ensure that the DVN setup supports your OApp's chain pathways, as they are the conduits for message routing. This is vital for efficient, error-free cross-chain communication.

Parameters

Below are the possible configurations you can customize in your OApp's Security Stack:

1. confirmations
2. Type
3. :uint64
4. Description
5. : The number of block confirmations to wait before a DVN should listen for thepayloadHash
6. . This setting can be used to ensure message finality on chains with frequent block reorganizations.
7. caution
8. If you set your block confirmations too low, and a reorg occurs after your confirmation, it can materially impact yourOApp
9. orOFT
10. .
11. requiredDVNCount
12. Type
13. :uint8
14. Description
15. : The quantity of required DVNs that will be used in your OApp.
16. optionalDVNCount
17. Type
18. :uint8
19. Description
20. : The quantity of optional DVNs that will be used in your OApp.
21. optionalDVNThreshold
22. Type
23. :uint8
24. Description
25. : The minimum number of verifications needed from optional DVNs. A message is deemed Verifiable if it receives verifications from at least the number of optional DVNs specified by theoptionalDVNsThreshold
26. , plus the required DVNs.
27. requiredDVNs
28. Type
29. :address[]
30. Description
31. : An array of addresses for all required DVNs.
32. optionalDVNs
33. Type

34. :address[]
35. Description
36. : An array of addresses for all optional DVNs.

Set Config

To use a custom Security Stack, the OApp owner or assigned delegate must call setConfig from the OApp's Endpoint.

info Eachconfig is specified per channel pathway (ie. from the current source to destination). Therefore, it's imperative to apply the same ULN configuration to both the send (source) and receive (destination) libraries of a given chain. This means adjusting the message library address (_lib) as necessary while maintaining consistentSetConfigParam settings.

Below is an an example configuration setup to see how to customize Security Stack.

Define Parameters

// Using ethers v5

const confirmations =

6 ;

// Arbitrary; Varies per remote chain const optionalDVNsThreshold =

1 ; const requiredDVNs =

['0xRequiredVerifier1' ,

'0xRequiredVerifier2'] ; const optionalDVNs =

['0xOptionalVerifier1' ,

'0xOptionalVerifier2'] ; const requiredDVNsCount = requiredDVNs . length ; const optionalDVNsCount = optionalDVNs . length ; // Configuration type const configTypeUln =

2 ;

// As defined for CONFIG_TYPE_ULN

Encode Parameters to Bytes

const ulnConfigStructType = 'tuple(uint64 confirmations, uint8 requiredDVNCount, uint8 optionalDVNCount, uint8 optionalDVNThreshold, address[] requiredDVNs, address[] optionalDVNs)' ;

const ulnConfigData =

{ confirmations , requiredDVNCount , optionalDVNCount , optionalDVNThreshold , requiredDVNs , optionalDVNs , } ; const ulnConfigEncoded = ethersV5 . utils . defaultAbiCoder . encode ([ulnConfigStructType] , [ulnConfigData] ,) ;

DefineSetConfigParam

const setConfigParamUln =

{ eid :

REMOTE_CHAIN_ENDPOINT_ID ,

// Replace with your remote chain's endpoint ID (source or destination) configType : configTypeUln , config : ulnConfigEncoded , } ;

CallsetConfig

const tx =

await endpointContract . setConfig (oappAddress , messageLibraryAddress ,

[setConfigParamUln ,]) ;

await tx . wait ( ) ; All set! Now your OApp will require confirmation from 2 required DVNs and 1 of 2 optional DVNs before a message can be successfully delivered from source to destination.

To save gas, you can seamlessly configure multiple chain paths in a single function call by including multipleSetConfigParam structs within the array argument ofsetConfig . Just ensure that each struct has a uniqueeid for its remote chain, along with the appropriately encoded configuration parameters for that specific chain.

// Call setConfig with multiple configurations const tx =

await oappContract . setConfig (oappAddress , messageLibraryAddress ,

[setUConfigParamUlnChainA ,

// Configuration for the first destination chain setConfigParamUlnChainB ,

// Configuration for the second destination chain // ... add as many configurations as needed]) ;

await tx . wait ( ) ;

Resetting Configurations

Resetting configurations involves callingsetConfig and passing in parameters with nil values. Here's how you can do it:

1. Encode Nil Parameters to Bytes

// Define nil values for int and address array types const confirmations =

0 ; const optionalDVNCount =

0 ; const requiredDVNCount =

0 ; const const optionalDVNThreshold =

0 ; const requiredDVNs =

[ ] ; const optionalDVNs =

[ ] ;

const ulnConfigStructType = 'tuple(uint64 confirmations, uint8 requiredDVNCount, uint8 optionalDVNCount, uint8 optionalDVNThreshold, address[] requiredDVNs, address[] optionalDVNs)'

// Configuration type remains the same const configTypeUln =

2 ;

// As defined for CONFIG_TYPE_ULN

const ulnConfigData =

{ confirmations , requiredDVNCount , optionalDVNCount , optionalDVNThreshold , requiredDVNs , optionalDVNs , } ; const ulnConfigEncoded = ethersV5 . utils . defaultAbiCoder . encode ([ulnConfigStructType] , [ulnConfigData]) ;

const resetConfigParamUln =

{ eid :

REMOTE_CHAIN_ENDPOINT_ID ,

// Replace with the endpoint ID you want to reset configType : configTypeUln , config : nilUlnConfigEncoded , } ; 1. CallsetConfig 2. for Reset

const resetTx =

await endpointContract . setConfig (oappAddress , messageLibraryAddress ,

[resetConfigParamUln ,]) ; await resetTx . wait ( ) ;

Snapshotting Configurations

Snapshotting ULN configurations involves:

- Locking your send and receive libraries to the defaults by callingsetSendLibrary
- andsetReceiveLibrary
- respectively, on the endpoint.
- note
- When configuring your OApp's ULN Config, setting your send and receive libraries is crucial. If they are not explicitly set when snapshotting configurations, any updates to LayerZero's default message libraries will reset your configurations. This ensures your OApp remains consistent with your security and functionality requirements, safeguarding against unintended changes from default library updates.
- CallinggetConfig
- and passing in the decoded default values intosetConfig
- to lock configuration settings for an OApp.

Locking the Send Library

const sendTx =

await endpointContract . setSendLibrary (oappAddress , eid , sendLibAddress) ; await sendTx . wait ( ) ;

Locking the ULN Config

1. CallgetConfig
2. from the Endpoint

Use thegetConfig function of the endpoint contract to retrieve the current configuration. This function requires the OApp address (_oapp), library address (_lib), endpoint ID (_eid), and configuration type (_configType). It returns the configuration as a bytes array.

Solidity

function

getConfig (address _oapp , address _lib , uint32 _eid , uint32 _configType)

external

view

returns

(bytes

memory config) ; Implementation

const ulnConfigBytes =

await contract . getConfig (oappAddress , libAddress , eid , configType) ; 1. Decode thegetConfig 2. bytes array

// Define the ABI for the UlnConfig struct const ulnConfigStructType =

['tuple(uint64 confirmations, uint8 requiredDVNCount, uint8 optionalDVNCount, uint8 optionalDVNThreshold, address[] requiredDVNs, address[] optionalDVNs)' ,] ;

const ulnConfigArray = ethers . utils . defaultAbiCoder . decode (ulnConfigStructType , ulnConfigBytes) ; 1. Encode Default Parameters to Bytes

// Configuration type remains the same for Executor const configTypeUln =

2 ;

// As defined for CONFIG_TYPE_ULN const ulnConfig = ulnConfigArray [0] ;

const ulnConfigData =

{ ulnConfig . confirmations , ulnConfig . requiredDVNCount , ulnConfig . optionalDVNCount , ulnConfig . optionalDVNThreshold , ulnConfig . requiredDVNs , ulnConfig . optionalDVNs , } ;

const snapshotUlnConfigEncoded = ethersV5 . utils . defaultAbiCoder . encode ([ulnConfigStructType] , [ulnConfigData]) ;

const snapshotConfigParamUln =

{ eid :

DEST_CHAIN_ENDPOINT_ID , configType : ulnConfigType , config : snapshotUlnConfigEncoded , } ; 1. CallsetConfig 2. for Snapshot

const snapshotTx =

await endpointContract . setConfig (oappAddress , messageLibraryAddress ,

[snapshotConfigParamUln ,]) ; await snapshotTx . wait ( ) Edit this page

Previous OApp Config Next Executors